EE613

Machine Learning for Engineers

LINEAR REGRESSION I

Sylvain Calinon
Robot Learning & Interaction Group
Idiap Research Institute

Oct. 31, 2019

EE613 - List of courses

19.09.2019 (JMO) Introduction
26.09.2019 (JMO) Generative I
03.10.2019 (JMO) Generative II
10.10.2019 (JMO) Generative III
17.10.2019 (JMO) Generative IV
24.10.2019 (JMO) Decision-trees
31.10.2019 (SC) Linear regression I
07.11.2019 (JMO) Kernel SVM
14.11.2019 (SC) Linear regression II
21.11.2019 (FF) MLP
28.11.2019 (FF) Feature-selection and boosting
05.12.2019 (SC) HMM and subspace clustering
12.12.2019 (SC) Nonlinear regression I
19.12.2019 (SC) Nonlinear regression II

Outline

Linear Regression I (Oct 31)

- Least squares
- Singular value decomposition (SVD)
- Kernels in least squares (nullspace)
- Ridge regression (Tikhonov regularization)
- Weighted least squares
- Iteratively reweighted least squares (IRLS)
- Recursive least squares

Linear Regression II (Nov 14)

- Logistic regression
- Tensor-variate regression

Hidden Markov model (HMM) & subspace clustering (Dec 5)

Nonlinear Regression I (Dec 12)

- Locally weighted regression (LWR)
- Gaussian mixture regression (GMR)

Nonlinear Regression II (Dec 19)

- Gaussian process regression (GPR)

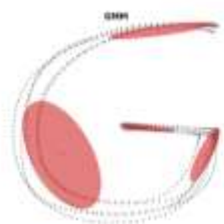
Labs

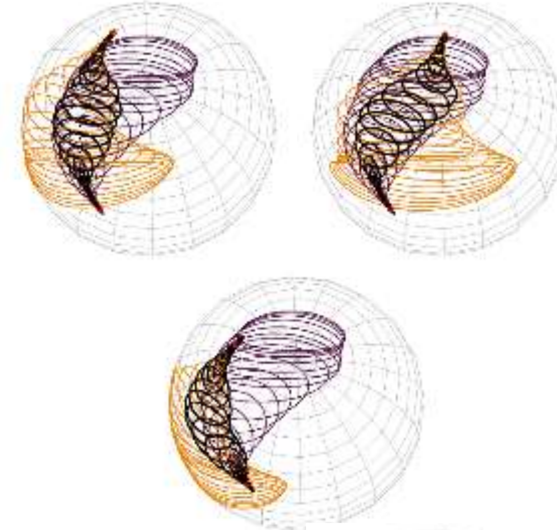
Teguh Lembono

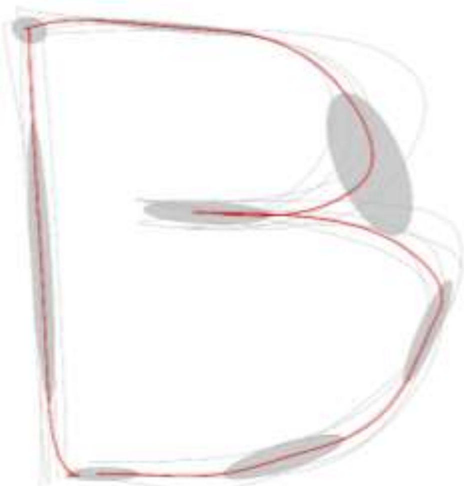
Python notebooks and labs exercises:

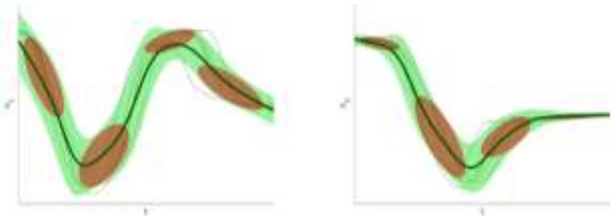
https://github.com/teguhSL/ee613-python

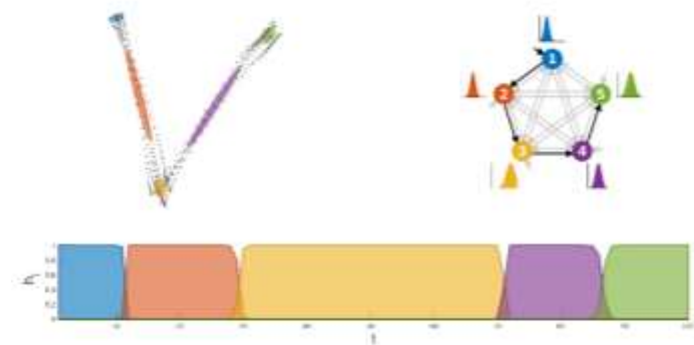
PbDlib

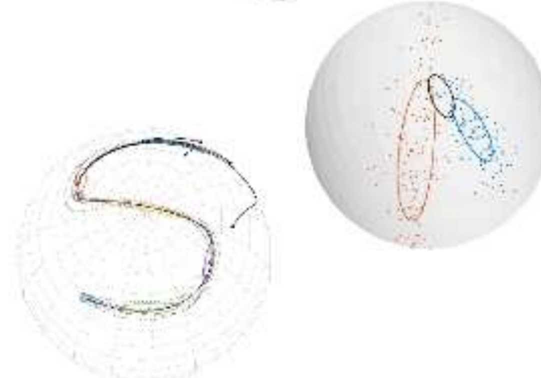


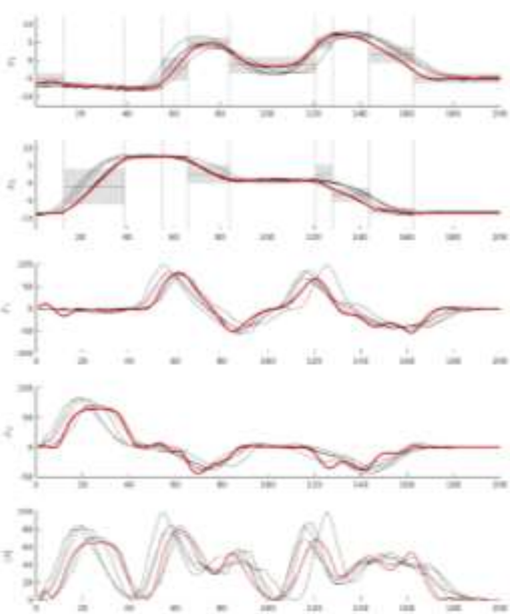


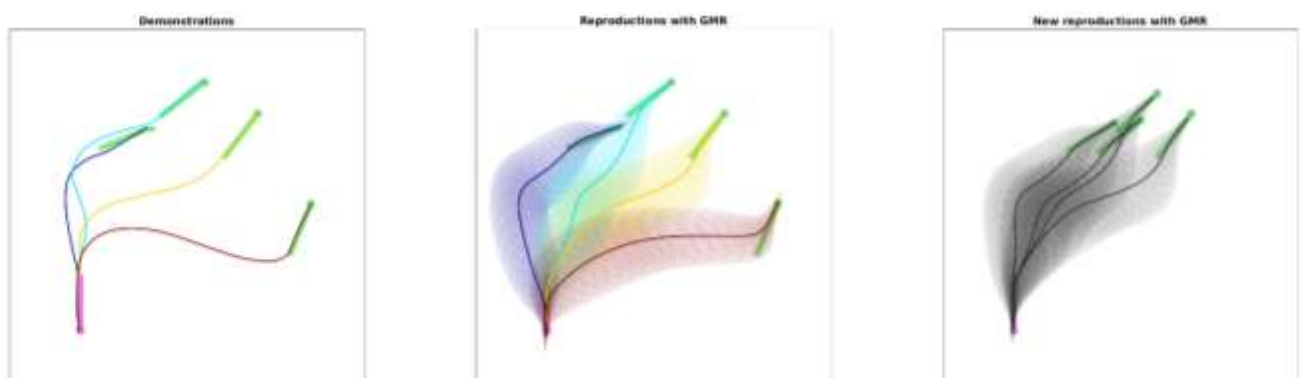












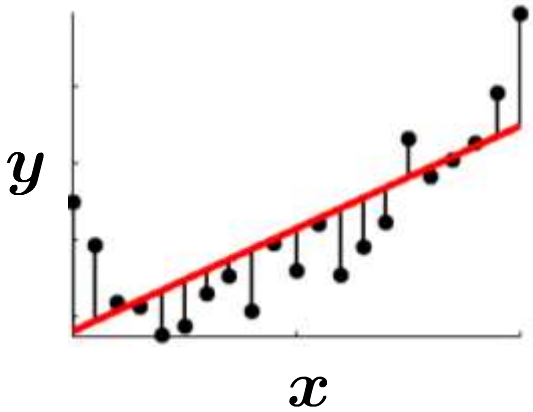
http://www.idiap.ch/software/pbdlib/

LEAST SQUARES

circa 1795

Least squares: a ubiquitous tool

$$\hat{a} = X^{\dagger}\, y$$



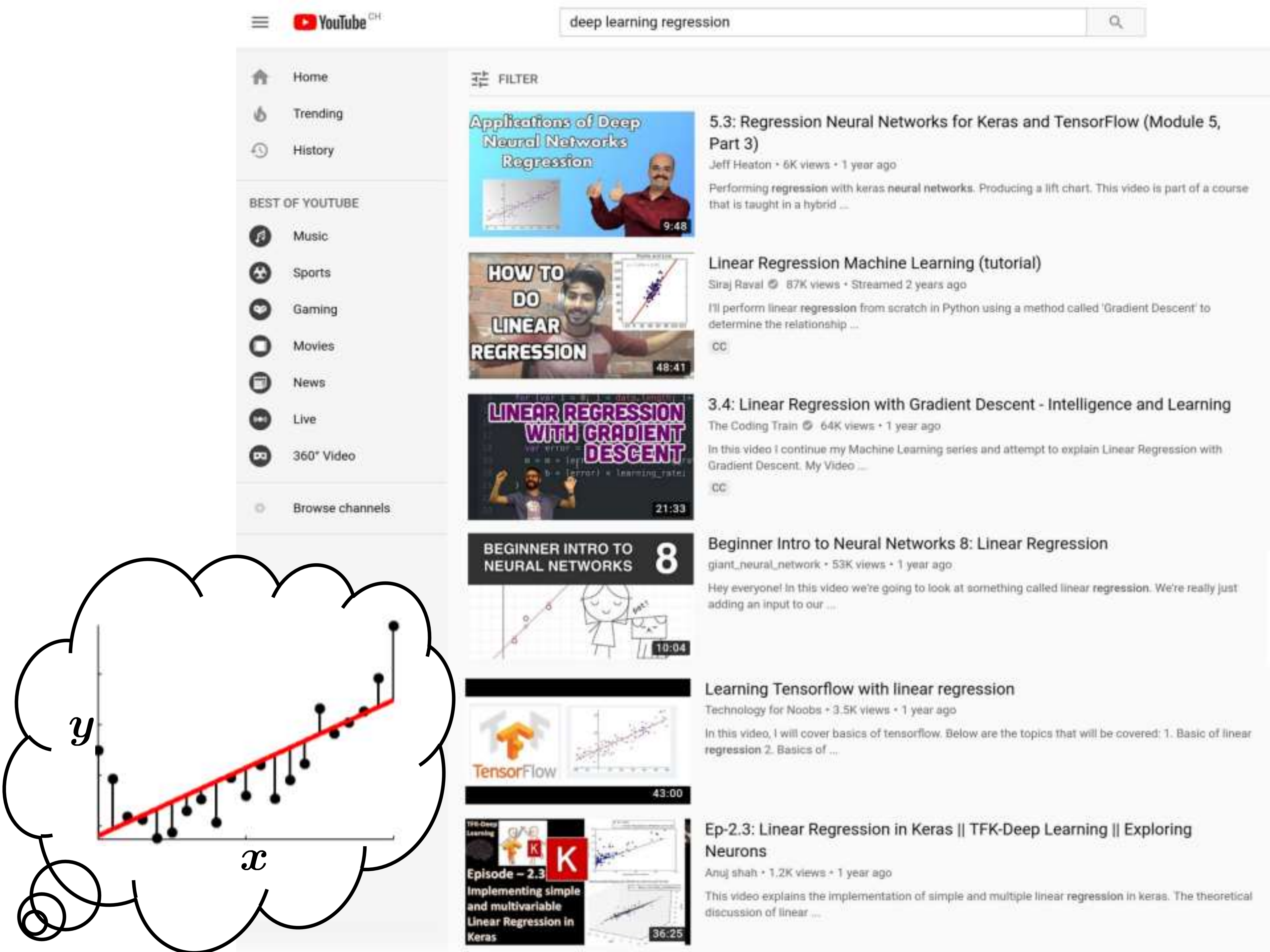
Weighted least squares?

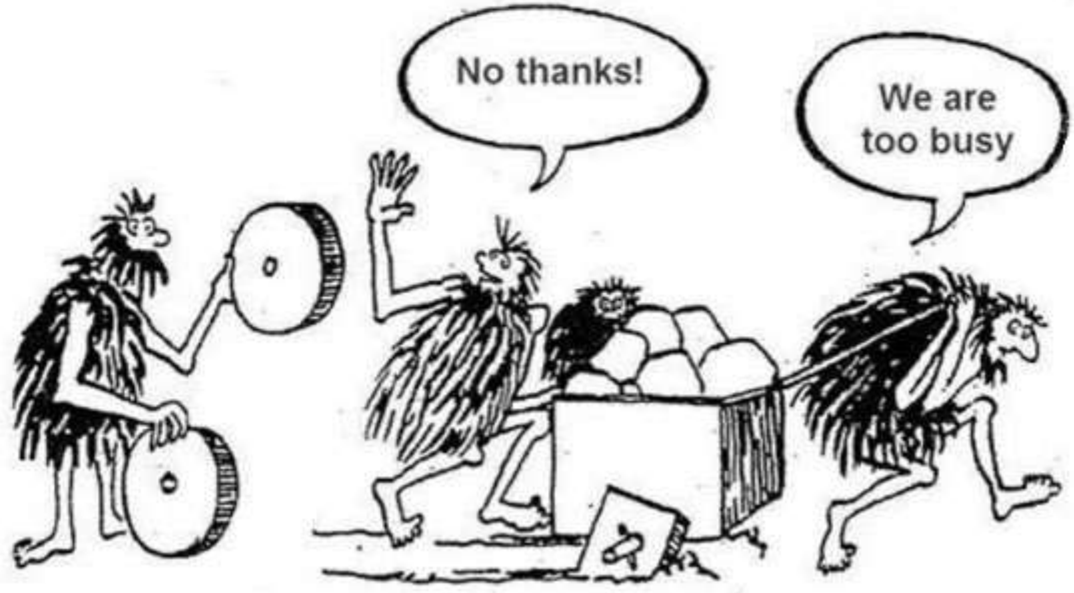
Regularized least squares?

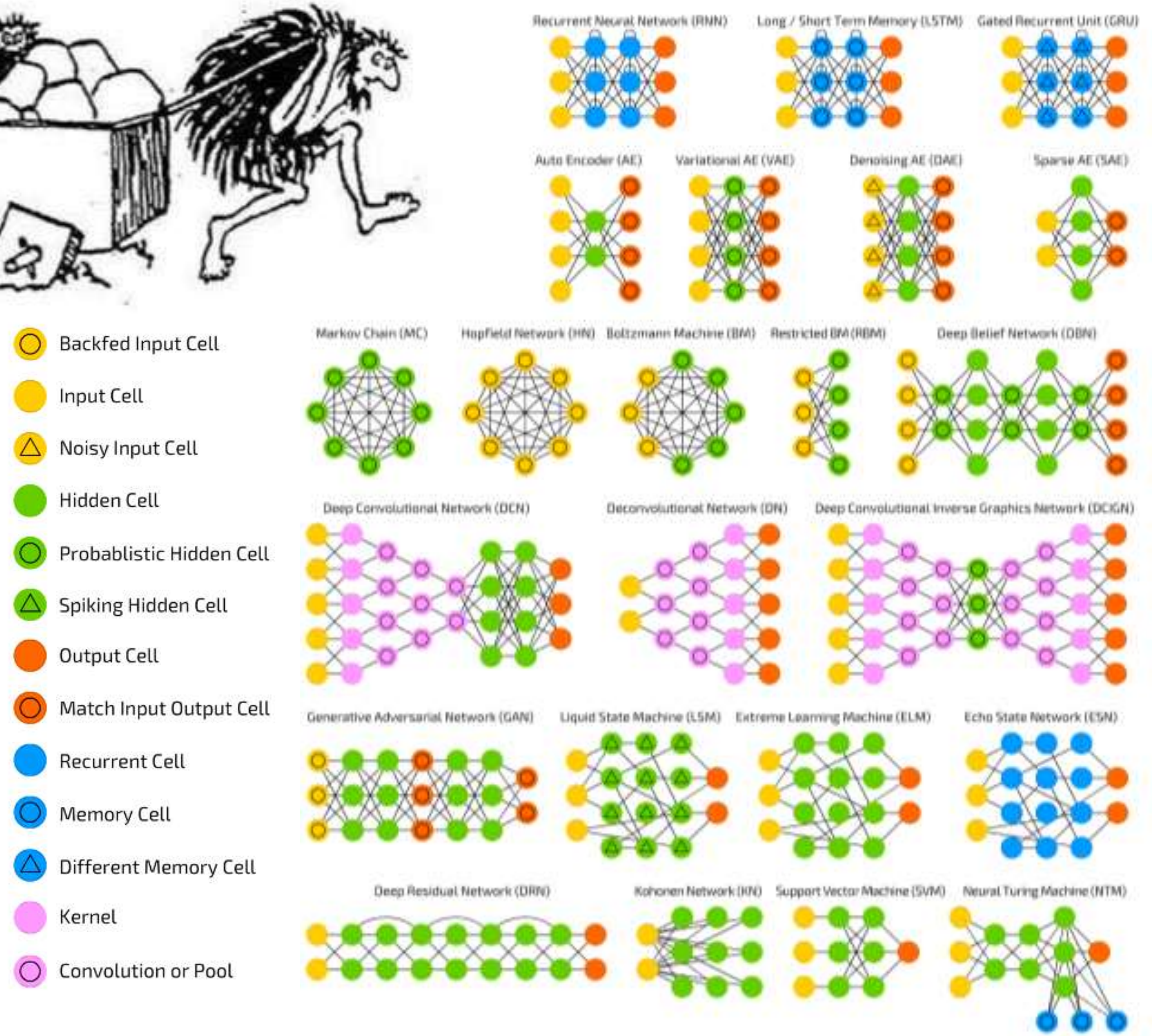
L1-norm instead of L2-norm?

Nullspace structure?

Recursive computation?


YouTube CH
deep learning regression
Home
Trending
History
BEST OF YOUTUBE
Music
Sports
Gaming
Movies
News
Live
360° Video
Browse channels
FILTER
Applications of Deep Neural Networks Regression
9:48
5.3: Regression Neural Networks for Keras and TensorFlow (Module 5, Part 3)
Jeff Heaton • 6K views • 1 year ago
Performing regression with keras neural networks. Producing a lift chart. This video is part of a course that is taught in a hybrid ...
HOW TO DO LINEAR REGRESSION
48:41
Linear Regression Machine Learning (tutorial)
Siraj Raval • 87K views • Streamed 2 years ago
I'll perform linear regression from scratch in Python using a method called 'Gradient Descent' to determine the relationship ...
CC
LINEAR REGRESSION WITH GRADIENT DESCENT
21:33
3.4: Linear Regression with Gradient Descent - Intelligence and Learning
The Coding Train • 64K views • 1 year ago
In this video I continue my Machine Learning series and attempt to explain Linear Regression with Gradient Descent. My Video ...
CC
BEGINNER INTRO TO NEURAL NETWORKS 8
10:04
Beginner Intro to Neural Networks 8: Linear Regression
giant_neural_network • 53K views • 1 year ago
Hey everyone! In this video we're going to look at something called linear regression. We're really just adding an input to our ...
TensorFlow
43:00
Learning Tensorflow with linear regression
Technology for Noobs • 3.5K views • 1 year ago
In this video, I will cover basics of tensorflow. Below are the topics that will be covered: 1. Basic of linear regression 2. Basics of ...
K
Episode – 2.3
Implementing simple and multivariable Linear Regression in Keras
36:25
Ep-2.3: Linear Regression in Keras || TFK-Deep Learning || Exploring Neurons
Anuj shah • 1.2K views • 1 year ago
This video explains the implementation of simple and multiple linear regression in keras. The theoretical discussion of linear ...
y
x


No thanks!
We are too busy


Recurrent Neural Network (RNN)
Long / Short Term Memory (LSTM)
Gated Recurrent Unit (GRU)
Auto Encoder (AE)
Variational AE (VAE)
Denoising AE (DAE)
Sparse AE (SAE)
Backfed Input Cell
Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Different Memory Cell
Kernel
Convolution or Pool
Markov Chain (MC)
Hopfield Network (HN)
Boltzmann Machine (BM)
Restricted BM (RBM)
Deep Belief Network (DBN)
Deep Convolutional Network (DCN)
Deconvolutional Network (DN)
Deep Convolutional Inverse Graphics Network (DCIGN)
Generative Adversarial Network (GAN)
Liquid State Machine (LSM)
Extreme Learning Machine (ELM)
Echo State Network (ESN)
Deep Residual Network (DRN)
Kohonen Network (KN)
Support Vector Machine (SVM)
Neural Turing Machine (NTM)

Linear regression

Python notebooks:
demo_LS.ipynb, demo_LS_polFit.ipynb

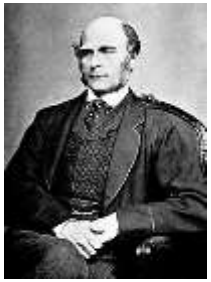
Matlab codes:
demo_LS01.m, demo_LS_polFit01.m

Linear regression

- **Least squares is everywhere**: from simple problems to large scale problems.
- It was the earliest form of regression, which was published by **Legendre** in 1805 and by **Gauss** in 1809. They both applied the method to the problem of determining the orbits of bodies around the Sun from astronomical observations.
- The term regression was only coined later by **Galton** to describe the biological phenomenon that the heights of descendants of tall ancestors tend to regress down towards a normal average.
- **Pearson** later provided the statistical context showing that the phenomenon is more general than a biological context.

Adrien-Marie Legendre

Carl Friedrich Gauss



Francis Galton

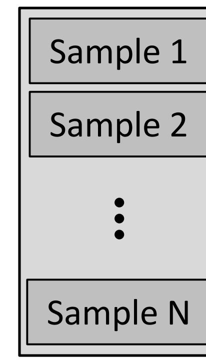
Karl Pearson

Multivariate linear regression

By describing the input data as $\boldsymbol{X} \in \mathbb{R}^{N \times D^{\mathcal{I}}}$ and the output data as $\boldsymbol{y} \in \mathbb{R}^{N}$, we want to find $\boldsymbol{a} \in \mathbb{R}^{D^{\mathcal{I}}}$ to have $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{a}$.

A solution can be found by minimizing the ℓ_2 norm

$$\begin{aligned}\hat{\boldsymbol{a}} &= \arg\min_{\boldsymbol{a}} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{a}\|^2 \\ &= \arg\min_{\boldsymbol{a}} (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{a})^\top (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{a}) \\ &= \arg\min_{\boldsymbol{a}} \boldsymbol{y}^\top \boldsymbol{y} - 2\boldsymbol{a}^\top \boldsymbol{X}^\top \boldsymbol{y} + \boldsymbol{a}^\top \boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{a}\end{aligned}$$



$\boldsymbol{X}$

By differentiating with respect to $\boldsymbol{a}$ and equating to zero

$$-2\boldsymbol{X}^\top \boldsymbol{y} + 2\boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{a} = \boldsymbol{0} \quad \Longleftrightarrow \quad \hat{\boldsymbol{a}} = \boxed{(\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top} \boldsymbol{y}$$

Moore-Penrose pseudoinverse $\boldsymbol{X}^\dagger$

Multiple multivariate linear regression

By describing the input data as $\boldsymbol{X} \in \mathbb{R}^{N \times D^{\mathcal{I}}}$ and the output data as $\boldsymbol{Y} \in \mathbb{R}^{N \times D^{\mathcal{O}}}$, we want to find $\boldsymbol{A} \in \mathbb{R}^{D^{\mathcal{I}} \times D^{\mathcal{O}}}$ to have $\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{A}$.

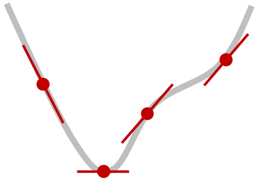
A solution can be found by minimizing the Frobenius norm

$$\begin{aligned}\hat{\boldsymbol{A}} &= \arg\min_{\boldsymbol{A}} \|\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{A}\|_{\mathrm{F}}^2 \\ &= \arg\min_{\boldsymbol{A}} \operatorname{tr}\Big((\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{A})^{\top}(\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{A})\Big) \\ &= \arg\min_{\boldsymbol{A}} \operatorname{tr}(\boldsymbol{Y}^{\top}\boldsymbol{Y} - 2\boldsymbol{A}^{\top}\boldsymbol{X}^{\top}\boldsymbol{Y} + \boldsymbol{A}^{\top}\boldsymbol{X}^{\top}\boldsymbol{X}\boldsymbol{A})\end{aligned}$$

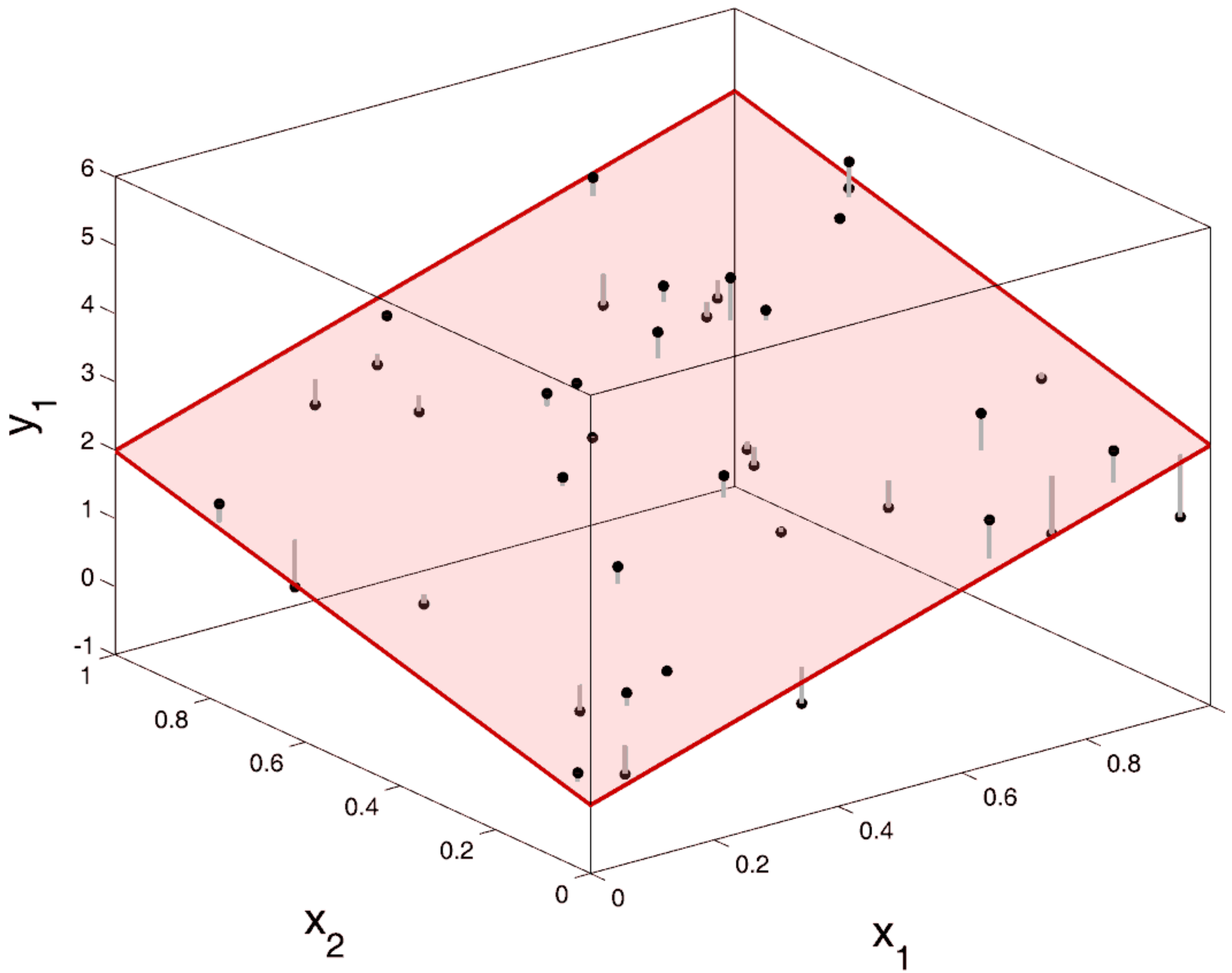
By differentiating with respect to $\boldsymbol{A}$ and equating to zero

$$-2\boldsymbol{X}^{\top}\boldsymbol{Y} + 2\boldsymbol{X}^{\top}\boldsymbol{X}\boldsymbol{A} = \boldsymbol{0} \quad \Longleftrightarrow \quad \hat{\boldsymbol{A}} = (\boldsymbol{X}^{\top}\boldsymbol{X})^{-1}\boldsymbol{X}^{\top}\boldsymbol{Y}$$

Moore-Penrose pseudoinverse $\boldsymbol{X}^{\dagger}$



Example of multivariate linear regression



$$\boldsymbol{x} = [\mathrm{x}_1, \mathrm{x}_2]$$

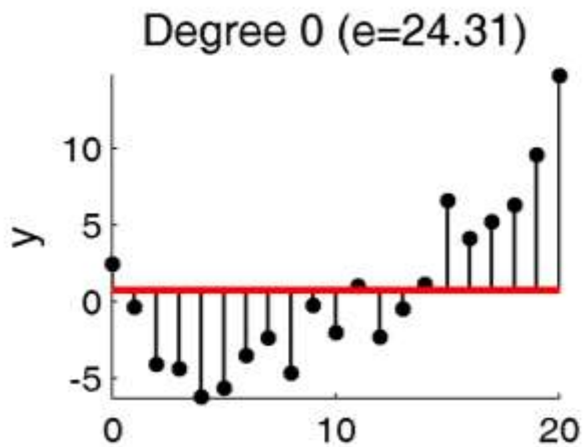
$$N = 40$$

$$D^{\mathcal{I}} = 2$$

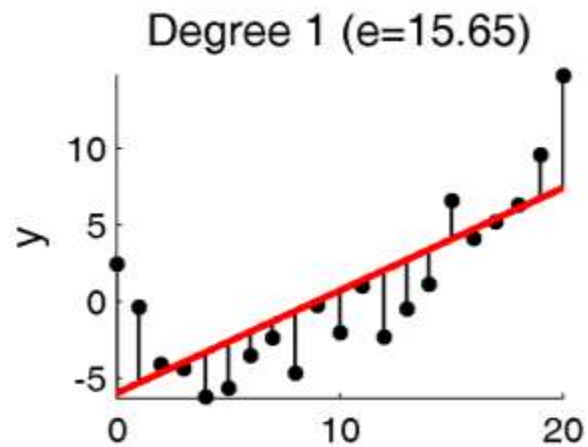
$$D^{\mathcal{O}} = 1$$

Polynomial fitting with least squares

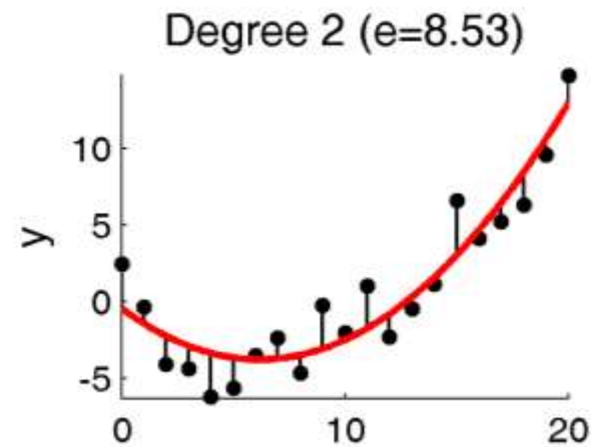
$\hat{A} = X^{\dagger} Y$



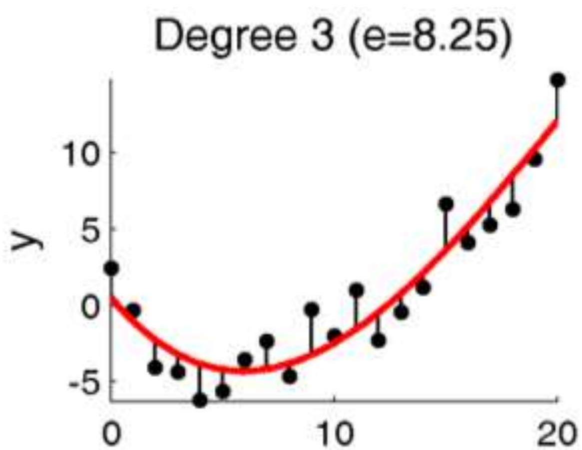
$x = 1$



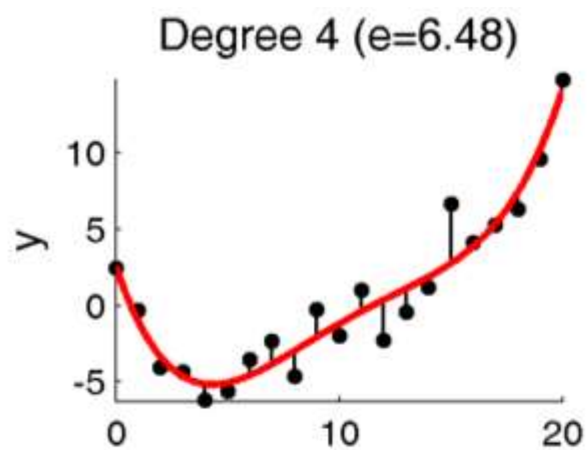
$\boldsymbol{x} = [1, \mathrm{x}]$



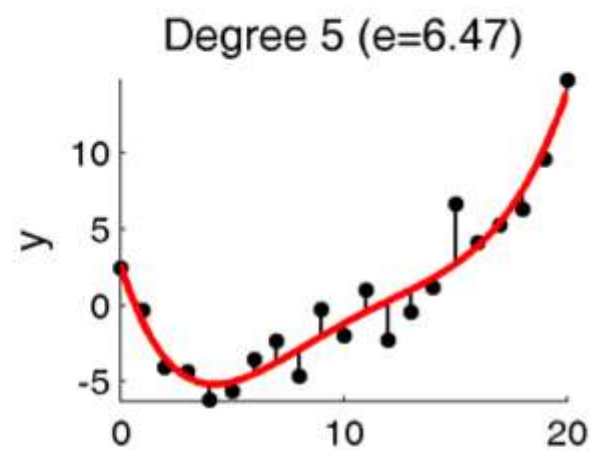
$\boldsymbol{x} = [1, \mathrm{x}, \mathrm{x}^2]$



$\boldsymbol{x} = [1, \mathrm{x}, \mathrm{x}^2, \mathrm{x}^3]$



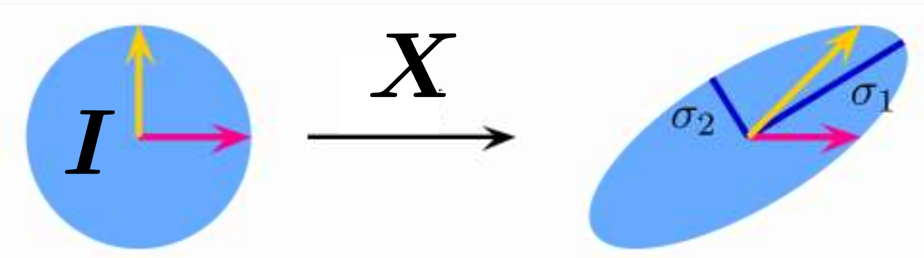
$\boldsymbol{x} = [1, \mathrm{x}, \mathrm{x}^2, \mathrm{x}^3, \mathrm{x}^4]$



$\boldsymbol{x} = [1, \mathrm{x}, \mathrm{x}^2, \mathrm{x}^3, \mathrm{x}^4, \mathrm{x}^5]$

Singular value decomposition (SVD)

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 2 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 \end{bmatrix}}_{\boldsymbol{X} \in \mathbb{R}^{N \times D^{\mathcal{I}}}} = \underbrace{\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 \end{bmatrix}}_{\boldsymbol{U} \in \mathbb{R}^{N \times N}} \underbrace{\begin{bmatrix} 4 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & \sqrt{5} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}}_{\boldsymbol{\Sigma} \in \mathbb{R}^{N \times D^{\mathcal{I}}}} \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ \sqrt{0.2} & 0 & 0 & 0 & \sqrt{0.8} \\ 0 & 0 & 0 & 1 & 0 \\ -\sqrt{0.8} & 0 & 0 & 0 & \sqrt{0.2} \end{bmatrix}}_{\boldsymbol{V}^\top \in \mathbb{R}^{D^{\mathcal{I}} \times D^{\mathcal{I}}}}$$



Matrix with non-negative diagonal entries
(singular values of X)

Unitary matrix (orthogonal)

Unitary matrix (orthogonal)

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$$

Least squares with SVD

$\boldsymbol{X} \in \mathbb{R}^{N \times D^{\mathcal{I}}}$

$$\hat{\boldsymbol{A}} = \overbrace{\boldsymbol{X}^\top (\boldsymbol{X}^\top \boldsymbol{X})^{-1}}^{\boldsymbol{X}^\dagger} \boldsymbol{Y}$$

$\boldsymbol{X}$ can be decomposed with the **singular value decomposition**

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$$

where $\boldsymbol{U}$ and $\boldsymbol{V}$ are $N \times N$ and $D^{\mathcal{I}} \times D^{\mathcal{I}}$ orthogonal matrices, and $\boldsymbol{\Sigma}$ is an $N \times D^{\mathcal{I}}$ matrix with all its elements outside of the main diagonal equal to 0. With this decomposition, a solution to the least squares problem is given by

$$\hat{\boldsymbol{A}} = \boldsymbol{V}\boldsymbol{\Sigma}^\dagger \boldsymbol{U}^\top \boldsymbol{Y}$$

where the pseudoinverse of $\boldsymbol{\Sigma}$ can be easily obtained by inverting the non-zero diagonal elements and transposing the resulting matrix.

Kernels in least squares (nullspace projection)

Python notebook:
demo_LS_polFit.ipynb

Matlab code:
demo_LS_polFit_nullspace01.m

Kernels in least squares (nullspace)

The pseudoinverse provides a single least norm solution, but we can sometimes obtain other solutions by employing a **nullspace projection operator $\boldsymbol{N}$**

$$\hat{\boldsymbol{A}} = \boldsymbol{X}^{\dagger}\boldsymbol{Y} + \overbrace{(\boldsymbol{I} - \boldsymbol{X}^{\dagger}\boldsymbol{X})}^{\boldsymbol{N}}\boldsymbol{V}$$

$\boldsymbol{V}$ can be any vector/matrix (typically, a gradient minimizing a secondary objective function).

The nullspace projection guarantees that $\|\boldsymbol{Y} - \boldsymbol{X}\hat{\boldsymbol{A}}\|_{\mathrm{F}}^2$ is still minimized.

Kernels in least squares (nullspace)

$$\hat{\boldsymbol{A}} = \boldsymbol{X}^{\dagger}\boldsymbol{Y} + \overbrace{(\boldsymbol{I} - \boldsymbol{X}^{\dagger}\boldsymbol{X})}^{\boldsymbol{N}}\boldsymbol{V}$$

An alternative way of computing the nullspace projection matrix is to exploit the singular value decomposition

$$\boldsymbol{X}^{\dagger} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\top}$$

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 2 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 \end{bmatrix}}_{\boldsymbol{X}^{\dagger} \in \mathbb{R}^{D^{\mathcal{I}} \times N}} = \underbrace{\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 \end{bmatrix}}_{\boldsymbol{U} \in \mathbb{R}^{D^{\mathcal{I}} \times D^{\mathcal{I}}}} \underbrace{\begin{bmatrix} 4 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & \sqrt{5} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}}_{\boldsymbol{\Sigma} \in \mathbb{R}^{D^{\mathcal{I}} \times N}} \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ \sqrt{0.2} & 0 & 0 & 0 & \sqrt{0.8} \\ 0 & 0 & 0 & 1 & 0 \\ -\sqrt{0.8} & 0 & 0 & 0 & \sqrt{0.2} \end{bmatrix}}_{\boldsymbol{V}^{\top} \in \mathbb{R}^{N \times N}}$$

$\tilde{U}$

to compute

$$\boldsymbol{N} = \tilde{\boldsymbol{U}}\tilde{\boldsymbol{U}}^{\top}$$

where $\tilde{\boldsymbol{U}}$ is a matrix formed by the columns of $\boldsymbol{U}$ that span for the corresponding zero rows in $\boldsymbol{\Sigma}$.

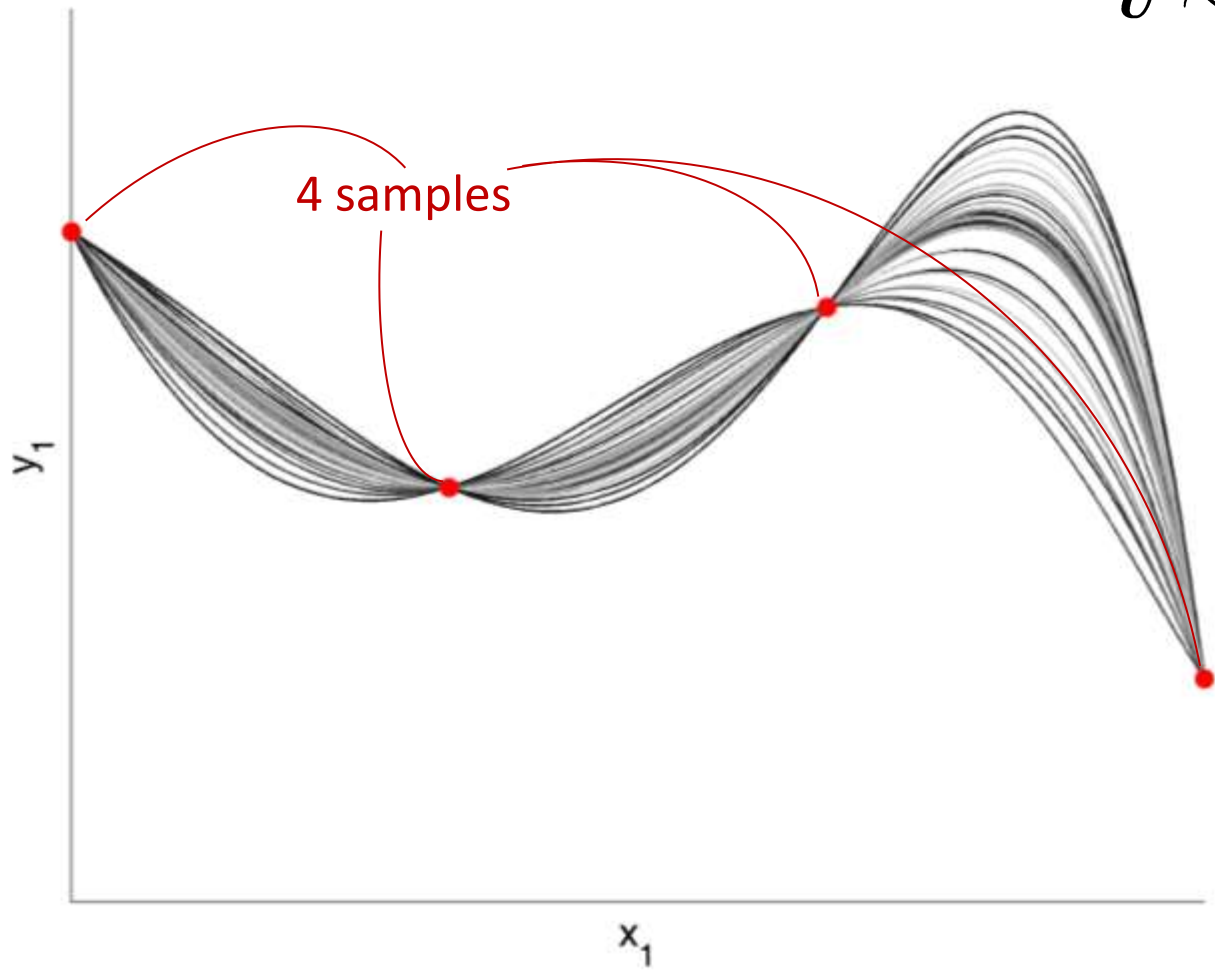
This can for example be implemented in Matlab/Octave with

```
[U,S,V] = svd(pinv(X))
sp = sum(S,2) < 1E-1
N = U(:,sp) * U(:,sp)'
```

Example with polynomial fitting

$$\hat{\boldsymbol{a}} = \boldsymbol{X}^{\dagger}\boldsymbol{y} + \boldsymbol{N}\boldsymbol{v} \quad \text{with} \quad \boldsymbol{x} = [1, \mathrm{x}, \mathrm{x}^2, \ldots, \mathrm{x}^6]$$

$$\boldsymbol{v} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$$

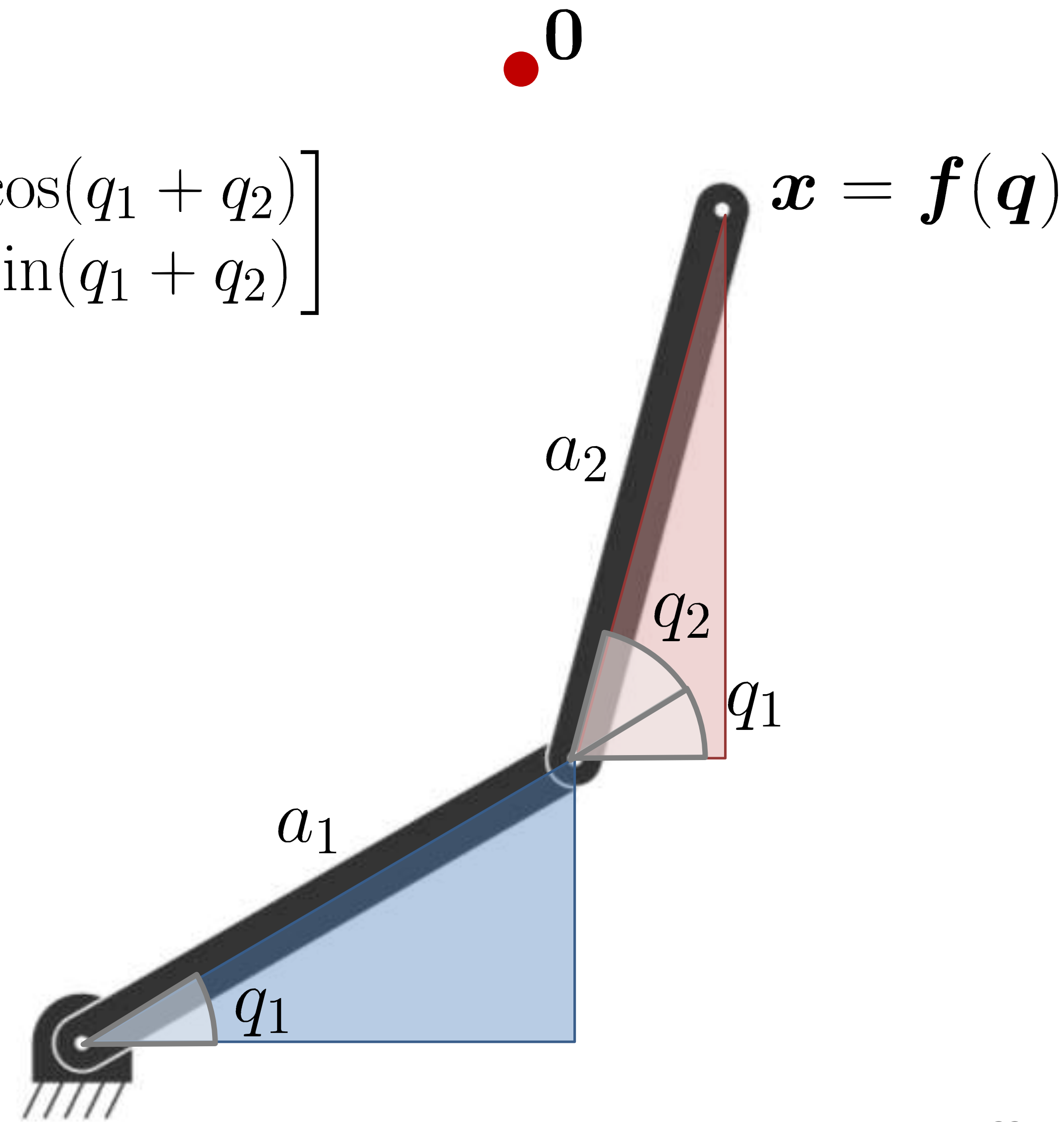


$$\boldsymbol{X} \in \mathbb{R}^{4\times 7}$$
$$\boldsymbol{y} \in \mathbb{R}^{4}$$
$$\hat{\boldsymbol{a}} \in \mathbb{R}^{7}$$

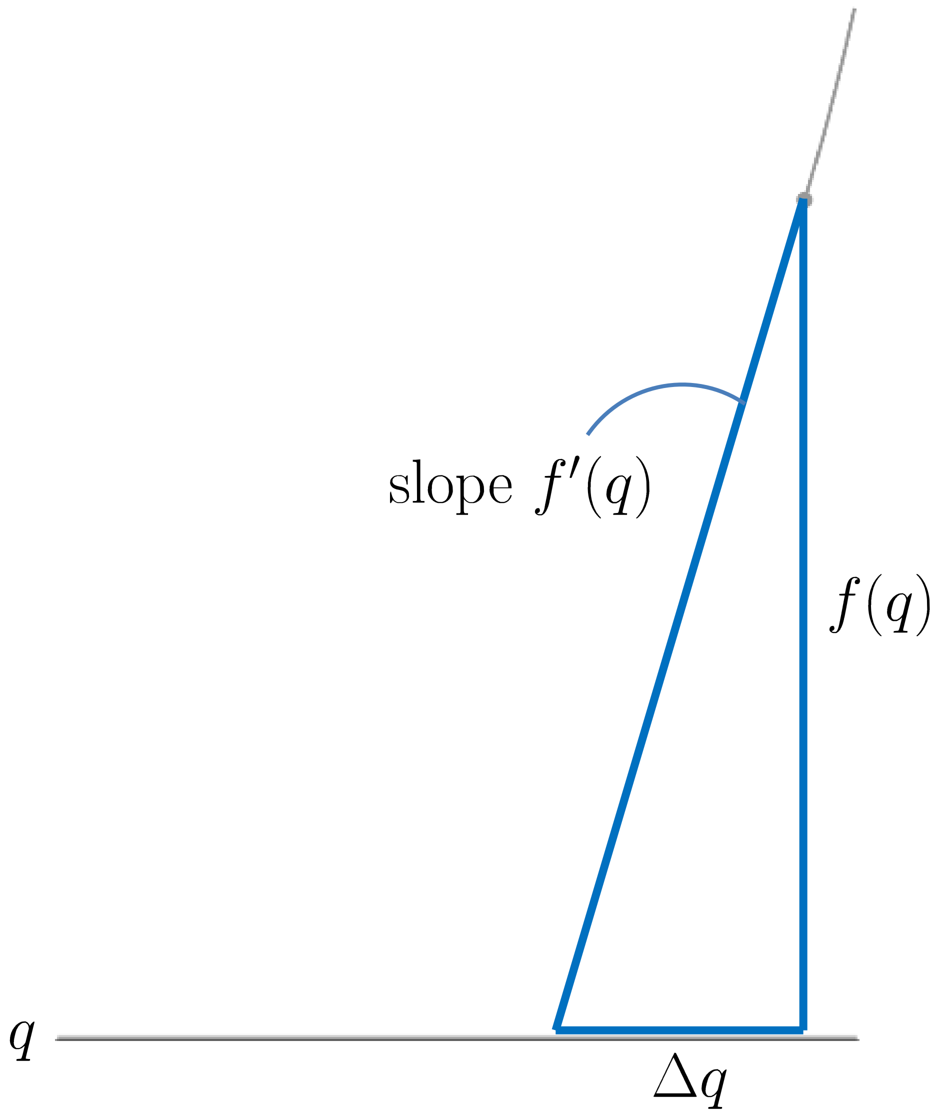
Example with robot inverse kinematics

Forward kinematics

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} a_1 \cos(q_1) + a_2 \cos(q_1 + q_2) \\ a_1 \sin(q_1) + a_2 \sin(q_1 + q_2) \end{bmatrix}$$



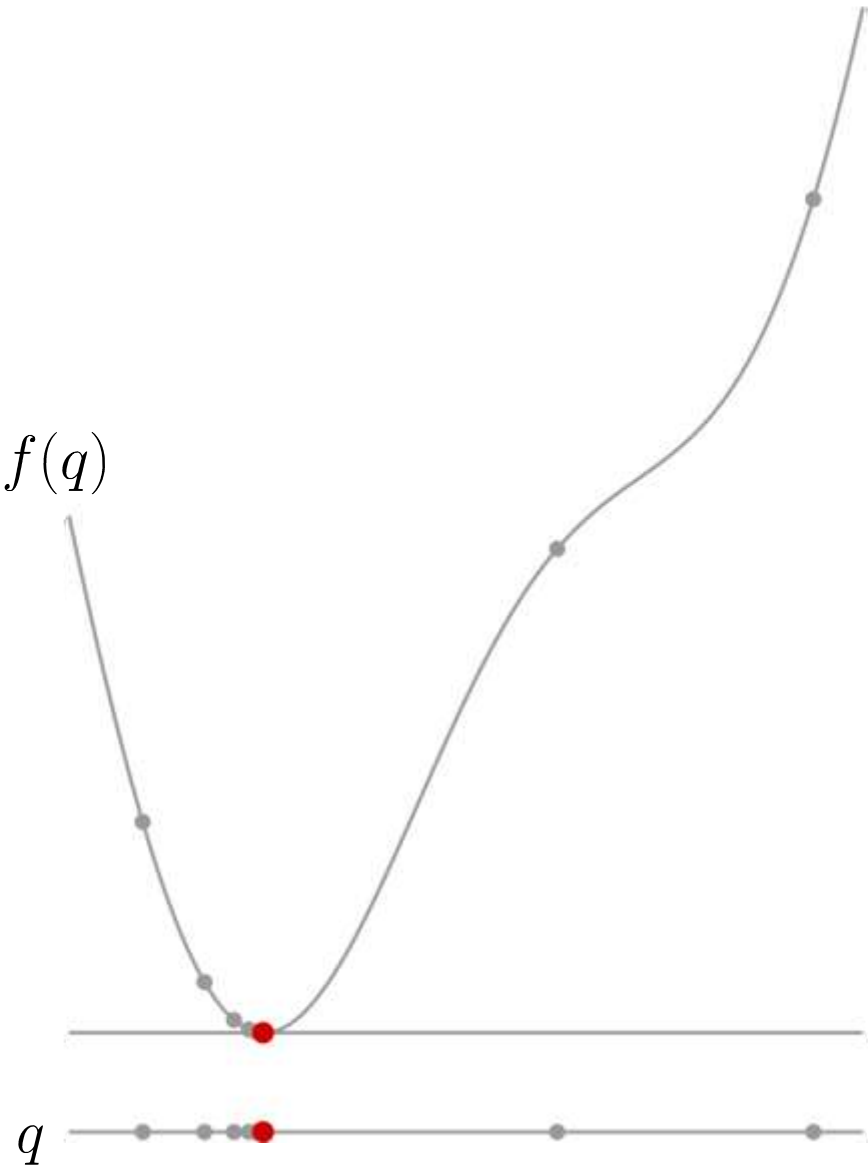
Find q to have f(q)=0



$$f'(q) = \frac{f(q)}{\Delta q}$$

$$\iff \quad \Delta q = \frac{f(q)}{f'(q)}$$

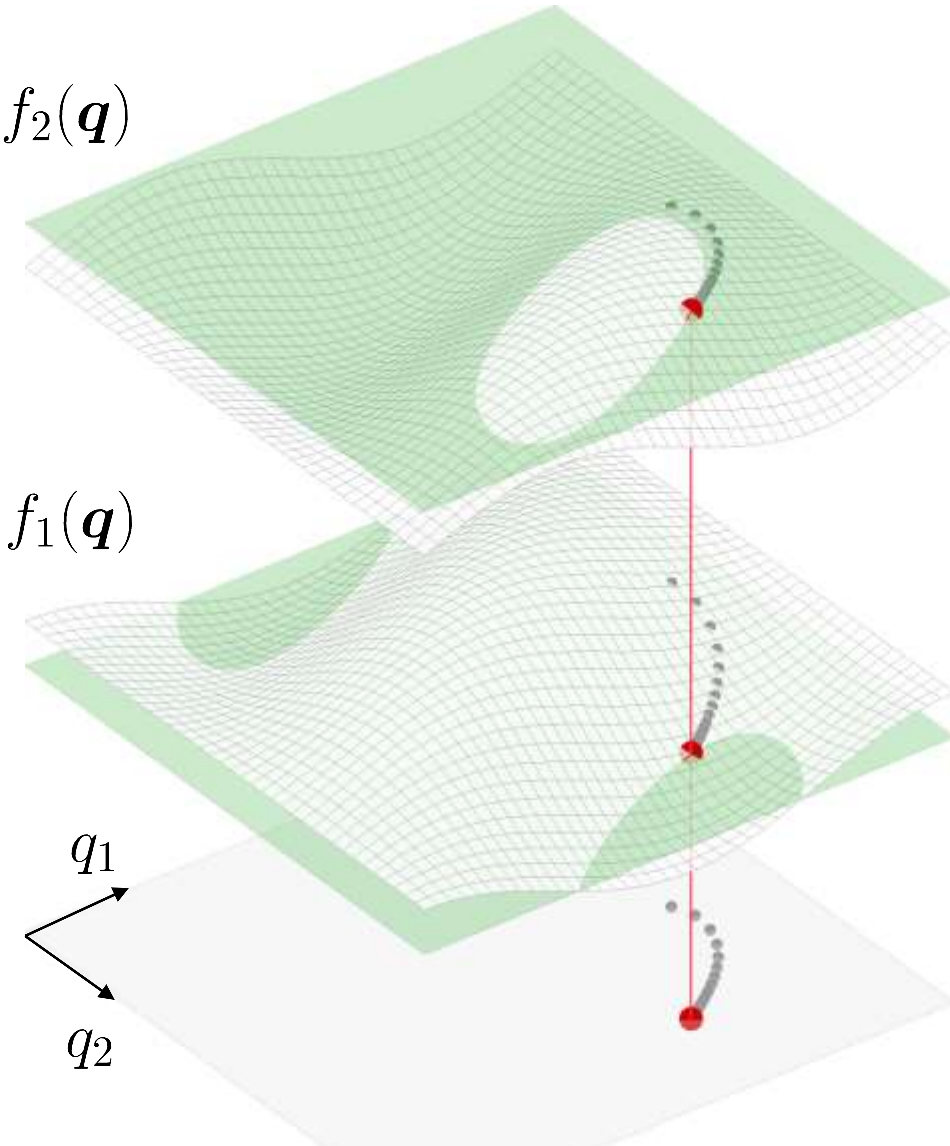
Gauss-Newton algorithm

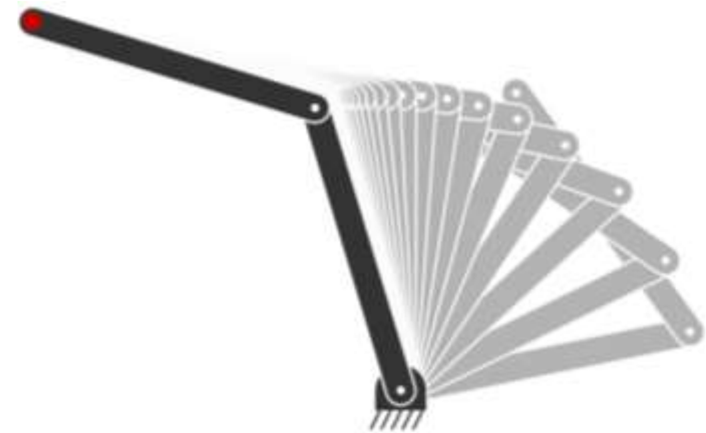


$$q \leftarrow q - \frac{f(q)}{f'(q)}$$

Example with robot inverse kinematics

Gauss-Newton algorithm



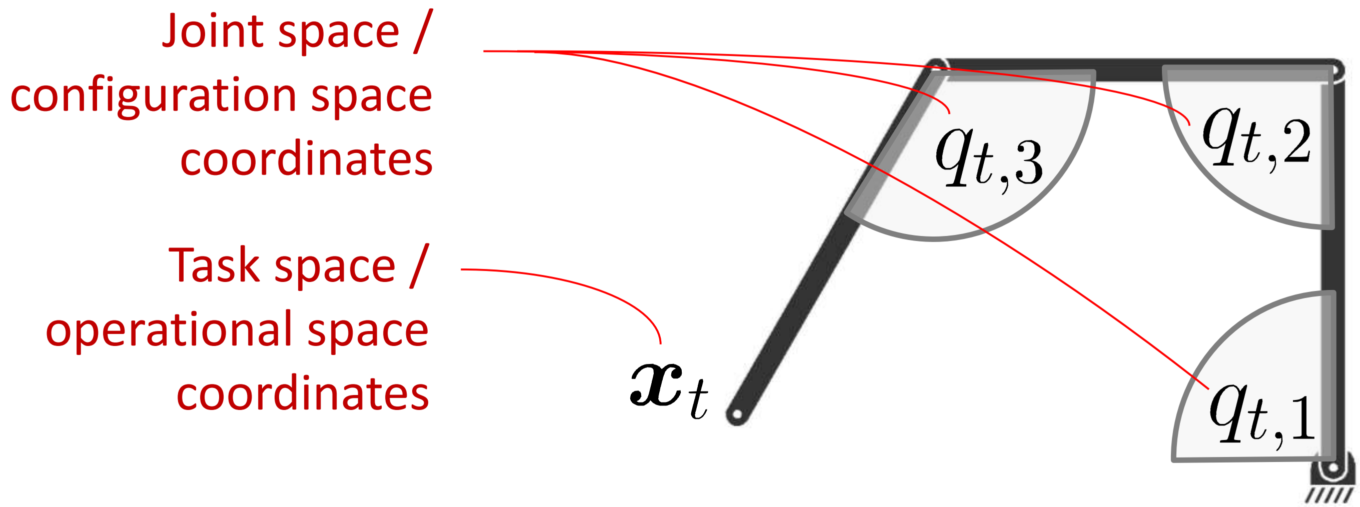


$$\boldsymbol{q} \;\leftarrow\; \boldsymbol{q} - \alpha\, \boldsymbol{J}^{\dagger}(\boldsymbol{q})\boldsymbol{f}(\boldsymbol{q})$$

$$\boldsymbol{J}(\boldsymbol{q}) = \begin{bmatrix} \frac{\partial f_1(\boldsymbol{q})}{\partial q_1} & \frac{\partial f_1(\boldsymbol{q})}{\partial q_2} \\ \frac{\partial f_2(\boldsymbol{q})}{\partial q_1} & \frac{\partial f_2(\boldsymbol{q})}{\partial q_2} \end{bmatrix}$$

$$\in \mathbb{R}^{2\times 2}$$

Example with robot inverse kinematics



Forward kinematics is computed with

$$\boldsymbol{x}_t = f(\boldsymbol{q}_t) \quad \Longleftrightarrow \quad \dot{\boldsymbol{x}}_t = \frac{\partial \boldsymbol{x}_t}{\partial t} = \frac{\partial f(\boldsymbol{q}_t)}{\partial \boldsymbol{q}_t} \frac{\partial \boldsymbol{q}_t}{\partial t} = \boldsymbol{J}(\boldsymbol{q}_t)\, \dot{\boldsymbol{q}}_t$$

where $\boldsymbol{J}(\boldsymbol{q}_t) = \frac{\partial f(\boldsymbol{q}_t)}{\partial \boldsymbol{q}_t}$ is a Jacobian matrix.

An inverse kinematics solution can be computed with

$$\hat{\dot{\boldsymbol{q}}}_t = \boldsymbol{J}^\dagger(\boldsymbol{q}_t)\, \dot{\boldsymbol{x}}_t + \boldsymbol{N}(\boldsymbol{q}_t)\, g(\boldsymbol{q}_t)$$

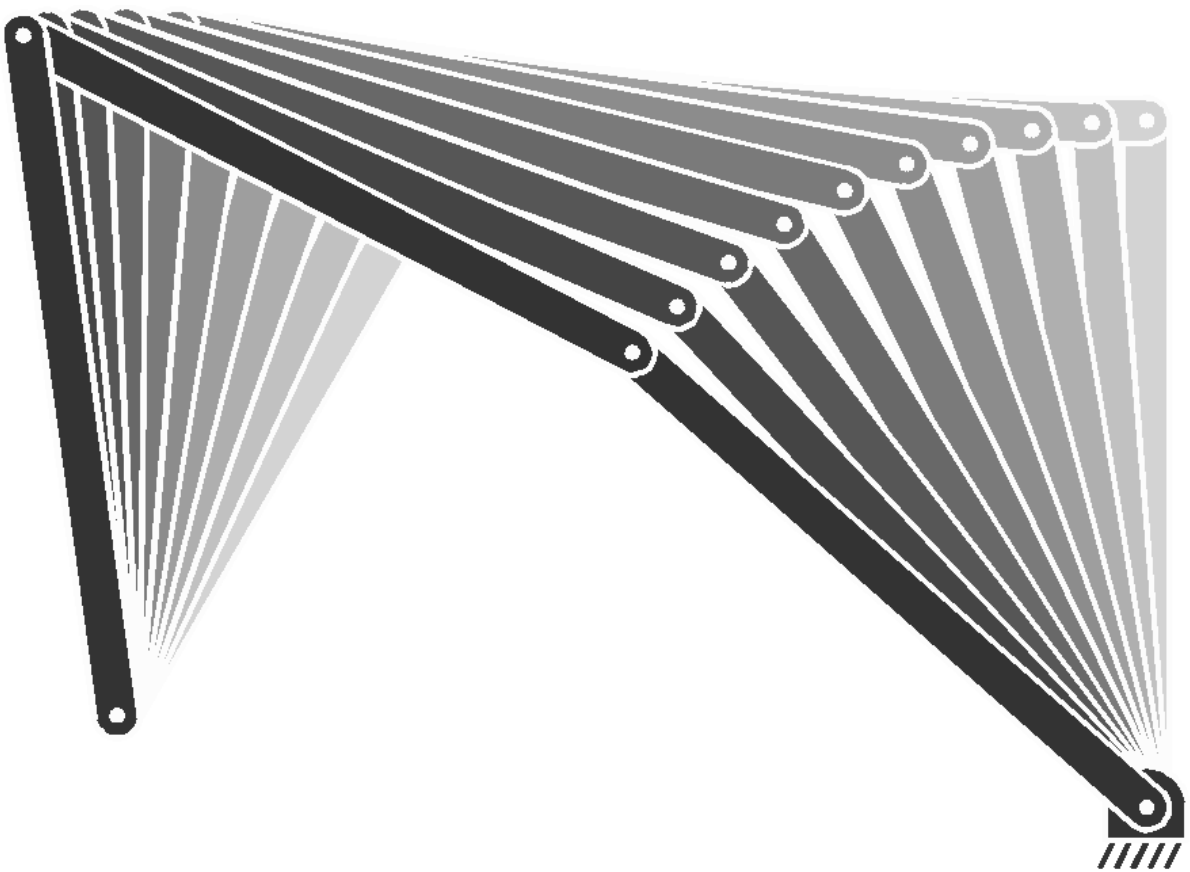
Example with robot inverse kinematics

$$\hat{\dot{\boldsymbol{q}}}_t = \boldsymbol{J}^{\dagger}(\boldsymbol{q}_t)\ \dot{\boldsymbol{x}}_t \quad + \boldsymbol{N}(\boldsymbol{q}_t)\ g(\boldsymbol{q}_t)$$

→ Primary constraint: keeping the tip of the robot still

$$= \boldsymbol{J}^{\dagger}(\boldsymbol{q}_t) \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad + \boldsymbol{N}(\boldsymbol{q}_t) \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

→ Secondary constraint: trying to move the first joint

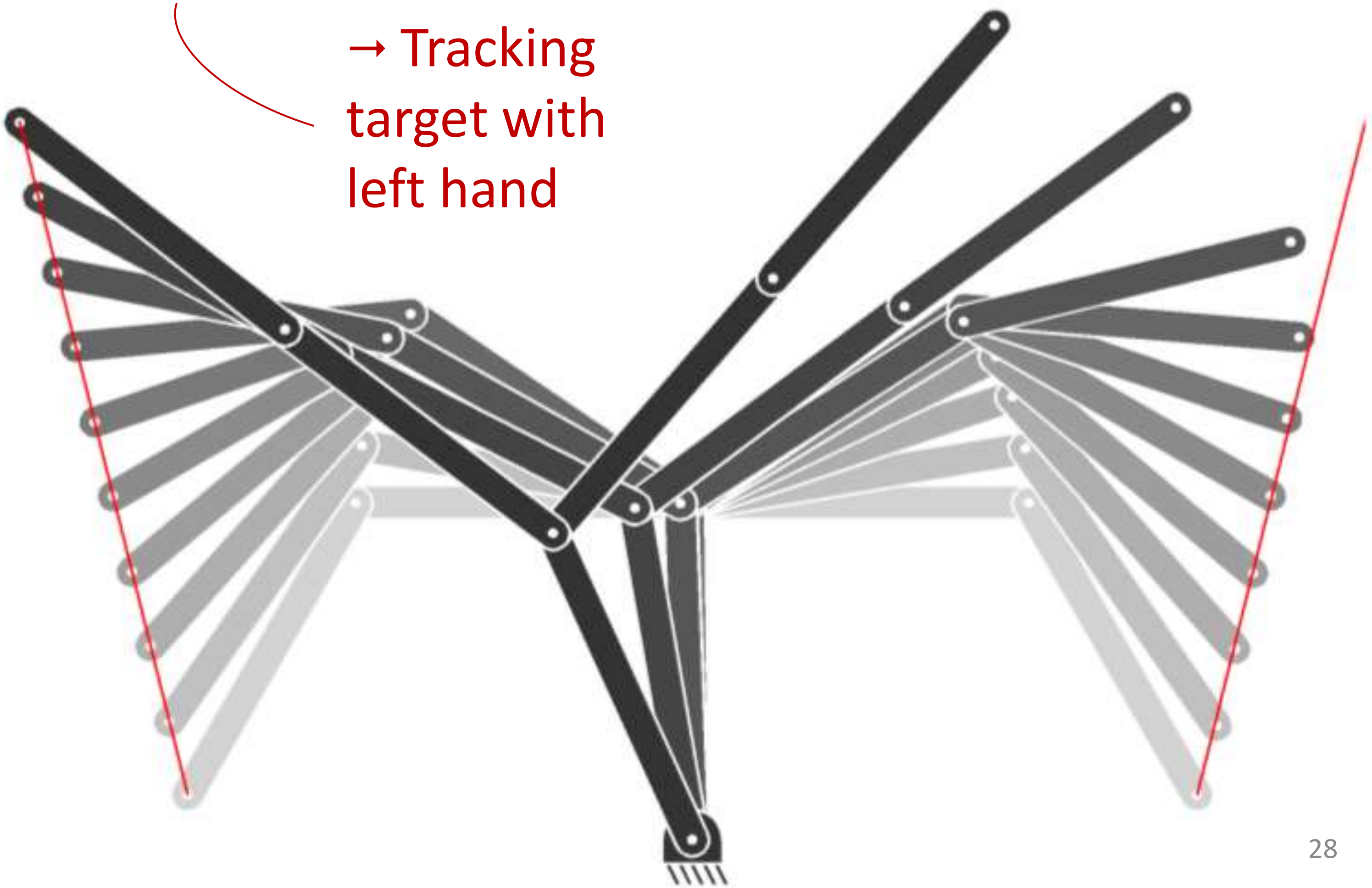


Example with robot inverse kinematics

$$\hat{\dot{q}}_t = J^{\mathcal{L}\dagger}\, \dot{x}_t^{\mathcal{L}} + N^{\mathcal{L}}\, J^{\mathcal{R}\dagger}\, \dot{x}_t^{\mathcal{R}}$$
$$= J^{\mathcal{L}\dagger}\, (\hat{x}_t^{\mathcal{L}} - x_t^{\mathcal{L}}) + N^{\mathcal{L}}\, J^{\mathcal{R}\dagger}\, (\hat{x}_t^{\mathcal{R}} - x_t^{\mathcal{R}})$$

→ Tracking target with right hand, if possible

→ Tracking target with left hand



Ridge regression (Tikhonov regularization, penalized least squares)

Python notebook:
demo_LS_polFit.ipynb

Matlab example:
demo_LS_polFit02.m

Ridge regression (Tikhonov regularization)

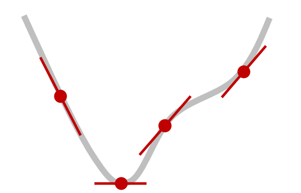
The least squares objective can be modified to give preference to a particular solution with

$$\hat{\boldsymbol{A}} = \arg\min_{\boldsymbol{A}} \|\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{A}\|_{\mathrm{F}}^2 + \|\boldsymbol{\Gamma}\boldsymbol{A}\|_{\mathrm{F}}^2$$

$$= \arg\min_{\boldsymbol{A}} \operatorname{tr}\Big((\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{A})^\top(\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{A})\Big) + \operatorname{tr}\Big((\boldsymbol{\Gamma}\boldsymbol{A})^\top\boldsymbol{\Gamma}\boldsymbol{A}\Big)$$

By differentiating with respect to $\boldsymbol{A}$ and equating to zero, we can see that

$$-2\boldsymbol{X}^\top\boldsymbol{Y} + 2\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{A} + 2\boldsymbol{\Gamma}^\top\boldsymbol{\Gamma}\boldsymbol{A} = \boldsymbol{0}$$



yielding

$$\hat{\boldsymbol{A}} = (\boldsymbol{X}^\top\boldsymbol{X} + \boldsymbol{\Gamma}^\top\boldsymbol{\Gamma})^{-1}\boldsymbol{X}^\top\boldsymbol{Y}$$

If $\boldsymbol{\Gamma} = \lambda\boldsymbol{I}$ with $\lambda \ll 1$ (i.e., giving preference to solutions with smaller norms), the process is known as **$\boldsymbol{\ell_2}$ regularization**.

Ridge regression (Tikhonov regularization)

Ridge regression can alternatively be computed with augmented matrices

$$\tilde{\boldsymbol{X}} = \begin{bmatrix} \boldsymbol{X} \\ \boldsymbol{\Gamma} \end{bmatrix} \qquad \tilde{\boldsymbol{Y}} = \begin{bmatrix} \boldsymbol{Y} \\ \boldsymbol{0} \end{bmatrix}$$

with $\boldsymbol{0} \in \mathbb{R}^{D^{\mathcal{I}} \times D^{\mathcal{O}}}$ and $\boldsymbol{\Gamma} \in \mathbb{R}^{D^{\mathcal{I}} \times D^{\mathcal{I}}}$, yielding

$$\begin{aligned}
\hat{\boldsymbol{A}} &= \big(\tilde{\boldsymbol{X}}^{\top} \tilde{\boldsymbol{X}}\big)^{-1} \tilde{\boldsymbol{X}}^{\top} \tilde{\boldsymbol{Y}} \\
&= \left(\begin{bmatrix} \boldsymbol{X} \\ \boldsymbol{\Gamma} \end{bmatrix}^{\top} \begin{bmatrix} \boldsymbol{X} \\ \boldsymbol{\Gamma} \end{bmatrix} \right)^{-1} \begin{bmatrix} \boldsymbol{X} \\ \boldsymbol{\Gamma} \end{bmatrix}^{\top} \begin{bmatrix} \boldsymbol{Y} \\ \boldsymbol{0} \end{bmatrix} \\
&= \big(\boldsymbol{X}^{\top} \boldsymbol{X} + \boldsymbol{\Gamma}^{\top} \boldsymbol{\Gamma}\big)^{-1} \boldsymbol{X}^{\top} \boldsymbol{Y}
\end{aligned}$$

$\boldsymbol{X} \in \mathbb{R}^{N \times D^{\mathcal{I}}}$
$\boldsymbol{Y} \in \mathbb{R}^{N \times D^{\mathcal{O}}}$
$\boldsymbol{A} \in \mathbb{R}^{D^{\mathcal{I}} \times D^{\mathcal{O}}}$

Ridge regression (Tikhonov regularization)

Ridge regression also has links with SVD. For the singular value decomposition

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$$

with σ_i the singular values in the diagonal of $\boldsymbol{\Sigma}$, a solution to the ridge regression problem is given by

$$\hat{\boldsymbol{A}} = \boldsymbol{V}\tilde{\boldsymbol{\Sigma}}\boldsymbol{U}^\top \boldsymbol{Y}$$

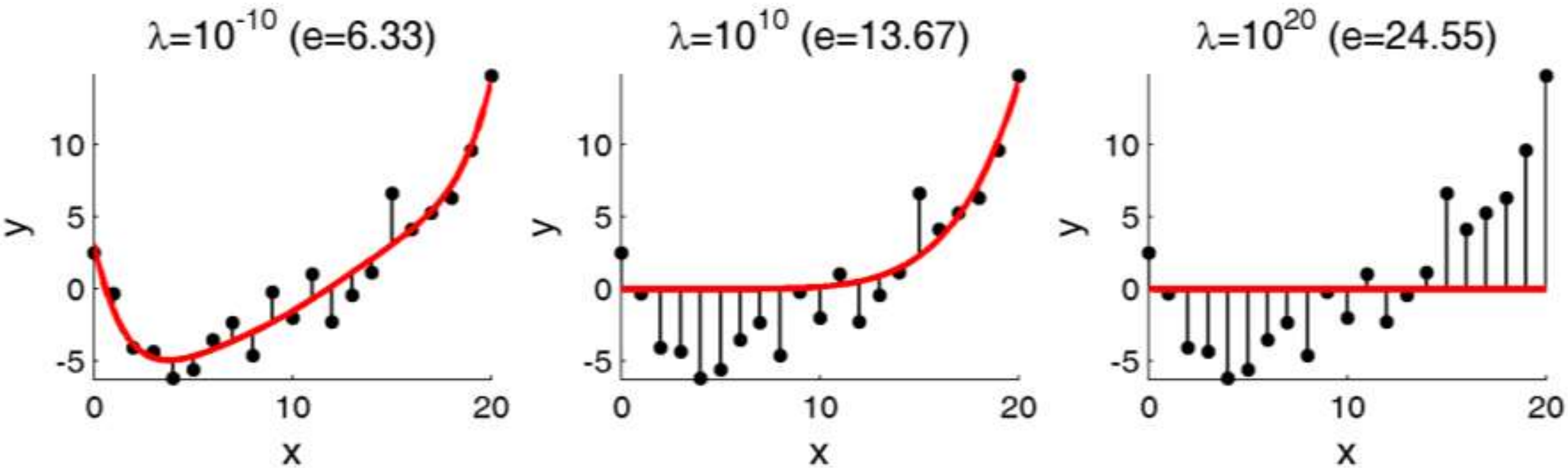
where $\tilde{\boldsymbol{\Sigma}}$ has diagonal values

$$\tilde{\sigma}_i = \frac{\sigma_i}{\sigma_i^2 + \lambda^2}$$

and has zeros elsewhere.

Ridge regression (Tikhonov regularization)

$D^{\mathcal{I}} = 7$ (polynomial of degree 7)



Weighted least squares (Generalized least squares)

Python notebook:
demo_LS_weighted.ipynb

Matlab example:
demo_LS_weighted01.m

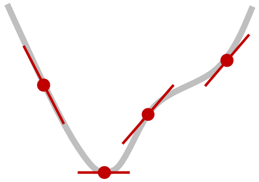
Weighted least squares

By describing the input data as $\boldsymbol{X} \in \mathbb{R}^{N \times D^{\mathcal{I}}}$ and the output data as $\boldsymbol{Y} \in \mathbb{R}^{N \times D^{\mathcal{O}}}$, with a weight matrix $\boldsymbol{W} \in \mathbb{R}^{N \times N}$, we want to minimize

$$\begin{aligned}
\hat{\boldsymbol{A}} &= \arg\min_{\boldsymbol{A}} \|\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{A}\|^2_{\mathrm{F},\boldsymbol{W}} \\
&= \arg\min_{\boldsymbol{A}} \mathrm{tr}\Big((\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{A})^\top \boldsymbol{W} (\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{A})\Big) \\
&= \arg\min_{\boldsymbol{A}} \mathrm{tr}(\boldsymbol{Y}^\top \boldsymbol{W}\boldsymbol{Y} - 2\boldsymbol{A}^\top \boldsymbol{X}^\top \boldsymbol{W}\boldsymbol{Y} + \boldsymbol{A}^\top \boldsymbol{X}^\top \boldsymbol{W}\boldsymbol{X}\boldsymbol{A})
\end{aligned}$$

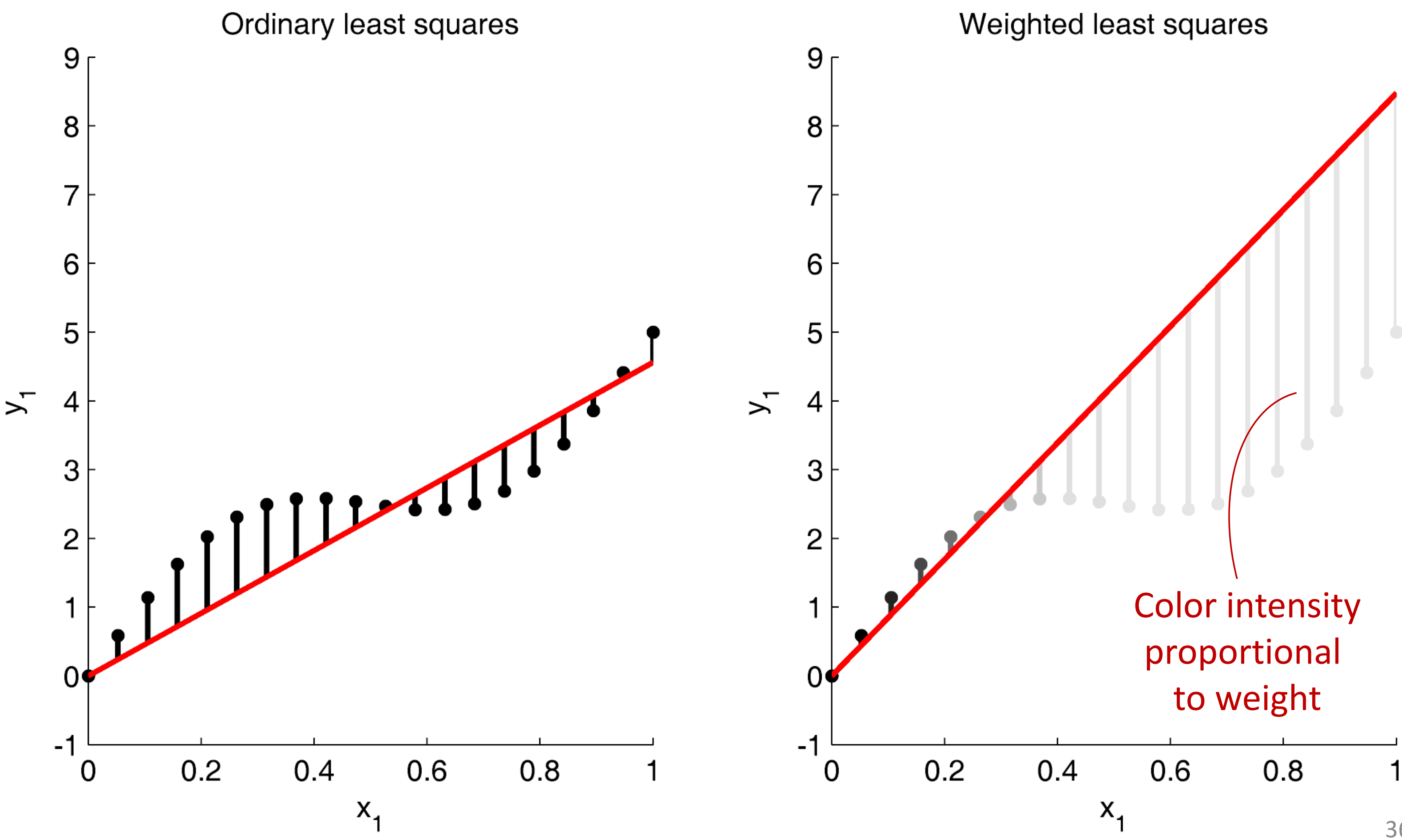
By differentiating with respect to $\boldsymbol{A}$ and equating to zero

$$-2\boldsymbol{X}^\top \boldsymbol{W}\boldsymbol{Y} + 2\boldsymbol{X}^\top \boldsymbol{W}\boldsymbol{X}\boldsymbol{A} = \boldsymbol{0} \iff \hat{\boldsymbol{A}} = \underbrace{(\boldsymbol{X}^\top \boldsymbol{W}\boldsymbol{X})^{-1}\boldsymbol{X}^\top \boldsymbol{W}}_{\boldsymbol{X}^\dagger_{\boldsymbol{W}}}\boldsymbol{Y}$$



Weighted least squares

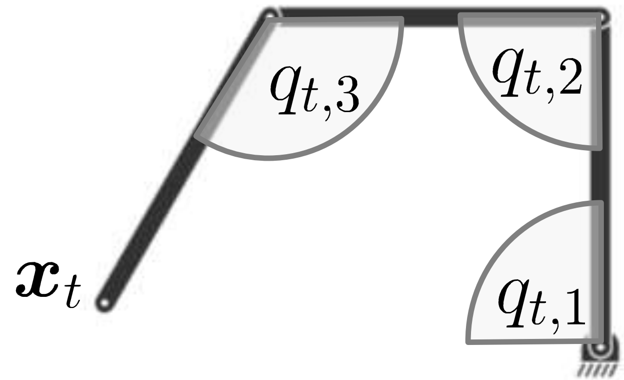
$$\hat{A} = (\boldsymbol{X}^\top \boldsymbol{W} \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{W} \boldsymbol{Y}$$

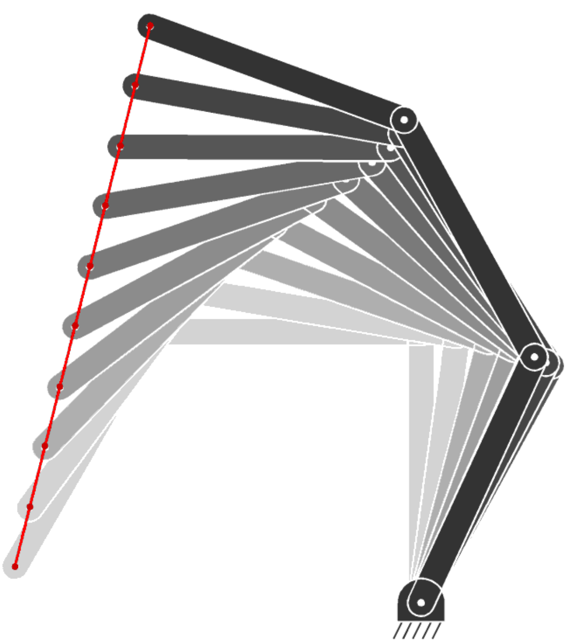


Weighted least squares - Example I

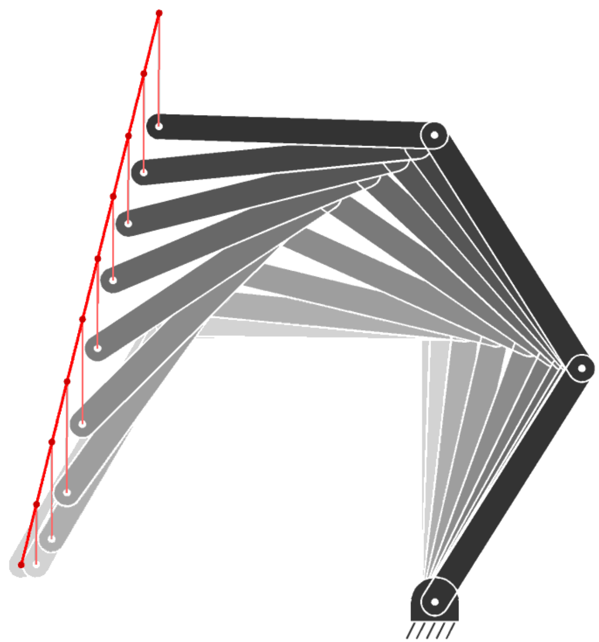
$$\hat{\boldsymbol{A}} = (\boldsymbol{X}^\top \boldsymbol{W} \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{W} \boldsymbol{Y}$$

$$\hat{\dot{\boldsymbol{q}}}_t = (\boldsymbol{J}^\top \boldsymbol{W}^x \boldsymbol{J})^{-1} \boldsymbol{J}^\top \boldsymbol{W}^x \, \dot{\boldsymbol{x}}_t$$

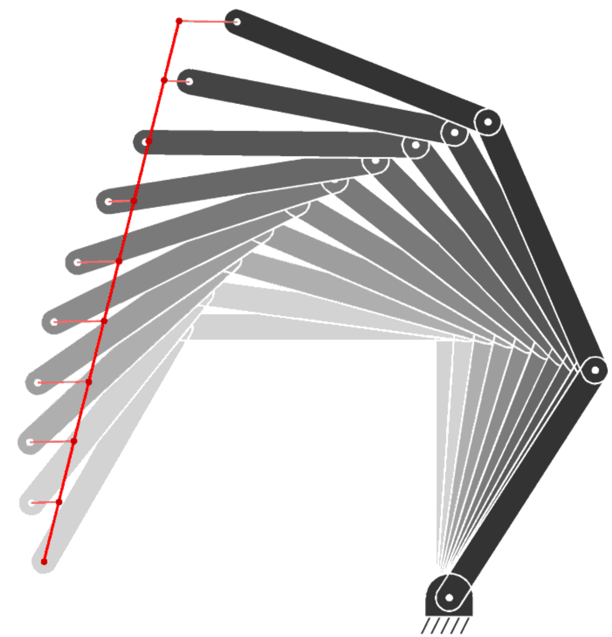




$$\boldsymbol{W}^x = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$



$$\boldsymbol{W}^x = \begin{bmatrix} 1 & 0 \\ 0 & .01 \end{bmatrix}$$

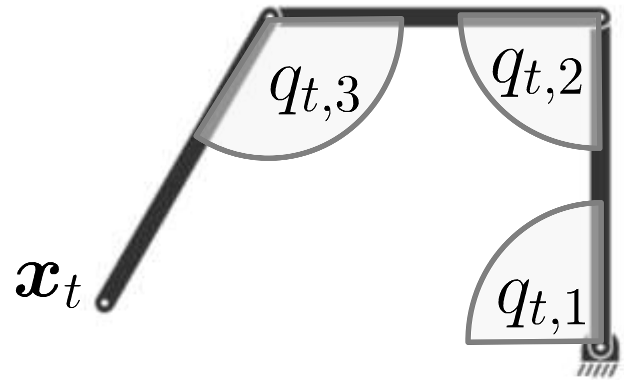


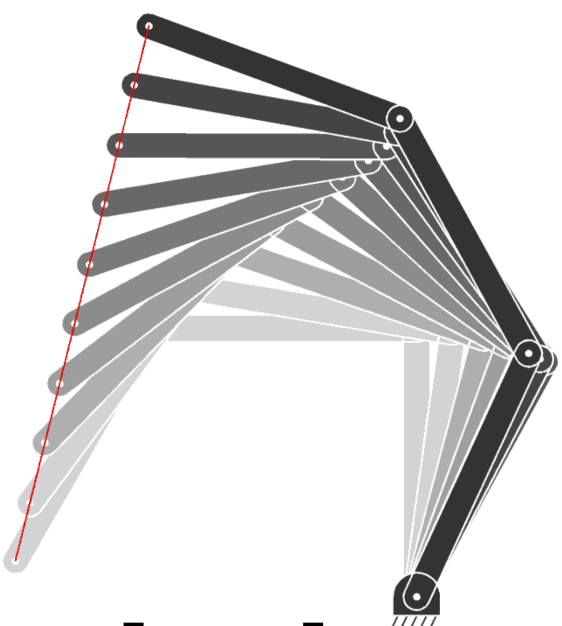
$$\boldsymbol{W}^x = \begin{bmatrix} .01 & 0 \\ 0 & 1 \end{bmatrix}$$

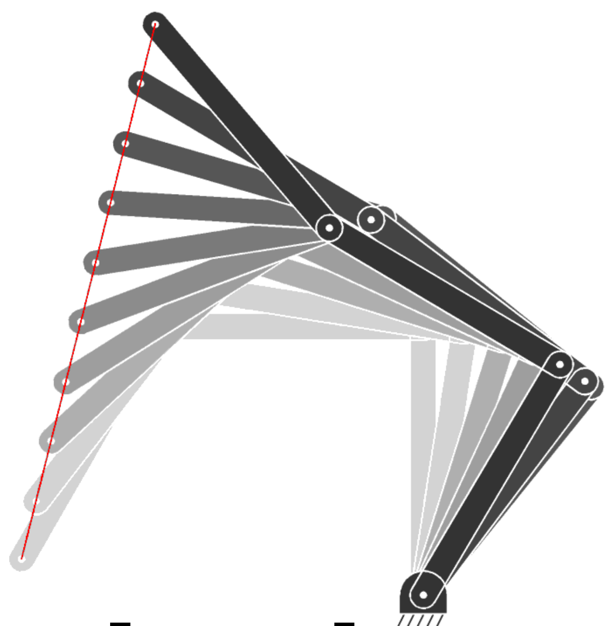
Weighted least squares - Example II

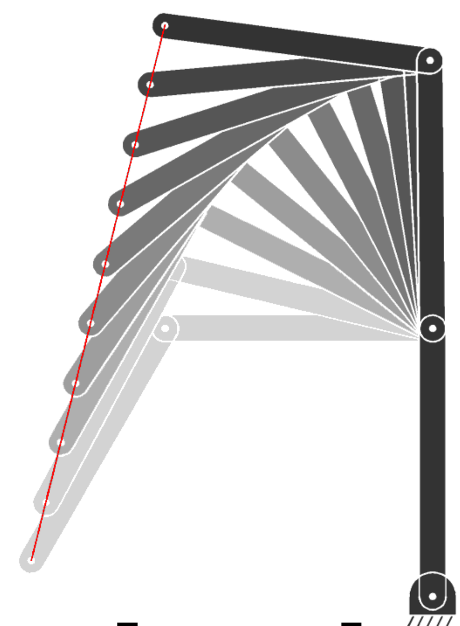
$$\hat{\boldsymbol{A}} = \boldsymbol{W}\boldsymbol{X}^{\top}(\boldsymbol{X}\boldsymbol{W}\boldsymbol{X}^{\top})^{-1}\boldsymbol{Y}$$

$$\hat{\dot{\boldsymbol{q}}}_t = \boldsymbol{W}^{\mathcal{Q}}\boldsymbol{J}^{\top}(\boldsymbol{J}\boldsymbol{W}^{\mathcal{Q}}\boldsymbol{J}^{\top})^{-1}\dot{\boldsymbol{x}}_t$$









$$\boldsymbol{W}^{\mathcal{Q}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \boldsymbol{W}^{\mathcal{Q}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & .01 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \boldsymbol{W}^{\mathcal{Q}} = \begin{bmatrix} .01 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Iteratively reweighted least squares (IRLS)

Python notebook:
demo_LS_weighted.ipynb

Matlab code:
demo_LS_IRLS01.m

Iteratively reweighted least squares (IRLS)

- **Iteratively Reweighted Least Squares** generalizes least squares by raising the error to a power that is less than 2: → can no longer be called "least squares"

- The strategy is that an error $|\mathbf{e}|^p$ can be rewritten as $|\mathbf{e}|^p = |\mathbf{e}|^{p-2}\, \mathbf{e}^2$.

- $|\mathbf{e}|^{p-2}$ can be interpreted as a weight, which is used to minimize $\mathbf{e}^2$ with **weighted least squares**.

- p=1 corresponds to **least absolute deviation regression**.

Iteratively reweighted least squares (IRLS)

$|e|^p = |e|^{p-2} e^2$

For an ℓ_p norm objective defined by

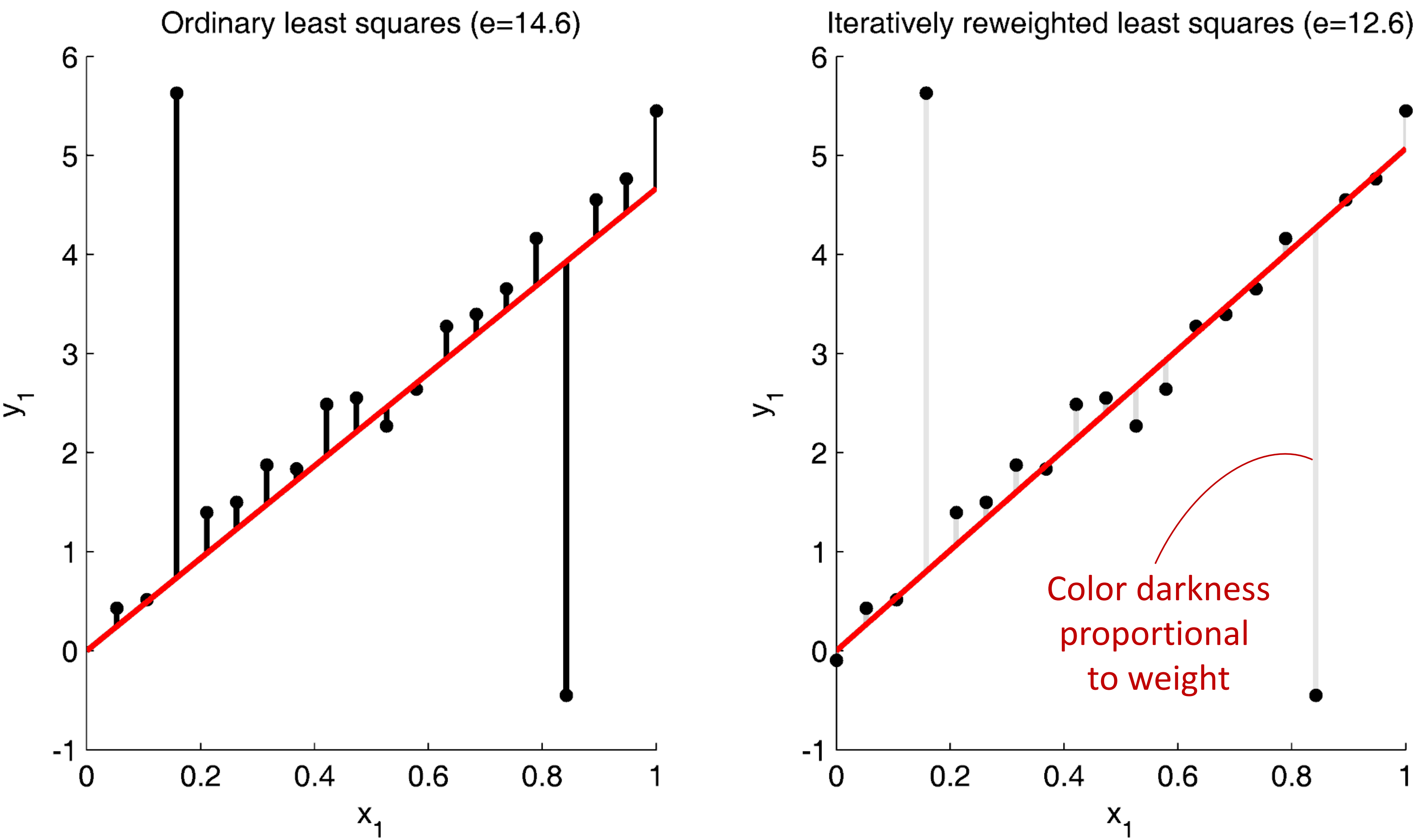
$$\hat{\boldsymbol{A}} = \arg\min_{\boldsymbol{A}} \|\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{A}\|_{\mathrm{F},p}^2$$

$\hat{\boldsymbol{A}}$ is estimated by starting from $\boldsymbol{W} = \boldsymbol{I}$ and iteratively computing

$$\hat{\boldsymbol{A}} \leftarrow \left(\boldsymbol{X}^\top \boldsymbol{W} \boldsymbol{X}\right)^{-1} \boldsymbol{X}^\top \boldsymbol{W} \boldsymbol{Y}$$

$$\boldsymbol{W}_{t,t} \leftarrow |\boldsymbol{Y}_t - \boldsymbol{X}_t \boldsymbol{A}|^{p-2} \quad \forall t \in \{1, \ldots, T\}$$

IRLS as regression robust to outliers



Recursive least squares

Python notebook:
demo_LS_recursive.ipynb

Matlab code:
demo_LS_recursive01.m

Recursive least squares

Sherman-Morrison-Woodbury relation:

$$\left(\boldsymbol{B}+\boldsymbol{U}\boldsymbol{V}\right)^{-1}=\boldsymbol{B}^{-1}-\overbrace{\boldsymbol{B}^{-1}\boldsymbol{U}\left(\boldsymbol{I}+\boldsymbol{V}\boldsymbol{B}^{-1}\boldsymbol{U}\right)^{-1}\boldsymbol{V}\boldsymbol{B}^{-1}}^{\boldsymbol{E}}$$

with $\boldsymbol{U}\in\mathbb{R}^{n\times m}$ and $\boldsymbol{V}\in\mathbb{R}^{m\times n}$.

When $m \ll n$, the correction term $\boldsymbol{E}$ can be computed more efficiently than inverting $\boldsymbol{B}+\boldsymbol{U}\boldsymbol{V}$.

By defining $\boldsymbol{B}=\boldsymbol{X}^{\top}\boldsymbol{X}$, the above relation can be exploited to update a least squares solution when new datapoints are available.

Recursive least squares

$$(\boldsymbol{B}+\boldsymbol{U}\boldsymbol{V})^{-1} = \boldsymbol{B}^{-1} - \overbrace{\boldsymbol{B}^{-1}\boldsymbol{U}\left(\boldsymbol{I}+\boldsymbol{V}\boldsymbol{B}^{-1}\boldsymbol{U}\right)^{-1}\boldsymbol{V}\boldsymbol{B}^{-1}}^{\boldsymbol{E}}$$

If $\boldsymbol{X}_{\text{new}} = [\boldsymbol{X}^\top, \boldsymbol{V}^\top]^\top$ and $\boldsymbol{Y}_{\text{new}} = [\boldsymbol{Y}^\top, \boldsymbol{C}^\top]^\top$, we then have

$$\begin{aligned}\boldsymbol{B}_{\text{new}} &= \boldsymbol{X}_{\text{new}}^\top \boldsymbol{X}_{\text{new}}\\ &= \boldsymbol{X}^\top\boldsymbol{X} + \boldsymbol{V}^\top\boldsymbol{V}\\ &= \boldsymbol{B} + \boldsymbol{V}^\top\boldsymbol{V}\end{aligned}$$

whose inverse can be computed with

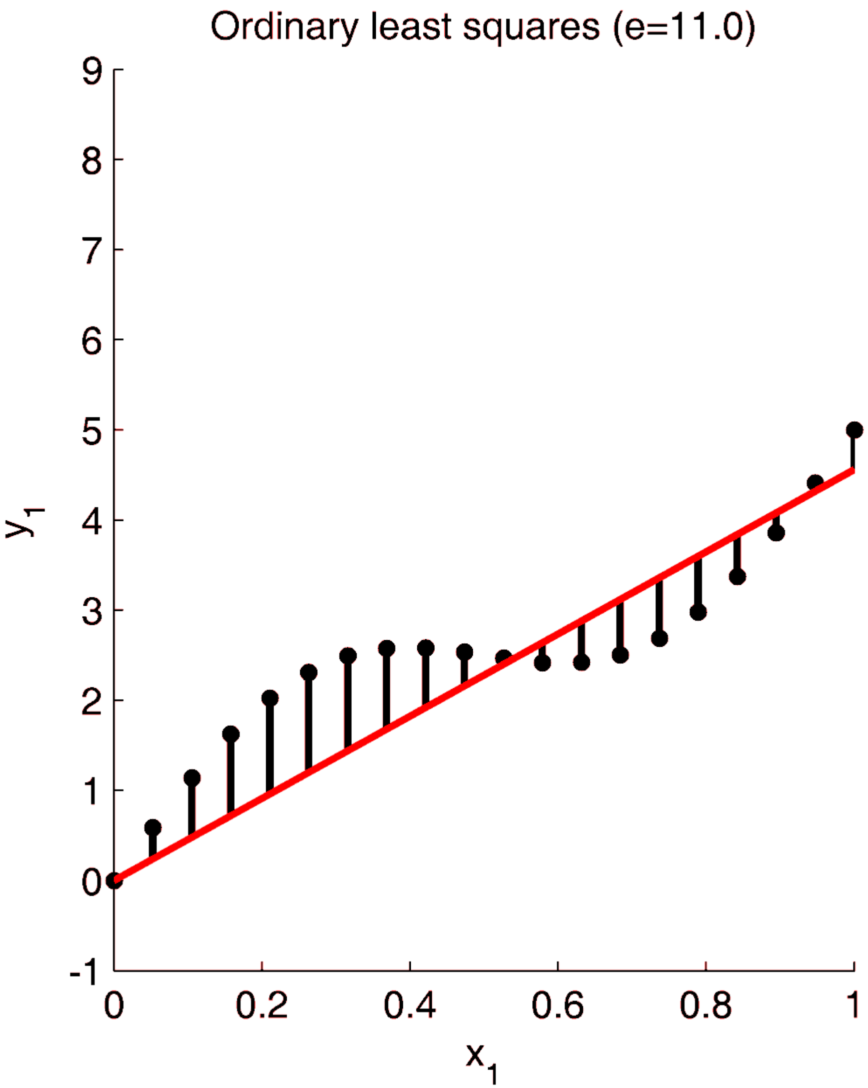
$$\boldsymbol{B}_{\text{new}}^{-1} = \boldsymbol{B}^{-1} - \boldsymbol{B}^{-1}\boldsymbol{V}^\top\left(\boldsymbol{I}+\boldsymbol{V}\boldsymbol{B}^{-1}\boldsymbol{V}^\top\right)^{-1}\boldsymbol{V}\boldsymbol{B}^{-1}$$

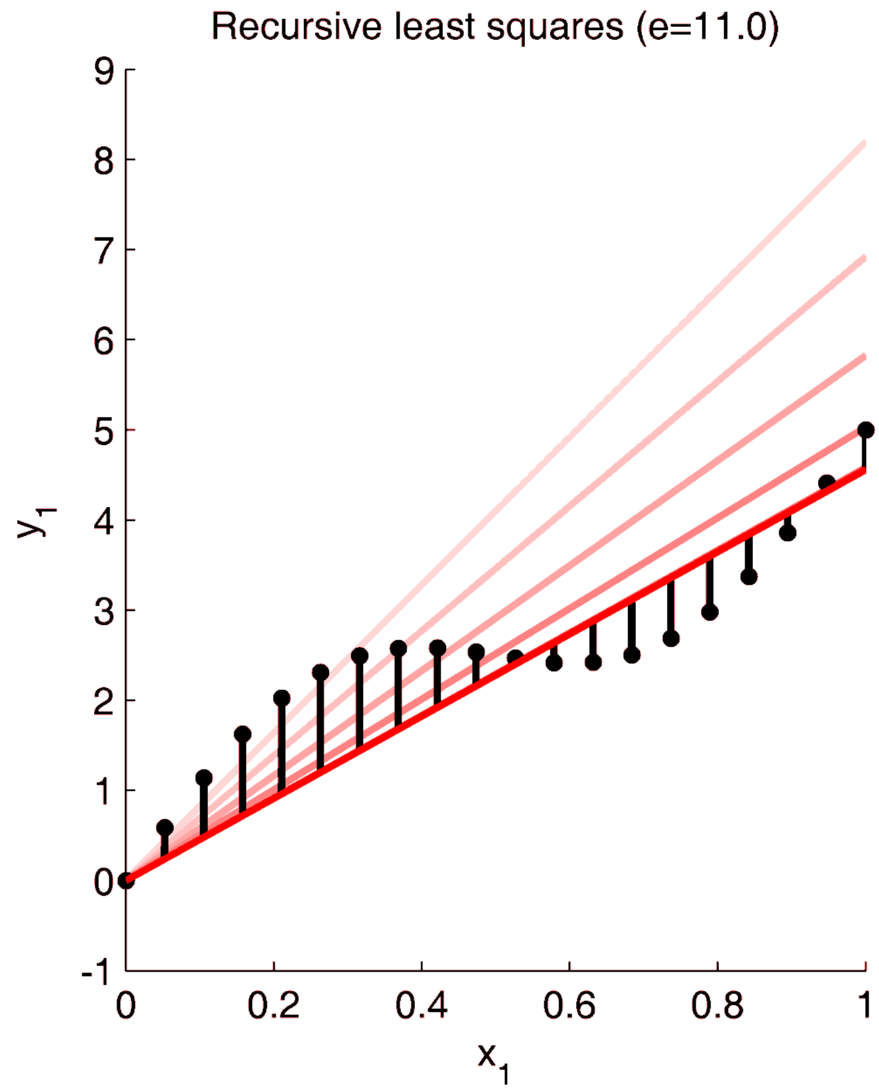
which is exploited to efficiently compute the update as

$$\hat{\boldsymbol{A}}_{\text{new}} = \hat{\boldsymbol{A}} + \boldsymbol{K}\Big(\boldsymbol{C} - \boldsymbol{V}\hat{\boldsymbol{A}}\Big)$$

with Kalman gain $\quad \boldsymbol{K} = \boldsymbol{B}^{-1}\boldsymbol{V}^\top\left(\boldsymbol{I}+\boldsymbol{V}\boldsymbol{B}^{-1}\boldsymbol{V}^\top\right)^{-1}$

Recursive least squares





Linear regression:

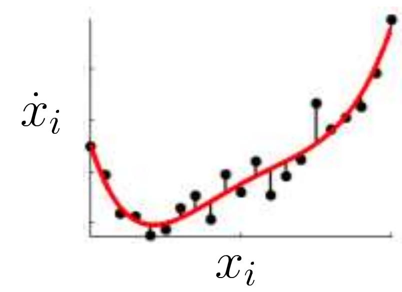
Examples of applications

Koopman operators in control

Nonlinear

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x})$$

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} \lambda_1 x_1 \\ \lambda_2(x_2 - x_1^2) \end{bmatrix}$$

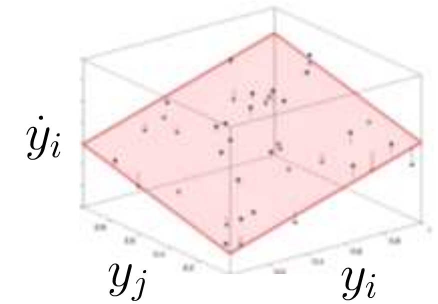


Linear in state space of higher dimension

$$\dot{\boldsymbol{y}} = \boldsymbol{A}\,\boldsymbol{y}$$

$$\begin{bmatrix} \dot{y}_1 \\ \dot{y}_2 \\ \dot{y}_3 \end{bmatrix} = \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & -\lambda_2 \\ 0 & 0 & 2\lambda_1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \quad \text{with} \quad \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_1^2 \end{bmatrix}$$

$$\dot{y}_3 = \frac{\partial y_3}{\partial x_1}\,\dot{x}_1 = 2x_1\,\lambda_1 x_1 = 2\lambda_1 y_3$$



Main challenge in Koopman analysis:
How to find these basis functions?

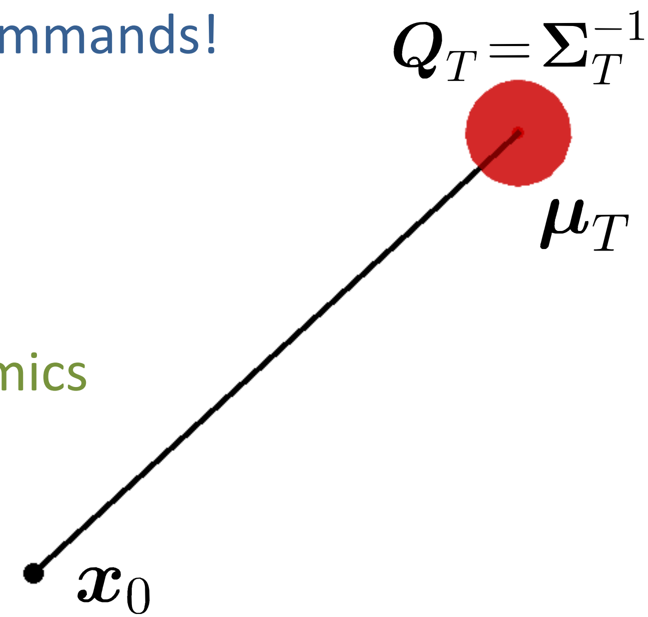
Linear quadratic tracking (LQT)

Track path! Use low control commands!

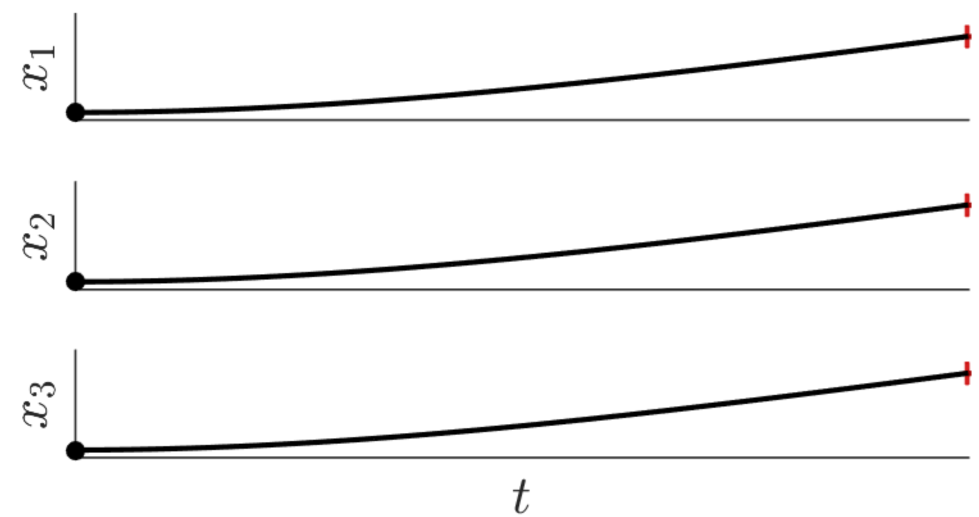
$$\min_{\boldsymbol{u}} \sum_{t=1}^{T} \left\|\boldsymbol{\mu}_t - \boldsymbol{x}_t\right\|_{\boldsymbol{Q}_t}^2 + \left\|\boldsymbol{u}_t\right\|_{\boldsymbol{R}_t}^2$$

$$\text{s.t.} \quad \boldsymbol{x}_{t+1} = \boldsymbol{A}\boldsymbol{x}_t + \boldsymbol{B}\boldsymbol{u}_t$$

System dynamics



- $\boldsymbol{x}_t$ state variable (position+velocity)
- $\boldsymbol{\mu}_t$ desired state
- $\boldsymbol{u}_t$ control command (acceleration)
- $\boldsymbol{Q}_t$ precision matrix
- $\boldsymbol{R}_t$ control weight matrix



How to solve this objective function?

Track path! Use low control commands!

$$\min_{\boldsymbol{u}} \sum_{t=1}^{T} \left\| \boldsymbol{\mu}_t - \boldsymbol{x}_t \right\|^2_{\boldsymbol{Q}_t} + \left\| \boldsymbol{u}_t \right\|^2_{\boldsymbol{R}_t}$$

$$\text{s.t.} \quad \boldsymbol{x}_{t+1} = \boldsymbol{A}\boldsymbol{x}_t + \boldsymbol{B}\boldsymbol{u}_t$$

System dynamics

Pontryagin's max. principle, Riccati equation, Hamilton-Jacobi-Bellman

(the Physicist perspective)

Dynamic programming

(the Computer Scientist perspective)

Linear algebra

(the Algebraist perspective)

Let's first re-organize the objective function...

$$c = \sum_{t=1}^{T} \Big((\boldsymbol{\mu}_t - \boldsymbol{x}_t)^\top \boldsymbol{Q}_t (\boldsymbol{\mu}_t - \boldsymbol{x}_t) \; + \; \boldsymbol{u}_t^\top \boldsymbol{R}_t \, \boldsymbol{u}_t \Big)$$

$$= (\boldsymbol{\mu} - \boldsymbol{x})^\top \boldsymbol{Q} (\boldsymbol{\mu} - \boldsymbol{x}) \; + \; \boldsymbol{u}^\top \boldsymbol{R} \boldsymbol{u}$$

$$\boldsymbol{Q} = \begin{bmatrix} \boldsymbol{Q}_1 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \boldsymbol{Q}_2 & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \boldsymbol{Q}_T \end{bmatrix} \qquad \boldsymbol{R} = \begin{bmatrix} \boldsymbol{R}_1 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \boldsymbol{R}_2 & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \boldsymbol{R}_T \end{bmatrix}$$

$$\boldsymbol{\mu} = \begin{bmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \\ \vdots \\ \boldsymbol{\mu}_T \end{bmatrix} \qquad \boldsymbol{x} = \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix} \qquad \boldsymbol{u} = \begin{bmatrix} \boldsymbol{u}_1 \\ \boldsymbol{u}_2 \\ \vdots \\ \boldsymbol{u}_T \end{bmatrix}$$

Let's then re-organize the constraint...

$$\boldsymbol{x}_{t+1} = \boldsymbol{A}\,\boldsymbol{x}_t + \boldsymbol{B}\,\boldsymbol{u}_t$$

$$\begin{aligned} \boldsymbol{x}_2 &= \boldsymbol{A}\boldsymbol{x}_1 + \boldsymbol{B}\boldsymbol{u}_1 \\ \boldsymbol{x}_3 &= \boldsymbol{A}\boldsymbol{x}_2 + \boldsymbol{B}\boldsymbol{u}_2 = \boldsymbol{A}(\boldsymbol{A}\boldsymbol{x}_1 + \boldsymbol{B}\boldsymbol{u}_1) + \boldsymbol{B}\boldsymbol{u}_2 \\ &\vdots \\ \boldsymbol{x}_T &= \boldsymbol{A}^{T-1}\boldsymbol{x}_1 + \boldsymbol{A}^{T-2}\boldsymbol{B}\boldsymbol{u}_1 + \boldsymbol{A}^{T-3}\boldsymbol{B}\boldsymbol{u}_2 + \cdots + \boldsymbol{B}_{T-1}\boldsymbol{u}_{T-1} \end{aligned}$$

$$\begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \boldsymbol{x}_3 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix} = \underbrace{\begin{bmatrix} \boldsymbol{I} \\ \boldsymbol{A} \\ \boldsymbol{A}^2 \\ \vdots \\ \boldsymbol{A}^{T-1} \end{bmatrix}}_{\boldsymbol{S}^x} \boldsymbol{x}_1 + \underbrace{\begin{bmatrix} 0 & 0 & \cdots & 0 & 0 \\ \boldsymbol{B} & 0 & \cdots & 0 & 0 \\ \boldsymbol{A}\boldsymbol{B} & \boldsymbol{B} & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \boldsymbol{A}^{T-2}\boldsymbol{B} & \boldsymbol{A}^{T-3}\boldsymbol{B} & \cdots & \boldsymbol{B} & 0 \end{bmatrix}}_{\boldsymbol{S}^u} \begin{bmatrix} \boldsymbol{u}_1 \\ \boldsymbol{u}_2 \\ \vdots \\ \boldsymbol{u}_T \end{bmatrix}$$

$$\boldsymbol{x} = \boldsymbol{S}^x\boldsymbol{x}_1 + \boldsymbol{S}^u\boldsymbol{u}$$

Linear quadratic tracking (LQT)

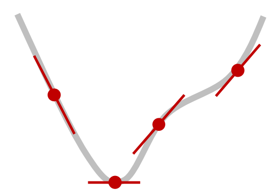
The constraint can then be put into the objective function:

$$\boldsymbol{x} = \boldsymbol{S}^{\boldsymbol{x}} \boldsymbol{x}_1 + \boldsymbol{S}^{\boldsymbol{u}} \boldsymbol{u}$$

$$c = (\boldsymbol{\mu} - \boldsymbol{x})^{\top} \boldsymbol{Q} (\boldsymbol{\mu} - \boldsymbol{x}) + \boldsymbol{u}^{\top} \boldsymbol{R} \boldsymbol{u}$$

$$= (\boldsymbol{\mu} - \boldsymbol{S}^{\boldsymbol{x}} \boldsymbol{x}_1 - \boldsymbol{S}^{\boldsymbol{u}} \boldsymbol{u})^{\top} \boldsymbol{Q} (\boldsymbol{\mu} - \boldsymbol{S}^{\boldsymbol{x}} \boldsymbol{x}_1 - \boldsymbol{S}^{\boldsymbol{u}} \boldsymbol{u}) + \boldsymbol{u}^{\top} \boldsymbol{R} \boldsymbol{u}$$

Solving for *u* results in the analytic solution:



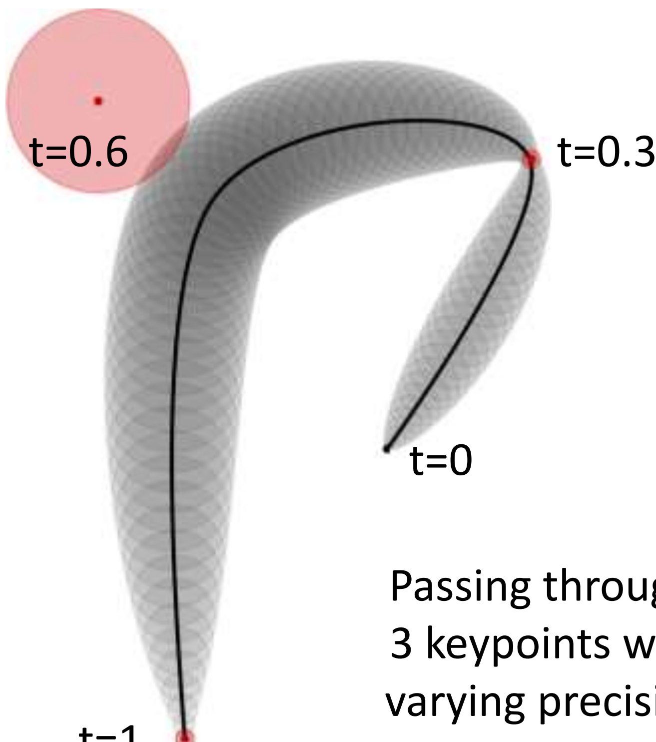
$$\hat{\boldsymbol{u}} = \left(\boldsymbol{S}^{\boldsymbol{u}\top} \boldsymbol{Q} \boldsymbol{S}^{\boldsymbol{u}} + \boldsymbol{R}\right)^{-1} \boldsymbol{S}^{\boldsymbol{u}\top} \boldsymbol{Q} \left(\boldsymbol{\mu} - \boldsymbol{S}^{\boldsymbol{x}} \boldsymbol{x}_1\right)$$

Linear quadratic tracking (LQT)

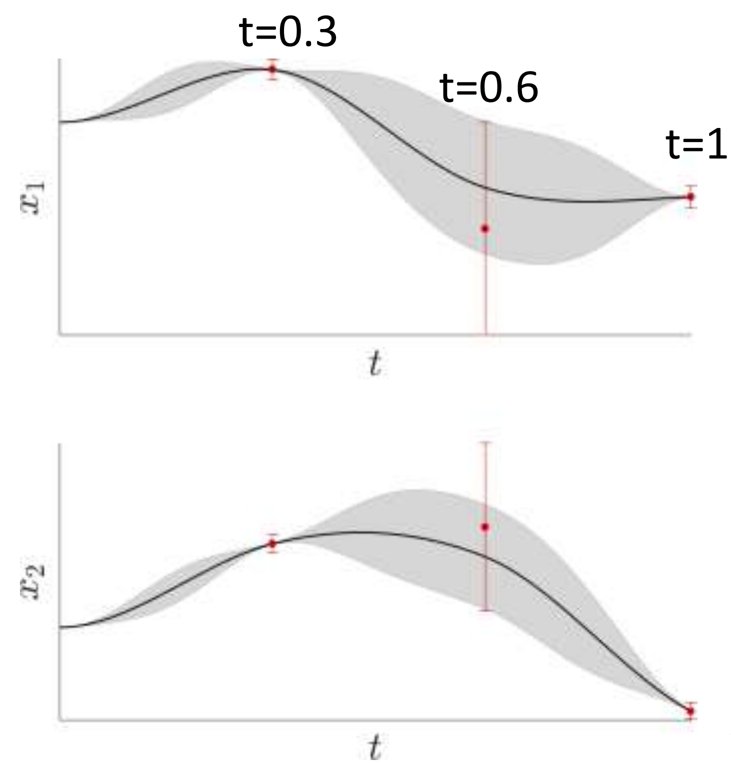
$$\hat{\boldsymbol{u}} = \left(\boldsymbol{S}^{u\top}\boldsymbol{Q}\boldsymbol{S}^{u} + \boldsymbol{R}\right)^{-1}\boldsymbol{S}^{u\top}\boldsymbol{Q}\,\left(\boldsymbol{\mu} - \boldsymbol{S}^{x}\boldsymbol{x}_1\right)$$
$$\hat{\boldsymbol{\Sigma}}^{u} = \left(\boldsymbol{S}^{u\top}\boldsymbol{Q}\boldsymbol{S}^{u} + \boldsymbol{R}\right)^{-1}$$

$$\hat{\boldsymbol{x}} = \boldsymbol{S}^{x}\boldsymbol{x}_1 + \boldsymbol{S}^{u}\hat{\boldsymbol{u}}$$
$$\hat{\boldsymbol{\Sigma}}^{x} = \boldsymbol{S}^{u}\left(\boldsymbol{S}^{u\top}\boldsymbol{Q}\boldsymbol{S}^{u} + \boldsymbol{R}\right)^{-1}\boldsymbol{S}^{u\top}$$

The distribution in control space can be projected back to the state space



Passing through 3 keypoints with varying precision



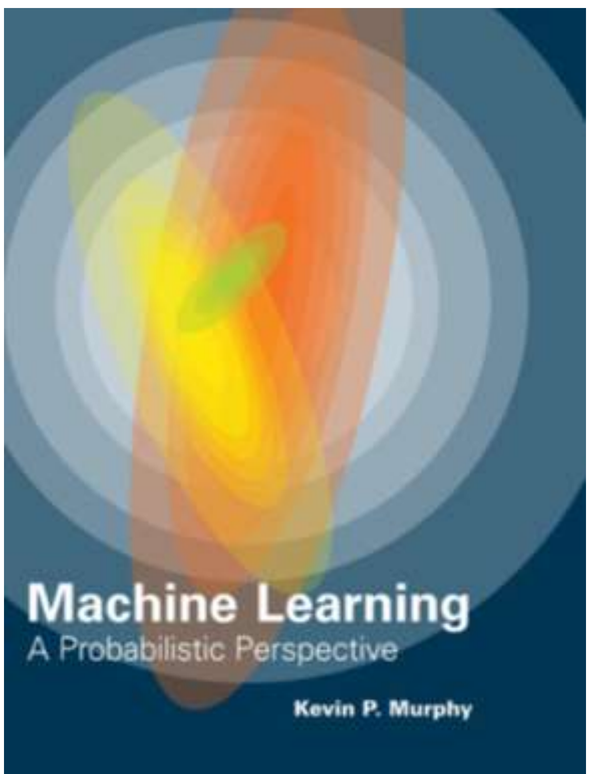
Main references

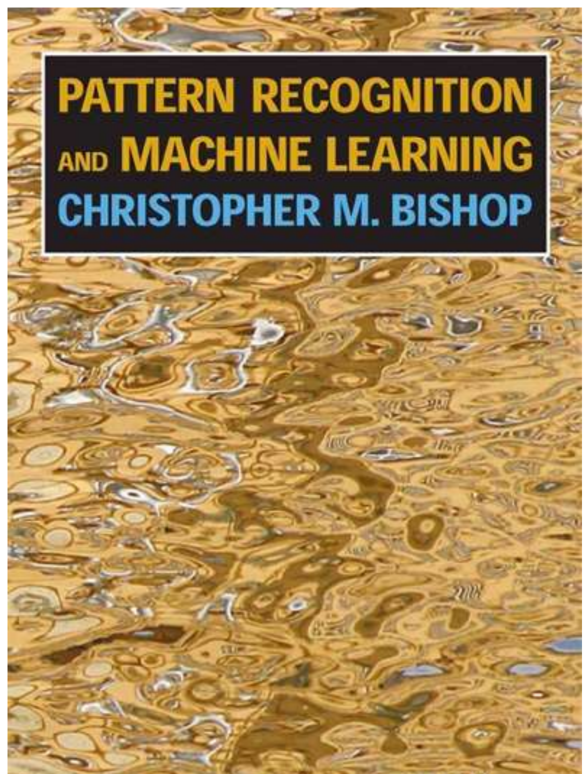
Regression

F. Stulp and O. Sigaud. Many regression algorithms, one unified model – a review. Neural Networks, 69:60–79, September 2015

W. W. Hager. Updating the inverse of a matrix. SIAM Review, 31(2):221–239, 1989

G. Strang. Introduction to Applied Mathematics. Wellesley-Cambridge Press, 1986





The Matrix Cookbook

Kaare Brandt Petersen
Michael Syskind Pedersen