Inteligência Artificial para Robótica Móvel

Introdução a Otimização

Métodos de Otimização de Busca Local

Professor: Marcos Maximo

Roteiro

- Motivação.
- Métodos Analíticos.
- Métodos dos Mínimos Quadrados (MMQ).
- Descida do Gradiente.
- Hill Climbing.
- Simulated Annealing.
- Busca Tabu.
- Dicas Práticas para Robótica.

Motivação

Otimização Matemática

- Encontrar o mínimo (ou máximo) de uma função:

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} J(\mathbf{x}) \text{ ou } \mathbf{x}^* = \arg\max_{\mathbf{x}} J(\mathbf{x})$$

- Nomenclatura:
  - $J(\mathbf{x})$: função objetivo, função de *fitness*, medida de qualidade, função de avaliação, função de custo etc.
  - $\mathbf{x}$: parâmetros.
  - $\mathbf{x}^*$: ótimo (mínimo ou máximo).
- Também é comum haver restrições:

$$\mathbf{g}(\mathbf{x}) \leq \mathbf{0}$$
$$\mathbf{h}(\mathbf{x}) = \mathbf{0}$$

Otimização Matemática

- Pode-se pensar como busca em que o caminho não importa.
- Diversos problemas de Engenharia podem ser escritos como problemas de otimização.
- Frequentemente, em tomada de decisão, comportamentos envolvem parâmetros a serem ajustados:
  - Exemplo: como ajustar os parâmetros do Roomba do lab 1 de modo a limpar o ambiente no menor tempo?
- Também usa-se o nome “programação matemática” (marketing): programação linear, programação quadrática, programação dinâmica etc.
- A área de Aprendizado de Máquina é construída em cima de otimização.

Otimização Numérica

- Quando solução analítica não é possível: otimização numérica.
- Há diversas técnicas.
- **Não** existe o melhor algoritmo (*no free lunch*).
- “Melhor” depende de quão difícil é o problema de otimização.
- Algoritmos específicos para certas classes:
  - Programação linear: função de custo e restrições lineares.
  - Programação quadrática: função de custo quadrática e restrições lineares.
  - Programação quadrática restrita quadraticamente: função de custo e restrições quadráticas.
- Comunidade de IA geralmente está preocupada com problemas de otimização muito difíceis (classes mais gerais).

Metaheurísticas

- A maioria dos algoritmos que veremos estão na classe de Metaheurísticas.
- Metaheurísticas são os algoritmos mais “força bruta”. ♣♦♥♠
- Para muitos desses algoritmos, não precisa nem da expressão de $J(\mathbf{x})$, basta conseguir amostrar.
- Sem muitas garantias matemáticas, mas funcionam muito bem na prática.
- Algoritmos apresentados com base na experiência do professor ☺.

Exploitation x Exploration

- Em problemas muito difíceis, não há garantia de encontrar o ótimo global.
- Surge o dilema de *exploitation* x *exploration*.
- *Exploitation*: tentar mais do que sei que é bom.
- *Exploration*: buscar por onde não fui antes.
- Focar demais em *exploitation* resulta em convergência prematura para ótimo local.
- Focar demais em *exploration* faz convergência demorar demais.

Métodos Analíticos

Métodos Analíticos

- Quando o problema é muito simples, a solução é analítica:

$$J'(x) = 0$$

- Se for uma função multivariável (campo escalar), usa-se o gradiente:

$$\nabla J(\mathbf{x}) = \mathbf{0}$$

$$\nabla J(\mathbf{x}) = \frac{\partial J}{\partial \mathbf{x}} = \left[\frac{\partial J}{\partial x_1} \quad \frac{\partial J}{\partial x_2} \quad \cdots \quad \frac{\partial J}{\partial x_n}\right]^T$$

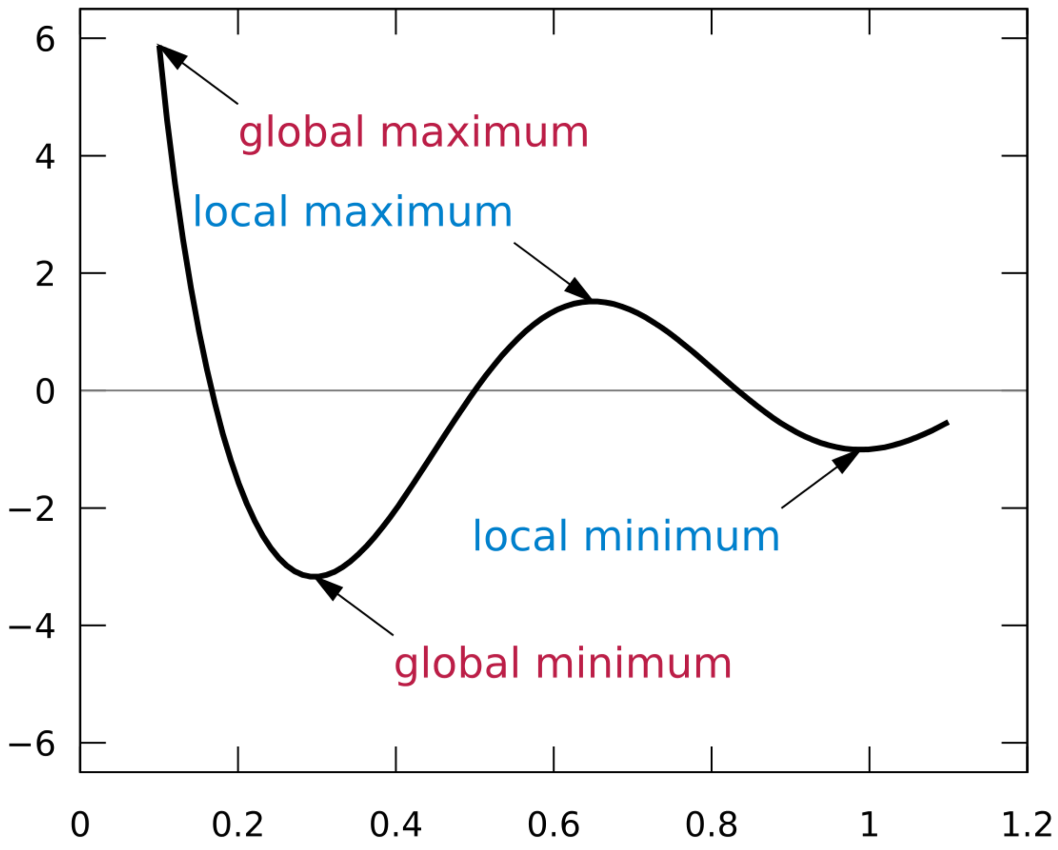
- Segunda derivada indica se é ponto de mínimo ou máximo local:

$$f''(x_0) > 0 \Rightarrow \text{mínimo local}$$

$$f''(x_0) < 0 \Rightarrow \text{máximo local}$$

$$f''(x_0) = 0 \Rightarrow \text{ponto de inflexão}$$

Ótimos Locais e Globais



Métodos Analíticos

- Por que mínimo ou máximo? Seja x na vizinhança de x_0:

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + f''(c)(x - x_0)^2 =$$
$$f(x_0) + f''(c)\underbrace{(x - x_0)^2}_{>0}$$

- Como c é arbitrariamente próximo de x_0, $f''(c)$ tem o mesmo sinal de $f''(x_0)$.
- Assim:

$$f''(x_0) > 0 \Rightarrow f(x_0) < f(x) \text{ (mínimo)}$$
$$f''(x_0) < 0 \Rightarrow f(x_0) > f(x) \text{ (máximo)}$$

Métodos Analíticos

- Para o caso multivariado, o teste da derivada segunda se refere à hessiana:

$$f(\mathbf{x}) \approx f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T(\mathbf{x} - \mathbf{x}_0) + (\mathbf{x} - \mathbf{x}_0)^T \mathbf{H}(\mathbf{x} - \mathbf{x}_0)$$

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_1 \partial x_2} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

Métodos Analíticos

- Para o caso multivariado, o teste da derivada segunda se refere à hessiana:

$$f(\mathbf{x}) \approx f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T(\mathbf{x} - \mathbf{x}_0) + (\mathbf{x} - \mathbf{x}_0)^T\mathbf{H}(\mathbf{x} - \mathbf{x}_0)$$

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 x_n} \\ \frac{\partial^2 f}{\partial x_1 \partial x_2} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

Métodos Analíticos

- Seja $\mathbf{H} = \mathbf{H}^T \in \mathbb{R}^{n \times n}$, diz-se que:
  - $\mathbf{H} > 0$ (positivo-definida) se $\mathbf{x}^T\mathbf{H}\mathbf{x} > 0, \forall \mathbf{x} \neq \mathbf{0}$.
  - $\mathbf{H} \geq 0$ (positivo-semidefinida) se $\mathbf{x}^T\mathbf{H}\mathbf{x} \geq 0, \forall \mathbf{x} \neq \mathbf{0}$.
  - $\mathbf{H} < 0$ (negativo-definida) se $\mathbf{x}^T\mathbf{H}\mathbf{x} < 0, \forall \mathbf{x} \neq \mathbf{0}$.
  - $\mathbf{H} \leq 0$ (negativo-semidefinida) se $\mathbf{x}^T\mathbf{H}\mathbf{x} \leq 0, \forall \mathbf{x} \neq \mathbf{0}$.
- Ideia: generalizar a noção de positivo/negativo para matrizes.

Métodos Analíticos

- Teste da segunda derivada para caso multivariado:

$$\mathbf{H} > 0 \Rightarrow \text{mínimo local}$$

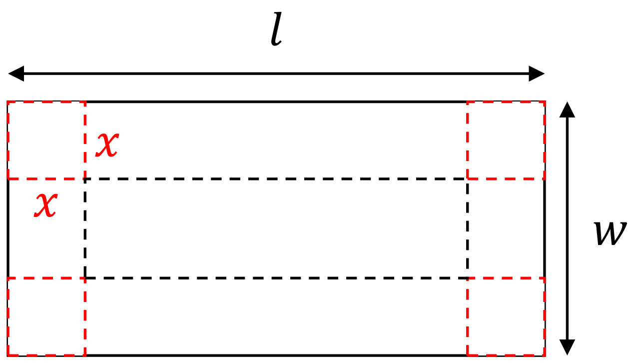
$$\mathbf{H} < 0 \Rightarrow \text{máximo local}$$

- Demonstração é análoga ao caso univariado.

Métodos Analíticos

- Exemplo:

Tem-se um pedaço de papelão de comprimento l e largura w. Deseja-se construir uma caixa (sem tampa) cortando-se quadrados de lado x nos cantos desse papelão e dobrando. Qual deve ser o valor de x para termos volume máximo?



Métodos Analíticos

$$J(x) = x(l - 2x)(w - 2x) \Rightarrow$$
$$J(x) = lwx - 2(w + l)x^2 + 4x^3$$
$$J'(x) = lw - 4(w + l)x + 12x^2 = 0 \Rightarrow$$
$$x = \frac{(w + l) \pm \sqrt{l^2 + w^2 - lw}}{6}$$

$x_1 = \frac{(w+l)+\sqrt{l^2+w^2-lw}}{6}$ não satisfaz pois lado menor seria negativo.

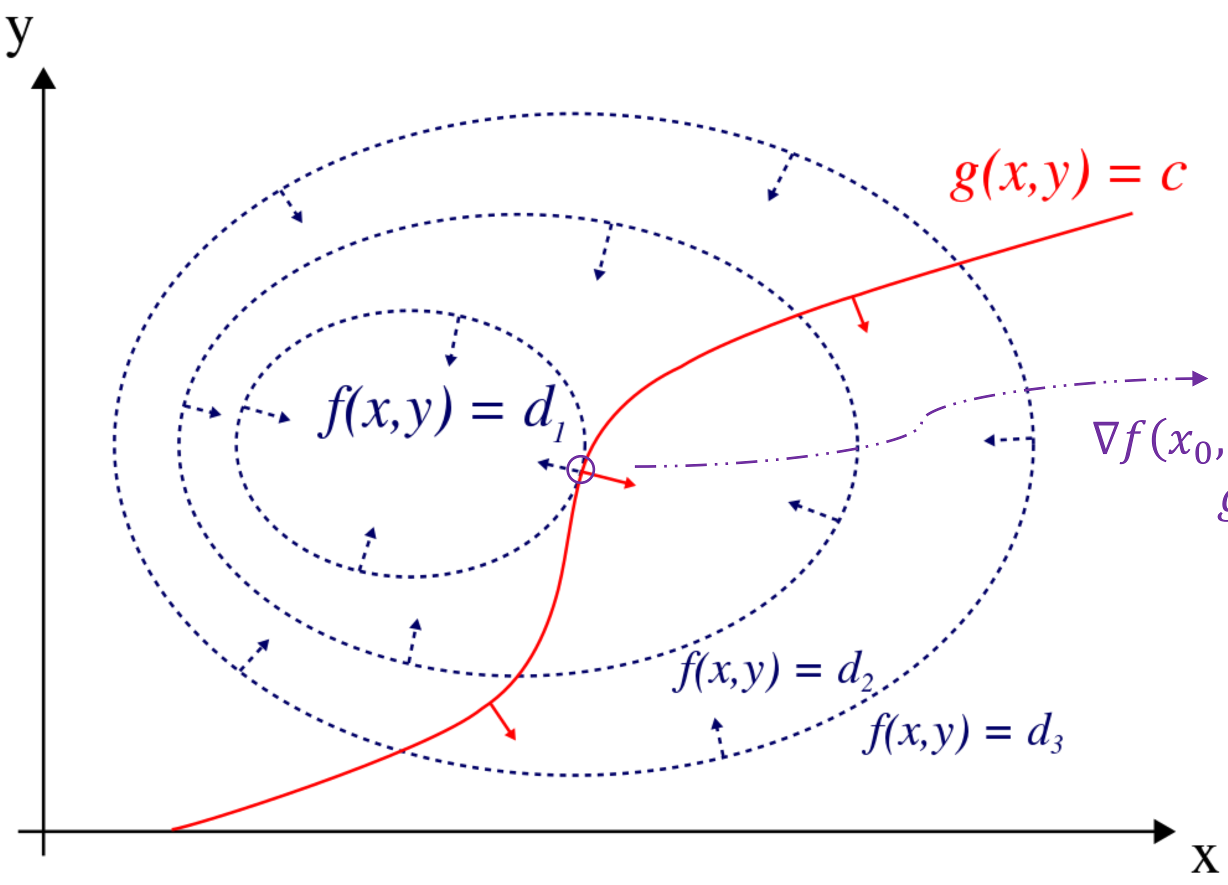
$J''(x) = -4(w + l) + 24x = -4\sqrt{l^2 + w^2 - lw} < 0$ (ponto de máximo)

Otimização com Restrições

$$\min_{\mathbf{x}} J(\mathbf{x}) \;\; s.a. \;\; \mathbf{g}(\mathbf{x}) = \mathbf{c}$$

- Solução analítica usa Multiplicadores de Lagrange.
- Converter em otimização sem restrições com função de custo:
$$\mathrm{L}(\mathbf{x}, \boldsymbol{\lambda}) = J(\mathbf{x}) + \boldsymbol{\lambda}^T(\mathbf{g}(\mathbf{x}) - \mathbf{c})$$

Multiplicadores de Lagrange



Condições:

$$\nabla f(x_0, y_0) = \lambda \nabla g(x_0, y_0)$$
$$g(x_0, y_0) = c$$

Fonte: https://en.wikipedia.org/wiki/Lagrange_multiplier

Otimização com Restrições

- Exemplo:

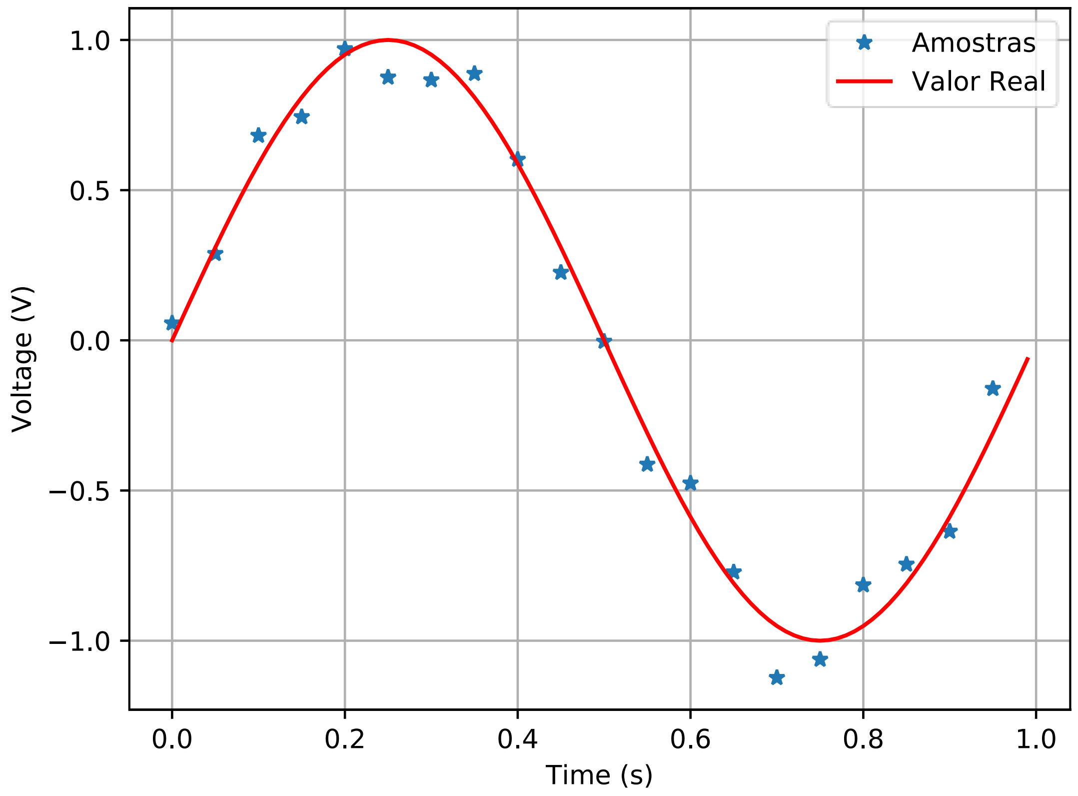
Uma caixa retangular sem tampa deve ser feita com $12\ m^2$ de papelão. Determine as dimensões x, y e z que fornecem o volume máximo de tal caixa.

Otimização com Restrições

$$J(x, y, z) = xyz$$
$$g(x, y, z) = xy + 2xz + 2yz = 12$$
$$L(x, y, z, \lambda) = xyz + \lambda(xy + 2xz + 2yz - 12)$$
$$\nabla L(x, y, z, \lambda) = \mathbf{0}$$
$$\begin{cases} x = 2 \\ y = 2 \\ z = 1 \end{cases}$$

Método dos Mínimos Quadrados (MMQ)

Método dos Mínimos Quadrados (MMQ)



Método dos Mínimos Quadrados (MMQ)

- Problema de ajuste de curvas.
- Assumir modelo para a curva:

$$f(x) = \theta_0\phi_0(x) + \theta_1\phi_1(x) + \theta_2\phi_2(x) + \cdots + \theta_n\phi_n(x)$$

- $\phi_j(x)$ são funções arbitrárias (*features*).
- $\boldsymbol{\theta} = [\theta_0 \quad \theta_1 \quad \cdots \quad \theta_n]^T$ é vetor de parâmetros.
- Coletar m pontos experimentais: $\{(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)\}$.
- Função de custo quadrática:

$$J(\theta_0, \theta_1, \dots, \theta_n) = \sum_{i=1}^{m} \big(y_i - f(x_i)\big)^2$$

- Espécie de "Aprendizado Supervisionado" muito simples!

Método dos Mínimos Quadrados (MMQ)

- Solução (analítica) recai na solução de sistema linear:

$$\begin{bmatrix} \sum_i \phi_0(x_i)\phi_0(x_i) & \sum_i \phi_0(x_i)\phi_1(x_i) & \cdots & \sum_i \phi_0(x_i)\phi_n(x_i) \\ \sum_i \phi_1(x_i)\phi_0(x_i) & \sum_i \phi_1(x_i)\phi_1(x_i) & \cdots & \sum_i \phi_1(x_i)\phi_n(x_i) \\ \vdots & \vdots & \ddots & \vdots \\ \sum_i \phi_n(x_i)\Phi_0(x_i) & \sum_i \phi_n(x_i)\phi_1(x_i) & \cdots & \sum_i \phi_n(x_i)\phi_n(x_i) \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} = \begin{bmatrix} \sum_i \phi_0(x_i)y_i \\ \sum_i \phi_1(x_i)y_i \\ \sum_i \phi_2(x_i)y_i \\ \vdots \\ \sum_i \phi_n(x_i)y_i \end{bmatrix}$$

- Demonstração (derivar e igualar a zero)!

Método dos Mínimos Quadrados (MMQ)

- Notação alternativa:

$$\mathbf{X}^T\mathbf{X}\boldsymbol{\theta} = \mathbf{X}^T\mathbf{y}$$

$$X_{ij} = \phi_j(x_i)$$
$$\boldsymbol{\theta} = [\theta_0 \quad \theta_2 \quad \cdots \quad \theta_n]^T$$
$$\mathbf{y} = [y_1 \quad y_2 \quad \cdots \quad y_m]$$

Método dos Mínimos Quadrados (MMQ)

- Há alguns casos especiais clássicos...
- Regressão Linear:

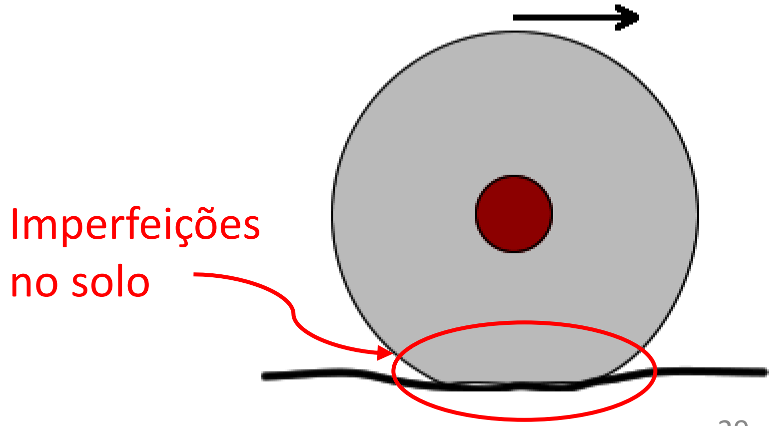
$$f(x) = \theta_0 + \theta_1 x$$

- Regressão Polinomial:

$$f(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \cdots + \theta_n x^n$$

Predição de Movimento da Bola

- Para que um robô consiga interceptar uma bola em movimento, é importante um bom modelo de predição do movimento da bola.
- Uma bola perfeitamente esférica num terreno perfeitamente plano teoricamente não sofre atrito.
- Na prática imperfeições na bola ou no solo geram *rolling friction*.
- Modelo comum: $v(t) = v_0 - ft$.
- f depende do campo e da bola.
- Interesse: aprender f.



Predição de Movimento da Bola

Procedimento:

1. Usar câmera para obter posições (x, y) da bola em cada instante.
2. Calcular velocidades em x e y usando diferenças finitas centradas:
$$v_x[k] = \frac{x[k+1]-x[k-1]}{2\Delta t},\ v_y[k] = \frac{y[k+1]-y[k-1]}{2\Delta t}$$
3. Calcular $v[k] = \sqrt{v_x^2[k] + v_y^2[k]}$.
4. Usar MMQ para obter v_0 e f.
5. Usar f para predições futuras.

MMQ Multivariável

- MMQ é facilmente extensível para múltiplas variáveis.
- Basta considerar que cada ϕ_{j} pode depender de múltiplas variáveis:

$$\phi_j(\mathbf{x}) = \phi_j(x_1, x_2, \dots, x_p)$$

- Deve-se resolver o mesmo sistema linear que antes:

$$\mathbf{X}^T\mathbf{X}\boldsymbol{\theta} = \mathbf{X}^T\mathbf{y}$$

Limitações do MMQ

- Resolver sistema linear pode ter alto custo computacional e cair em problemas numéricos.
- Função do modelo tem que ser combinação linear das *features*.

$$f(\mathbf{x}) = \sum_j \theta_j \Phi_j(\mathbf{x})$$

- Quando é não-linear, às vezes dá para transformar em linear:

$$f(x) = Acos(x + \phi) = A(cos(x)cos(\phi) - sen(x)sen(\phi) = \theta_0 \cos(x) + \theta_1 sen(x)$$

$$\theta_0 = Acos\phi, \theta_1 = -Asen\phi$$

- De modo geral, não dá para fazer isso (e.g. rede neural).

Métodos de Otimização de Busca Local

Métodos de Otimização de Busca Local

- Métodos de busca local buscam iterativamente ótimos locais próximo ao ponto inicial (chute inicial).
- Convergem rápido, mas são muito suscetíveis a ficar preso em ótimo local.
- Em geral, funcionam bem quando se tem poucos parâmetros e um bom chute inicial.

Descida do Gradiente

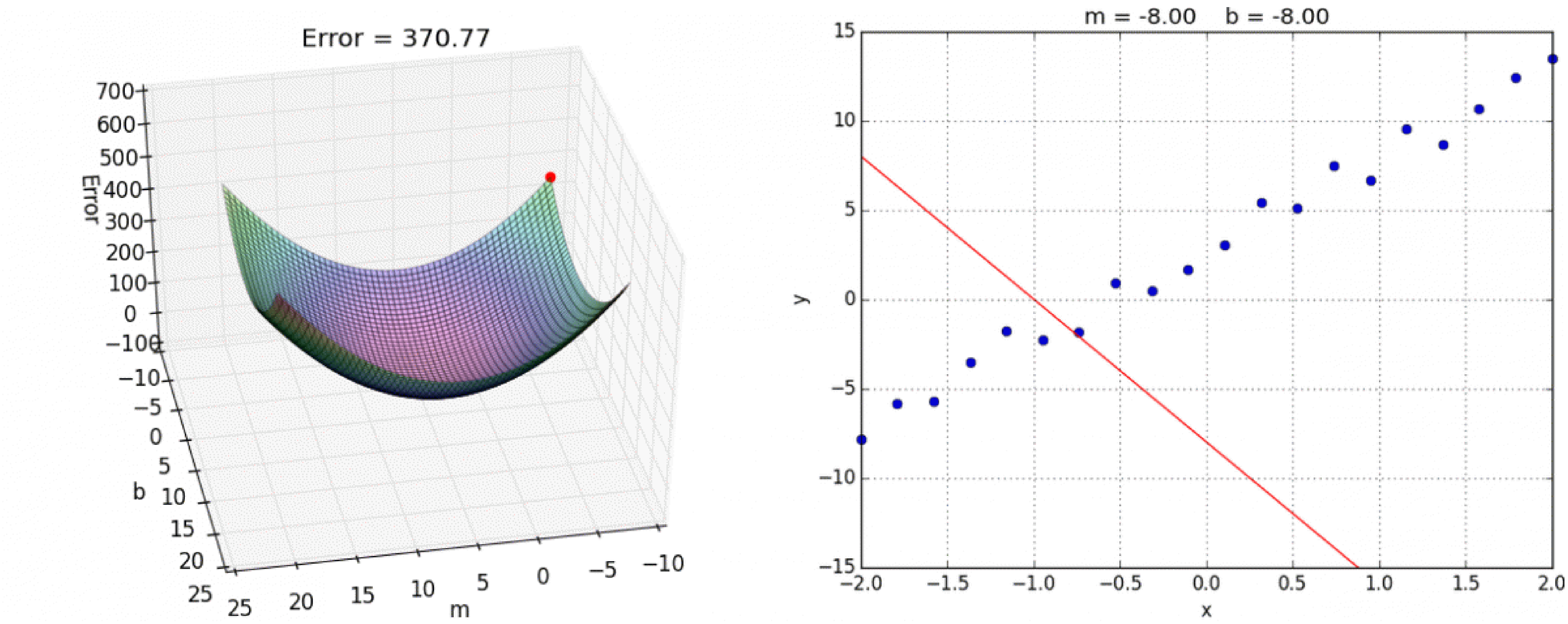
Descida do Gradiente

- Inglês: *Gradient Descent* (GD).
- Método popularizado devido ao uso para treinar redes neurais.
- Funciona bem se for possível calcular o gradiente analiticamente.
- MAT: gradiente dá a direção de máximo crescimento da função.
- Ideia do algoritmo: seguir na direção contrária à do gradiente (máximo decrescimento).

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

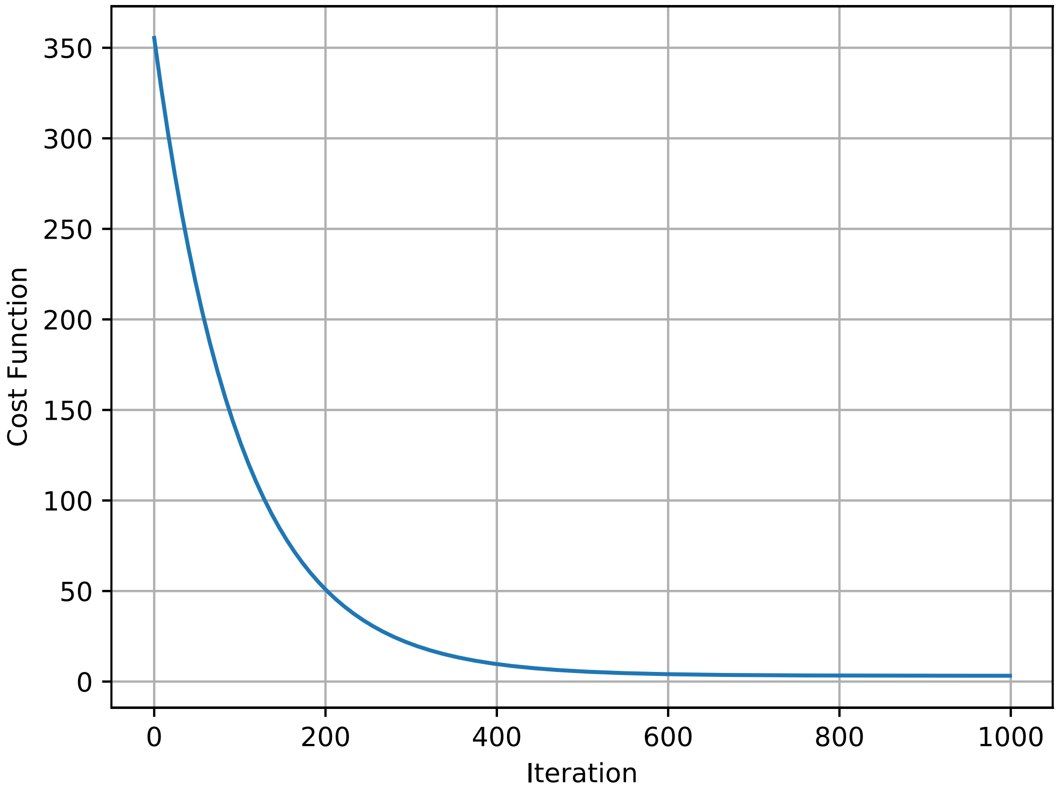
- α: taxa de aprendizagem (hiperparâmetro).
  - Ajustado por tentativa e erro.
  - Depende do problema.

Descida do Gradiente



Fonte: https://giphy.com/gifs/gradient-O9rcZVmRcEGqI

Descida do Gradiente



Descida do Gradiente

- Regressão linear:

$$f(x) = \theta_0 + \theta_1 x$$

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

$$\frac{\partial J}{\partial \theta_0} = \frac{1}{m} \sum_i (f(x_i) - y_i), \frac{\partial J}{\partial \theta_1} = \frac{1}{m} \sum_i (f(x_i) - y_i) x_i$$

$$\theta_0 = \theta_0 - \alpha \frac{1}{m} \sum_i (f(x_i) - y_i)$$

$$\theta_1 = \theta_1 - \alpha \frac{1}{m} \sum_i (f(x_i) - y_i) x_i$$

Descida do Gradiente

```
def gradient_descent(dJ, theta, alpha):
      while not check_stopping_condition():
            theta = theta – alpha * dJ(theta)
      return theta
```

Descida do Gradiente

- Também resolve problemas em que $f(x)$ envolve combinação não-linear das *features*, desde que seja possível calcular $\partial J/\partial \boldsymbol{\theta}$.
- Muito dependente do chute inicial.
- É comum usar um *schedule* para o α (começa com valor alto e vai diminuindo).
- Veremos melhorias do algoritmo quando estudarmos Deep Learning.

Descida do Gradiente

- Fica “preso” em mínimos locais.
- Critérios de parada comuns:
  - Número de iterações.
  - $J < \varepsilon_1$.
  - $|J_{k+1} - J_k| < \varepsilon_2$.
  - Monitora curva de J e para quando achar que está bom (usado na prática).
- O que fazer quando não se tem expressão de $\partial J / \partial \boldsymbol{\theta}$?
  - Possível calcular numericamente (alto custo).
  - Em geral, usa-se outros métodos.

Hill Climbing

Hill Climbing

- Português: subida de encosta.
- "*Climbing*" dá ideia de maximização, mas pode usar para minimização.
- Avalia vizinhos da posição atual.
- Vai para vizinho melhor avaliado.
- Continua até atingir critério de parada.
- Não requer cálculo da derivada.
- Funciona quando espaço de busca é discreto.

Hill Climbing

```
# Assuming maximization
def hill_climbing(J, theta):
      while not check_stopping_condition():
            best = None # J(None) = -inf
            for neighbor in neighbors(theta):
                  if J(neighbor) > J(best):
                        best = neighbor
            if J(best) < J(theta):
                  return theta
            theta = best
      return theta
```

Obs.: Na prática, guardar em variáveis os valores de $J(\mathbf{x})$.

Escolha de Vizinhos

- Para problema contínuo, é comum adotar:

$$\theta_i' = \theta_i \pm \alpha$$

(cada vizinho soma/subtrai α em uma dimensão)

- Quando a dimensão do espaço é muito grande, escolhe-se dimensões aleatórias para expansão: *Stochastic Hill Climbing* (SHC).

Hill Climbing

- Muito dependente do chute inicial.
- É comum o uso com reinício aleatório.
  - Executa-se HC várias vezes com diferentes chutes iniciais.

Simulated Annealing

Simulated Annealing

- Português: têmpera simulada.
- Semelhante ao *Hill Climbing*, mas permite transição para estados piores.
- Motivo: fuga de mínimos locais.
- Inspiração vem do processo de têmpera da metalurgia (aquecimento seguido de resfriamento lento para tratamento de um metal).

Simulated Annealing

```
# Assuming maximization
def simulated_annealing(J, theta):
	while not check_stopping_condition():
		T = temperature_schedule(i)
		if T < 0.0:
			return theta
		neighbor = random_neighbor(theta)
		deltaE = J(neighbor) – J(theta)
		if deltaE > 0:
			theta = neighbor
		else:
			r = random_uniform(0.0, 1.0) # Draws random number w/ uniform dist.
			if r <= exp(deltaE / T):
				theta = neighbor
	return theta
```

Simulated Annealing

- Garantia matemática: se T diminui suficientemente lento, então sempre SA atinge o melhor estado.
- Garantia de pouca utilidade prática: “suficientemente lento” pode significar **muito** tempo.
- Exemplos de *schedule* de temperatura:

$$T_i = T_0 \beta^i$$

$$T_i = \frac{T_0}{1 + \beta i}$$

Busca Tabu

Busca Tabu

- Inglês: *Tabu Search*.
- Modificação no *Hill Climbing* em que se mantém uma lista de estados já visitados (lista tabu) para que se evite voltar a eles.
- Ajuda a evitar mínimos locais.
- Vai para melhor vizinho que não está na lista tabu, mesmo que piore.
- Mas guarda melhor até agora.

Busca Tabu

```
# Assuming maximization
def tabu_search(J, theta, max_tabu_len):
        previous_best = theta
        while not check_stopping_condition():
                current_best = None # J(None) = -inf
                for neighbor in neighbors(previous_best):
                        if (not neighbor in tabu_list) and J(neighbor) > J(current_best):
                                current_best = neighbor
                tabu_list.append(current_best)
                if J(current_best) > J(theta):
                        theta = current_best
                if len(tabu_list) > max_tabu_len:
                        tabu_list.remove_first() # tabu_list operates in a FIFO fashion
                previous_best = current_best
        return theta
```

Dicas Práticas para Robótica

Chute Inicial

- Como encontrar um bom chute inicial?
- Tentar resolver problema “simplificado”.
  - Ignorar efeitos da Física até ter solução analítica.
  - Ignorar “acoplamento” (e.g. resolver problema 2D primeiro).
  - Usar outra técnica (e.g. visibility graph e A* para chute inicial de caminho).
- Tentativa e erro (testar alguns valores).

Função de Custo

- Comumente, o $J(.)$ em problemas de Robótica é estocástico (aleatório), i.e. $J(\boldsymbol{\theta})$ pode dar valores diferentes para o mesmo $\boldsymbol{\theta}$.
- Nesse caso, costuma-se avaliar $J(\boldsymbol{\theta})$ várias vezes e tomar uma média.
- A variância (“erro”) de $J(\boldsymbol{\theta})$ fica divididida por $\sqrt{r}$, em que r é o número de repetições.

Custo Computacional

- Tipicamente, em problemas de Robótica, o gargalo é o cálculo de $J(.)$.
- Isso é principalmente verdade se $J(.)$ envolver uma simulação.
- Simulações em geral são implementadas usando CPU, logo problema é *CPU bounded*.
- Por outro lado, otimização costuma ser muito paralelizável.
- Otimizações podem demorar alguns dias.

Para Saber Mais

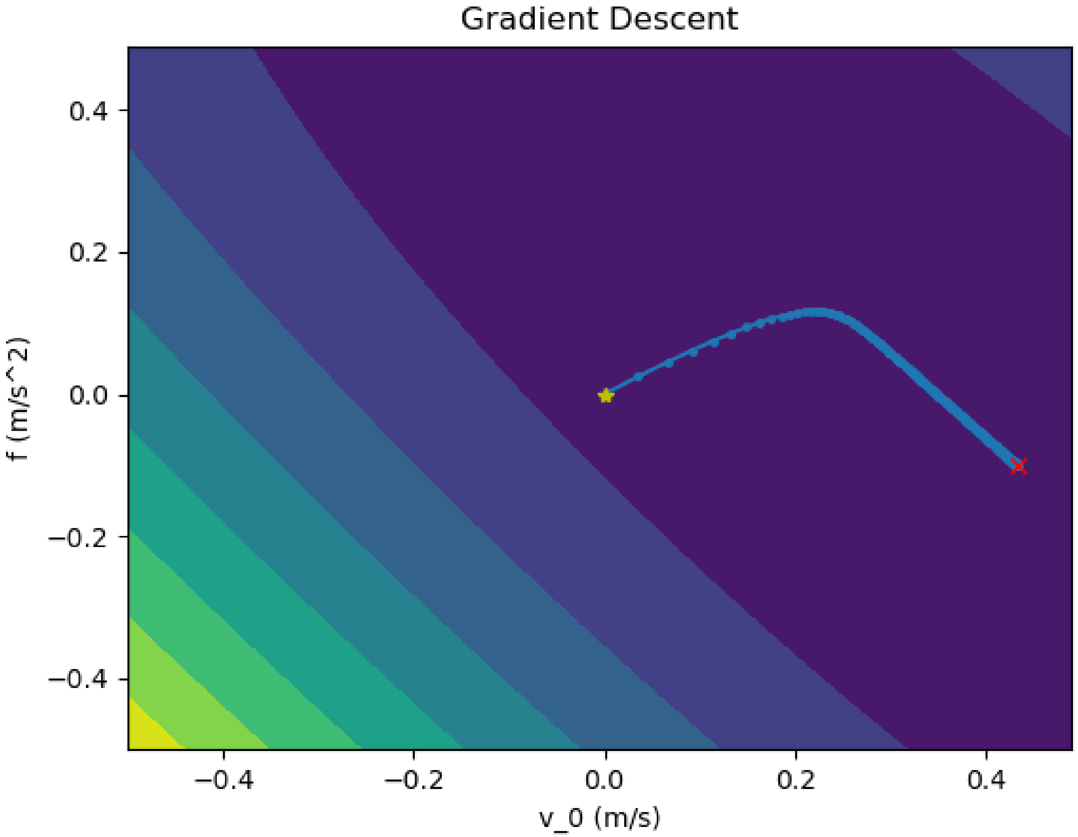
- Descida de Gradiente:
  - Capítulos 5 e 8 do livro *Deep Learning* de Goodfellow, Begin & Courville.
  - Primeiras duas aulas de Aprendizado Supervisionado ☺.
  - Cursos 1 e 2 da especialização de Deep Learning do Coursera (Andrew Ng).
- Hill Climbing e Simulated Annealing:
  - Capítulo NORVIG, Peter; RUSSELL, Stuart. *Artificial Intelligence: A Modern Approach*. Pearson, 2009.

Laboratório 3

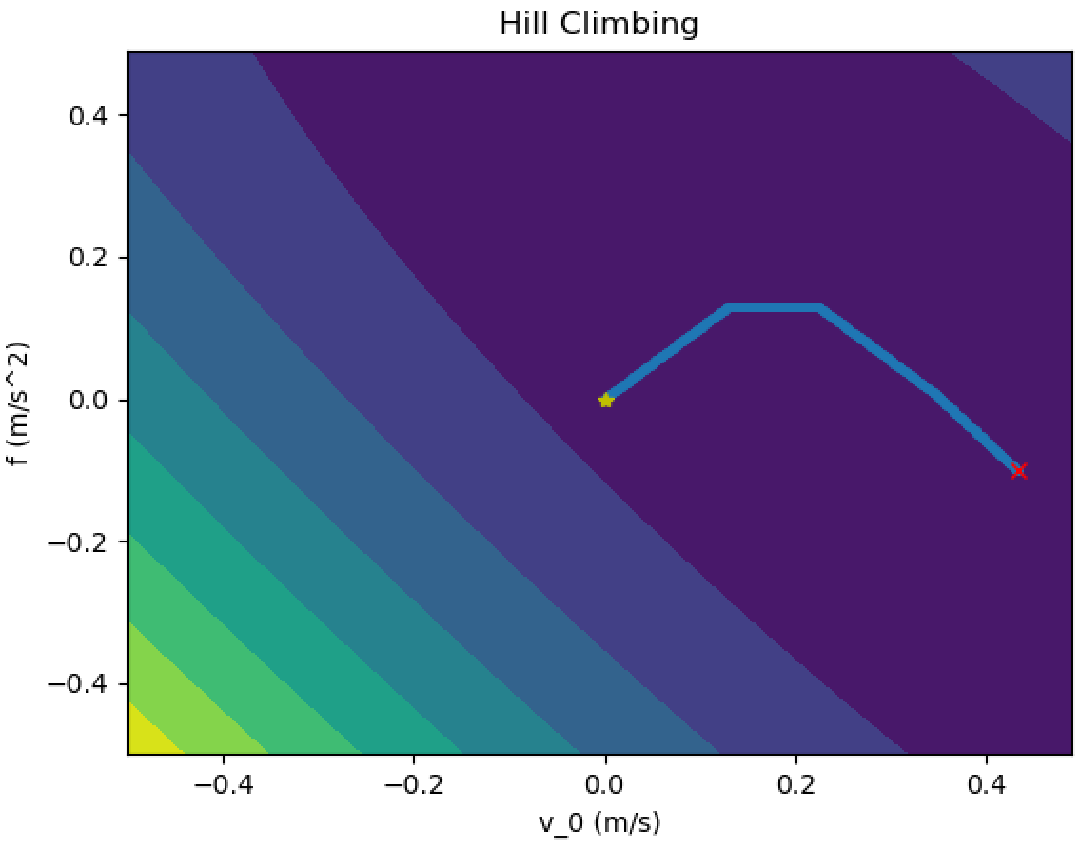
Laboratório 3

- Implementar algoritmos de otimização:
  - Descida do Gradiente.
  - *Hill Climbing*.
  - *Simulated Annealing*.
- Problema: regressão linear (achar v_0 e f no problema de *fit* da bola).
- Método dos Mínimos Quadrados já implementado (comparar solução).

Descida do Gradiente



Hill Climbing



Simulated Annealing