Instituto Tecnológico de Aeronáutica - ITA
CT-213 - Inteligência Artificial aplicada à Robótica Móvel
Aluno: Ulisses Lopes da Silva

Relatório do Laboratório 3 - Otimização com Métodos de Busca Local

---

1 Breve Explicação em Alto Nível da Implementação

1.1 Descida do Gradiente

O método de descida do gradiente baseia-se em pequenas iterações subtrativas sobre o gradiente da função de custo, o qual consiste no vetor de derivadas de J em relação a θ_0 e em relação a θ_1, que são as *features* a serem otimizadas. No caso do problema em questão, as iterações são subtrativas porque estamos buscando minimizar o custo. Dessa forma, deseja-se levar o vetor gradiente de um ponto qualquer até o ponto de mínimo da função de custo que, neste caso, possui apenas um mínimo local, já que é quadrática, conforme abaixo:

$$J(x) = \frac{1}{2m} \sum_{i}^{m} (f(x_i) - y_i)^2$$

As iterações "deslocam"o vetor até o ponto de mínimo local por meio de iterações e um critério de parada, que pode ser um erro ϵ aceitável em relação ao valor real, o número de iterações, a magnitude da variação do gradiente etc., ou até todos eles. As equações iterativas seguem abaixo:

$$\theta_{0_{n+1}} = \theta_{0_n} - \alpha \frac{1}{m} \sum_{i}^{m} (f(x_i) - y_i)$$

$$\theta_{1_{n+1}} = \theta_{1_n} - \alpha \frac{1}{m} \sum_{i}^{m} (f(x_i) - y_i) x_i$$

Quando as *features* são ajustadas te tal forma que o custo chega ao seu mínimo (ou o mais próximo deste), as iterações se encerram. Por ser um método que fica "preso"em mínimos locais (perfil de *exploitation*, ele é mais adequado para *fitting* de funções quadráticas, pois estas possuem apenas um ponto de mínimo ou de máximo.

Conforme se observa na figura do seu resultado de exploração, a Descida do Gradiente é um método bastante uniforme na convergência, sobretudo por ser um método analítico-matemático, no sentido de que usa propriedades matemáticas mais "exatas"para cálculo do custo e *fitting* das *features*. Cabe ressaltar, contudo, que essa uniformidade pode não ocorrer, caso seja escolhido um α (hiperparâmetro) ruim, fazendo o gradiente oscilar no ponto de mínimo, nunca convergindo de fato.

1.2 *Hill Climbing*

O método do algoritmo *Hill Climbing* se baseia em avaliar um certo número de vizinhos num determinado raio, a partir do "chute"inicial. No caso do problema que foi desenvolvido no presente laboratório, desejava-se minimizar o custo. Dessa forma, o algoritmo foi ajustado para que, nas iterações, os vizinhos escolhidos tivessem um custo menor em relação ao ponto da iteração anterior.

Escolheu-se o padrão de oito-conectado, que é relativamente fácil de se calcular (pouco custo computacional), e também fornece direção e sentido razoáveis para onde os pontos devem convergir. além disso, de modo semelhante ao método de descida do Gradiente, os critérios de parada estabelecidos foram o número de iterações e o erro ϵ pré-definido.

Para uso na função principal, foi implementada a função `neighbors()`, responsável por retornar uma lista dos oito vizinhos do ponto em determinado instante i da iteração.

Apesar de sua implementação mais simples e de caráter mais discreto, ele converge bem para o ponto objetivo. Contudo, é um método sujeito a ficar preso em mínimos locais, de forma semelhante à Descida do Gradiente. Todavia, como se observa na figura de seu resultado, ele progride de maneira quase uniforme.

1.3 *Simulated Annealing*

O método *Simulated Annealing* é parecido com o método Hill Climbing, com a diferença de que ele permite que o sistema ocupe pontos piores momentaneamente durante as iterações, a fim de se fugir de mínimos locais.

Para sua implementação, antes foram desenvolvidas funções de cálculo da temperatura e do vizinho aleatório. Esta última função, `random_neighbor()`, obtinha um valor aleatório de ângulo no intervalo $[-\pi, \pi)$ e utilizava esse valor para calular o vizinho. Em seguida, o algoritmo calcula a temperatura daquela iteração, por meio da função pré-definida no código. Se a temperatura fosse negativa significaria que a iteração não iria mais se otimizar e deveria retornar o ponto *theta* final encontrado. Após essa comparação, verificavam-se os custos, comparando o ponto atual com o vizinho. Se o vizinho tivesse um custo menor (pois buscava-se a minimização), *theta* receberia o valor do vizinho imediatamente. Caso não fosse, obtinha-se um valor randômico entre 0 e 1 a fim de verificar, por probabilidade, se o *theta* iria ser trocado. Essa "propabilidade"seria controlada por meio de uma comparação de um fator r com uma exponencial, conforme exemplificado abaixo:

$$r \leq exp(\Delta E/T)$$

Observe que, se T for muito grande, apesar de a exponencial ser negativa (conforme implementado no código), seu módulo se aproxima de 1, aumentando as chances de trocar *theta* por seu vizinho. Contudo, se T for pequeno (após várias iterações), o número se torna pequeno e as chances de se escolher o vizinho diminuem.

Como se espera desse método e pode ser verificado na imagem de seu resutado, ele se torna mais ruidoso devido à aleatoriedade e, portanto, demora mais para convergir para o objetivo, examinhando mais pontos.

2 Figuras Comprovando Funcionamento do Código

2.1 Descida do Gradiente

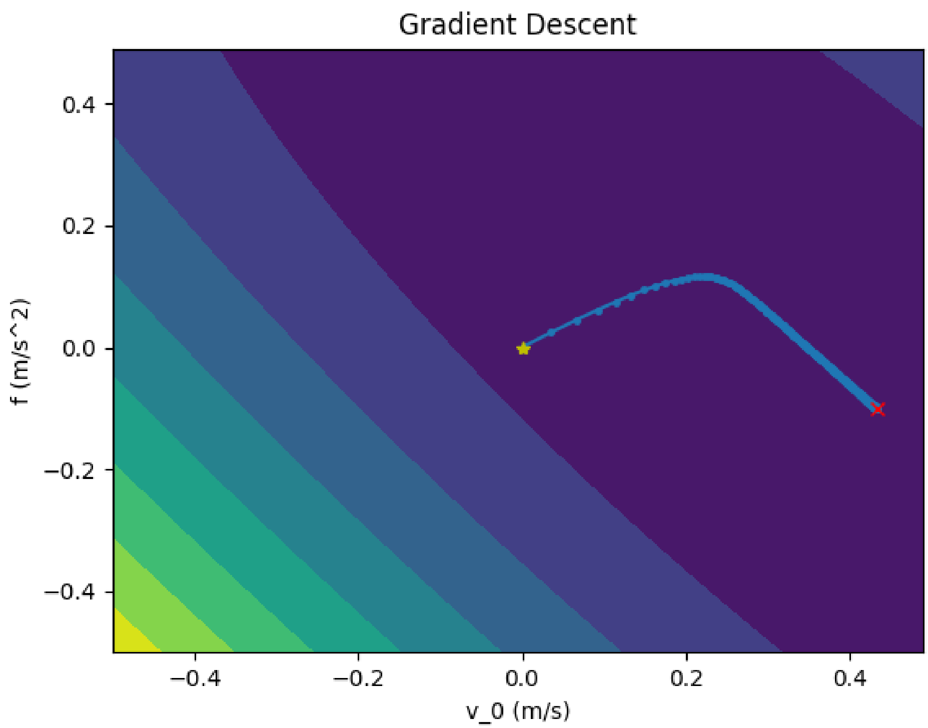

Fig. 1: Percurso obtido pelo algoritmo *Descida do Gradiente*

2.2 *Hill Climbing*

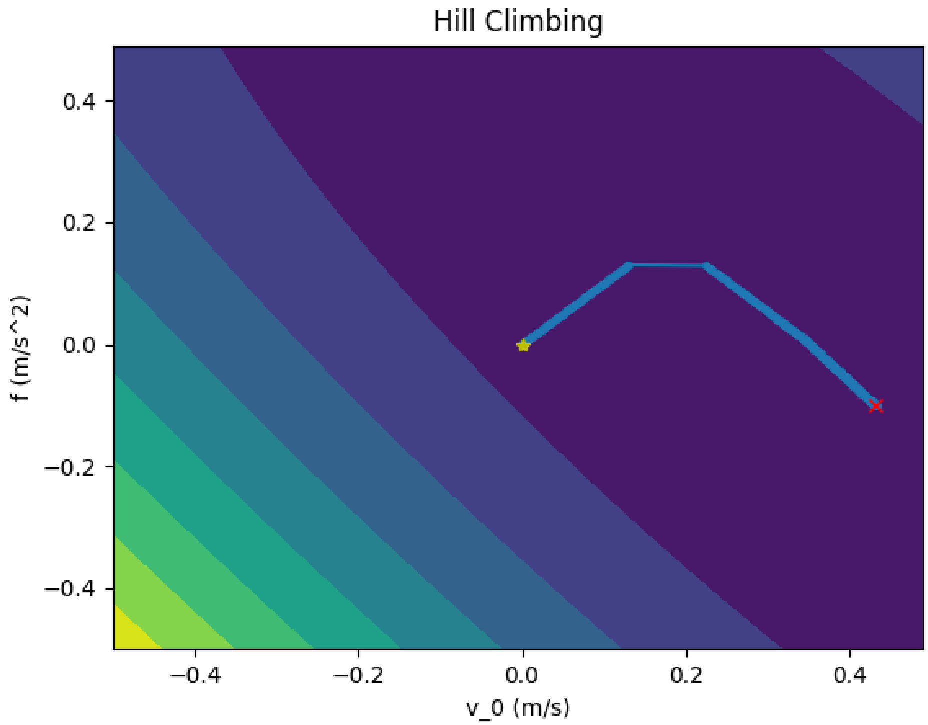

Fig. 2: Percurso obtido pelo algoritmo *Hill Climbing*

2.3 *Simulated Annealing*

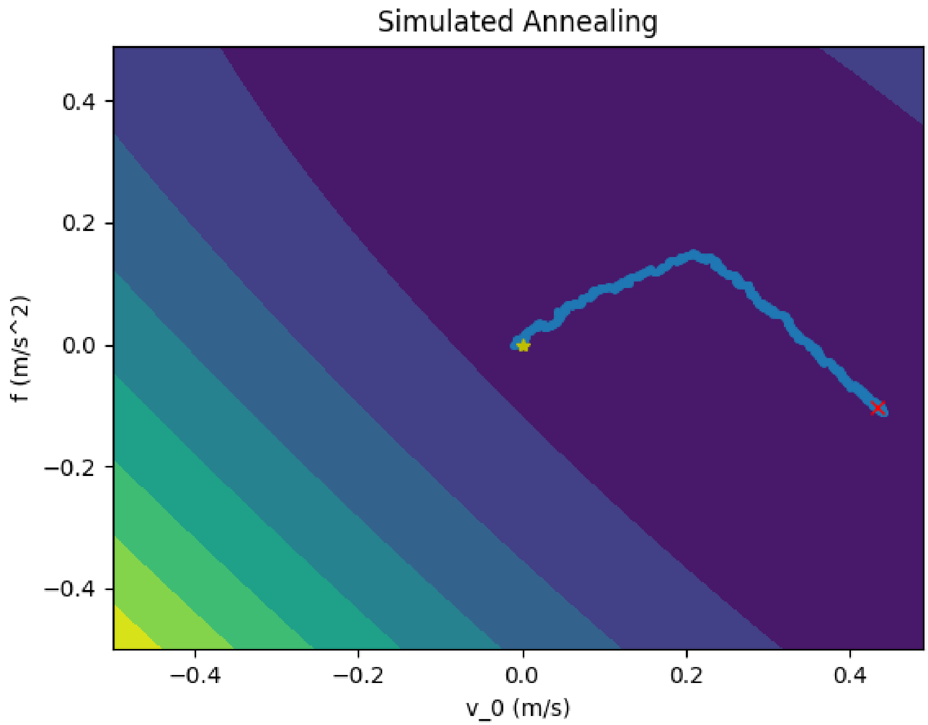

Fig. 3: Percurso obtido pelo algoritmo *Simulated Annealing*

3 Comparação entre os métodos

Tabela 1 com 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	0.434080	-0.101759

Como se verifica na tabela, todos os três métodos se aproximaram muito bem (cerca de 3 casas decimais) do resultado obtido pelo método analítico.

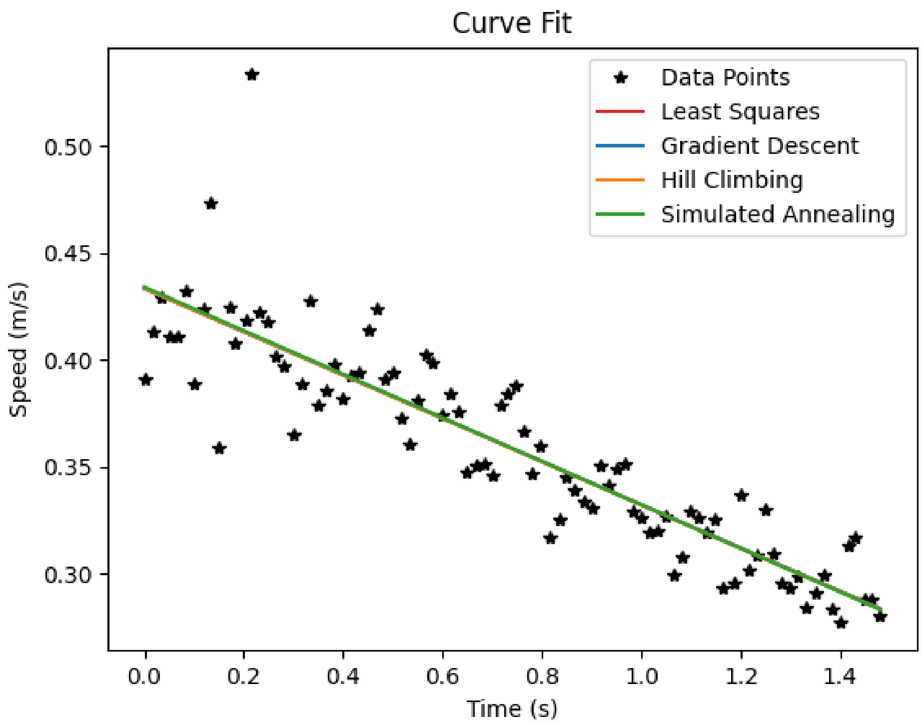

Fig. 4: *Fitting* comparativo entre os três algoritmos

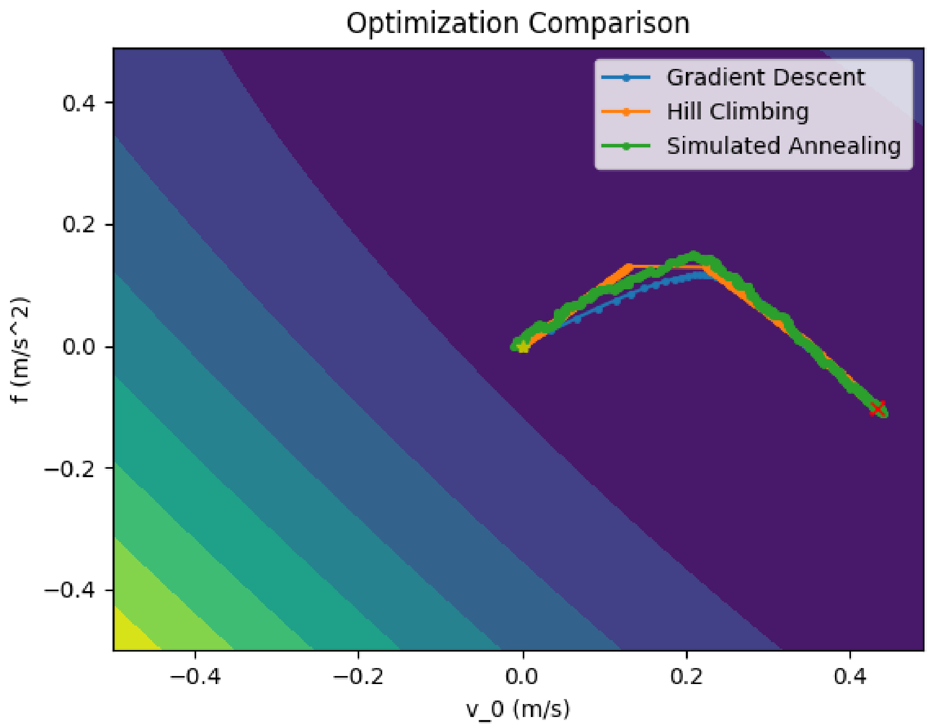

Fig. 5: Percursos comparados entre os três algoritmos