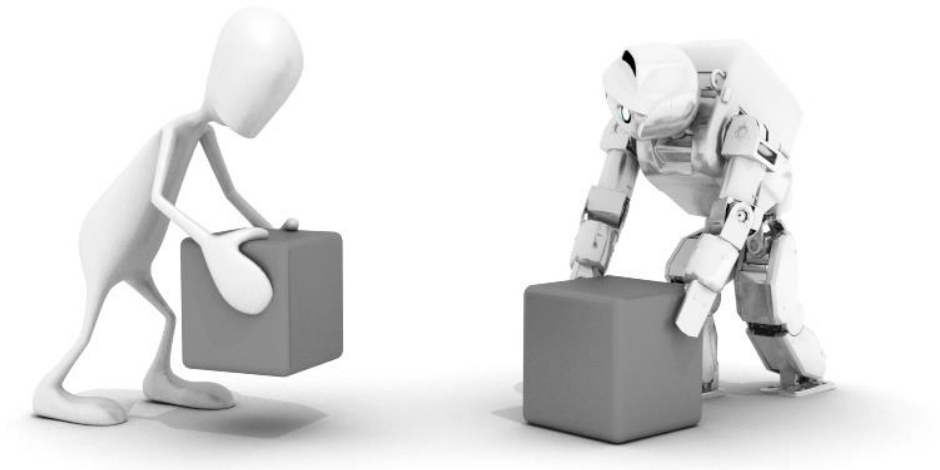
EE613 - Machine Learning for Engineers

LINEAR REGRESSION

Sylvain Calinon
Robot Learning and Interaction Group
Idiap Research Institute
Sep 30, 2021

Robot Learning and Interaction group

Idiap Research Institute



Learning from demonstration

EE613 labs (INM11):

Dr João Silvério

Outline

Linear regression (Sep 30)

- Least squares
- Singular value decomposition (SVD)
- Nullspace projection (kernels in least squares)
- Ridge regression (Tikhonov regularization)
- Weighted least squares
- Iteratively reweighted least squares (IRLS)
- Recursive least squares
- Logistic regression

Tensor regression (Oct 7)

- Tensor factorization
- Tensor-variate regression

Nonlinear regression (Oct 14)

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

Hidden Markov model (HMM) (Nov 11)

Regression

Nonlinear regression:

Approximation function
(predictor function)
(forecasting function)

$$y = f(x)$$

Output
(response variable)
(outcome variable)

Input
(explanatory variable)
(feature variable)

$\in \mathbb{R}$

Linear regression:

$$y = Ax$$

Tensor regression:

Similar to $y = Ax$, but with x a tensor instead of a vector

Nonlinear regression on x reformulated as linear regression on f:

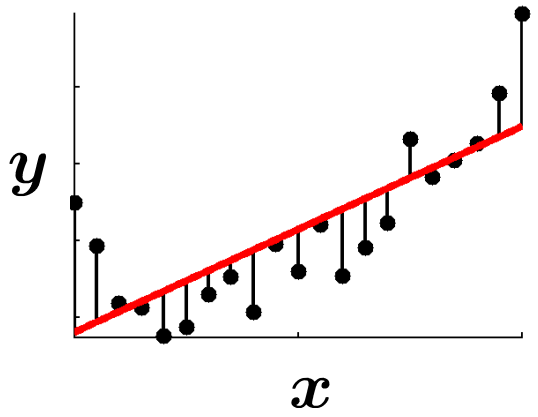
$$y = A \begin{bmatrix} f_1(x) \\ f_2(x) \\ \vdots \end{bmatrix}$$

LEAST SQUARES

circa 1795

Least squares: a ubiquitous tool

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



Weighted least squares?

Regularized least squares?

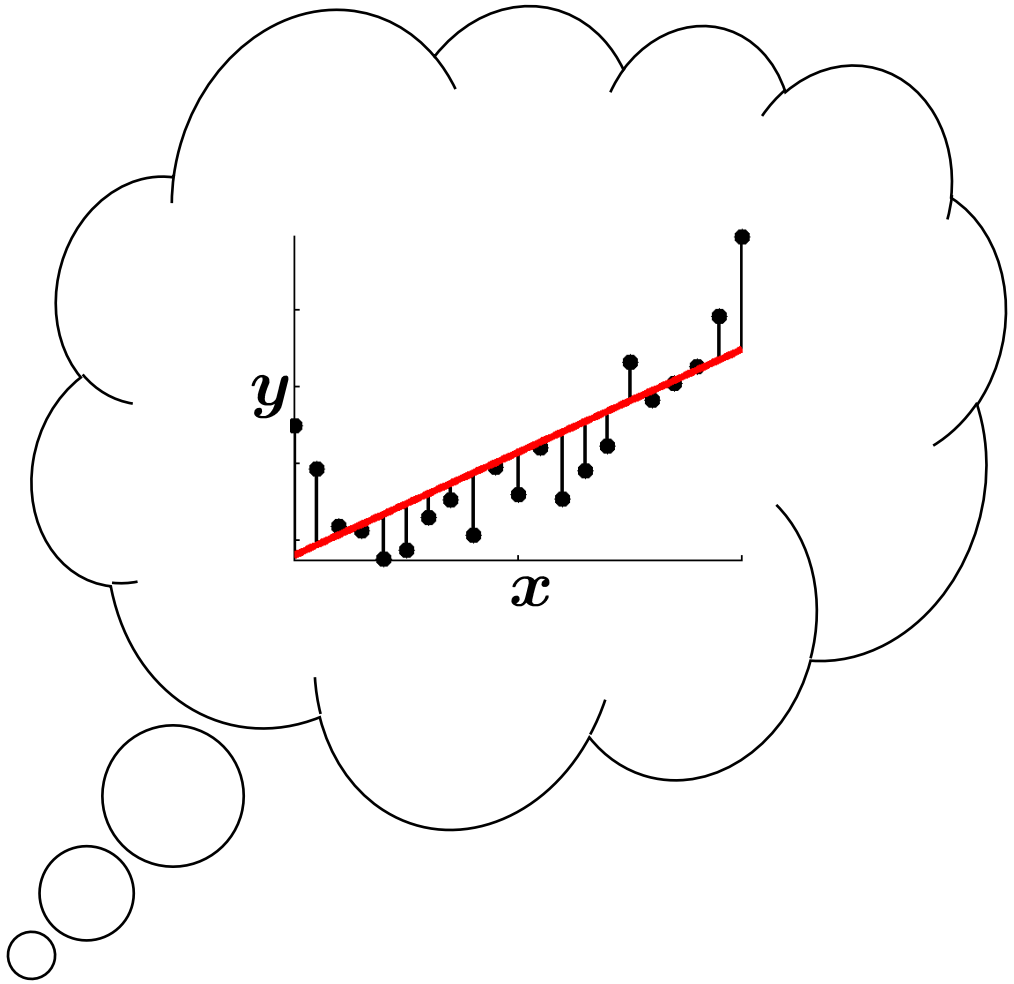
L1-norm instead of L2-norm?

Solution with a secondary objective?

Fast and robust implementation?

Recursive computation?

Youtube search for "deep learning regression"



YouTube CH

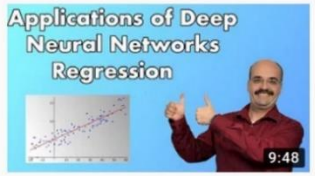
deep learning regression

Home
Trending
History

BEST OF YOUTUBE
Music
Sports
Gaming
Movies
News
Live
360° Video

Browse channels

FILTER



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 ...

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 ...

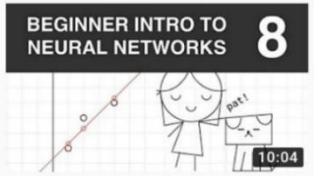
CC

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: Linear Regression

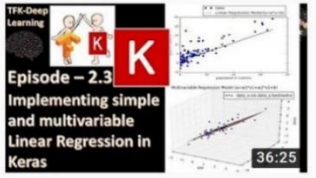
giant_neural_network • 53K views • 1 year ago

Hey everyone! In this video we're going to look at something called linear **regression**. We're really just adding an input to our ...

Learning Tensorflow with linear regression

Technology for Noobs • 3.5K views • 1 year ago

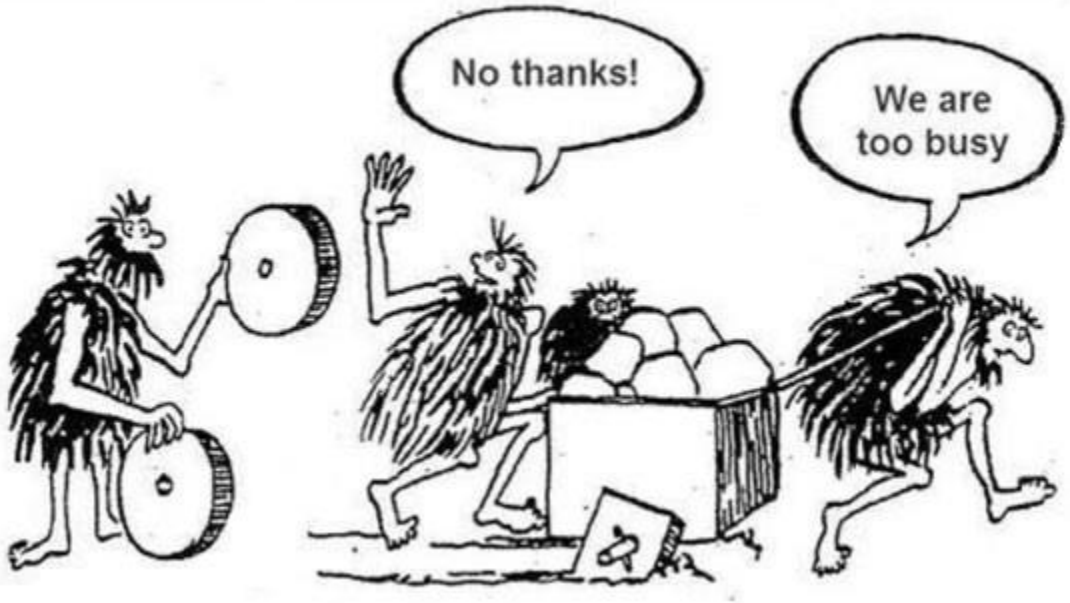
In this video, I will cover basics of tensorflow. Below are the topics that will be covered: 1. Basic of linear **regression** 2. Basics of ...



Ep-2.3: Linear Regression in Keras || TFK-Deep Learning || Exploring Neurons

Anuj shah • 1.2K views • 1 year ago

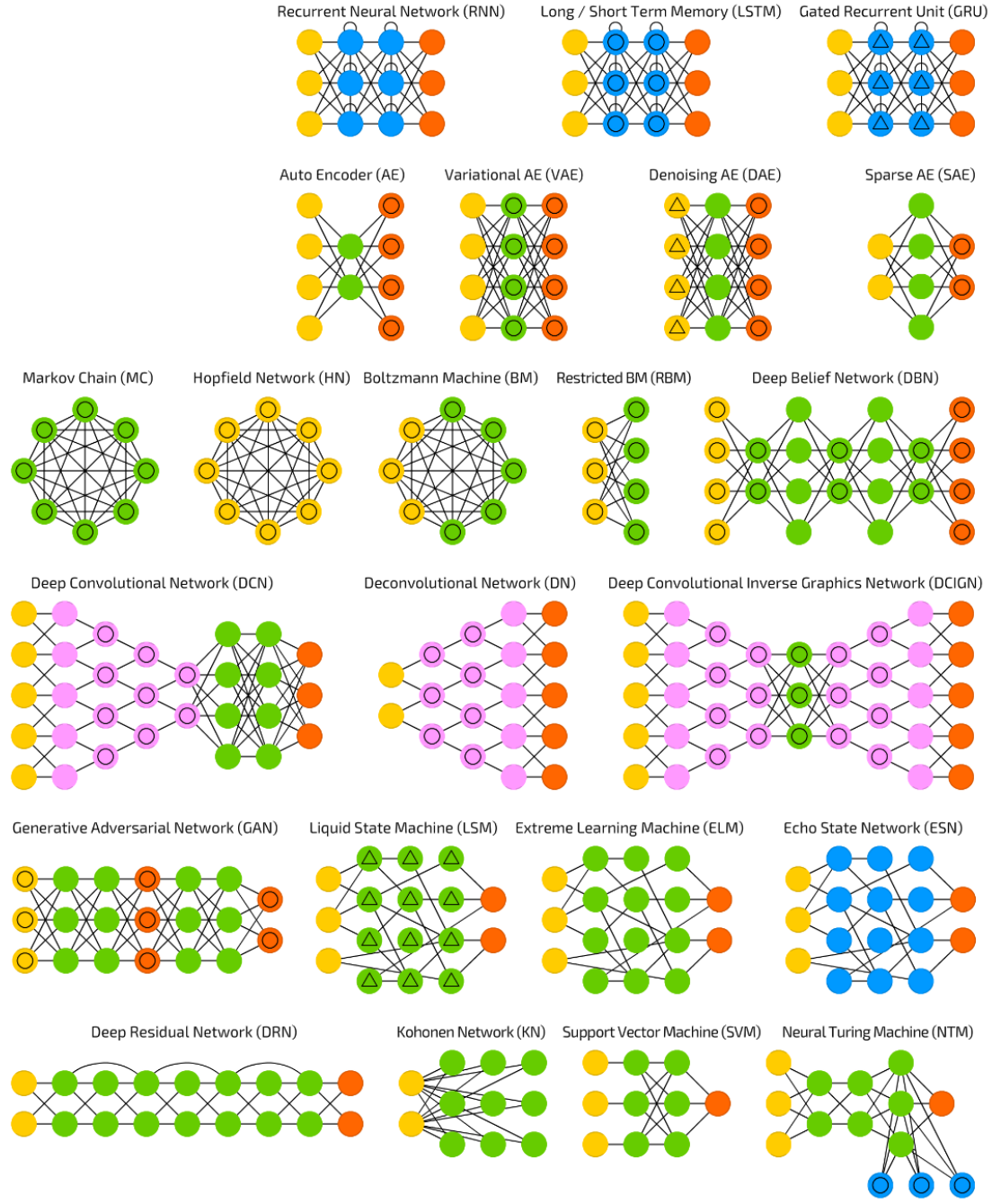
This video explains the implementation of simple and multiple linear **regression** in keras. The theoretical discussion of linear ...



The **pinv-net!** ☺
single layer,
single node,
linear activation!

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

- 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

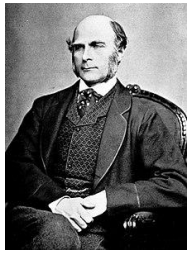


Multivariate linear regression (ordinary least squares)

- **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

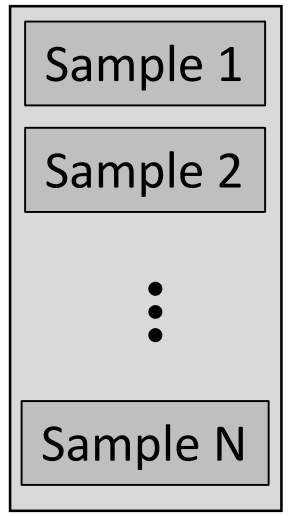


Francis Galton

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 $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{a}$.



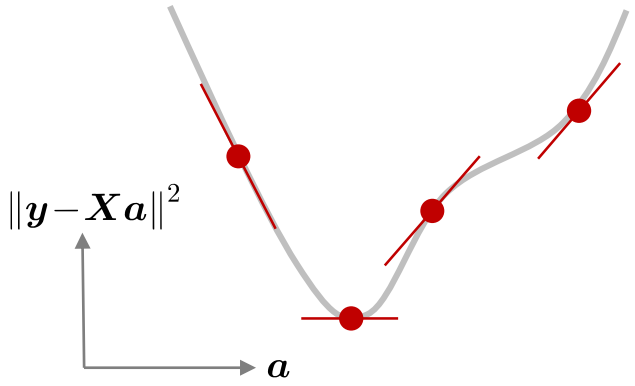
A solution can be found by minimizing the ℓ_2 norm

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

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

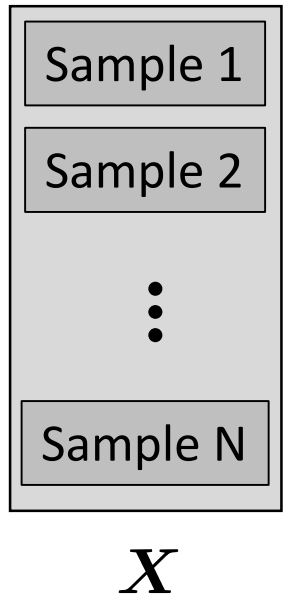
$$-2\boldsymbol{X}^\top \boldsymbol{y} + 2\boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{a} = \boldsymbol{0} \quad \Longleftrightarrow \quad \hat{\boldsymbol{a}} = \left(\boldsymbol{X}^\top \boldsymbol{X}\right)^{-1} \boldsymbol{X}^\top \boldsymbol{y}$$

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 $\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{A}$.



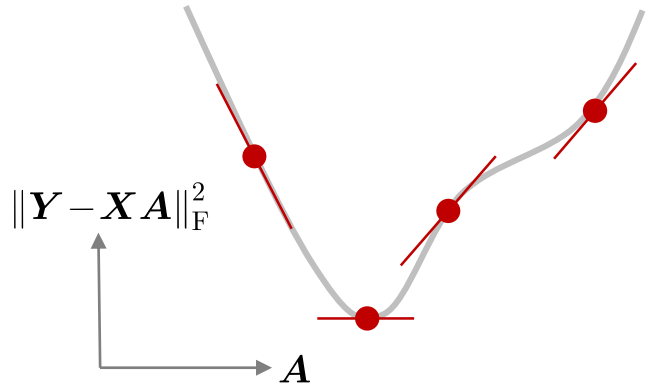
A solution can be found by minimizing the Frobenius norm

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

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

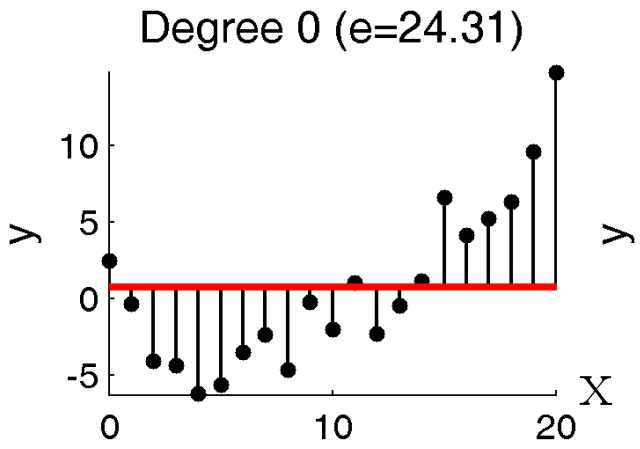
$$-2\boldsymbol{X}^\top\boldsymbol{Y} + 2\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{A} = \boldsymbol{0} \quad \Longleftrightarrow \quad \hat{\boldsymbol{A}} = \boxed{(\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top}\boldsymbol{Y}$$

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

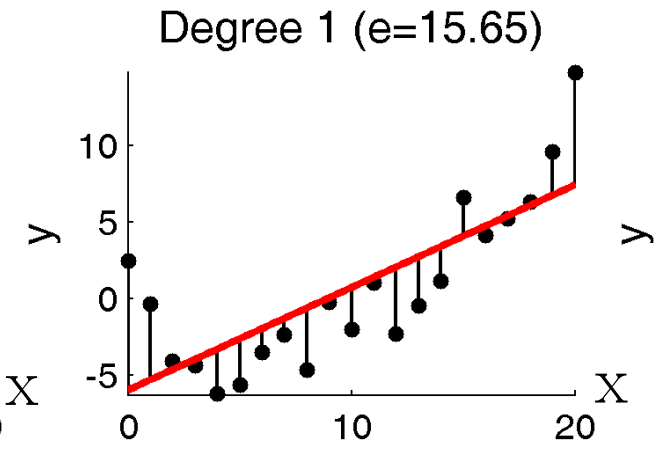


Polynomial fitting with least squares

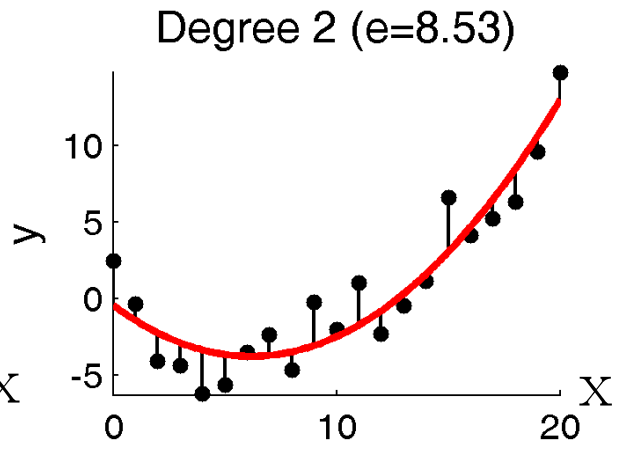
$$\hat{A} = X^{\dagger} Y$$



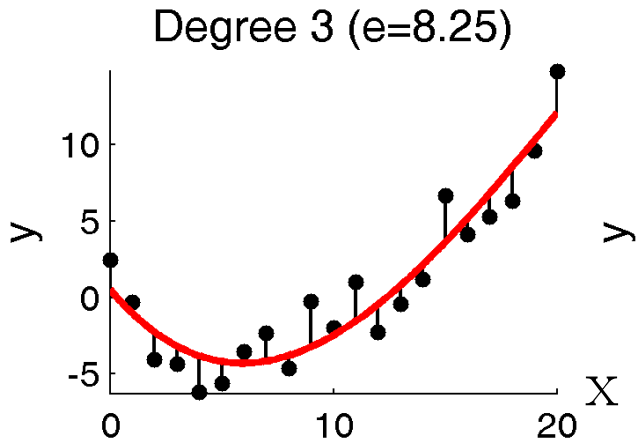
$x = 1$



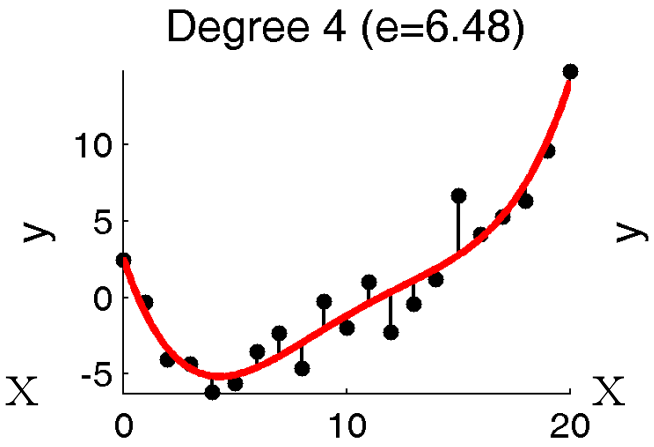
$\boldsymbol{x} = [1, \mathrm{x}]$



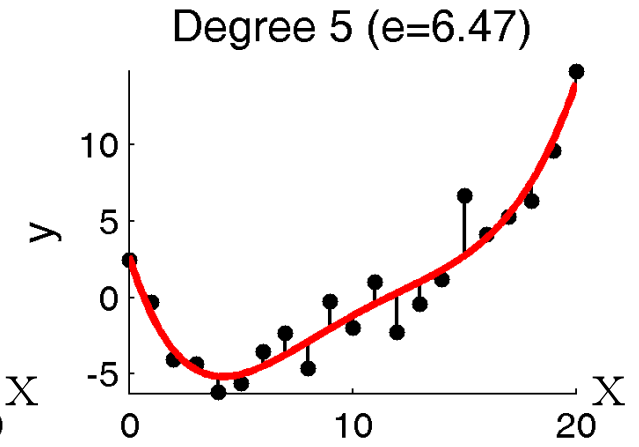
$\boldsymbol{x} = [1, \mathrm{x}, \mathrm{x}^2]$



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



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



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

Least squares computed with SVD

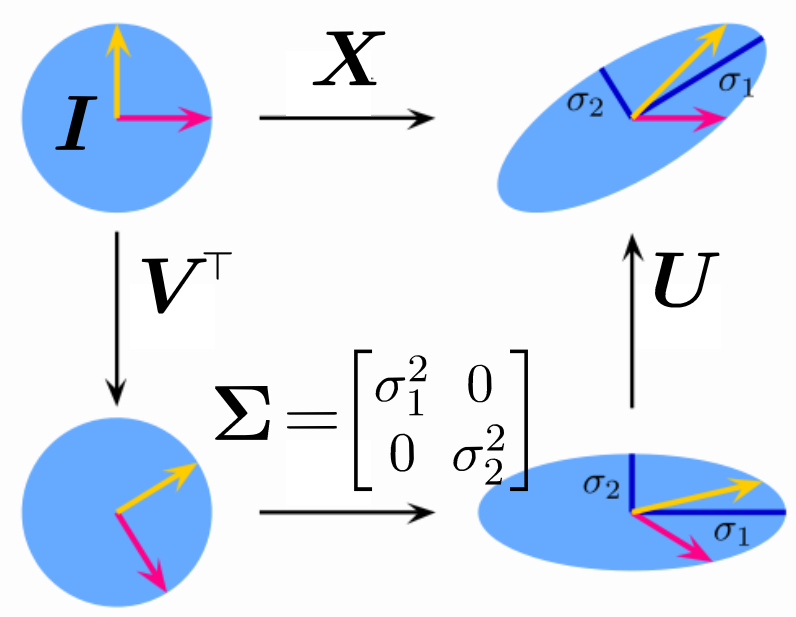
Singular value decomposition (SVD)

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

Unitary matrix (orthogonal)

Unitary matrix (orthogonal)

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



Least squares computed with SVD

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

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

$\boldsymbol{X}$ 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.

$$\boldsymbol{\Sigma} = \begin{bmatrix} 4 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

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.

$$\boldsymbol{\Sigma}^\dagger = \begin{bmatrix} 0.25 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Nullspace projection (kernels in least squares)

Python notebook:
demo_LS_polFit.ipynb

Matlab code:
demo_LS_polFit_nullspace01.m

Nullspace projection

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

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

$\boldsymbol{V}$ 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.

Nullspace projection computed with SVD

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

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

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

to compute

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

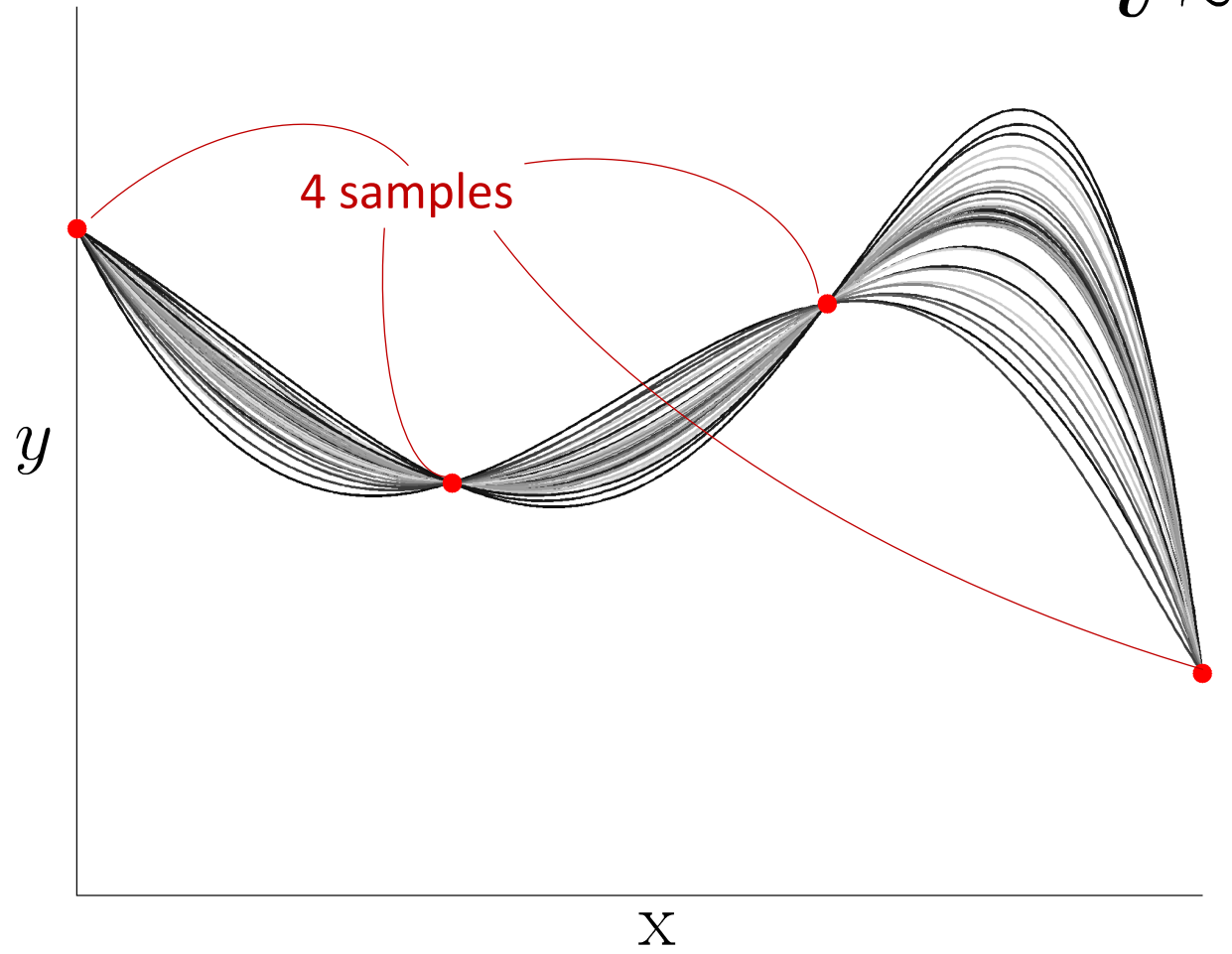
where $\tilde{\boldsymbol{U}}$ is a matrix formed by the columns of $\boldsymbol{U}$ that span for the corresponding zero rows in $\boldsymbol{\Sigma}$.

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

$\tilde{\boldsymbol{U}}$

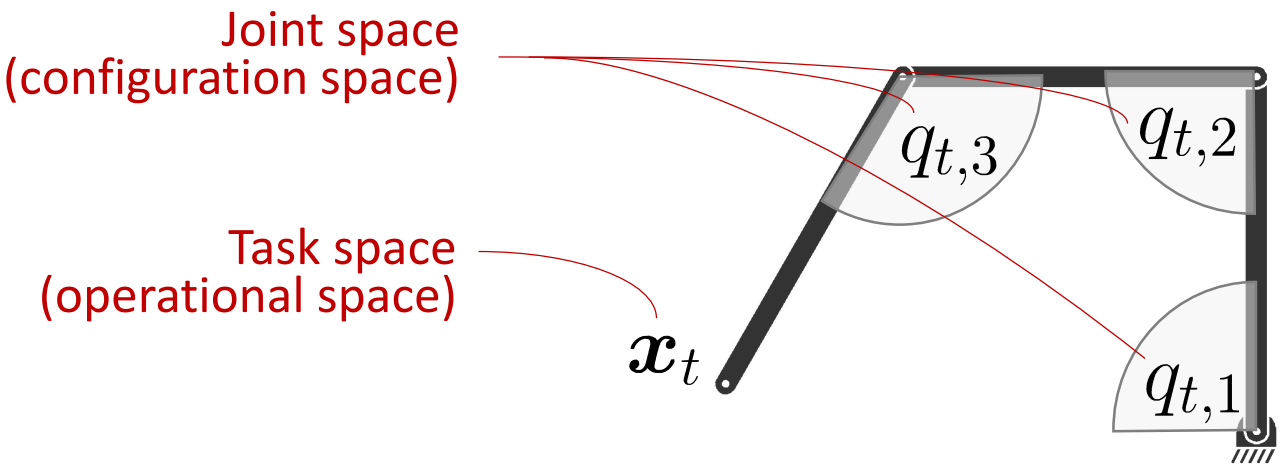
Example: Polynomial fitting

$$\hat{\boldsymbol{a}} = \boldsymbol{X}^{\dagger}\boldsymbol{y} + \boldsymbol{N}\boldsymbol{v} \quad \text{with} \quad \boldsymbol{x} = [1, \mathrm{x}, \mathrm{x}^2, \ldots, \mathrm{x}^6]$$
$$\boldsymbol{v} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$$



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

Example: Robot inverse kinematics



Forward kinematics is computed with

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

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

An inverse kinematics solution can be computed with

$$\hat{\dot{\boldsymbol{q}}}_t = \boldsymbol{J}^{\dagger}(\boldsymbol{q}_t)\, \dot{\boldsymbol{x}}_t + \boldsymbol{N}(\boldsymbol{q}_t)\, \boldsymbol{g}(\boldsymbol{q}_t), \quad \text{where} \quad \boldsymbol{N}(\boldsymbol{q}_t) = \boldsymbol{I} - \boldsymbol{J}^{\dagger}(\boldsymbol{q}_t)\boldsymbol{J}(\boldsymbol{q}_t).$$

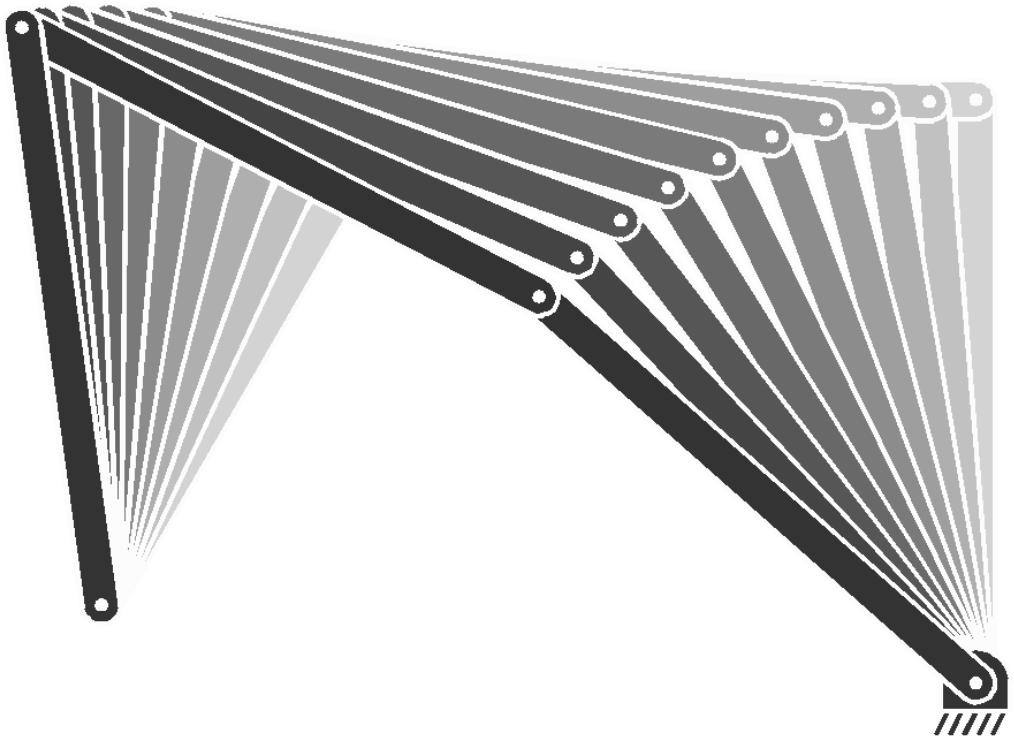
Example: Robot inverse kinematics

Primary constraint:
keeping the tip of
the robot still

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

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

Secondary constraint:
moving the
first joint



Example: Robot inverse kinematics

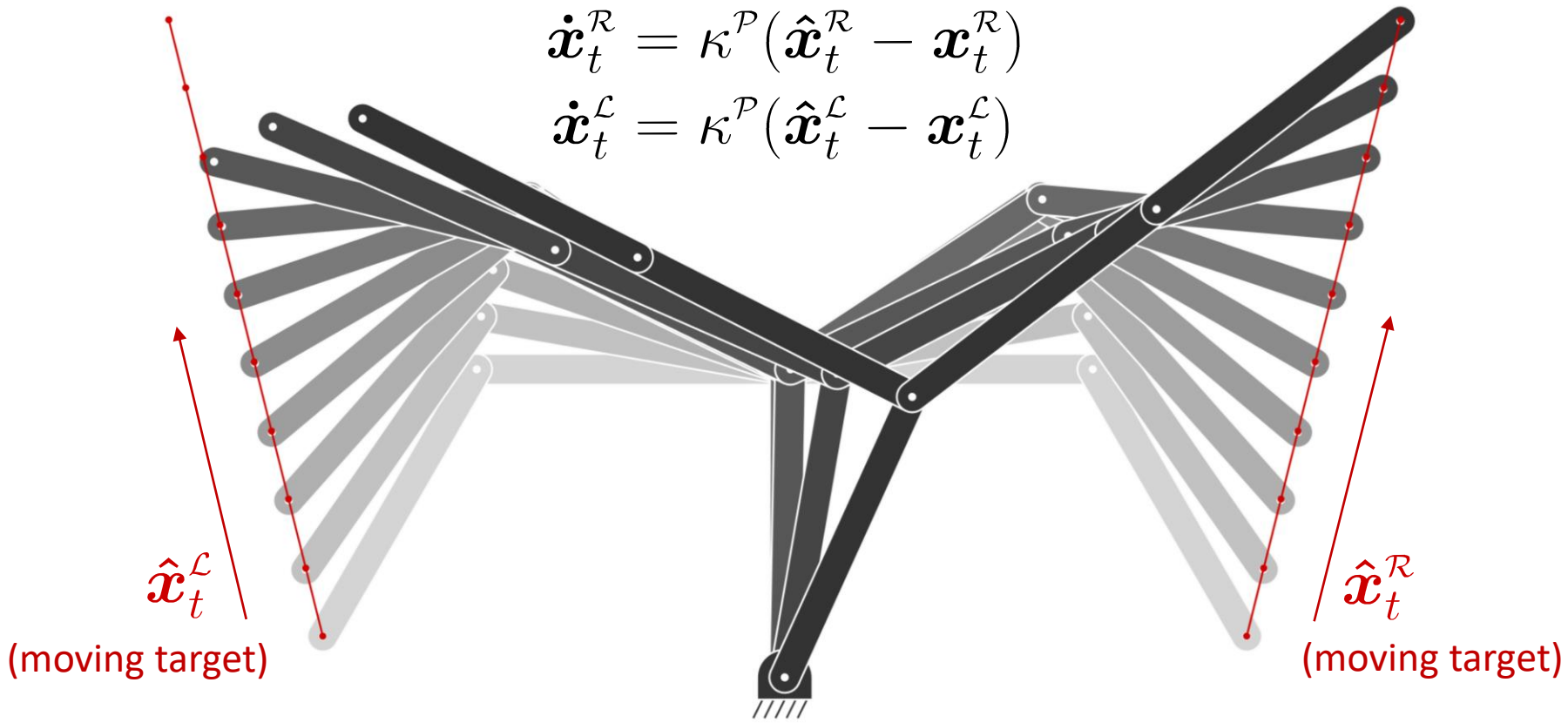
$$\hat{\dot{\boldsymbol{q}}}_t = \boldsymbol{J}_t^{\mathcal{R}\dagger}\, \dot{\boldsymbol{x}}_t^{\mathcal{R}} + \boldsymbol{N}_t^{\mathcal{R}}\, \dot{\boldsymbol{q}}_t^{\mathcal{L}}$$

Tracking target with right hand

Tracking target with left hand, if possible

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

$$\dot{\boldsymbol{x}}_t^{\mathcal{R}} = \kappa^{\mathcal{P}}(\hat{\boldsymbol{x}}_t^{\mathcal{R}} - \boldsymbol{x}_t^{\mathcal{R}})$$
$$\dot{\boldsymbol{x}}_t^{\mathcal{L}} = \kappa^{\mathcal{P}}(\hat{\boldsymbol{x}}_t^{\mathcal{L}} - \boldsymbol{x}_t^{\mathcal{L}})$$



Ridge regression

(robust regression, Tikhonov regularization, penalized least squares)

Python notebook:
demo_LS_polFit.ipynb

Matlab example:
demo_LS_polFit02.m

Ridge regression

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

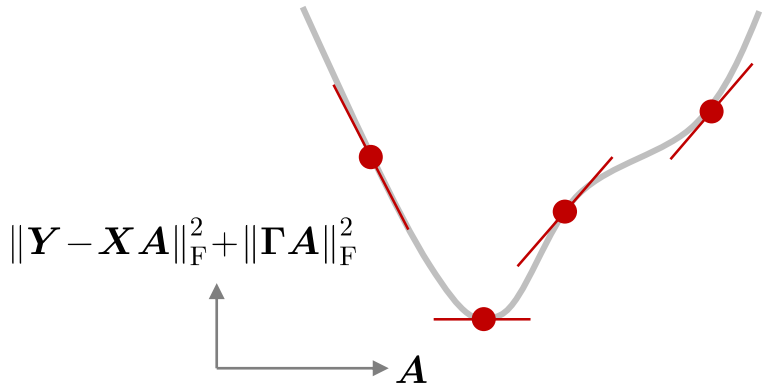
$$\begin{aligned}\hat{\boldsymbol{A}} &= \arg\min_{\boldsymbol{A}} \|\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{A}\|_{\mathrm{F}}^2 + \|\boldsymbol{\Gamma}\boldsymbol{A}\|_{\mathrm{F}}^2 \\ &= \arg\min_{\boldsymbol{A}} \operatorname{tr}\Big((\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{A})^\top(\boldsymbol{Y}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

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 $0 < \lambda \ll 1$ (i.e., giving preference to solutions with smaller norms), the process is known as **ℓ_2 regularization**.

Ridge regression

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

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

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

Ridge regression computed 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

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

and has zeros elsewhere.

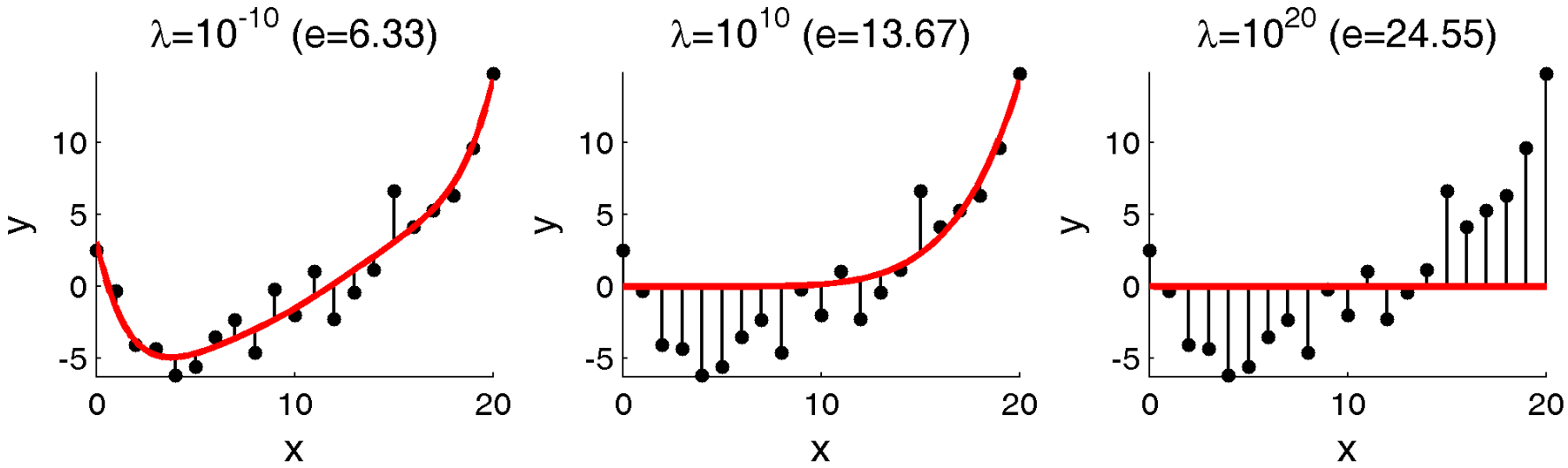
$$\boldsymbol{\Sigma} = \begin{bmatrix} 4 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\tilde{\boldsymbol{\Sigma}} = \begin{bmatrix} 0.2498 & 0 & 0 & 0 \\ 0 & 0.4988 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

(for $\lambda = 0.01$)

Example: Polynomial fitting

$D^{\mathcal{I}} = 7$ (polynomial of degree 6)



Weighted least squares
(Generalized least squares)

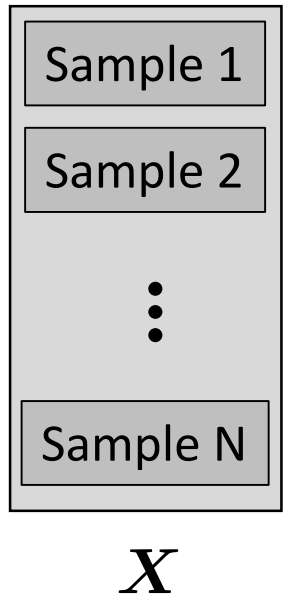
Python notebook:
demo_LS_weighted.ipynb

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

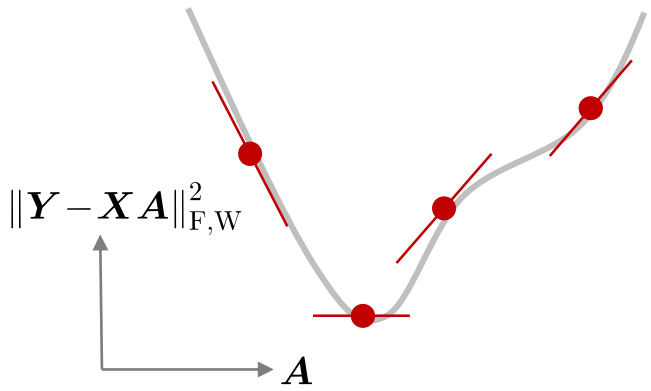
$$\begin{aligned}\hat{\boldsymbol{A}} &= \arg\min_{\boldsymbol{A}} \|\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{A}\|^2_{\mathrm{F},\boldsymbol{W}} \\ &= \arg\min_{\boldsymbol{A}} \operatorname{tr}\Big((\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{A})^\top \boldsymbol{W}(\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{A})\Big)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



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

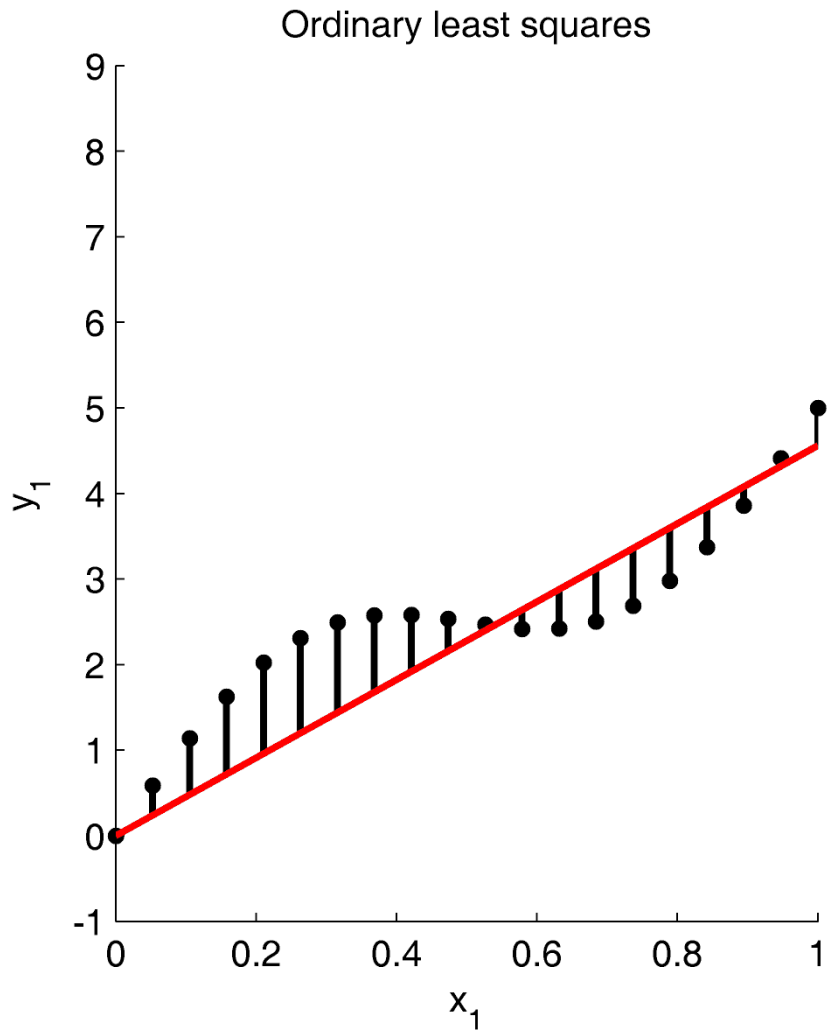
$$-2\boldsymbol{X}^\top\boldsymbol{W}\boldsymbol{Y} + 2\boldsymbol{X}^\top\boldsymbol{W}\boldsymbol{X}\boldsymbol{A} = \boldsymbol{0} \iff \hat{\boldsymbol{A}} = \boxed{(\boldsymbol{X}^\top\boldsymbol{W}\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{W}}\boldsymbol{Y}$$

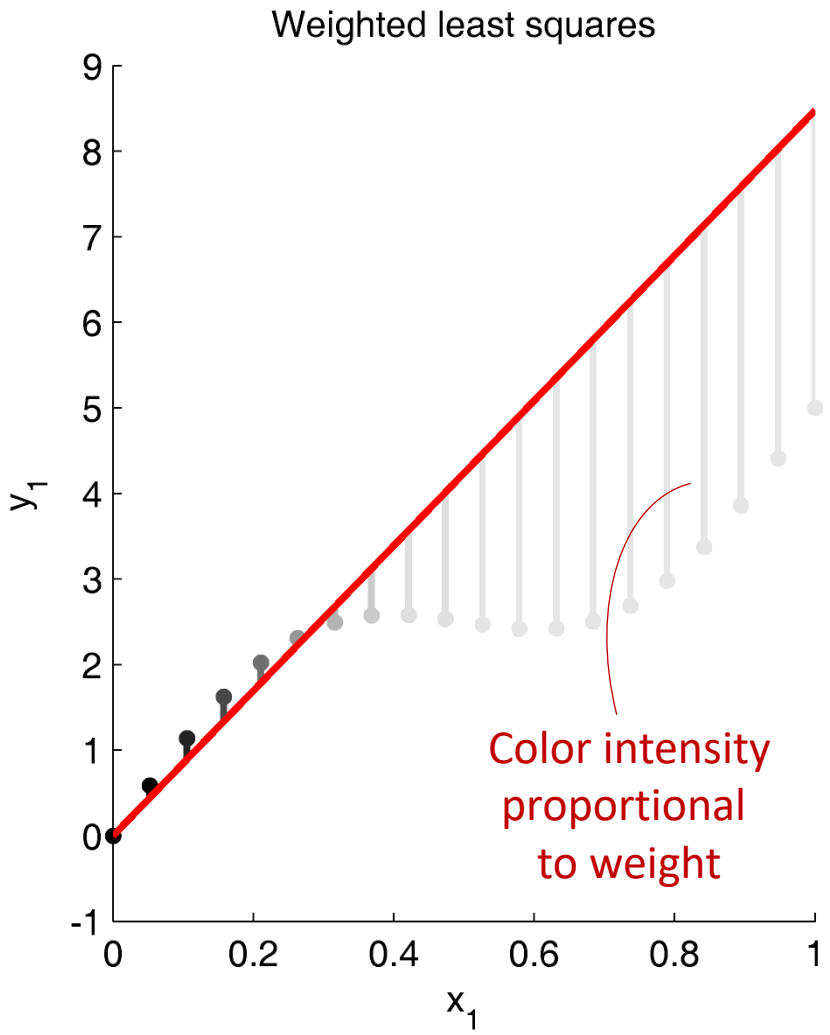
$$\boldsymbol{X}^\dagger_{\boldsymbol{W}}$$



Weighted least squares

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

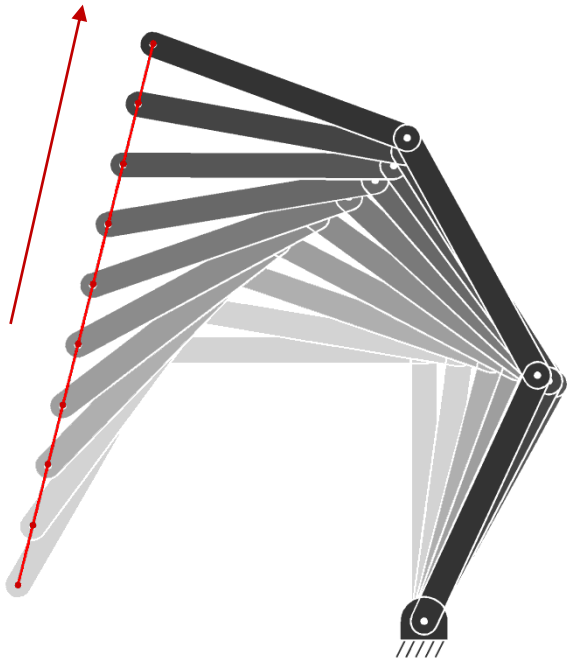


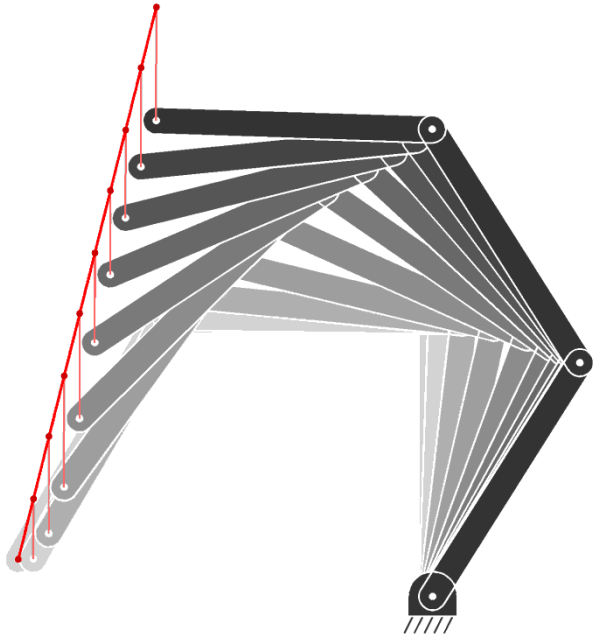


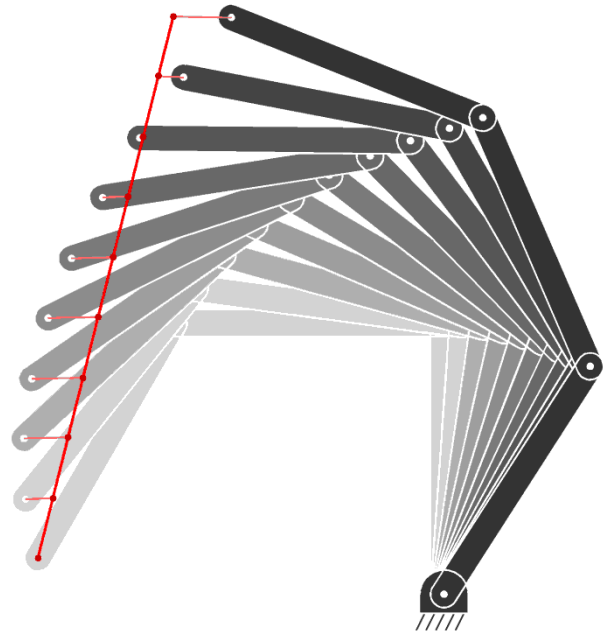
Example: Robot inverse kinematics (weights in task space)

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

(moving target)







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

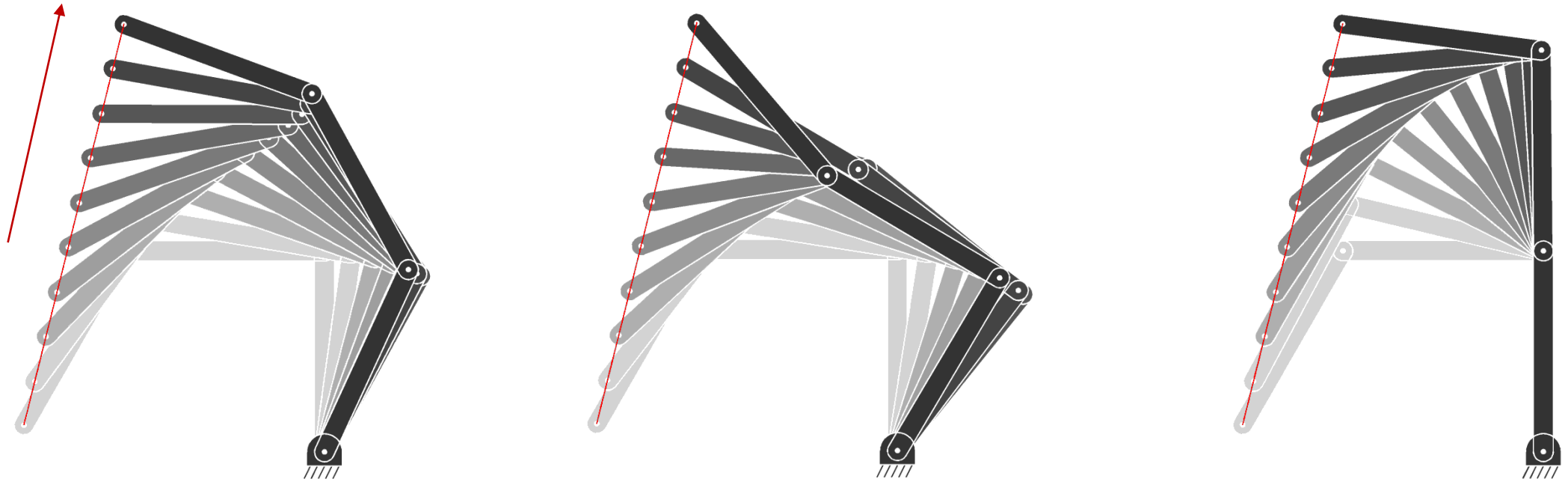
$$\boldsymbol{W}^x = \begin{bmatrix} 1 & 0 \\ 0 & .01 \end{bmatrix}$$

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

Example: Robot inverse kinematics (weights in joint space)

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

(moving target)



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

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

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

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 (IRLS)** generalizes least squares by raising the error term to a power that is less than 2:
  → the optimization problem can no longer be called “least squares”

- The strategy of IRLS 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**.
  → we solve a least squares problem at each iteration of the algorithm

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

Iteratively reweighted least squares (IRLS)

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

transformed as weight **W**

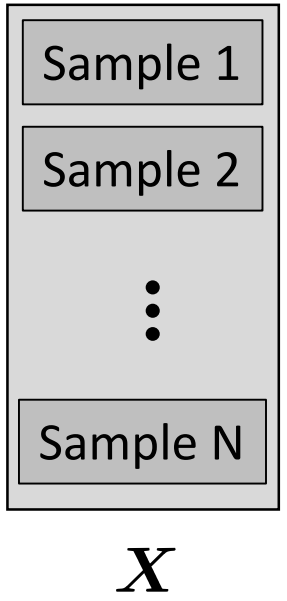
For an ℓ_p norm cost function defined by

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

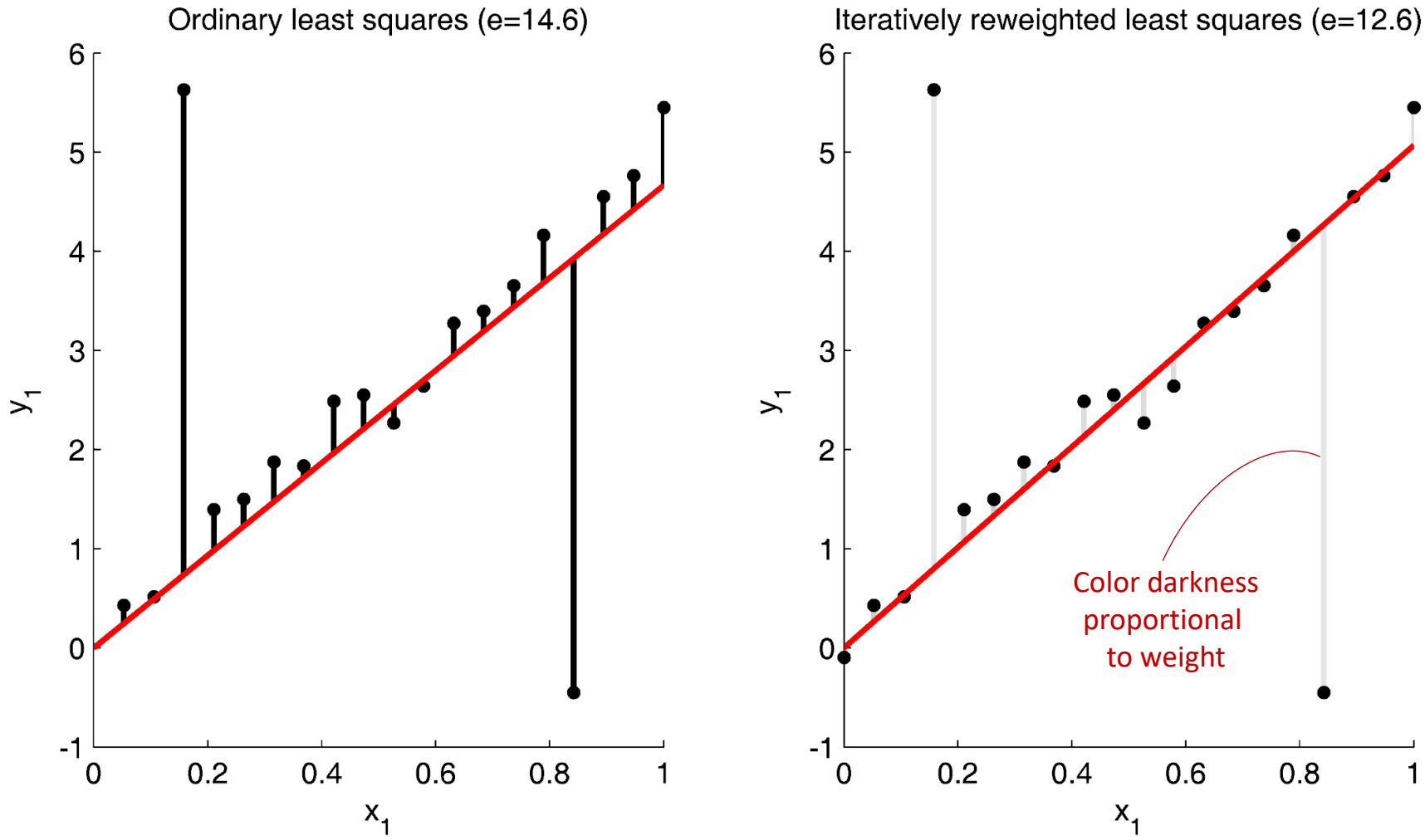
$\hat{\boldsymbol{A}}$ is estimated by starting from $\boldsymbol{W} = \boldsymbol{I}$ and iteratively computing

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

$$\boldsymbol{W}_{n,n} \leftarrow |\boldsymbol{Y}_n - \boldsymbol{X}_n \boldsymbol{A}|^{p-2} \quad \forall n \in \{1, \ldots, N\}$$



Iteratively reweighted least squares (IRLS)



→ regression that can be more 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:

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

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.

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

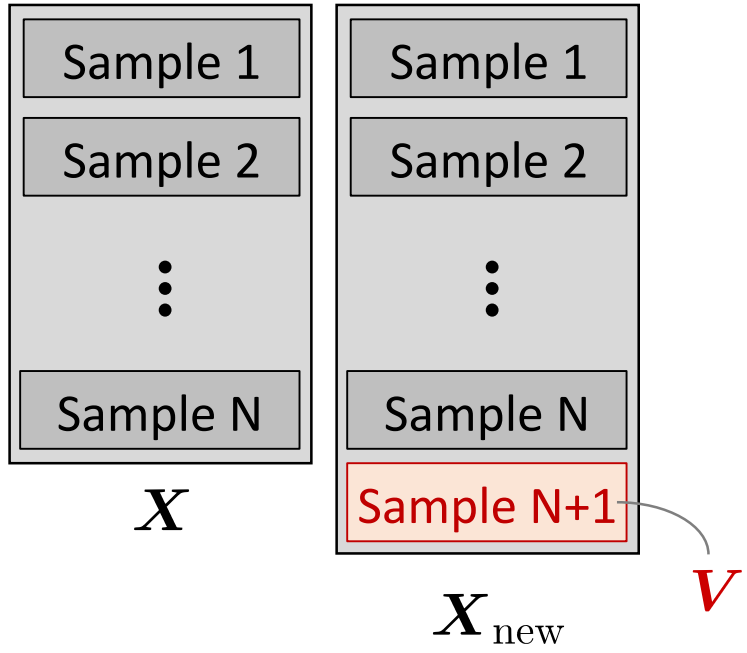
$$\boldsymbol{X}\in\mathbb{R}^{N\times D^{\mathcal{I}}}$$

$$\boldsymbol{X}_{\text{new}}\in\mathbb{R}^{N_{\text{new}}\times D^{\mathcal{I}}}$$

Recursive least squares

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

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

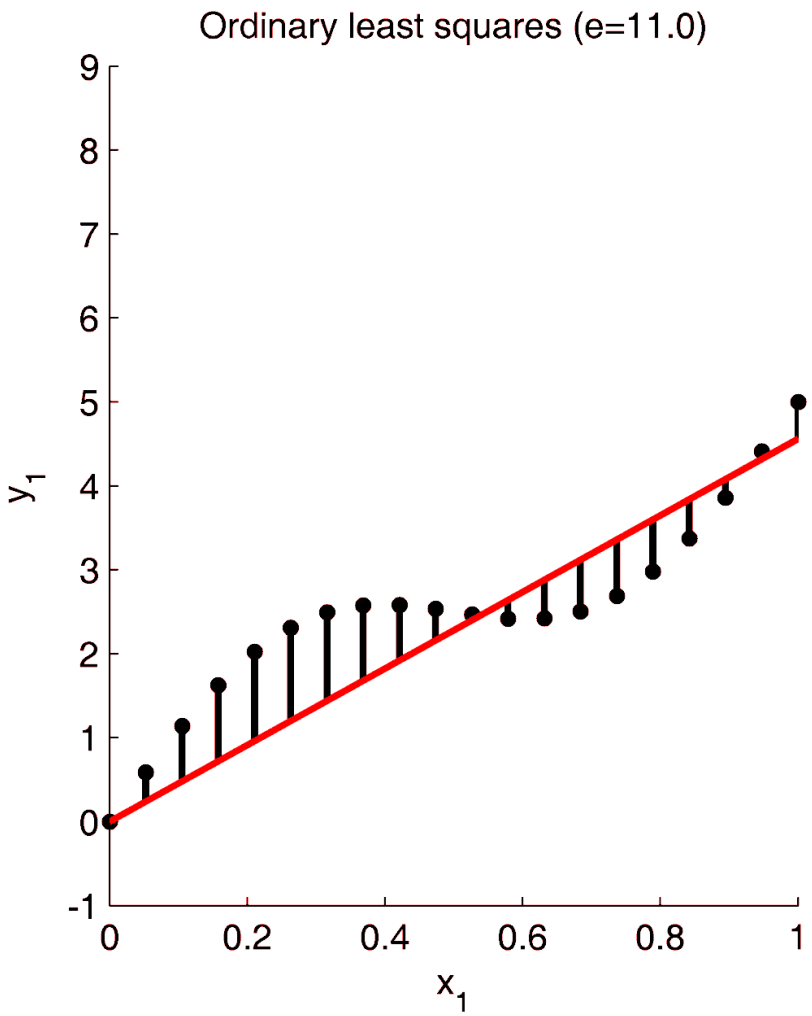
which is exploited to efficiently compute the lease squares update as

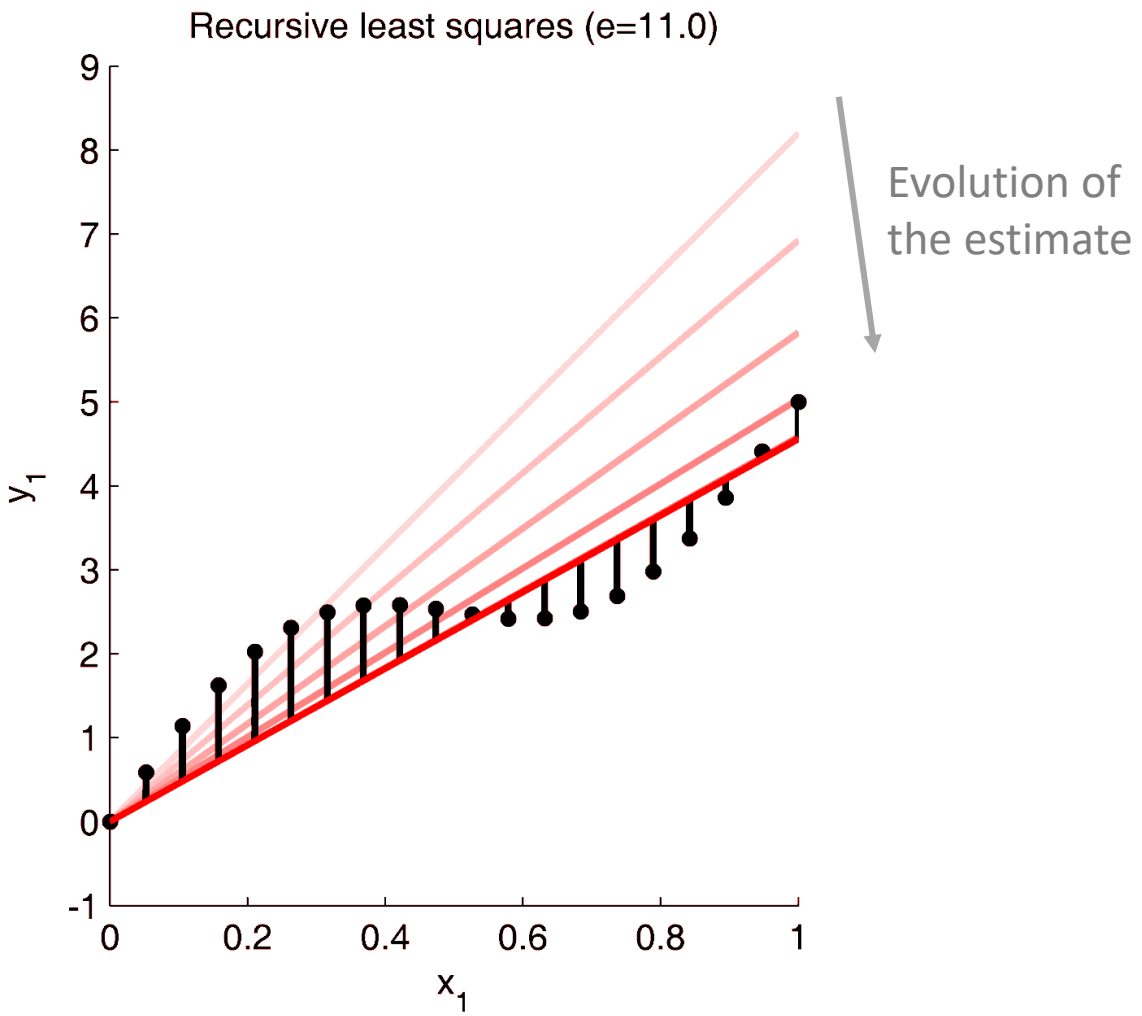
$$\hat{\boldsymbol{A}}_{\text{new}} = \hat{\boldsymbol{A}} + \boldsymbol{K}\left(\boldsymbol{C} - \boldsymbol{V}\hat{\boldsymbol{A}}\right)$$

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

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

Recursive least squares





→ the least squares estimate is the same in the two cases

Linear regression:

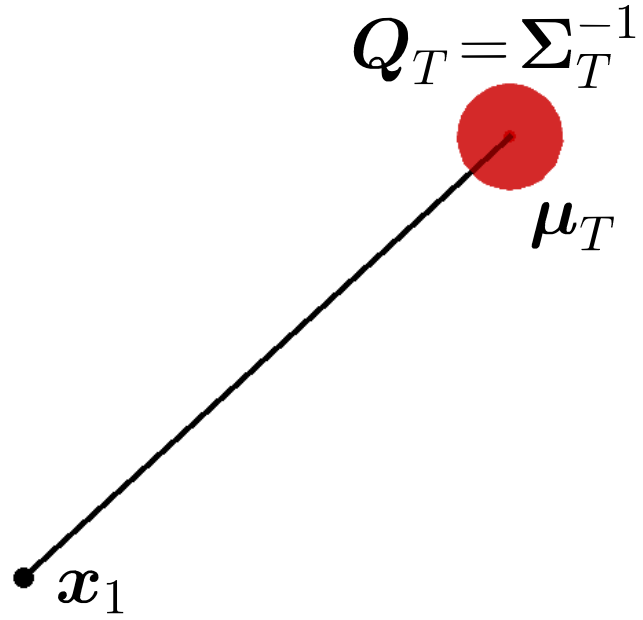
Example of application

Linear quadratic tracking (LQT)

Track path! Use low control commands!

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

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



$\boldsymbol{x}_t$ state variable (position+velocity)

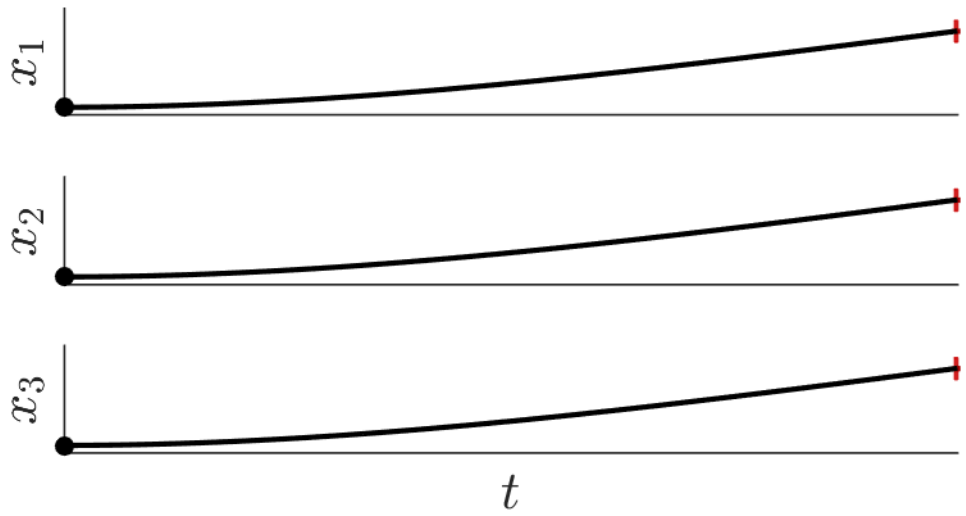
$\boldsymbol{\mu}_t$ desired state

$\boldsymbol{u}_t$ control command (acceleration)

$\boldsymbol{Q}_t$ precision matrix

$\boldsymbol{R}_t$ control weight matrix

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



Linear quadratic tracking (LQT)

Track path! Use low control commands!

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

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

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)

Linear quadratic tracking (LQT)

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

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

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

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

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

Linear quadratic tracking (LQT)

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

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

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

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

Linear quadratic tracking (LQT)

The constraint can then be inserted in the cost function:

$$x = S^x x_1 + S^u u$$

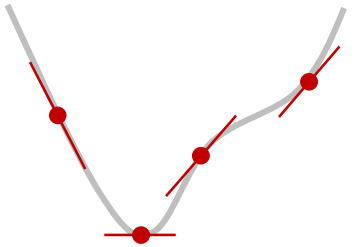
$$c = (\mu - x)^\top Q(\mu - x) + u^\top R u$$

$$= (\mu - S^x x_1 - S^u u)^\top Q (\mu - S^x x_1 - S^u u) + u^\top R u$$

Solving for *u* is similar to a weighted ridge regression problem, and results in the analytic solution:

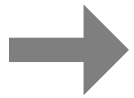
$$\hat{u} = (S^{u\top} Q S^u + R)^{-1} S^{u\top} Q (\mu - S^x x_1)$$

$$\hat{u} = \begin{bmatrix} \hat{u}_1 \\ \hat{u}_2 \\ \vdots \\ \hat{u}_T \end{bmatrix}$$



Linear quadratic tracking (LQT)

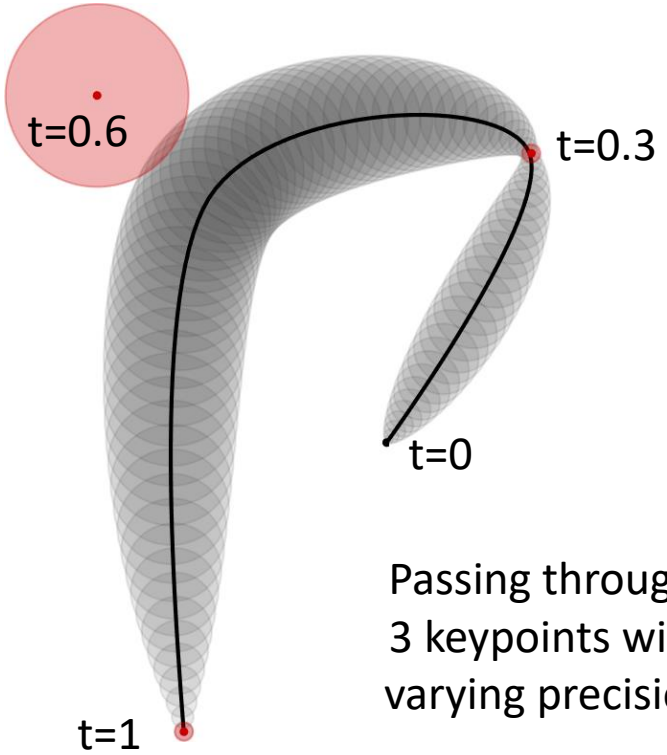
$$\hat{\boldsymbol{u}} = \left(\boldsymbol{S}^{\boldsymbol{u}\top}\boldsymbol{Q}\boldsymbol{S}^{\boldsymbol{u}} + \boldsymbol{R}\right)^{-1}\boldsymbol{S}^{\boldsymbol{u}\top}\boldsymbol{Q}\left(\boldsymbol{\mu} - \boldsymbol{S}^{\boldsymbol{x}}\boldsymbol{x}_1\right)$$
$$\hat{\boldsymbol{\Sigma}}^{\boldsymbol{u}}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



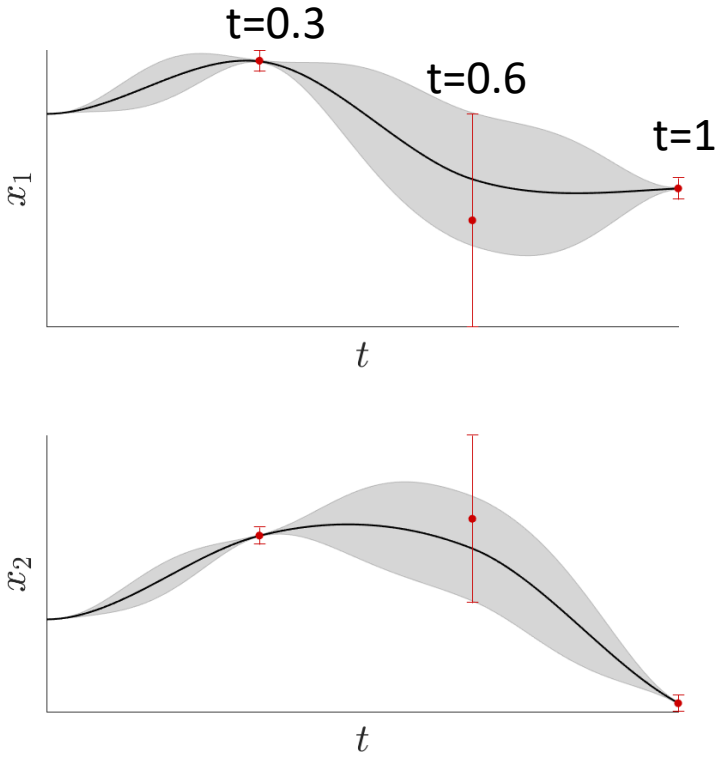
$$\hat{\boldsymbol{x}} = \boldsymbol{S}^{\boldsymbol{x}}\boldsymbol{x}_1 + \boldsymbol{S}^{\boldsymbol{u}}\hat{\boldsymbol{u}}$$
$$\hat{\boldsymbol{\Sigma}}^{\boldsymbol{x}} = \boldsymbol{S}^{\boldsymbol{u}}\left(\boldsymbol{S}^{\boldsymbol{u}\top}\boldsymbol{Q}\boldsymbol{S}^{\boldsymbol{u}} + \boldsymbol{R}\right)^{-1}\boldsymbol{S}^{\boldsymbol{u}\top}$$

The error on the regression estimate can be used to form a distribution

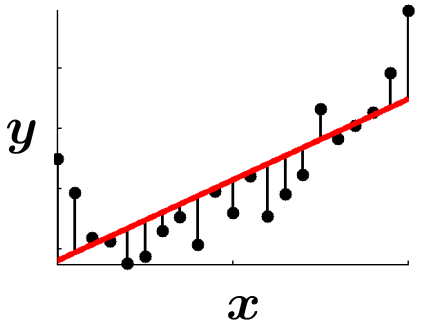
The distribution in control space can then be projected back to the state space



Passing through
3 keypoints with
varying precision



Linear quadratic tracking (LQT) - Example



For $t \leq \frac{T}{4}$ (left hand holding the ball), we have

$$\underbrace{\begin{bmatrix} \dot{\mathbf{x}}_{1,t} \\ \ddot{\mathbf{x}}_{1,t} \\ \dot{\boldsymbol{f}}_{1,t} \\ \hline \dot{\mathbf{x}}_{2,t} \\ \ddot{\mathbf{x}}_{2,t} \\ \dot{\boldsymbol{f}}_{2,t} \\ \hline \dot{\mathbf{x}}_{3,t} \\ \ddot{\mathbf{x}}_{3,t} \\ \dot{\boldsymbol{f}}_{3,t} \end{bmatrix}}_{\dot{\boldsymbol{x}}_t} = \underbrace{\left[\begin{array}{ccccccccc} \mathbf{0} & \boldsymbol{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \boldsymbol{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{I} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{array}\right]}_{\boldsymbol{A}_t^c} \underbrace{\begin{bmatrix} \mathbf{x}_{1,t} \\ \dot{\mathbf{x}}_{1,t} \\ \boldsymbol{f}_{1,t} \\ \hline \mathbf{x}_{2,t} \\ \dot{\mathbf{x}}_{2,t} \\ \boldsymbol{f}_{2,t} \\ \hline \mathbf{x}_{3,t} \\ \dot{\mathbf{x}}_{3,t} \\ \boldsymbol{f}_{3,t} \end{bmatrix}}_{\boldsymbol{x}_t} + \underbrace{\left[\begin{array}{cc} \mathbf{0} & \mathbf{0} \\ \boldsymbol{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{I} \\ \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{array}\right]}_{\boldsymbol{B}_t^c} \underbrace{\left[\begin{array}{c} \boldsymbol{u}_{1,t} \\ \hline \boldsymbol{u}_{2,t} \end{array}\right]}_{\boldsymbol{u}_t}$$

$$\boldsymbol{f}_{i,t} = m_i \boldsymbol{g} \quad \text{with} \quad \boldsymbol{g} = \begin{bmatrix} 0 \\ -9.81 \end{bmatrix}$$

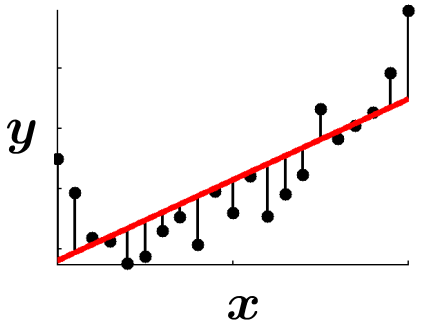
At $t = \frac{T}{2}$ (right hand hitting the ball), we have

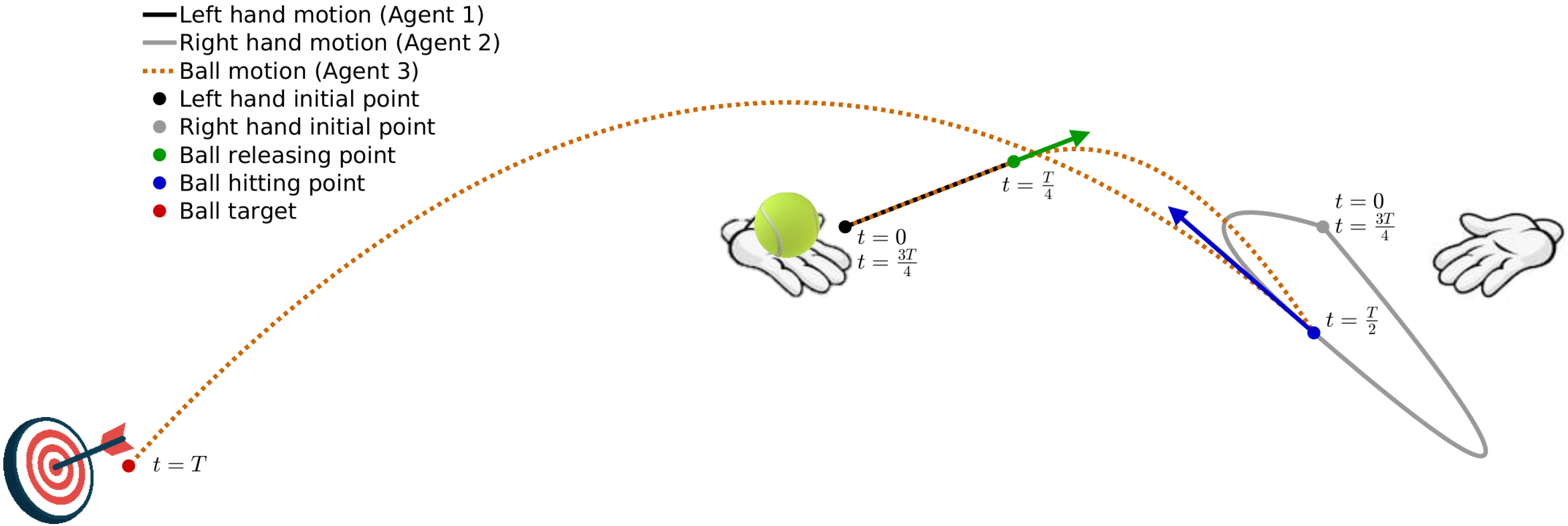
$$\underbrace{\begin{bmatrix} \dot{\mathbf{x}}_{1,t} \\ \ddot{\mathbf{x}}_{1,t} \\ \dot{\boldsymbol{f}}_{1,t} \\ \hline \dot{\mathbf{x}}_{2,t} \\ \ddot{\mathbf{x}}_{2,t} \\ \dot{\boldsymbol{f}}_{2,t} \\ \hline \dot{\mathbf{x}}_{3,t} \\ \ddot{\mathbf{x}}_{3,t} \\ \dot{\boldsymbol{f}}_{3,t} \end{bmatrix}}_{\dot{\boldsymbol{x}}_t} = \underbrace{\left[\begin{array}{ccccccccc} \mathbf{0} & \boldsymbol{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{I} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{array}\right]}_{\boldsymbol{A}_t^c} \underbrace{\begin{bmatrix} \mathbf{x}_{1,t} \\ \dot{\mathbf{x}}_{1,t} \\ \boldsymbol{f}_{1,t} \\ \hline \mathbf{x}_{2,t} \\ \dot{\mathbf{x}}_{2,t} \\ \boldsymbol{f}_{2,t} \\ \hline \mathbf{x}_{3,t} \\ \dot{\mathbf{x}}_{3,t} \\ \boldsymbol{f}_{3,t} \end{bmatrix}}_{\boldsymbol{x}_t} + \underbrace{\left[\begin{array}{cc} \mathbf{0} & \mathbf{0} \\ \boldsymbol{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{I} \\ \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{array}\right]}_{\boldsymbol{B}_t^c} \underbrace{\left[\begin{array}{c} \boldsymbol{u}_{1,t} \\ \hline \boldsymbol{u}_{2,t} \end{array}\right]}_{\boldsymbol{u}_t}$$

For $\frac{T}{4} < t < \frac{T}{2}$ and $t > \frac{T}{2}$ (free motion of the ball), we have

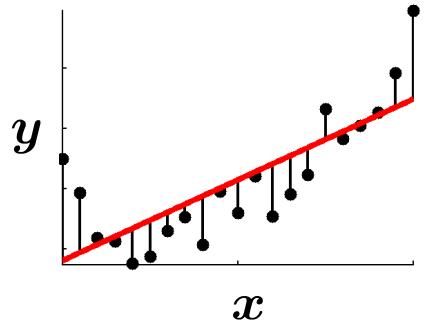
$$\underbrace{\begin{bmatrix} \dot{\mathbf{x}}_{1,t} \\ \ddot{\mathbf{x}}_{1,t} \\ \dot{\boldsymbol{f}}_{1,t} \\ \hline \dot{\mathbf{x}}_{2,t} \\ \ddot{\mathbf{x}}_{2,t} \\ \dot{\boldsymbol{f}}_{2,t} \\ \hline \dot{\mathbf{x}}_{3,t} \\ \ddot{\mathbf{x}}_{3,t} \\ \dot{\boldsymbol{f}}_{3,t} \end{bmatrix}}_{\dot{\boldsymbol{x}}_t} = \underbrace{\left[\begin{array}{ccccccccc} \mathbf{0} & \boldsymbol{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{I} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{array}\right]}_{\boldsymbol{A}_t^c} \underbrace{\begin{bmatrix} \mathbf{x}_{1,t} \\ \dot{\mathbf{x}}_{1,t} \\ \boldsymbol{f}_{1,t} \\ \hline \mathbf{x}_{2,t} \\ \dot{\mathbf{x}}_{2,t} \\ \boldsymbol{f}_{2,t} \\ \hline \mathbf{x}_{3,t} \\ \dot{\mathbf{x}}_{3,t} \\ \boldsymbol{f}_{3,t} \end{bmatrix}}_{\boldsymbol{x}_t} + \underbrace{\left[\begin{array}{cc} \mathbf{0} & \mathbf{0} \\ \boldsymbol{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{I} \\ \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{array}\right]}_{\boldsymbol{B}_t^c} \underbrace{\left[\begin{array}{c} \boldsymbol{u}_{1,t} \\ \hline \boldsymbol{u}_{2,t} \end{array}\right]}_{\boldsymbol{u}_t}$$

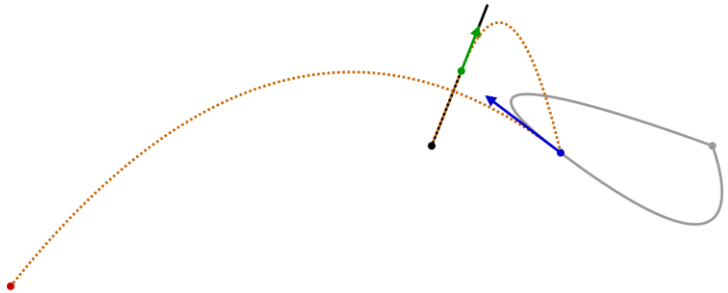
Linear quadratic tracking (LQT) - Example

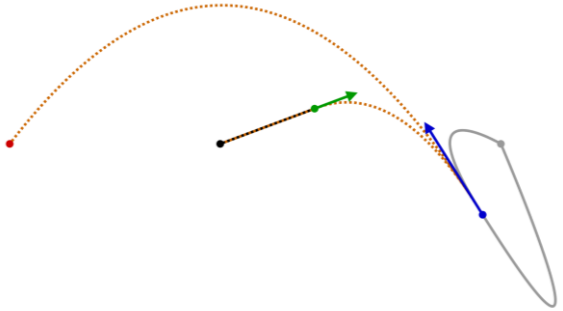


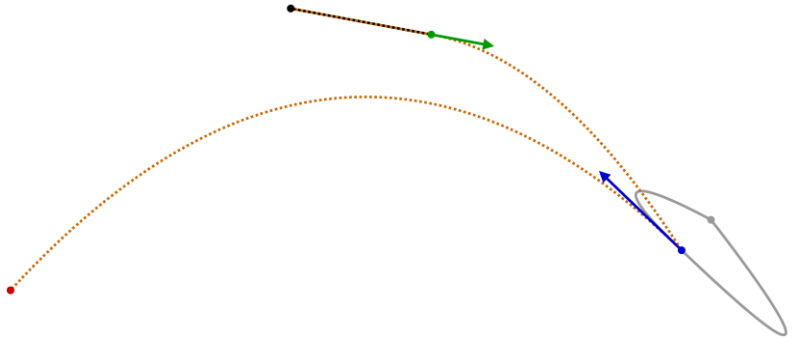


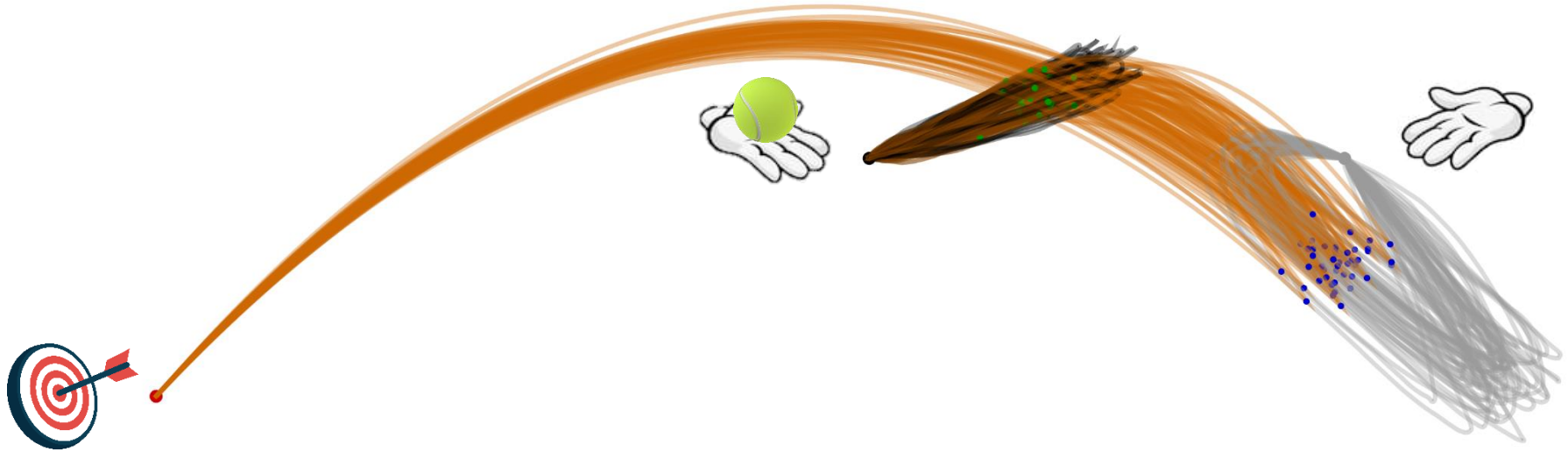
Linear quadratic tracking (LQT) - Example











Logistic regression

Python notebook:
demo_LS_IRLS_logRegr.ipynb

Matlab code:
demo_LS_IRLS_logRegr01.m

Costs functions and associated regression solutions

Univariate output y:

$$\hat{\boldsymbol{a}} = \arg\min_{\boldsymbol{a}} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{a}\|^2 \quad = \quad \overbrace{(\boldsymbol{X}^\top \boldsymbol{X})^{-1}\boldsymbol{X}^\top}^{\boldsymbol{X}^\dagger}\, \boldsymbol{y}$$

$$\hat{\boldsymbol{a}} = \arg\min_{\boldsymbol{a}} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{a}\|^2_{\mathrm{F},\boldsymbol{W}} \quad = \quad (\boldsymbol{X}^\top \boldsymbol{W}\boldsymbol{X})^{-1}\boldsymbol{X}^\top \boldsymbol{W}\, \boldsymbol{y}$$

$$\hat{\boldsymbol{a}} = \arg\min_{\boldsymbol{a}} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{a}\|^2_{\mathrm{F}} + \|\boldsymbol{\Gamma}\boldsymbol{a}\|^2_{\mathrm{F}} \quad = \quad (\boldsymbol{X}^\top \boldsymbol{X} + \boldsymbol{\Gamma}^\top\boldsymbol{\Gamma})^{-1}\boldsymbol{X}^\top \boldsymbol{y}$$

$$\hat{\boldsymbol{a}} = \arg\min_{\boldsymbol{a}} f_{\boldsymbol{a}}(\boldsymbol{X}, \boldsymbol{y})$$

Multivariate output y:

$$\hat{\boldsymbol{A}} = \arg\min_{\boldsymbol{A}} \|\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{A}\|^2 \quad = \quad \overbrace{(\boldsymbol{X}^\top \boldsymbol{X})^{-1}\boldsymbol{X}^\top}^{\boldsymbol{X}^\dagger}\, \boldsymbol{Y}$$

$$\hat{\boldsymbol{A}} = \arg\min_{\boldsymbol{A}} \|\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{A}\|^2_{\mathrm{F},\boldsymbol{W}} \quad = \quad (\boldsymbol{X}^\top \boldsymbol{W}\boldsymbol{X})^{-1}\boldsymbol{X}^\top \boldsymbol{W}\, \boldsymbol{Y}$$

$$\hat{\boldsymbol{A}} = \arg\min_{\boldsymbol{A}} \|\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{A}\|^2_{\mathrm{F}} + \|\boldsymbol{\Gamma}\boldsymbol{A}\|^2_{\mathrm{F}} \quad = \quad (\boldsymbol{X}^\top \boldsymbol{X} + \boldsymbol{\Gamma}^\top\boldsymbol{\Gamma})^{-1}\boldsymbol{X}^\top \boldsymbol{Y}$$

Logistic regression

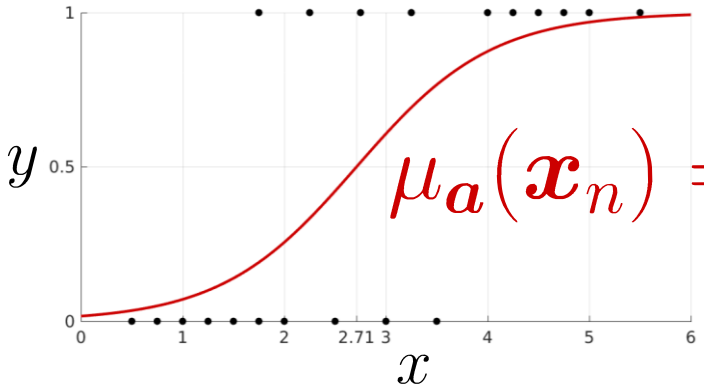
Bernoulli distribution *(for binary variables)*:

$$\mathcal{L}_n = \begin{cases} \overbrace{p}^{\mathcal{P}(y_n=1)} & \text{if } y_n = 1, \\ \underbrace{1-p}_{\mathcal{P}(y_n=0)} & \text{if } y_n = 0. \end{cases}$$

$$= p^{y_n} \, (1-p)^{(1-y_n)}$$

Logistic function:

$$y_n \in \{0, 1\}$$



$$\mu_{\boldsymbol{a}}(\boldsymbol{x}_n) = \frac{1}{1+e^{-\boldsymbol{a}^\intercal \boldsymbol{x}_n}}$$

Likelihood of *n*th datapoint:

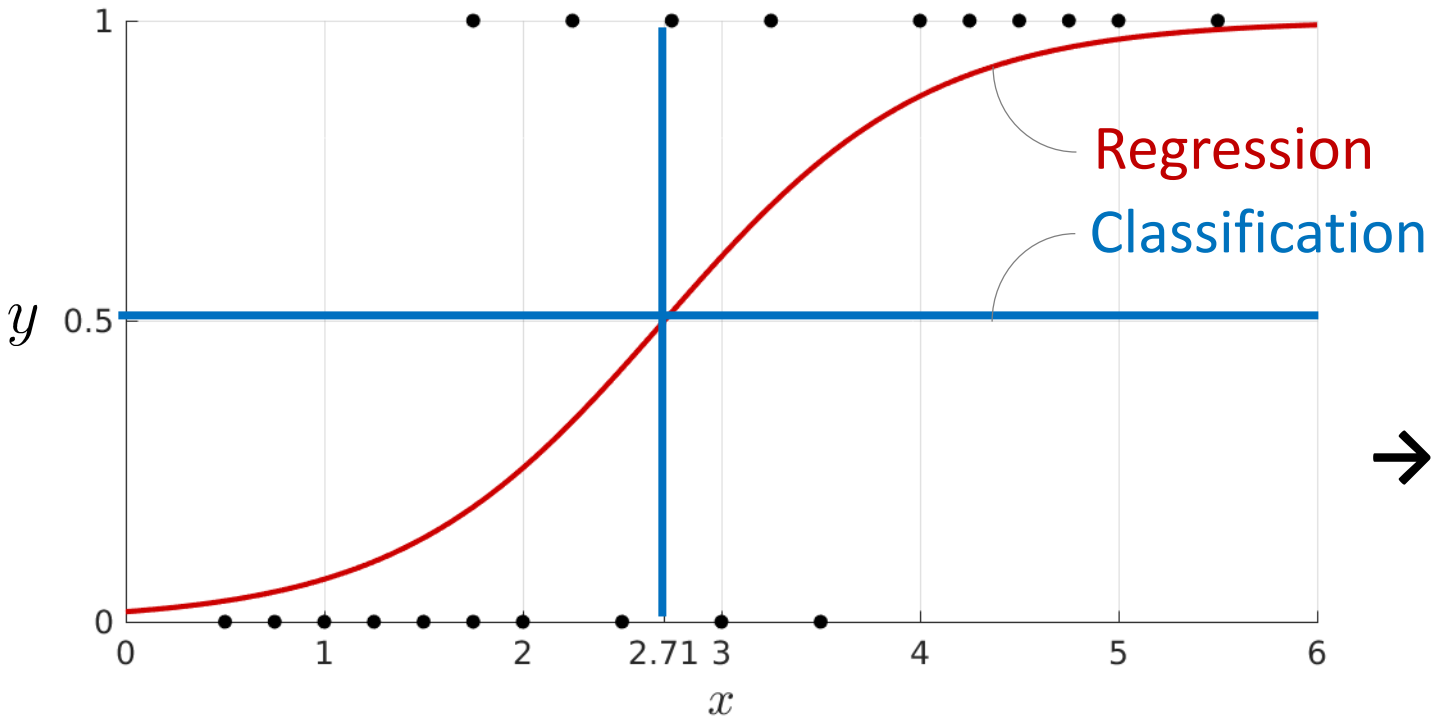
$$\mathcal{L}_n = \mu_{\boldsymbol{a}}(\boldsymbol{x}_n)^{y_n} \big(1 - \mu_{\boldsymbol{a}}(\boldsymbol{x}_n)\big)^{(1-y_n)}$$

Likelihood of N datapoints *(independence assumption)*:

$$\mathcal{L} = \prod_n \mu_{\boldsymbol{a}}(\boldsymbol{x}_n)^{y_n} \big(1 - \mu_{\boldsymbol{a}}(\boldsymbol{x}_n)\big)^{(1-y_n)}$$

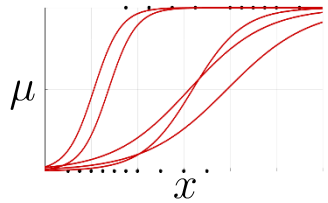
Logistic regression

Pass/fail in function of the time spent to study at an exam:



→ **Regression exploited for classification problem**

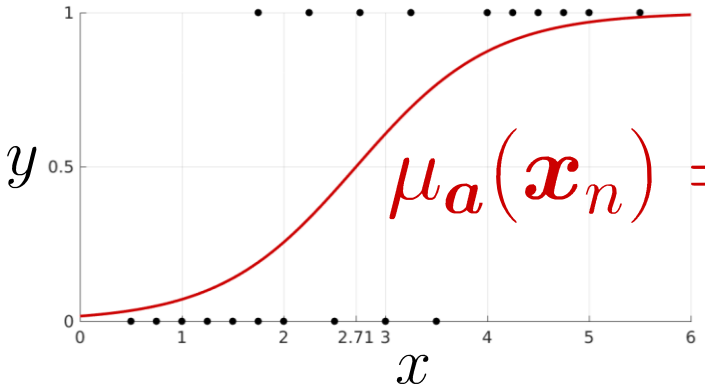
Logistic function:



$$\mu_{\boldsymbol{a}}(\boldsymbol{x}) = \frac{1}{1+e^{-\boldsymbol{a}^\top \boldsymbol{x}}} \qquad \mu_{\boldsymbol{a}}(x) = \frac{1}{1+e^{-(a_1+a_2 x)}}$$

Logistic regression

$y_n \in \{0, 1\}$



$$\mu_{\boldsymbol{a}}(\boldsymbol{x}_n) = \frac{1}{1+e^{-\boldsymbol{a}^\top \boldsymbol{x}_n}}$$

Likelihood of N datapoints:

$$\mathcal{L} = \prod_n \mu_{\boldsymbol{a}}(\boldsymbol{x}_n)^{y_n} \left(1 - \mu_{\boldsymbol{a}}(\boldsymbol{x}_n)\right)^{(1-y_n)}$$

Cost function as negative log-likelihood:

$\log(a^b) = b\log(a)$ — $\log(ab) = \log(a) + \log(b)$

$$c = -\sum_n y_n \log\left(\mu_{\boldsymbol{a}}(\boldsymbol{x}_n)\right) + (1 - y_n) \log\left(1 - \mu_{\boldsymbol{a}}(\boldsymbol{x}_n)\right)$$

$\frac{\partial}{\partial x}\log(x) = \frac{1}{x}$

$$\frac{\partial c}{\partial \boldsymbol{a}} = -\sum_n y_n \, \mu_{\boldsymbol{a}}^{-1} \mu_{\boldsymbol{a}} \left(1 - \mu_{\boldsymbol{a}}\right) \boldsymbol{x}_n - (1 - y_n)(1 - \mu_{\boldsymbol{a}})^{-1} \mu_{\boldsymbol{a}} \left(1 - \mu_{\boldsymbol{a}}\right) \boldsymbol{x}_n$$

$$= -\sum_n y_n \left(1 - \mu_{\boldsymbol{a}}\right) \boldsymbol{x}_n - \left(1 - y_n\right) \mu_{\boldsymbol{a}} \, \boldsymbol{x}_n$$

$$= \sum_n (\mu_{\boldsymbol{a}} - y_n) \, \boldsymbol{x}_n$$

$$\mu(x) = \frac{1}{1+e^{-x}}$$

$$\frac{\partial \mu}{\partial x} = \mu(1 - \mu)$$

Logistic regression

$$\frac{\partial c}{\partial \boldsymbol{a}} = \sum_n (\mu_{\boldsymbol{a}} - y_n)\boldsymbol{x}_n$$

$$\mu_{\boldsymbol{a}}(\boldsymbol{x}_n) = \frac{1}{1+e^{-\boldsymbol{a}^\top \boldsymbol{x}_n}}$$

It can for example be solved with Newton's method, by iterating

$$\boldsymbol{a} \leftarrow \boldsymbol{a} - \boldsymbol{H}^{-1}\boldsymbol{g},$$

with gradient $\boldsymbol{g} = \sum_n (\mu_{\boldsymbol{a}}(\boldsymbol{x}_n) - y_n)\boldsymbol{x}_n = \boldsymbol{X}^\top(\boldsymbol{\mu}_{\boldsymbol{a}} - \boldsymbol{y})$ and Hessian $\boldsymbol{H} = \boldsymbol{X}^\top \boldsymbol{W} \boldsymbol{X}$,
with diagonal matrix $\boldsymbol{W} = \mathrm{diag}\big(\boldsymbol{\mu}_{\boldsymbol{a}} * (\boldsymbol{1} - \boldsymbol{\mu}_{\boldsymbol{a}})\big)$.

$*$ Hadamard (elementwise) product

We then obtain

$$\begin{aligned}
\boldsymbol{a} &\leftarrow \boldsymbol{a} - \boldsymbol{H}^{-1}\boldsymbol{g} \\
&\leftarrow \boldsymbol{a} - (\boldsymbol{X}^\top \boldsymbol{W} \boldsymbol{X})^{-1}\boldsymbol{X}^\top(\boldsymbol{\mu}_{\boldsymbol{a}} - \boldsymbol{y}) \\
&\leftarrow (\boldsymbol{X}^\top \boldsymbol{W} \boldsymbol{X})^{-1}\big(\boldsymbol{X}^\top \boldsymbol{W} \boldsymbol{X}\boldsymbol{a} - \boldsymbol{X}^\top(\boldsymbol{\mu}_{\boldsymbol{a}} - \boldsymbol{y})\big) \\
&\leftarrow (\boldsymbol{X}^\top \boldsymbol{W} \boldsymbol{X})^{-1}\boldsymbol{X}^\top(\boldsymbol{W}\boldsymbol{X}\boldsymbol{a} + \boldsymbol{y} - \boldsymbol{\mu}_{\boldsymbol{a}}) \\
&\leftarrow (\boldsymbol{X}^\top \boldsymbol{W} \boldsymbol{X})^{-1}\boldsymbol{X}^\top \boldsymbol{W}\boldsymbol{z},
\end{aligned}$$

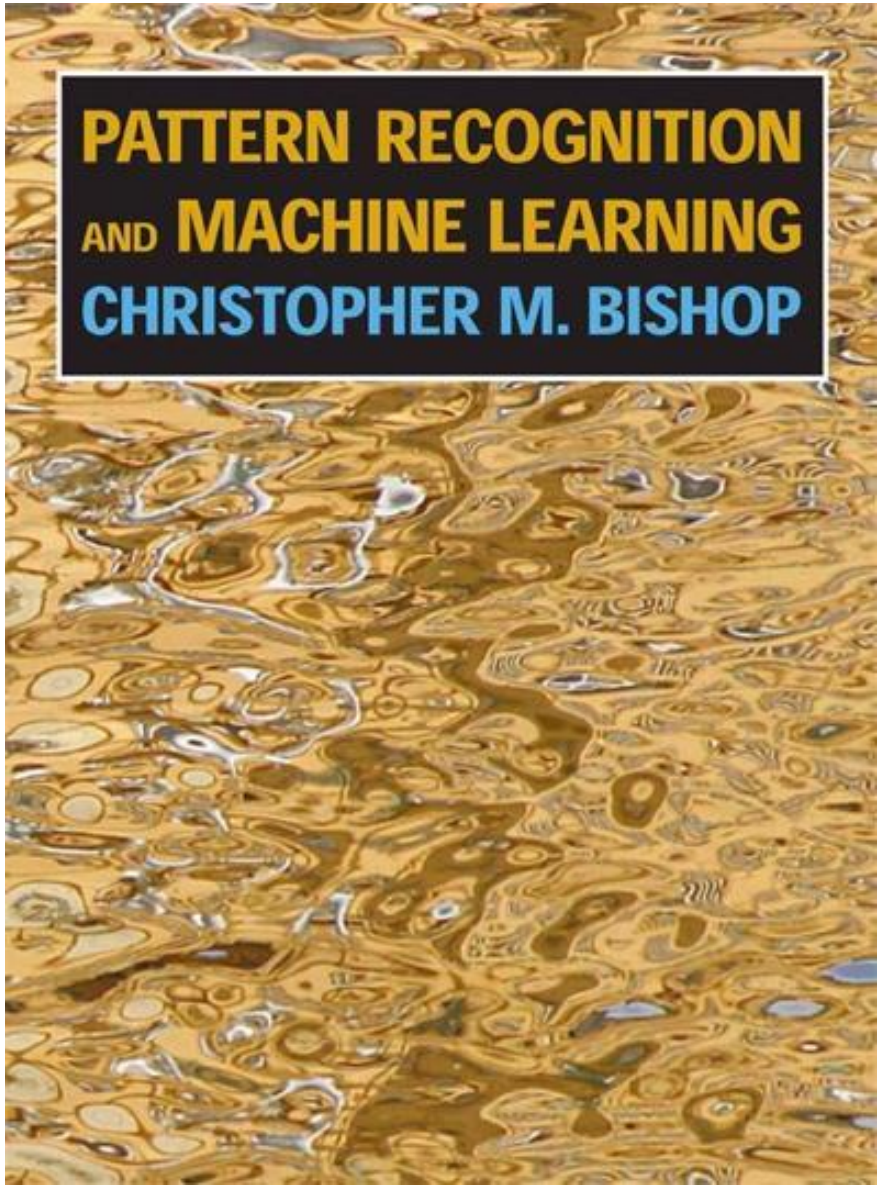
with *working response* $\boldsymbol{z} = \boldsymbol{X}\boldsymbol{a} + \boldsymbol{W}^{-1}(\boldsymbol{y} - \boldsymbol{\mu}_{\boldsymbol{a}})$.

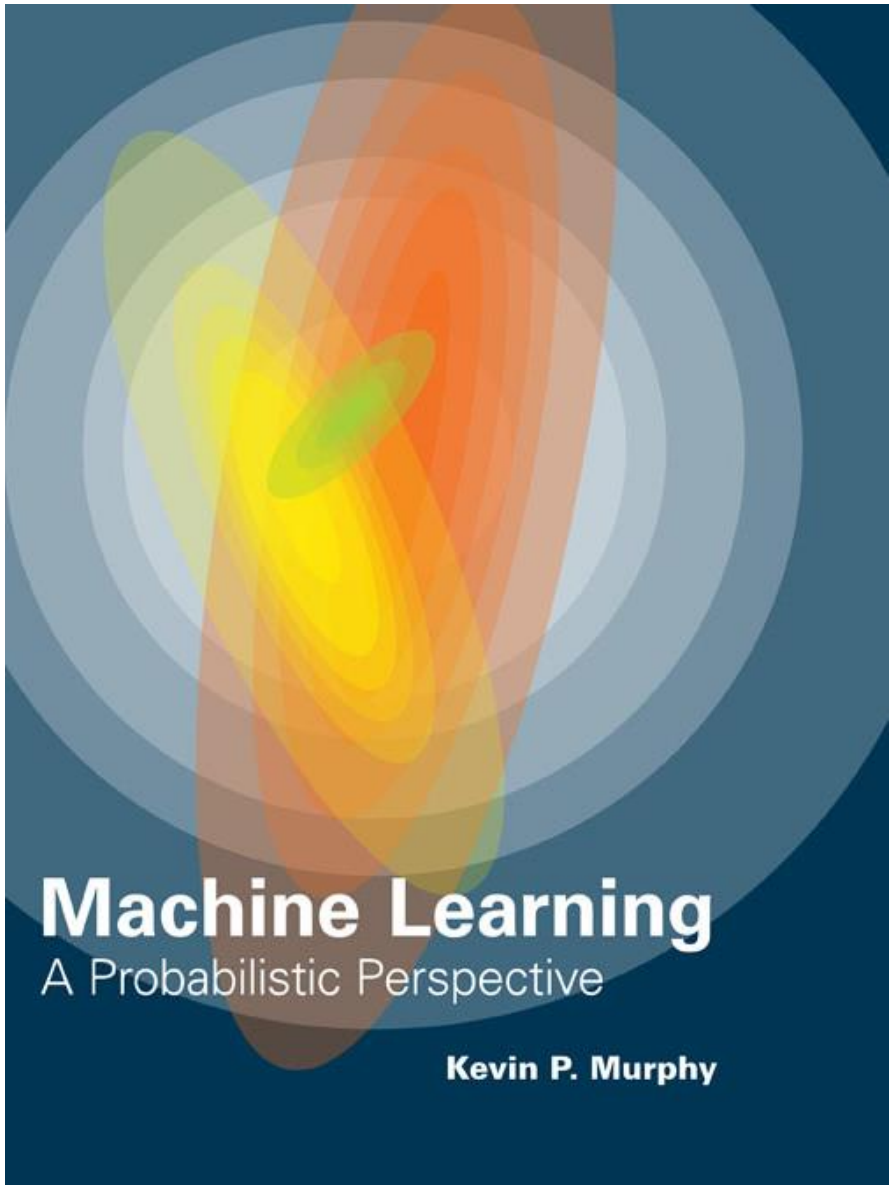
$$\mu(x) = \frac{1}{1+e^{-x}}$$

$$\frac{\partial \mu}{\partial x} = \mu(1-\mu)$$

→ IRLS algorithm

Recommended reading material





The Matrix Cookbook

Kaare Brandt Petersen
Michael Syskind Pedersen

Labs this afternoon

https://moodle.epfl.ch/course/view.php?id=16819

INM11, 14:15-16:00

Dr João Silvério