19.5.2015

Optimal and Learning Control

for

Autonomous Robots

Lecture 13

Jonas Buchli
Agile & Dexterous Robotics Lab

Class logistics

Exercise 3 Due: Tue. May 26 - midnight

Interviews: Thu/Fri. May 28/29
https://ethz.doodle.com/bsi7gvkycvrmht6t

A D R L

ETH Zürich

L13

Review on: Gradient descent methods

Analytical optimum

Minima (and maxima) of functions

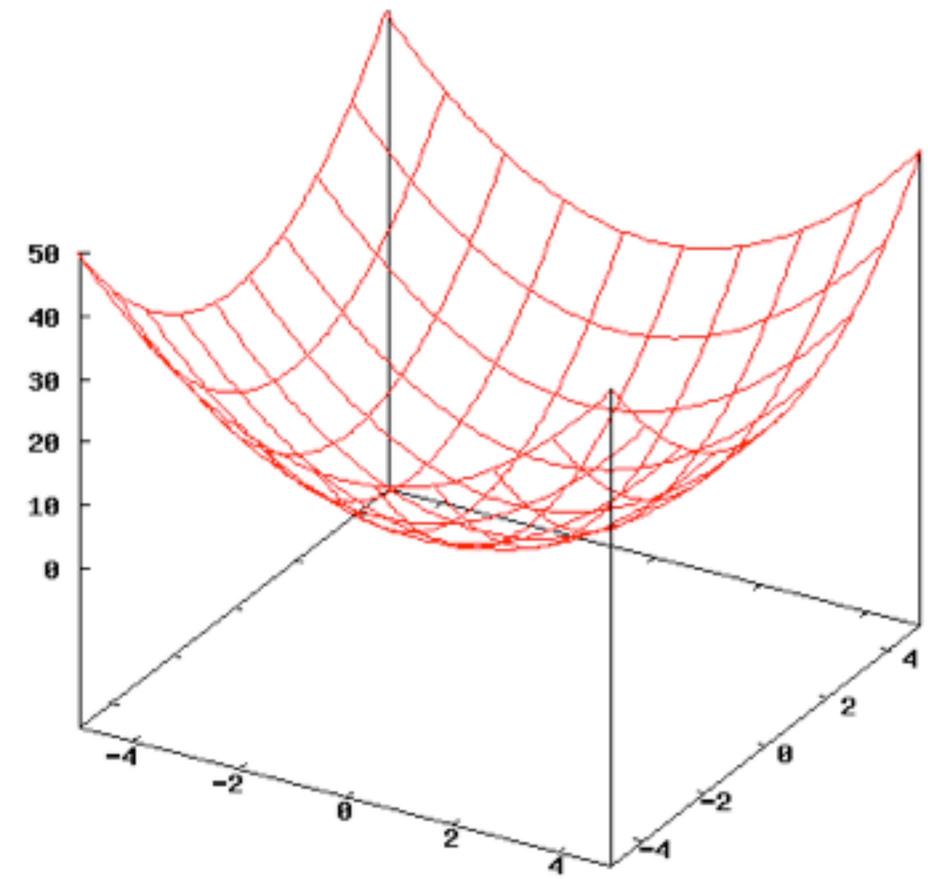
n-dimensional: $C = f(x_1, \ldots, x_n)$

$$\frac{\partial C}{\partial x_i}$$

$$\frac{\partial C}{\partial x_i} = 0$$

$$\nabla C = \left[\frac{\partial C}{\partial x_i}, \ldots, \frac{\partial C}{\partial x_i}\right]^T$$

$$\nabla C = 0$$



Minimum is an ‘inflection point’

- slope is 0

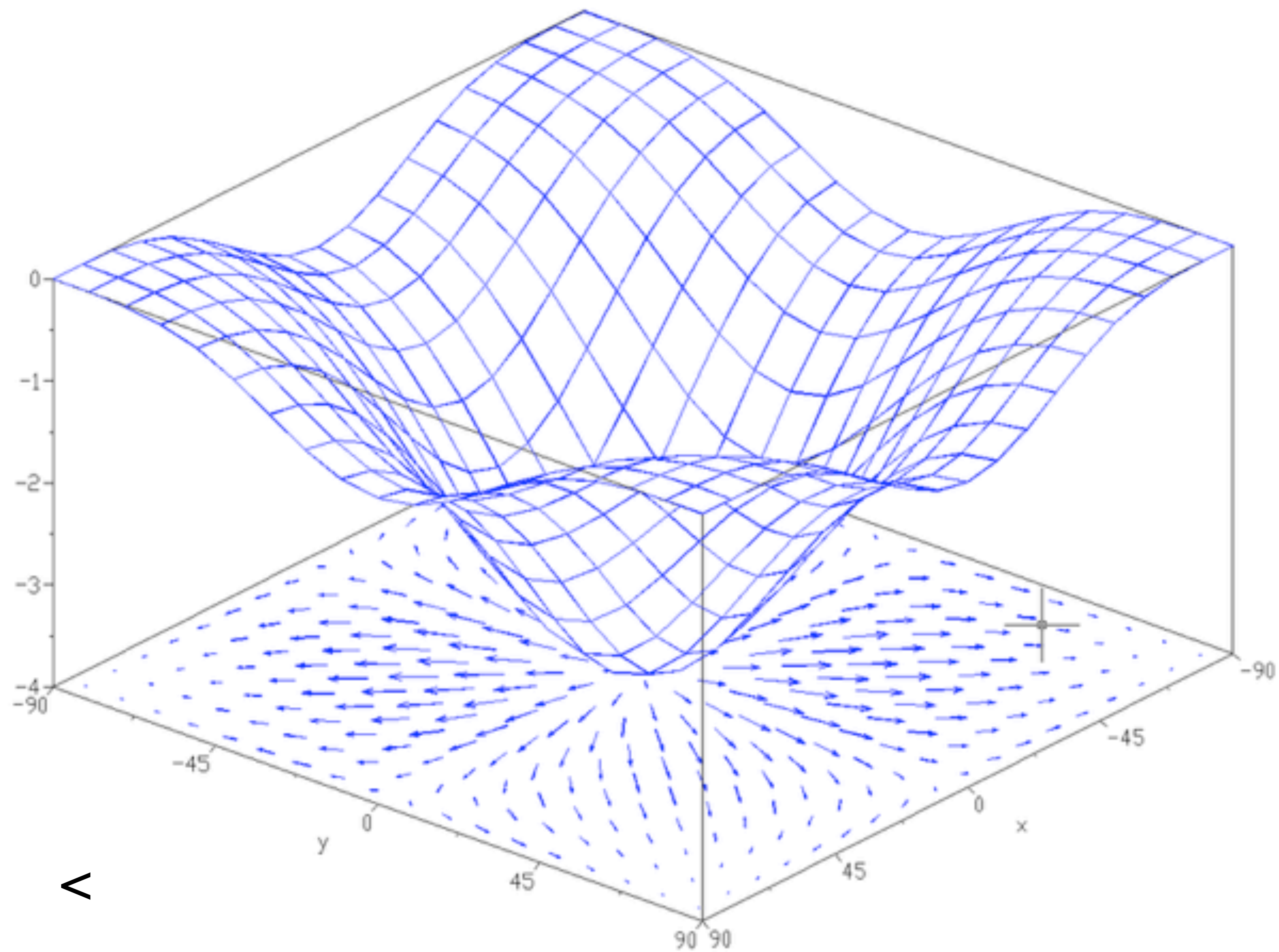
A D R L

ETH Zürich

Gradient descent

- Start from initial x_0
- Loop until convergence
  - $x_{m+1} = x_m - \alpha_m \left[\frac{\partial C(x)}{\partial x} \right]_{x_m}$

Learning Rate



<

Idea: Go to the direction where C decreases, negative direction of gradient vector

Optimal Control problem with parameterized policy

What if instead of finding the optimal control input (which is a function) we find an approximation of that (function approximation)

$$J_{\cos t} = \Phi(t_f) + \int_0^{t_f} L(x_t, u_t)\,\mathrm{d}t$$

$$s.t. \quad \dot{x} = f(x, u)$$

$$J_{\cos t} = \Phi(x_{t_f}) + \sum_{k=t_0}^{t_f} L(x_k, u_k)$$

$$s.t. \quad x_{n+1} = f(x_n, u_n)$$

$$u(n, x, \theta)$$

$$\theta = [\theta_1, \ldots, \theta_p]^T \qquad p \ll \frac{t_f}{\Delta t}$$

Policy Parameterization

Optimal Control problem:

$$J = \phi[\mathbf{x}(t_f), t_f] + \int_{t_0}^{t_f} \mathscr{L}[\mathbf{x}(t), \mathbf{u}(t), t]\, dt$$

w.r.t

$$\dot{\mathbf{x}}(t) = \mathbf{f}[\mathbf{x}(t), \mathbf{u}(t), t], \qquad \mathbf{x}(t_0) = \mathbf{x}_0$$

Parameterize the control

$$\mathbf{u}(t) = \mathbf{u}(\mathbf{k}, t), \qquad t_0 \leq t \leq t_f$$

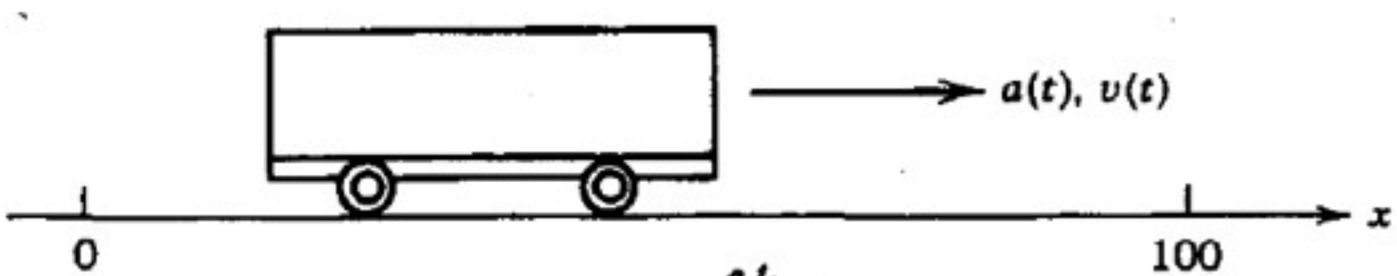
$$\mathbf{u}(\mathbf{k}, t) = \mathbf{k}_1 + \mathbf{k}_2 t + \mathbf{k}_3 t^2 + \cdots$$

$$\mathbf{u}(\mathbf{k}, t) = \sum_{i=1}^{N} \left[\mathbf{k}_{1i} \sin \frac{i\pi t}{(t_f - t_0)} + \mathbf{k}_{2i} \cos \frac{i\pi t}{(t_f - t_0)} \right]$$

If we can write **cost** as an **explicit function** of **parameter vector**

$$\frac{\partial J}{\partial \mathbf{k}} = 0$$

Example: Cart on a track



$$J = q(x_{1_f} - 100)^2 + r\int_0^{t_f} u^2\,dt$$

$$\dot{x} = v \qquad \dot{x}_1 = x_2$$

$$\dot{v} = \frac{f}{m} \qquad \dot{x}_2 = u$$

$$u = k_1 + k_2 t \quad \Rightarrow \quad x_1(t) = \frac{k_1 t^2}{2} + \frac{k_2 t^3}{6}, \quad x_2(t) = k_1 t + \frac{k_2 t^2}{2}$$

$$\Rightarrow \quad J = q[(50k_1 + 166.7k_2) - 100]^2 + r(10k_1^2 + 100k_1k_2 + 333.3k_2^2)$$

$$\frac{\partial J}{\partial k_1} = k_1(500q + 2r) + k_2(166.7q + 10r) - 1000q$$

$$\frac{\partial J}{\partial k_2} = k_1(1666.7q + 10r) + k_2(5555.6q + 66.7r) - 3333.3q$$

$$\Rightarrow \quad \begin{bmatrix} (500q+2r) & (1666.7q+10r) \\ (1666.7q+10r) & (5555.6q+66.7r) \end{bmatrix} \begin{bmatrix} k_1 \\ k_2 \end{bmatrix} = \begin{bmatrix} 1000 \\ 3333.3 \end{bmatrix} q$$

$$\mathbf{A}\mathbf{k} = \mathbf{B}q$$

$$\mathbf{k} = \mathbf{A}^{-1}\mathbf{B}q$$

Analytical gradient descent

$$J_{reward} = \sum_{k=0}^{H} \gamma^k r(x_k, u_k)$$

$$s.t. \quad x_{n+1} = f(x_n, u_n)$$

Policy parameterization

$$u_n = u(n, \mathrm{x}; \theta)$$

Solve the system dynamics analytically

$$x_{n+1} = f(x_n, u_n)$$

$$u_n = u(n, x; \theta)$$

$\rightarrow \; x_n(\theta) \; \rightarrow \; J_{reward}(\theta)$

Simple gradient descent

Algorithm 11 Gradient Descent Algorithm

given

A method to compute $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ for all $\boldsymbol{\theta}$

An initial value for the parameter vector: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_0$

repeat

Compute the cost function gradient at $\boldsymbol{\theta}$

$\mathbf{g} = \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$

Update the parameter vector

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \omega \mathbf{g}$

until convergence

$$x_{n+1} = f(x_n, u_n)$$
$$u_n = u(n, x; \theta)$$

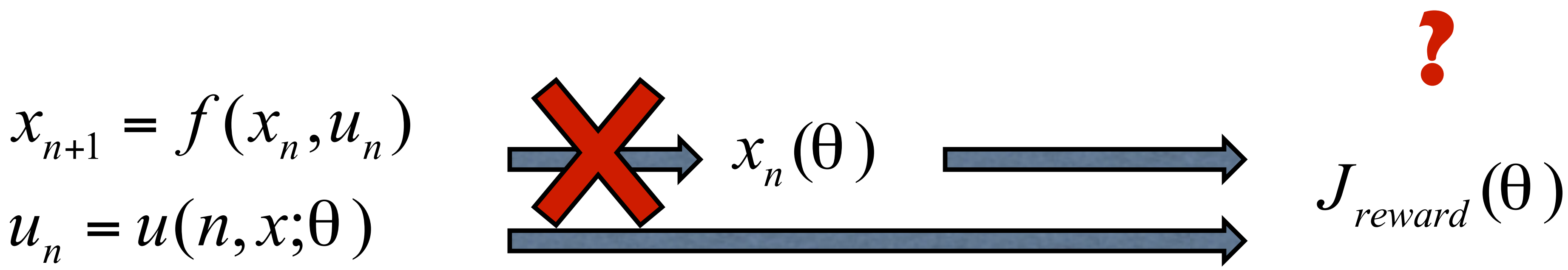
$\rightarrow x_n(\theta) \rightarrow J_{reward}(\theta)$

A D R L

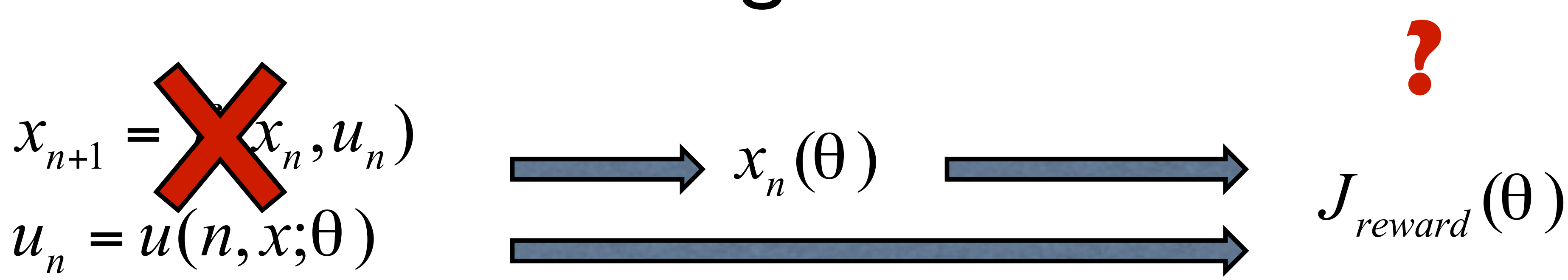
ETH Zürich

Why analytic GD might not be feasible...

- For a general nonlinear system it is not possible



- If the model is not given



A D R L

ETH Zürich

Black box/Model free gradient descent

What if I have no model? I.e. C = f(x) is not known.

Estimate the gradient $g = \left[\nabla J_{reward}(\theta)\right]_{\theta_m}$

Use the environment as 'model': probe, and update based on experience

Estimation of gradient

$$\frac{dJ(\theta)}{d\theta} = \lim_{d\theta \to 0} \frac{J(\theta + d\theta) - J(\theta)}{d\theta}$$

OR

$$\frac{dJ(\theta)}{d\theta} = \lim_{d\theta \to 0} \frac{J(\theta + d\theta/2) - J(\theta - d\theta/2)}{d\theta}$$

$$\frac{dJ(\theta)}{d\theta} \approx \frac{J(\theta + \Delta\theta) - J(\theta)}{\Delta\theta}$$

one-sided estimate

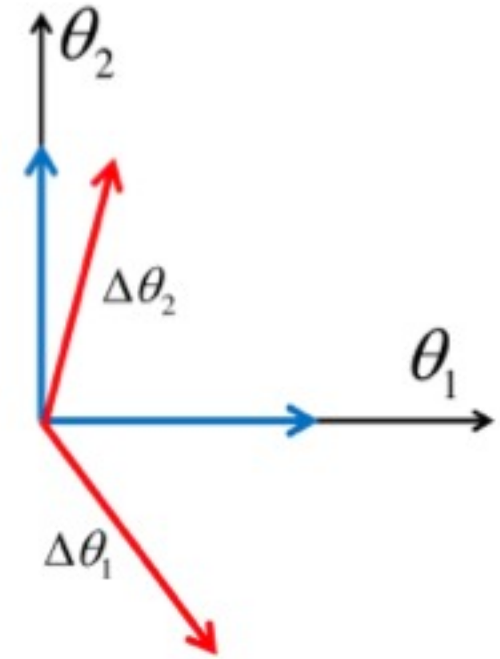
$$\frac{dJ(\theta)}{d\theta} \approx \frac{J(\theta + \Delta\theta/2) - J(\theta - \Delta\theta/2)}{\Delta\theta}$$

two-sided estimate

$$\nabla J(\theta_1, \theta_2, \ldots, \theta_p) = \left[\frac{\partial J}{\partial \theta_1}, \frac{\partial J}{\partial \theta_2}, \ldots, \frac{\partial J}{\partial \theta_p}\right]$$

Leads to **Finite Difference Method**

Example



Assume two parameters $\theta = [\theta_1 \quad \theta_2]^T$

Thus need three samples of the cost:

(e.g. measurement at point of interest, and two one-sided estimates)

$$J(\boldsymbol{\theta}),\ J(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}_1),\ J(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}_2)$$

$$\frac{dJ(\theta)}{d\theta} \approx \frac{J(\theta + \Delta\theta) - J(\theta)}{\Delta\theta}$$

$$\nabla J(\theta_1, \theta_2, \dots, \theta_p) = \left[\frac{\partial J}{\partial\theta_1}, \frac{\partial J}{\partial\theta_2}, \dots, \frac{\partial J}{\partial\theta_p}\right]$$

$$J(\boldsymbol{\theta} + \Delta\theta_1) \approx J(\boldsymbol{\theta}) + \Delta\boldsymbol{\theta}_1^T \nabla J(\boldsymbol{\theta})$$

$$J(\boldsymbol{\theta} + \Delta\theta_2) \approx J(\boldsymbol{\theta}) + \Delta\boldsymbol{\theta}_2^T \nabla J(\boldsymbol{\theta})$$

$$J(\boldsymbol{\theta} + \Delta\theta_1) \approx J(\boldsymbol{\theta}) + \Delta\boldsymbol{\theta}_1^T \nabla J(\boldsymbol{\theta})$$

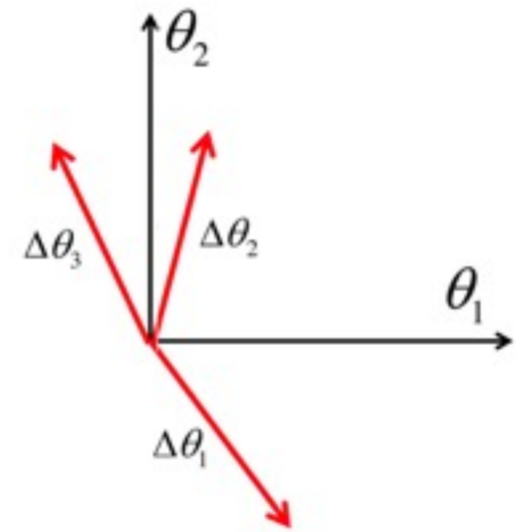
$$J(\boldsymbol{\theta} + \Delta\theta_2) \approx J(\boldsymbol{\theta}) + \Delta\boldsymbol{\theta}_2^T \nabla J(\boldsymbol{\theta})$$

$$\begin{bmatrix} \Delta\boldsymbol{\theta}_1^T \\ \Delta\boldsymbol{\theta}_2^T \end{bmatrix} \nabla J(\boldsymbol{\theta}) = \begin{bmatrix} J(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}_1) - J(\boldsymbol{\theta}) \\ J(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}_2) - J(\boldsymbol{\theta}) \end{bmatrix}$$

if perturbations not parallel $\begin{bmatrix} \Delta\boldsymbol{\theta}_1^T \\ \Delta\boldsymbol{\theta}_2^T \end{bmatrix}$ is a 2-by-2 invertible matrix

$$\nabla J(\boldsymbol{\theta}) = \begin{bmatrix} \Delta\boldsymbol{\theta}_1^T \\ \Delta\boldsymbol{\theta}_2^T \end{bmatrix}^{-1} \begin{bmatrix} J(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}_1) - J(\boldsymbol{\theta}) \\ J(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}_2) - J(\boldsymbol{\theta}) \end{bmatrix}$$

Example 2



Three one sided perturbations

$$J(\boldsymbol{\theta}+\Delta\boldsymbol{\theta}_1),\ J(\boldsymbol{\theta}+\Delta\boldsymbol{\theta}_2),\ J(\boldsymbol{\theta}+\boldsymbol{\theta}_3)$$

$$J(\boldsymbol{\theta}+\Delta\boldsymbol{\theta}_1) = J(\boldsymbol{\theta}) + \Delta\boldsymbol{\theta}_1^T \nabla J(\boldsymbol{\theta})$$
$$J(\boldsymbol{\theta}+\Delta\boldsymbol{\theta}_2) = J(\boldsymbol{\theta}) + \Delta\boldsymbol{\theta}_2^T \nabla J(\boldsymbol{\theta})$$
$$J(\boldsymbol{\theta}+\Delta\boldsymbol{\theta}_3) = J(\boldsymbol{\theta}) + \Delta\boldsymbol{\theta}_3^T \nabla J(\boldsymbol{\theta})$$

unknown $J(\boldsymbol{\theta})$ and $\nabla J(\boldsymbol{\theta})$

$$J(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}_1) = J(\boldsymbol{\theta}) + \Delta\boldsymbol{\theta}_1^T \nabla J(\boldsymbol{\theta})$$
$$J(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}_2) = J(\boldsymbol{\theta}) + \Delta\boldsymbol{\theta}_2^T \nabla J(\boldsymbol{\theta})$$
$$J(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}_3) = J(\boldsymbol{\theta}) + \Delta\boldsymbol{\theta}_3^T \nabla J(\boldsymbol{\theta})$$

in matrix from

$$\begin{bmatrix} \Delta\boldsymbol{\theta}_1^T & 1 \\ \Delta\boldsymbol{\theta}_2^T & 1 \\ \Delta\boldsymbol{\theta}_3^T & 1 \end{bmatrix} \underbrace{\begin{bmatrix} \nabla J(\boldsymbol{\theta}) \\ J(\boldsymbol{\theta}) \end{bmatrix}}_{\text{unknown}} = \begin{bmatrix} J(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}_1) \\ J(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}_2) \\ J(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}_3) \end{bmatrix}$$

if perturbations pair-wise independent, invertible:

$$\begin{bmatrix} \nabla J(\boldsymbol{\theta}) \\ J(\boldsymbol{\theta}) \end{bmatrix} = \begin{bmatrix} \Delta\boldsymbol{\theta}_1^T & 1 \\ \Delta\boldsymbol{\theta}_2^T & 1 \\ \Delta\boldsymbol{\theta}_3^T & 1 \end{bmatrix}^{-1} \begin{bmatrix} J(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}_1) \\ J(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}_2) \\ J(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}_3) \end{bmatrix}$$

General FD

General case for p+1 samples:

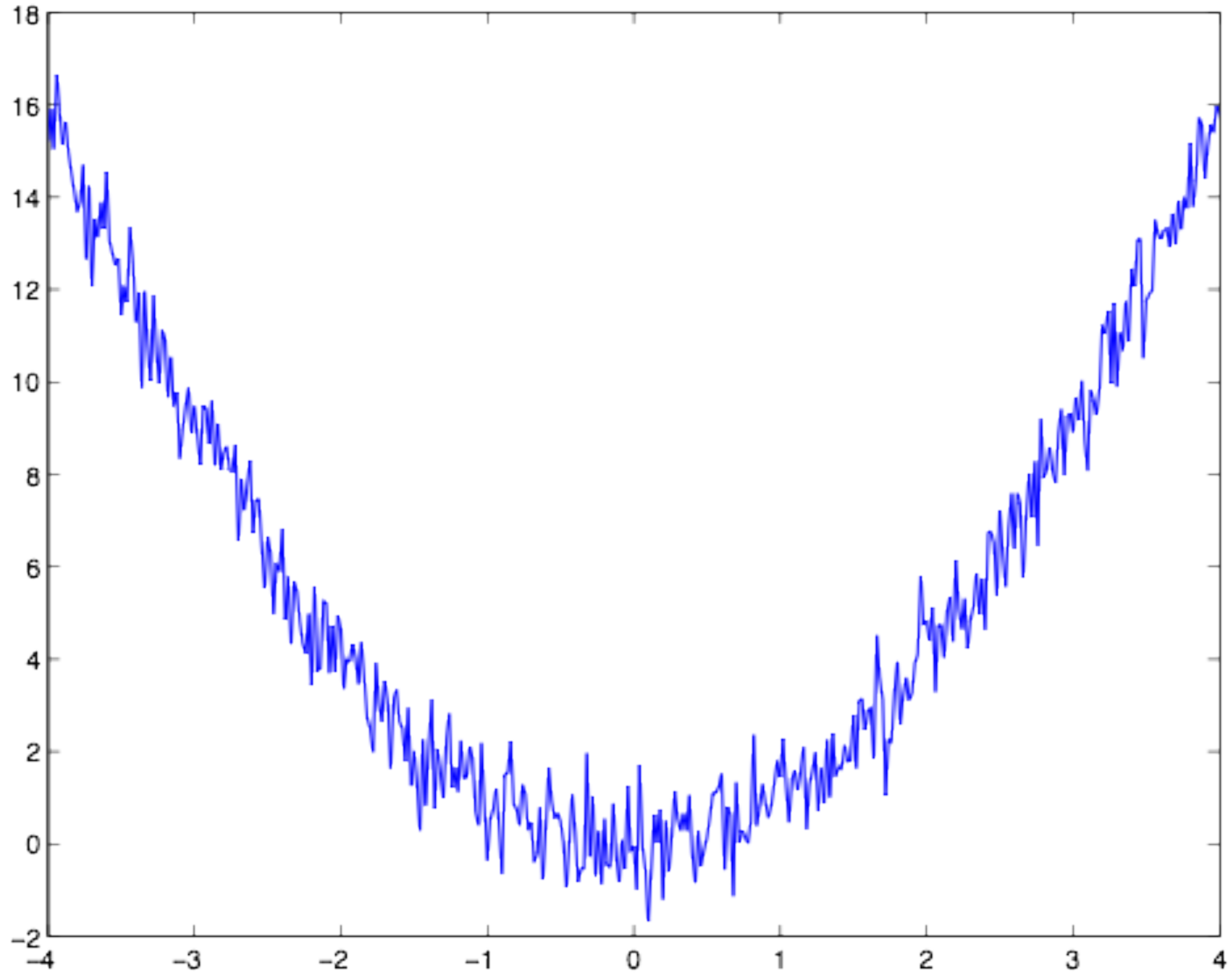
$$
\begin{aligned}
J(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}_1) &= J(\boldsymbol{\theta}) + \Delta\boldsymbol{\theta}_1^T \nabla J(\boldsymbol{\theta}) \\
J(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}_2) &= J(\boldsymbol{\theta}) + \Delta\boldsymbol{\theta}_2^T \nabla J(\boldsymbol{\theta}) \\
&\vdots \\
J(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}_p) &= J(\boldsymbol{\theta}) + \Delta\boldsymbol{\theta}_p^T \nabla J(\boldsymbol{\theta}) \\
J(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}_{p+1}) &= J(\boldsymbol{\theta}) + \Delta\boldsymbol{\theta}_{p+1}^T \nabla J(\boldsymbol{\theta})
\end{aligned}
\implies
\underbrace{\begin{bmatrix} J(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}_1) \\ J(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}_2) \\ \vdots \\ J(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}_{p+1}) \end{bmatrix}}_{\mathbf{J}}
=
\underbrace{\begin{bmatrix} \Delta\boldsymbol{\theta}_1^T & 1 \\ \Delta\boldsymbol{\theta}_2^T & 1 \\ \vdots & \\ \Delta\boldsymbol{\theta}_{p+1}^T & 1 \end{bmatrix}}_{\Delta\boldsymbol{\Theta}}
\begin{bmatrix} \nabla J(\boldsymbol{\theta}) \\ J(\boldsymbol{\theta}) \end{bmatrix}
$$

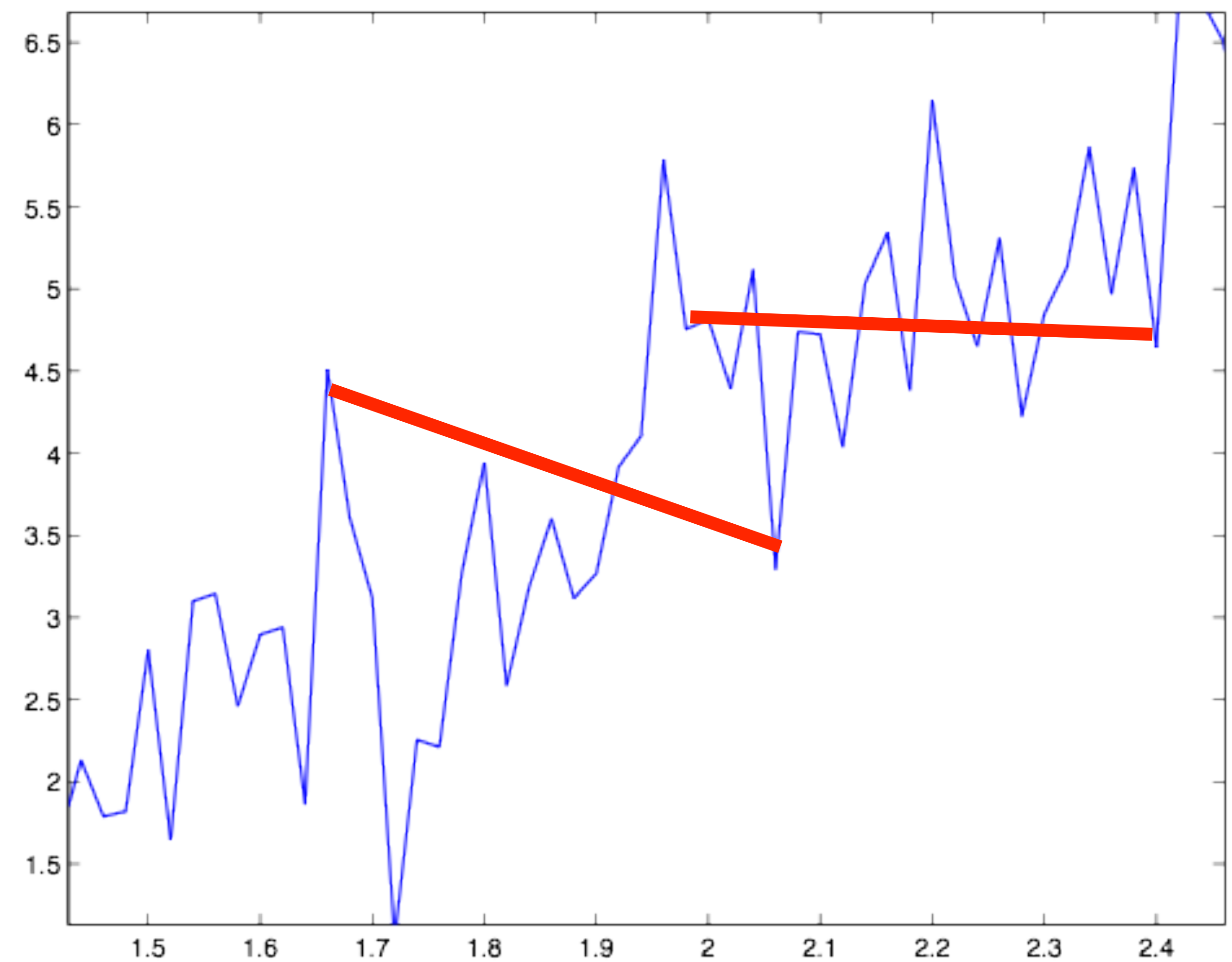
$$
\begin{bmatrix} \nabla J(\boldsymbol{\theta}) \\ J(\boldsymbol{\theta}) \end{bmatrix} = \Delta\boldsymbol{\Theta}^{-1}\mathbf{J}
$$

Stochastic Case

Stochastic problems

Noisy cost function





What about gradient now?

Stochastic FD

Return for a single rollout

$$R = \Phi(\mathbf{x}(N)) + \sum_{k=0}^{N-1} L_k\left(\mathbf{x}(k), \mathbf{u}(k)\right)$$

Cost is expected return

$$J = E[R]$$

Approximate return by averaging (K rollouts)

need $K \times (p+1)$ evaluations of the cost function

A D R L

ETH Zürich

Expected gradient

need $K \times (p+1)$ evaluations of the cost function

$$\underbrace{\begin{bmatrix} \frac{1}{K}\sum_{k=1}^{K} R^k(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}_1) \\ \frac{1}{K}\sum_{k=1}^{K} R^k(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}_2) \\ \vdots \\ \frac{1}{K}\sum_{k=1}^{K} R^k(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}_{p+1}) \end{bmatrix}}_{\mathbf{J}} = \underbrace{\begin{bmatrix} \Delta\boldsymbol{\theta}_1^T & 1 \\ \Delta\boldsymbol{\theta}_2^T & 1 \\ \vdots & \\ \Delta\boldsymbol{\theta}_{p+1}^T & 1 \end{bmatrix}}_{\Delta\boldsymbol{\Theta}} \begin{bmatrix} \nabla J(\boldsymbol{\theta}) \\ J(\boldsymbol{\theta}) \end{bmatrix}$$

$$\begin{bmatrix} \nabla J(\boldsymbol{\theta}) \\ J(\boldsymbol{\theta}) \end{bmatrix} = \Delta\boldsymbol{\Theta}^{-1}\mathbf{J}$$

Instead of doing many (similar) perturbations

$$N \leq K \times (p+1)$$

$$\underbrace{\begin{bmatrix} R(\boldsymbol{\theta}+\Delta\boldsymbol{\theta}_1) \\ R(\boldsymbol{\theta}+\Delta\boldsymbol{\theta}_2) \\ \vdots \\ R(\boldsymbol{\theta}+\Delta\boldsymbol{\theta}_N) \end{bmatrix}}_{\mathbf{R}} = \underbrace{\begin{bmatrix} \Delta\boldsymbol{\theta}_1^T & 1 \\ \Delta\boldsymbol{\theta}_2^T & 1 \\ \vdots & \\ \Delta\boldsymbol{\theta}_N^T & 1 \end{bmatrix}}_{\Delta\boldsymbol{\Theta}} \begin{bmatrix} \nabla J(\boldsymbol{\theta}) \\ J(\boldsymbol{\theta}) \end{bmatrix}$$

$\Delta\boldsymbol{\Theta}$ is $N \times (p+1)$

If $N \geq p+1$

$\Delta\boldsymbol{\Theta}$ rank $p+1$

use left pseudoinverse

$$\begin{bmatrix} \nabla J(\boldsymbol{\theta}) \\ J(\boldsymbol{\theta}) \end{bmatrix} = \Delta\boldsymbol{\Theta}^{\dagger}\mathbf{R} = (\Delta\boldsymbol{\Theta}^T\Delta\boldsymbol{\Theta})^{-1}\Delta\boldsymbol{\Theta}^T\mathbf{R}$$

Finite difference - general

$$\begin{bmatrix} \nabla J(\boldsymbol{\theta}) \\ J(\boldsymbol{\theta}) \end{bmatrix} = \Delta\boldsymbol{\Theta}^{\dagger}\mathbf{R} = (\Delta\boldsymbol{\Theta}^T\Delta\boldsymbol{\Theta} + \lambda\mathbf{I})^{-1}\Delta\boldsymbol{\Theta}^T\mathbf{R}$$

$$\mathbf{R} = \begin{bmatrix} R(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}_1) \\ R(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}_2) \\ \vdots \\ R(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}_N) \end{bmatrix}, \qquad \Delta\boldsymbol{\Theta} = \begin{bmatrix} \Delta\boldsymbol{\theta}_1^T & 1 \\ \Delta\boldsymbol{\theta}_2^T & 1 \\ \vdots & \\ \Delta\boldsymbol{\theta}_N^T & 1 \end{bmatrix}$$

Algorithm 12 Gradient Descend Algorithm with Finite Difference Method

given

The cost function:

$$J = E\left[\Phi(\mathbf{x}(N)) + \sum_{k=0}^{N-1} L_k\left(\mathbf{x}(k), \mathbf{u}(k)\right)\right]$$

A policy (function approximation) for the control input: $\mathbf{u}(n, \mathbf{x}) = \mu(n, \mathbf{x}; \boldsymbol{\theta})$

An initial value for the parameter vector: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_0$

The parameter exploration standard deviation: c

The regularization coefficient: λ

The learning rate: ω

repeat

Create N rollouts of the system with the perturbed parameters $\boldsymbol{\theta} + \Delta\boldsymbol{\theta}, \ \Delta\boldsymbol{\theta} \sim \mathcal{N}(\mathbf{0}, c^2\mathbf{I})$

Calculate the return from the initial time and state for the nth rollout:

$$R(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}_n) = \Phi(\mathbf{x}(N)) + \sum_{k=0}^{N-1} L_k\left(\mathbf{x}(k), \mathbf{u}(k)\right)$$

Construct $\mathbf{R}$ and $\boldsymbol{\Theta}$ matrices as:

$$\mathbf{R}_{N\times 1} = \left[R(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}_n)\right]_n, \ \boldsymbol{\Theta}_{N\times(p+1)} = \left[\Delta\boldsymbol{\theta}_n^T \ 1\right]_n$$

Calculate the value and gradient of the cost function at $\boldsymbol{\theta}$

$$\begin{bmatrix} \nabla J(\boldsymbol{\theta}) \\ J(\boldsymbol{\theta}) \end{bmatrix} = (\Delta\boldsymbol{\Theta}^T \Delta\boldsymbol{\Theta} + \lambda\mathbf{I})^{-1} \Delta\boldsymbol{\Theta}^T \mathbf{R}$$

Update the parameter vector:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \omega\nabla J(\boldsymbol{\theta})$$

until convergence

Learning Rate
&
Stability of gradient descent

Stability of maps

$$x_{m+1} = x_m - \gamma_m \left[\frac{\partial C(x)}{\partial x} \right]_{x_m} \leftrightarrow x_{m+1} = T(\mathrm{x}_m)$$

Small perturbation should be damped out

$$x'_m = \mathrm{x}_m + \Delta \mathrm{x}_m$$

$$x'_{m+1} = \mathrm{x}_{m+1} + \Delta \mathrm{x}_{m+1}$$

$$\Delta \mathrm{x}_{m+1} = \left[\frac{dT}{d\,\mathrm{x}} \right]_{\mathrm{x}_m} \Delta \mathrm{x}_m$$

$\Leftrightarrow$

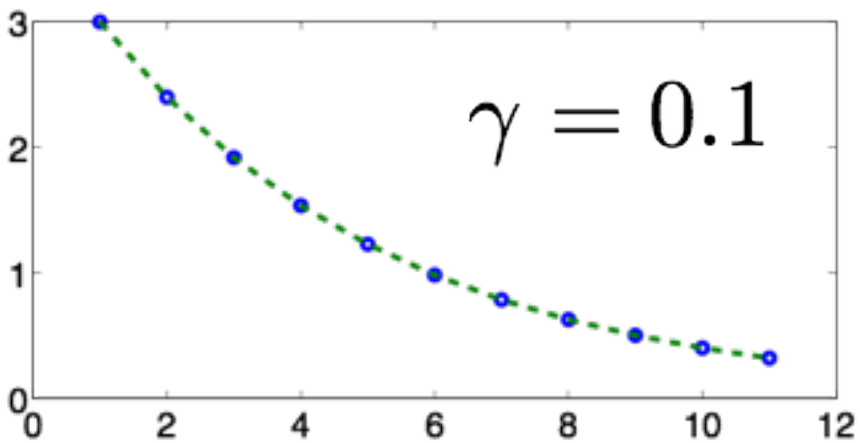
$$-1 < \left[\frac{dT}{d\,\mathrm{x}} \right]_{\mathrm{x}_m} < 1$$

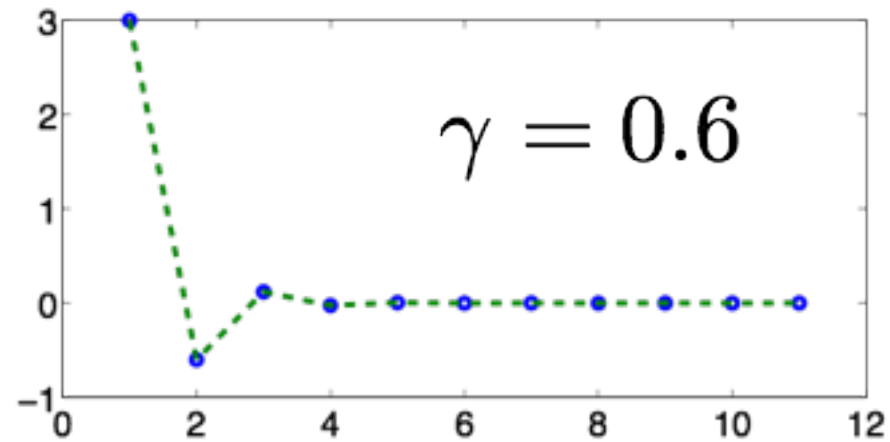
Taylor first order approximation

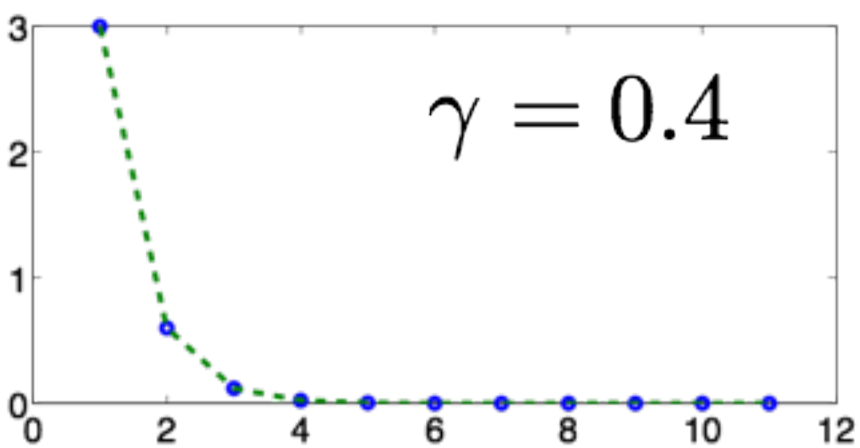
A D R L

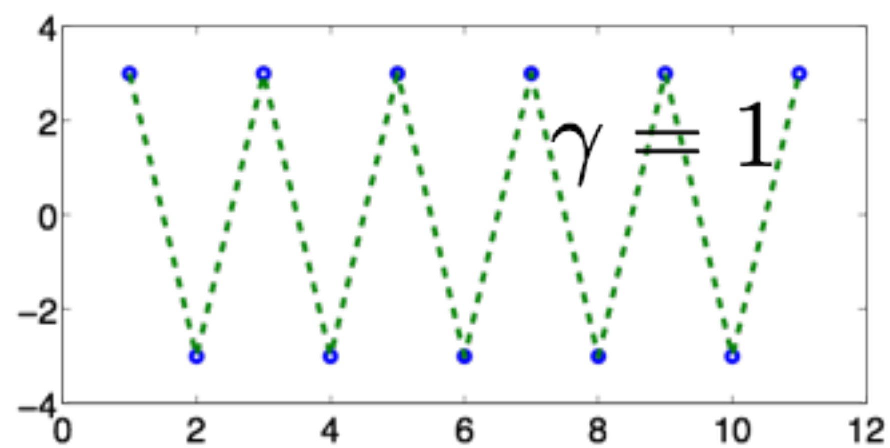
ETH Zürich

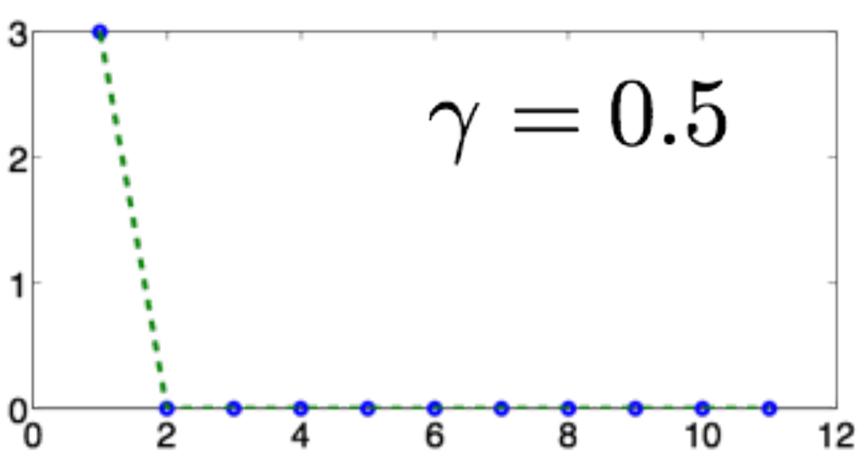
Numerical examples











$$C = x^2 + b \quad \Rightarrow \quad \frac{dC}{dx} = 2x$$

$$1 > \frac{d}{dx}(x - \gamma 2x) > -1$$

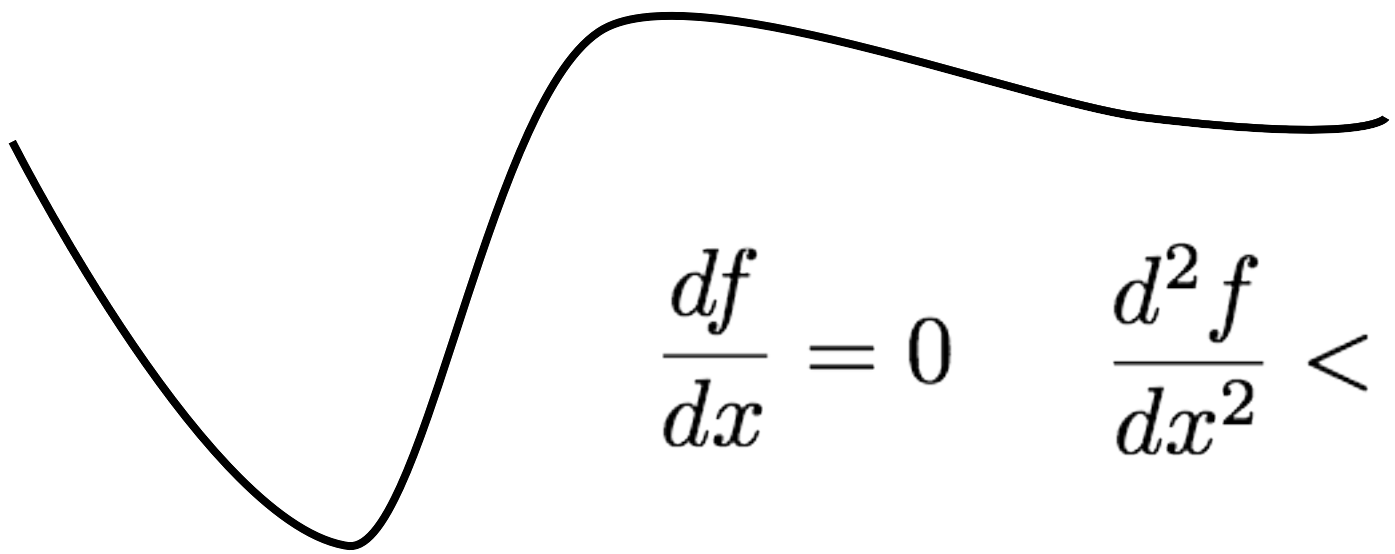
$$1 > (1 - 2\gamma) > -1 \quad \Rightarrow \quad \gamma \in (0, 1)$$

A D R L

ETH Zürich

Stability of gradient descent

Basic stability:



$$\frac{df}{dx} = 0 \qquad \frac{d^2 f}{dx^2} < 0$$

2-nd derivative tells about stability,
for n-DOF: Hessian

Step size:

Exploitation vs. Exploration

Choice of learning rate parameter vs. local minima, convergence speed

Solution is often adaptive learning rate, but...

- freezing to quickly, get stuck in local minimum
- freezing too slowly, slow convergence, wild oscillations in solutions

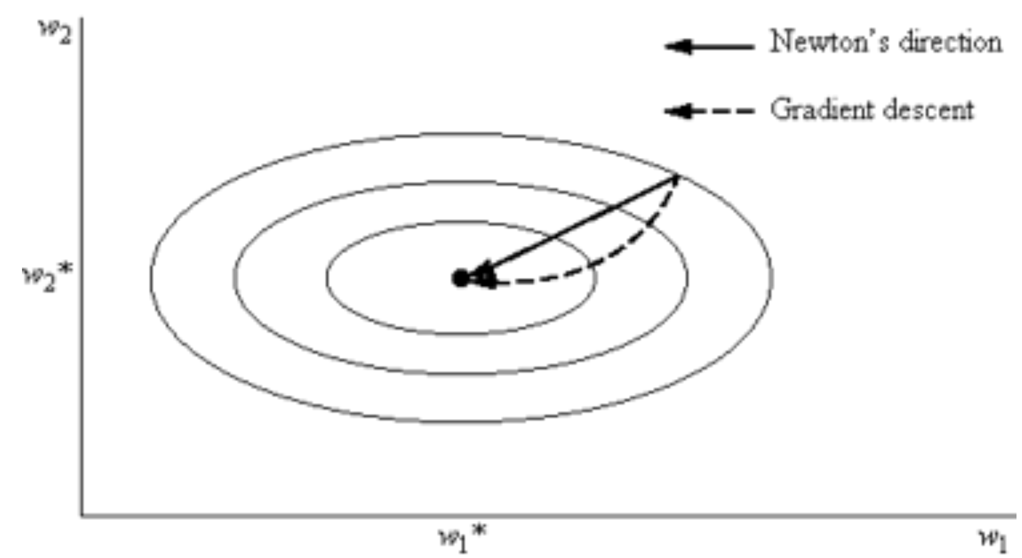
Newton Method

- Faster convergence
- Less sensitivity to the learning rate

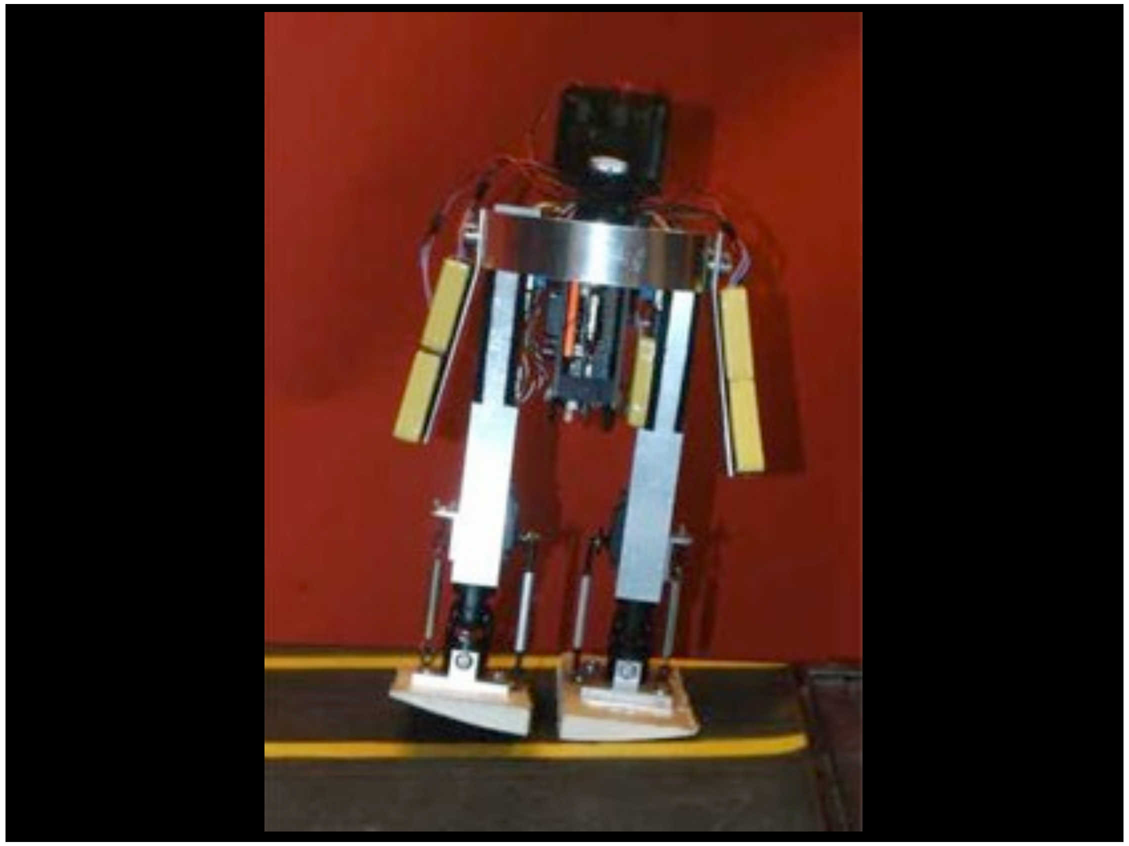
$$x_{m+1} = x_m - \gamma_m \left(H_c(x_m)\right)^{-1} \nabla C(x_m)$$

$$\nabla C(x) = \frac{\partial C}{\partial x}$$

$$H_c(x) = \frac{\partial^2 C}{\partial x^2}$$



A real implementation on a robot



Watch on YouTube "Robot Learning to Walk (Toddler)"

https://www.youtube.com/watch?v=goqWX7bC-ZY

Locomotion Skills for Simulated Quadrupeds

Stelian Coros[1,2] Andrej Karpathy[1] Ben Jones[1]
Lionel Reveret[3] Michiel van de Panne[1]

[1]University of British Columbia [2]Disney Research Zurich
[3]INRIA, Grenoble University, CNRS

Flexible Muscle-Based Locomotion for Bipedal Creatures

SIGGRAPH ASIA 2013

Thomas Geijtenbeek
Michiel van de Panne
Frank van der Stappen

Problems of FD

- Multiple minima
- Non-smooth cost functions
- performance
- robustness
- step size - exploration vs. exploitation
- noise
- at min. gradient = 0
- when converged?
- Lots of algorithmic parameters
- One update might be computationally intensive