Instituto Tecnológico de Aeronáutica - ITA
Inteligência Artificial para Robótica Móvel - CT-213
Aluno: Danilo de Farias Matos

Relatório do Laboratório 3 - Otimização com Métodos de Busca Local

1. Breve Explicação em Alto Nível da Implementação

1.1. Descida do Gradiente

Iniciei implementando um loop for para limitar a quantidade de iterações à variável "max_interations", na sequência, apliquei a fórmula da descida do gradiente adicionando cada um dos vetores encontrados ao histórico para conseguir plotar o mapa e adicionei no final uma condição de parada a mais para levar em conta o epsilon (que ficou com o valor mínimo de custo que já aceitaríamos como excelente).

1.2. Hill Climbing

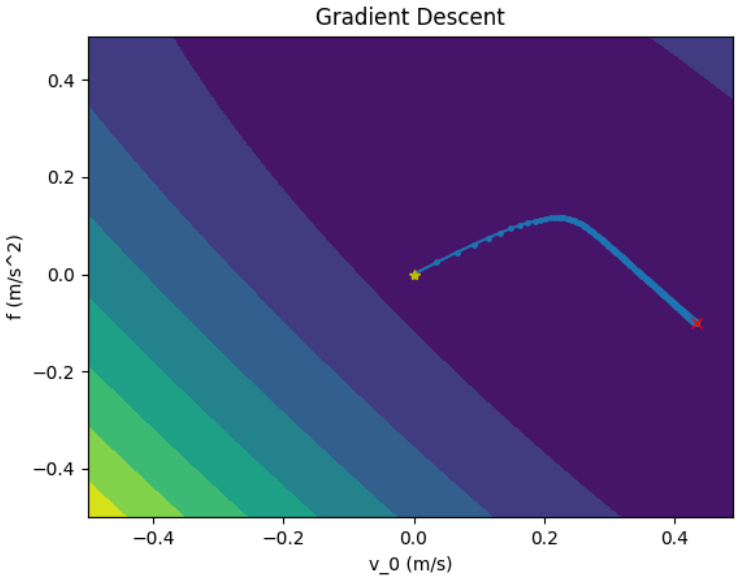
Deve haver um jeito mais elegante de se fazer isso… Mas implementei a função "neighbors()" com o uso de trigonometria (raízes quadradas para encontrar o valor de x e y necessários para serem adicionados ao vetor inicial a fim de obter os vetores dos 8 conectados). Após isso, iniciei o método do "hill_climbing()" propriamente dito criando uma variável para guardar o custo atual e criei um loop for para limitar as iterações à variável "max_iterations", já coloquei a condição de parada adicional para levar em consideração o epsilon também. Após isso foi a vez de inicializar mais algumas variáveis para controle e fazer mais um loop for para iterar entre todos os vizinhos encontrados, ficando com o que gerasse o menor custo. No fim do algoritmo eu atualizo as variáveis pertinentes e adiciono o vetor encontrado na iteração para o histórico (para possibilitar a criação do mapa).

1.3. Simulated Annealing

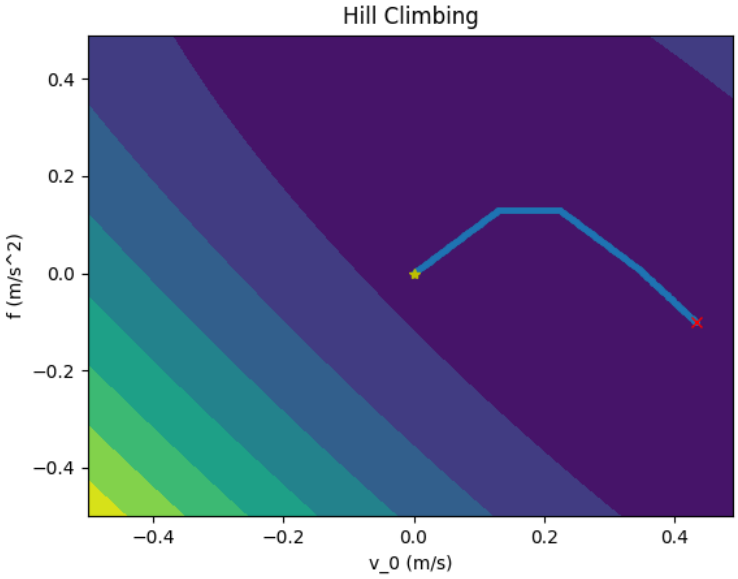
Iniciei implementando a função "random_neighbor()" com apenas um passo simples para escolher um valor aleatório de ângulo e possibilitar encontrar um novo vetor aleatório que respeitasse a máxima distância "delta". Depois, foi a vez da função "schedule()" que foi diretamente uma implementação de atualização de temperatura no estilo $T0/1+(b.i)^2$. Por fim, iniciei a implementação do método "simulated_annealing()" com a criação de um loop for para levar em conta a quantidade máxima de iterações " max_iterations" e já coloquei a condição de parada adicional para levar em consideração o epsilon também. Depois foi a vez de adicionar também a condição de parada para o caso do valor da temperatura ficar muito baixo (que poderia provocar um erro numérico). Por último foi a vez de implementar o código para comparar os custos obtidos pelo vetor atual e seu vizinho e, caso o custo diminua, atualizar o vizinho para que se torne o próximo vetor a ser usado como referência e, caso o custo seja maior, foi necessário implementar a característica principal do simulated annealing que poderia considerar um custo maior como possível caminho a seguir a depender da probabilidade obtida por meio do teste de valores da função exponencial e do valor aleatório entre 0 e 1.

2. Figuras Comprovando Funcionamento do Código

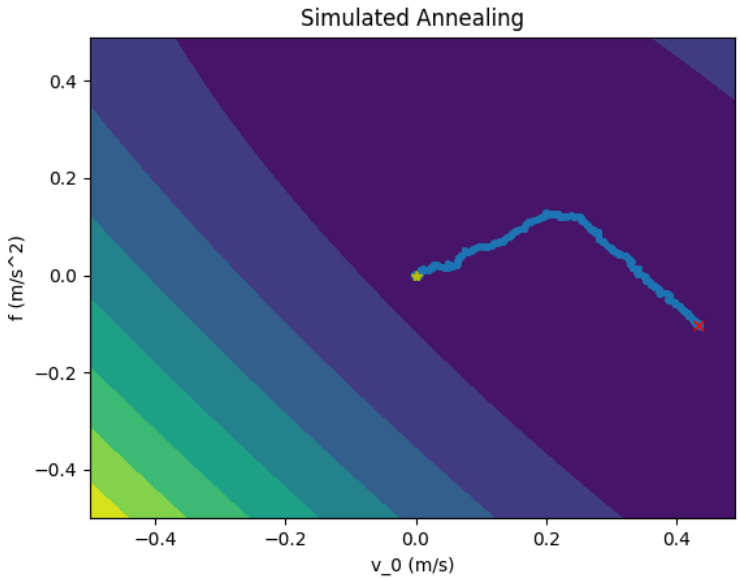
2.1. Descida do Gradiente



2.2. Hill Climbing



2.3. Simulated Annealing



3. Comparação entre os Métodos

Basta preencher a tabela e colocar as figuras da trajetória de otimização e das curvas da regressão linear.

Tabela 1 com a comparação dos 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	0.433057	-0.100678

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