# Relatório do Lab 3 de CT-213

## Trabalho 3 – Otimização com métodos de Busca Local

## Com *Gradient Descent, Hill Climbing* e *Simulated Annealing*


**Aluno:**

Bruno Costa Alves Freire

**Turma:**

T 22.4

**Professor:**

Marcos Ricardo Omena de Albuquerque Máximo


Data:

30/03/2019


**Instituto Tecnológico de Aeronáutica – ITA**
**Departamento de Computação**

# 1. Implementação dos algoritmos

## 1.1 Gradient Descent

A implementação do algoritmo da Descida do Gradiente foi feita conforme mostrada nos slides do curso. A cada iteração da descida, o passo de atualização do vetor de parâmetros `theta` é o mesmo, de valor constante `alpha`. Além disso, em cada iteração é armazenado na lista `history` o novo valor do vetor de parâmetros, para posterior plotagem.

As condições de parada seguidas são a limitação da função de custo (verifica-se se a função de custo é menor que um limiar `epsilon` dado, caso o seja, o algoritmo é encerrado) e o número de iterações, limitado pelo hiperparâmetro `max_iterations`.

O histórico dos pontos visitados pelo algoritmo, bem como o ponto de parada, para um chute inicial padronizado na origem, são plotados no gráfico da figura 1.

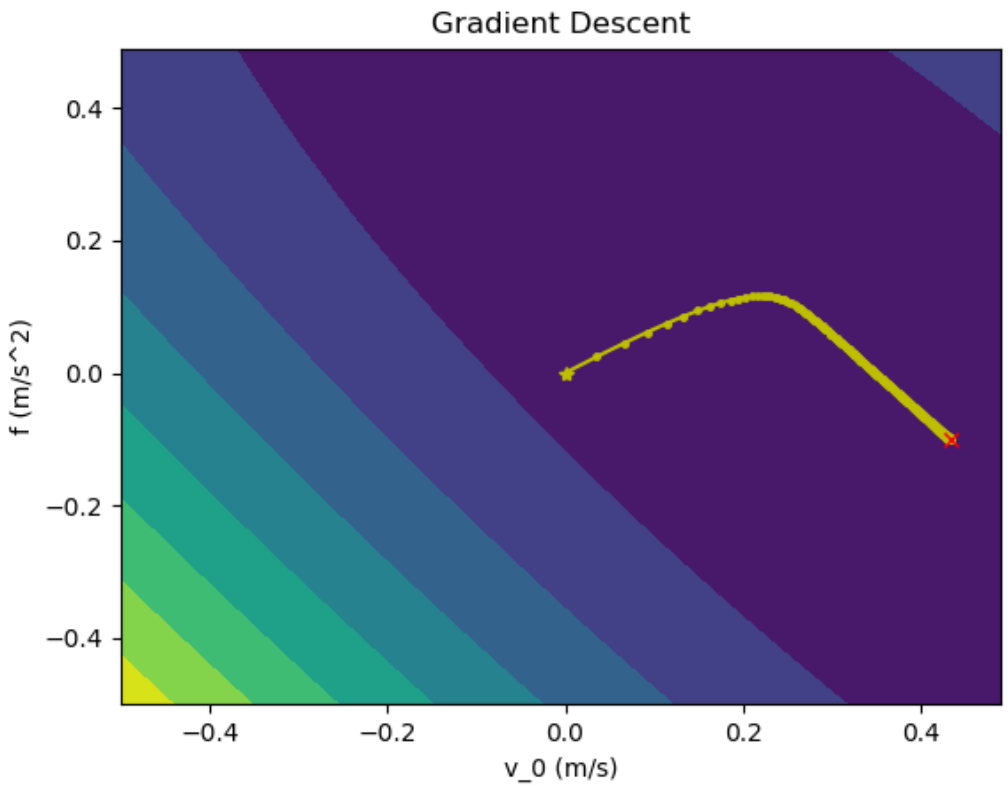


Figura 1: Plot da busca pela solução ótima com o algoritmo Gradient Descent

### 1.2 Hill Climbing

A implementação do algoritmo Hill Climbing (para minimização) foi feita conforme apresentado nos slides do curso, com as modificações necessárias para a minimização.

A função `neighbors` foi implementada de modo a retornar um octeto de vizinhos a uma distância `delta` do vetor `theta`, distribuídos ao longo da circunferência com um espaçamento angular uniforme (de $\pi/4$ radianos entre cada vizinho).

A iteração do método é realizada com as mesmas condições de parada do método Gradient Descent. Um detalhe da implementação é que, como a cada iteração é inicializada uma variável `best` como `None`, não é possível passa-la como argumento para a função `cost_function`. Para driblar essa tecnicalidade, foi utilizada uma variável auxiliar `cost_for_best`, que armazena o valor da função de custo do melhor vizinho.

Os demais detalhes da implementação seguem totalmente o código apresentado no curso, com a adição do armazenamento do histórico dos pontos visitados pelo método, para posterior plot.

O caminho percorrido pelo algoritmo até encontrar a solução ótima foi plotado no gráfico da figura 2, tendo como ponto inicial o mesmo do algoritmo anterior (chute inicial na origem).

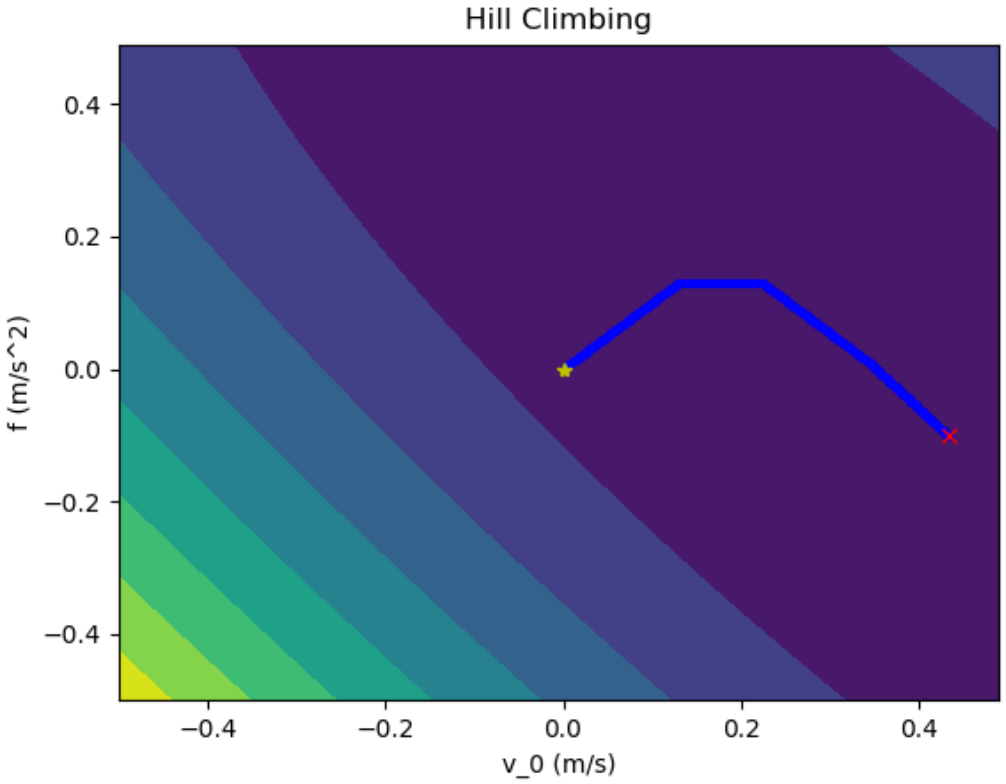


Figura 2: Plot da busca pela solução ótima com o algoritmo Hill Climbing

### 1.3 Simulated Annealing

A implementação do algoritmo Simulated Annealing foi feita conforme pedido no roteiro do laboratório. A função de escolha do vizinho aleatório a cada iteração, `random_neighbor`, foi implementada escolhendo-se um ângulo uniformemente aleatório entre -π e π, e uma distância `delta`. A função de decrescimento da temperatura, `schedule`, foi implementada segundo a fórmula:

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

Na iteração do método, as mesmas condições de parada dos demais métodos são checadas. A diferença de implementação para que o algoritmo realize minimização será na variável `delta_e`, que mede a variação da função de custo entre o vizinho sorteado e o ponto atual. A troca será realizada se `delta_e` for negativo, em vez de positivo. Por esse mesmo motivo, na etapa de decidir se um vizinho pior que o ponto atual deve ser visitado, deve-se comparar um valor aleatório entre 0 e 1 com a exponencial de `-delta_e/temperature`.

No mais, a cada iteração se registra o histórico dos pontos visitados pelo método. A curva de evolução do algoritmo até a solução ótima é plotada na figura 3.

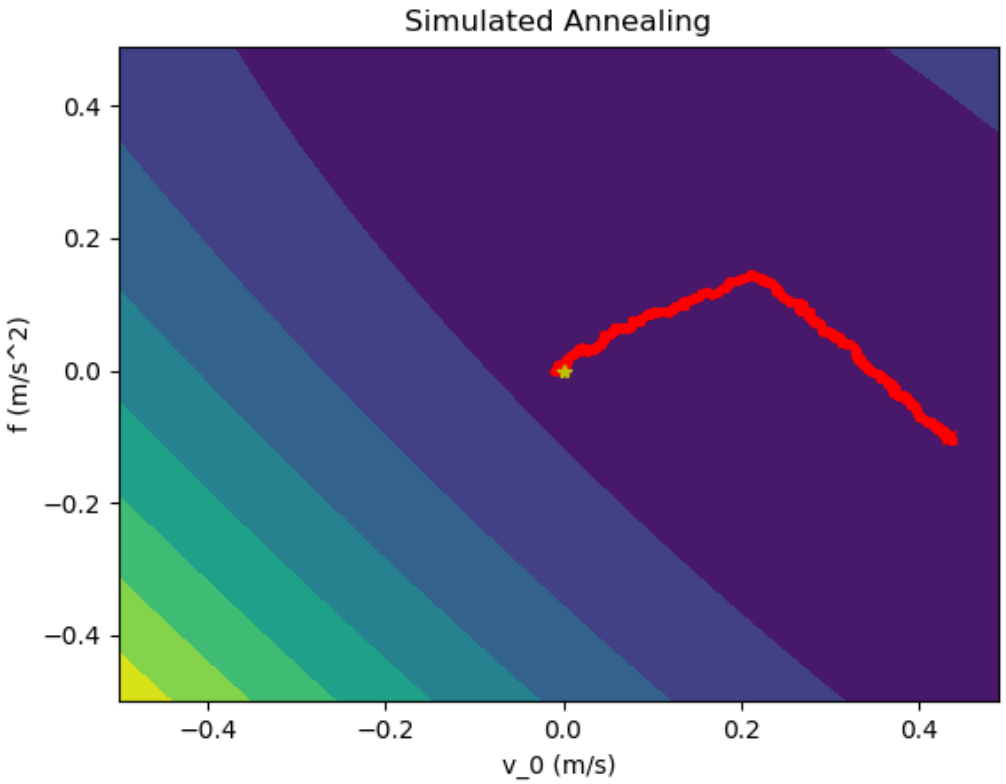


Figura 3: Plot da busca pela solução ótima com o algoritmo Simulated Annealing

# 2. Comparação dos Resultados

Após a resolução do problema com cada um dos algoritmos, para fins de comparação é feito um plot com todas as curvas de evolução no mesmo gráfico, o qual é mostrado na figura 4.

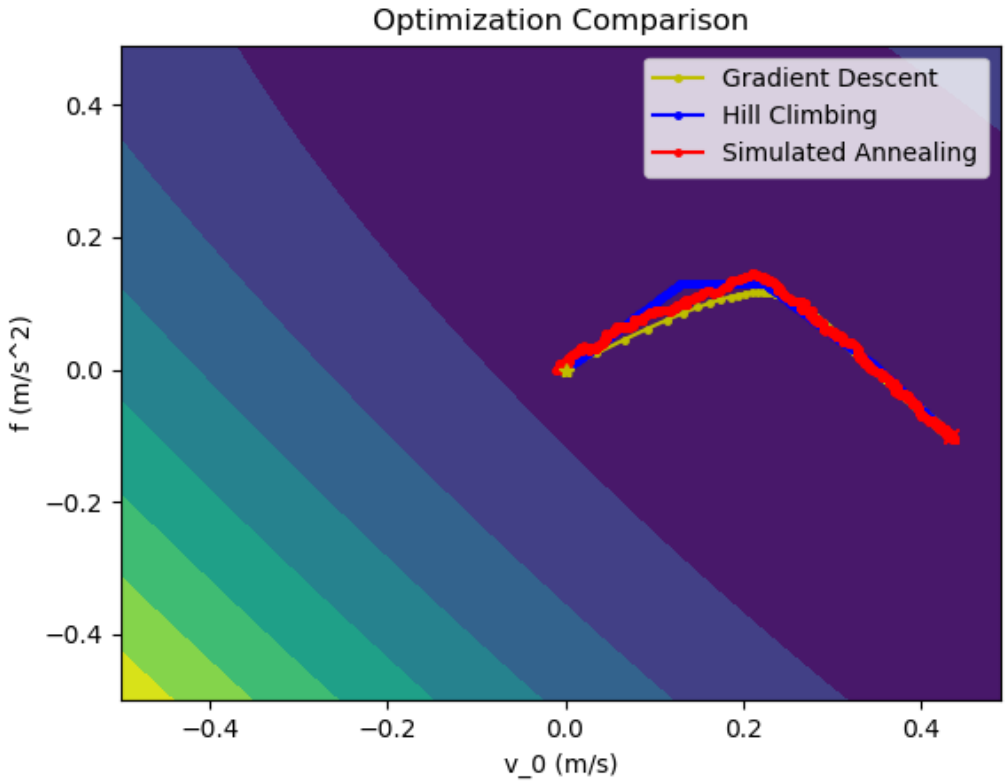


Figura 4: Comparação dos caminhos de cada algoritmo até encontrar a solução ótima.

Para uma comparação mais profunda, as soluções finais de cada algoritmo foram compiladas na tabela 1, juntamente com a solução encontrada pelo método de Quadrados Mínimos (Least Squares Solution).

Tabela 1: Parâmetros encontrados pelos métodos

| **Algoritmo** | $\theta_0 = v_0$ | $\theta_1 = f$ |
|---|---|---|
| *Least Squares Solution* | 0.43337277 | -0.10102096 |
| *Gradient Descent* | 0.43337067 | -0.10101846 |
| *Hill Climbing* | 0.43341125 | -0.10119596 |
| *Simulated Annealing* | 0.43397656 | -0.10134529 |

A partir da tabela 1, podemos notar que as soluções encontradas por todos os métodos diferem apenas na quarta casa após a vírgula em cada

componente, ou seja, uma discrepância da ordem de $10^{-3}$. Calculando a função de custo, vemos que a distância (ao quadrado) entre as soluções fica da ordem de $10^{-10}$, que é o valor do hiperparâmetro `epsilon`. Depurando o código, é possível constatar que apenas o método Hill Climbing parou por ter atingido o limiar da função de custo, enquanto os outros dois pararam por exaurir a quantidade de iterações.

Na figura 5 podemos comparar o *fit* obtido por cada um dos métodos (e da solução analítica, dada pelo Least Squares), com o conjunto de dados.

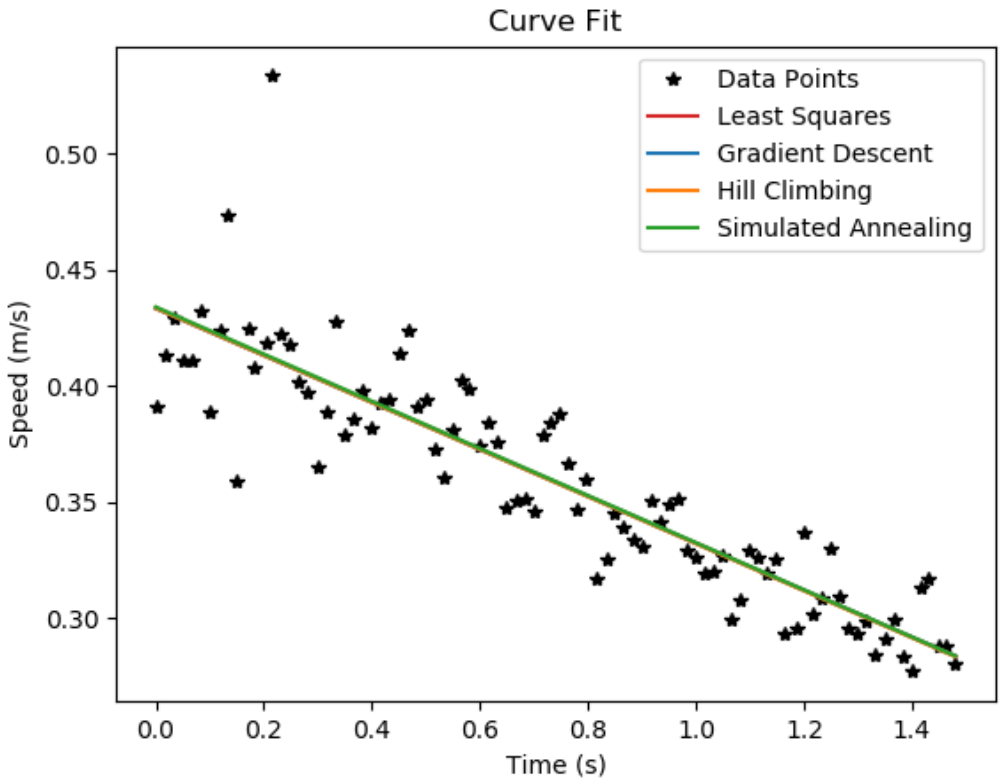


Figura 5: Comparação do *fit* produzido pelos métodos e pela solução analítica.

Pela figura 5, podemos ver uma sobreposição quase total das retas obtidas por todos os métodos, evidenciando a concordância entre as soluções obtidas.

Note que, em todos os gráficos deste relatório, as cores das curvas e das legendas foram alteradas (propositalmente), por meio de uma mudança no método `plot_optimization`, no arquivo `ball_fit.py`. A alteração foi a inserção de um parâmetro de cor para personalizar o plot. O motivo dessa alteração é que, originalmente, o gráfico obtido pela implementação teria exatamente o mesmo aspecto que o gráfico constante no roteiro do laboratório, na figura 1 do mesmo, de forma que seria impossível determinar se a figura gerada pela minha implementação de fato havia sido gerada no código ou apenas copiada do roteiro.