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

Relatório do Laboratório 3 - Otimização com Métodos de Busca Local

1 Breve Explicação em Alto Nível da Implementação

1.1 Descida do Gradiente

Seguindo a implementação em aula, implementou-se o loop de modo que o algoritmo parasse quando o custo alcançasse o ϵ fornecido.

```
cost = cost_function(theta)
k = 1
        while True:
                if cost < epsilon or k > max_iterations:
                        break
                else:
                        # cálculo do novo theta com a função gradiente:
                        theta = theta - alpha*gradient_function(theta)
                        cost = cost_function(theta)
                        # Armazena histórico
                        k = k + 1
```

1.2 *Hill Climbing*

Como particularidade, este método teve duas funções a serem implementadas. Para a seleção dos possíveis vizinhos a serem testados, utilizou-se:

```
for t in np.linspace(0, 2*pi, num = num_neighbors, endpoint=False):
        neighbors_list = np.append(neighbors_list,
                        [theta[0] + delta*cos(t), theta[1] + delta*sin(t)])
neighbors_list = neighbors_list.reshape(num_neighbors, 2)
```

Para a função do algoritmo em si, seguiu-se a implementação em aula, com a diferença que neste problema buscamos a minização da função objetivo. Dentro do laço foram alteradas as comparações, ficando implementado como a seguir:

```
for neighbor in neighbors(theta):
        if cost_function(neighbor) < cost_best:
                best = neighbor
                cost_best = cost_function(best)
        if cost_best > cost:
                break
```

1.3 *Simulated Annealing*

Foram implementadas duas funções auxiliares além da função do algoritmo:

```
def random_neighbor(theta):
        t = random.uniform(-pi, pi)
        neighbor = theta + [delta * cos(t), delta * sin(t)]
        return neighbor

def schedule(i):
        return temperature0/(1+beta*i**2)
```

Análogo ao método anterior, precisou-se alterar o método apresentado em sala por se tratar de um problema de minimização da função objetivo:

```
deltaE = - cost_function(neighbor) + cost_function(theta)
```

Alterou-se o deltaE desta forma.

2 Figuras Comprovando Funcionamento do Código

2.1 Descida do Gradiente

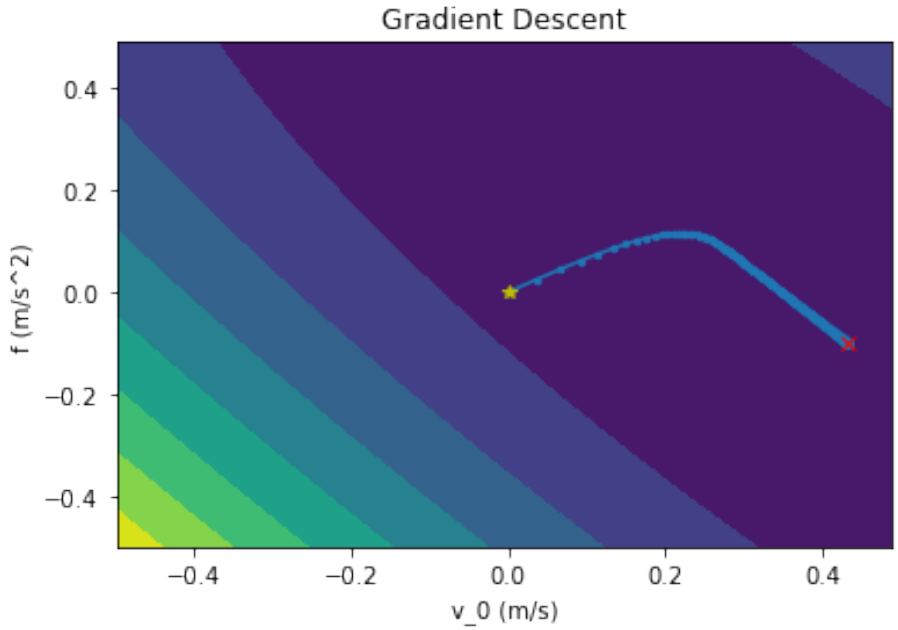

Figura 1: Histórico dos algoritmo gradiente.

2.2 *Hill Climbing*

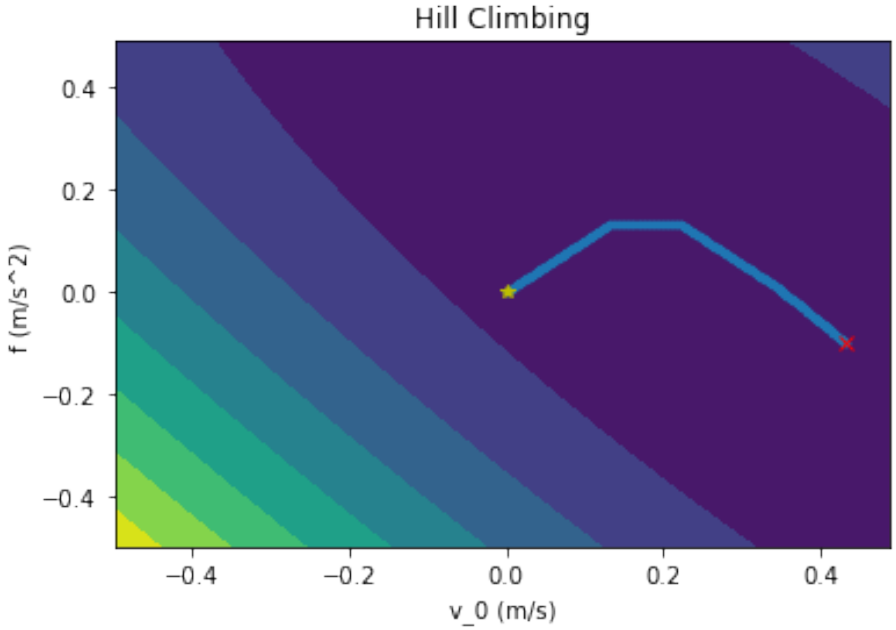

Figura 2: Histórico dos algoritmo hill climbing.

2.3 *Simulated Annealing*

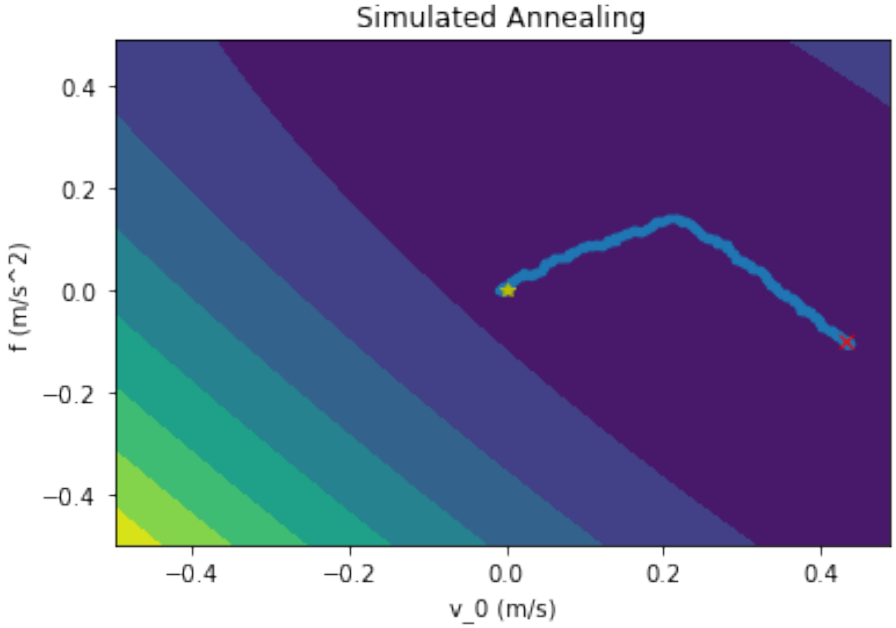

Figura 3: Histórico dos algoritmo simulated annealing.

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	0.433279	-0.101545

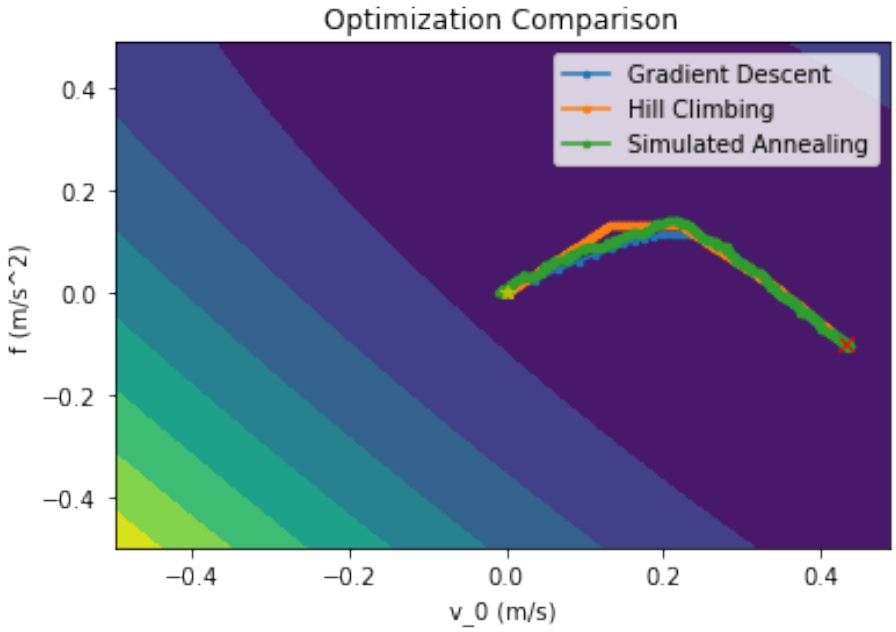

Figura 4: Histórico dos algoritmos até encontrar o ótimo.

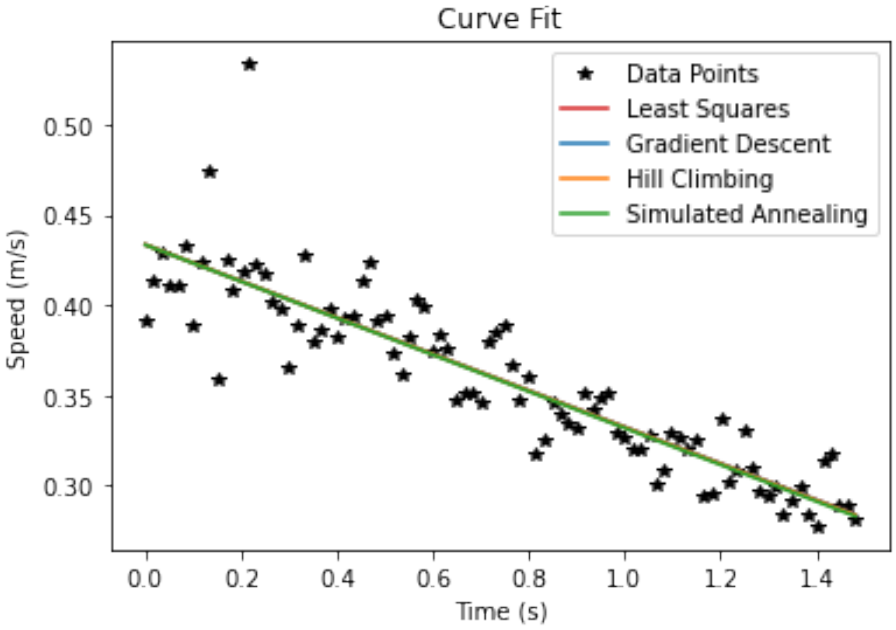

Figura 5: Comparação dos algoritmos em regressão.