EE613
Machine Learning for Engineers

LINEAR REGRESSION

Sylvain Calinon
Robot Learning & Interaction Group
Idiap Research Institute
Nov. 9, 2017

Outline

- Multivariate ordinary least squares
  Matlab code: *demo_LS01.m, demo_LS_polFit01.m*
- Singular value decomposition (SVD) and Cholesky decomposition
  Matlab code: *demo_LS_polFit_nullspace01.m*
- Kernels in least squares (nullspace projection)
  Matlab code: *demo_LS_polFit_nullspace01.m*
- Ridge regression (Tikhonov regularization)
  Matlab code: *demo_LS_polFit02.m*
- Weighted least squares (WLS)
  Matlab code: *demo_LS_weighted01.m*
- Iteratively reweighted least squares (IRLS)
  Matlab code: *demo_LS_IRLS01.m*
- Recursive least squares (RLS)
  Matlab code: *demo_LS_recursive01.m*

Multivariate ordinary least squares

Matlab codes:
demo_LS01.m
demo_LS_polFit01.m

Multivariate 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 (a phenomenon also known as regression toward the mean).
- **Pearson** later provided the statistical context showing that the phenomenon is more general than a biological context.

Multivariate ordinary least squares

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

A solution can be found by minimizing the L_2-norm of the residuals

$$\begin{aligned}
\hat{\boldsymbol{A}} &= \arg\min_{\boldsymbol{A}} \|\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{A}\|_2 \\
&= \arg\min_{\boldsymbol{A}} (\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{A})^\top (\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{A}) \\
&= \arg\min_{\boldsymbol{A}} (\boldsymbol{Y}^\top \boldsymbol{Y} - 2\boldsymbol{A}^\top \boldsymbol{X}^\top \boldsymbol{Y} + \boldsymbol{A}^\top \boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{A})
\end{aligned}$$

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

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

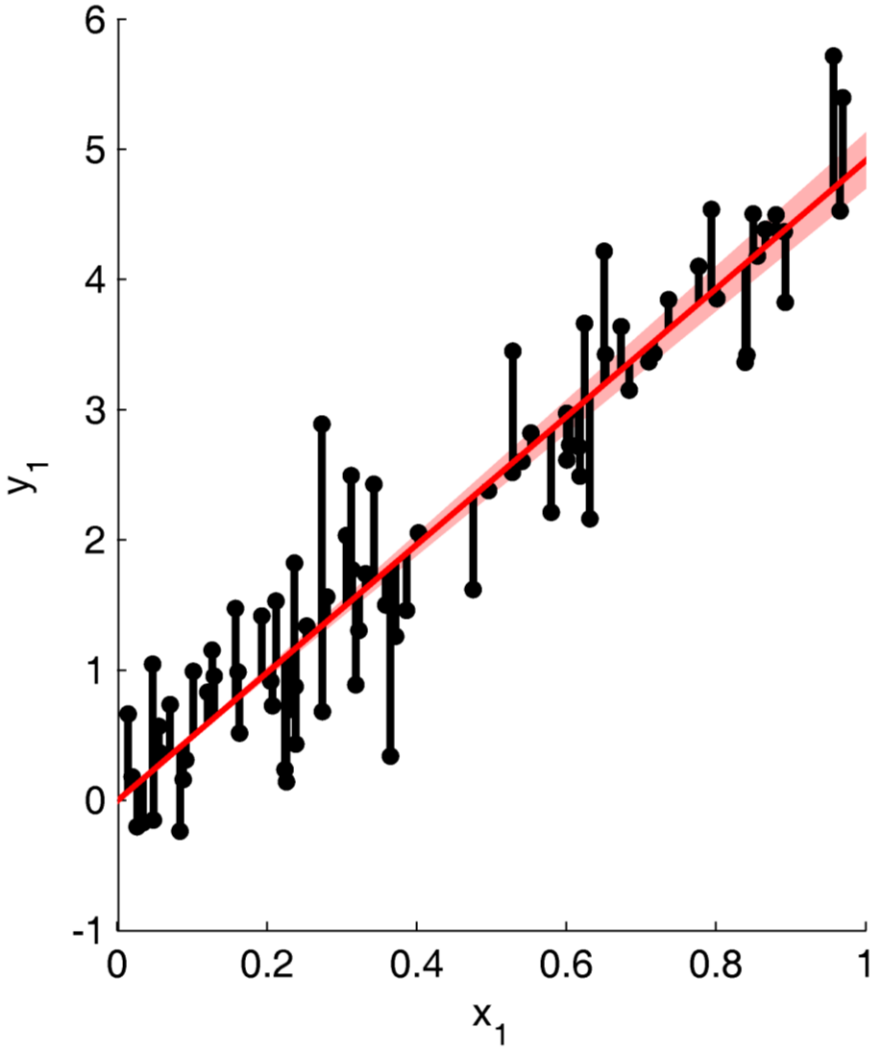
$$-2\boldsymbol{X}^\top \boldsymbol{Y} + 2\boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{A} = \boldsymbol{0} \quad \Longleftrightarrow \quad \hat{\boldsymbol{A}} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{Y}$$

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

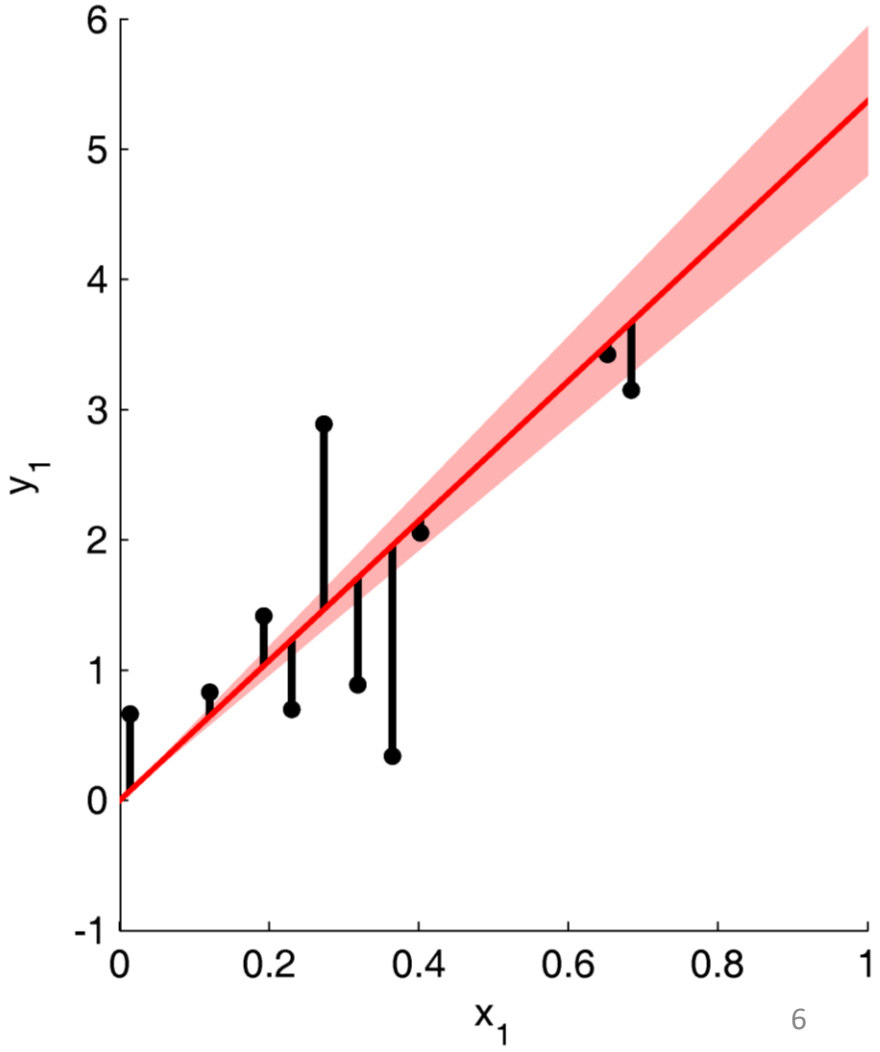
with residuals (parameters errors) given by $\hat{\boldsymbol{\Sigma}}^{\boldsymbol{A}} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1}$

Multivariate ordinary least squares

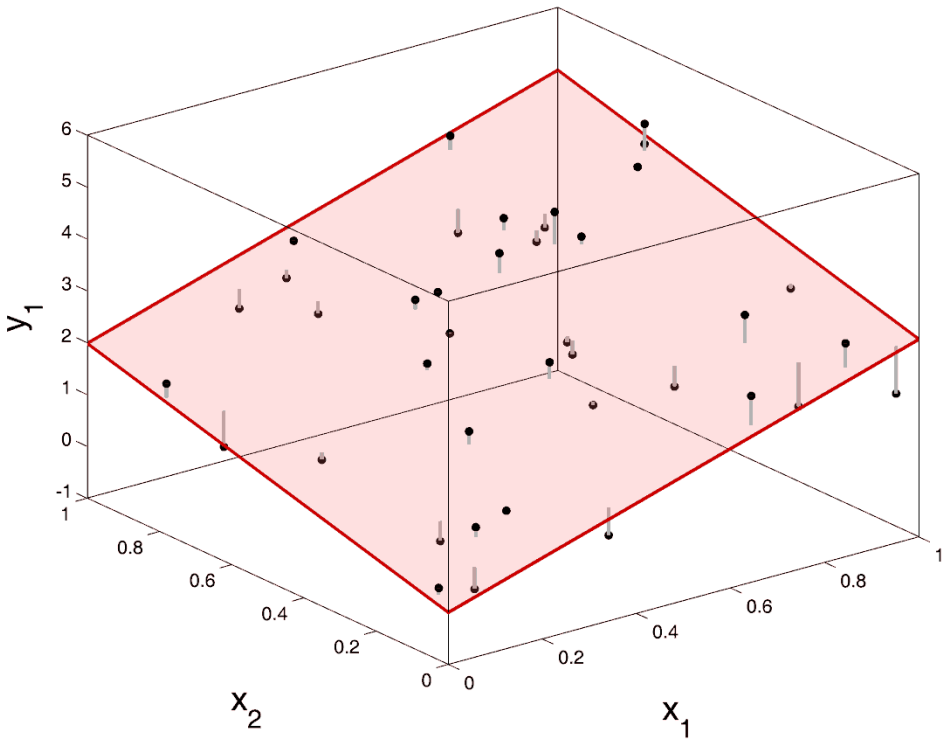
$D^{\mathcal{I}} = 1,\ D^{\mathcal{O}} = 1,\ N = 80$



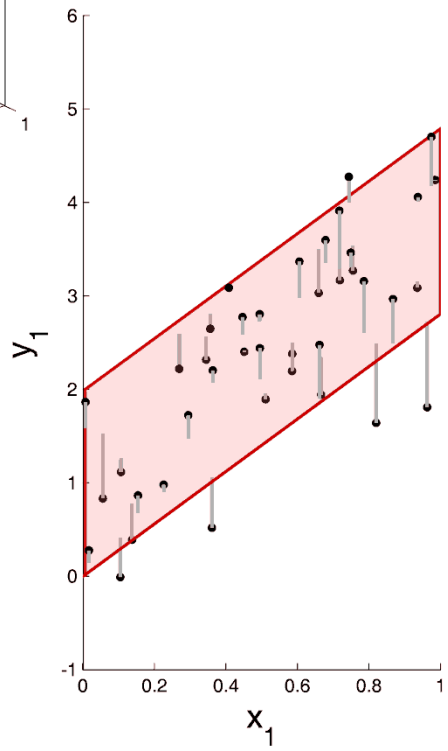
$D^{\mathcal{I}} = 1,\ D^{\mathcal{O}} = 1,\ N = 10$

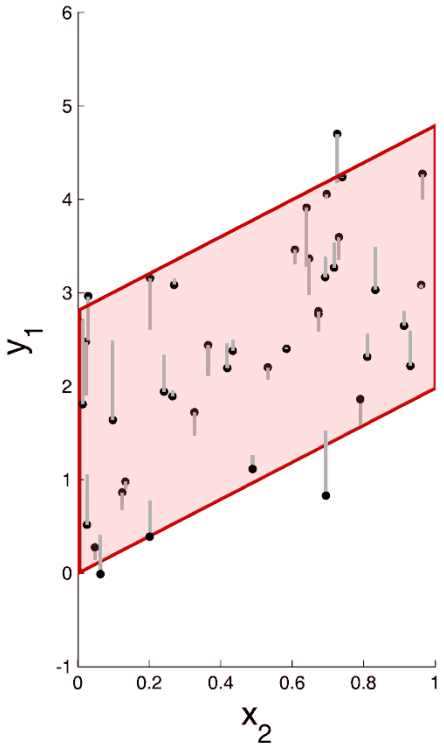


Multivariate ordinary least squares



$$D^{\mathcal{I}} = 2,\ D^{\mathcal{O}} = 1,\ N = 40$$





Multivariate ordinary least squares

$$\hat{\boldsymbol{A}} = \arg\min_{\boldsymbol{A}} (\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{A})^\top (\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{A})$$

The result minimizes the sum of squared residuals

$$(\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{A})^\top (\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{A}) = \sum_{t=1}^{N} (\boldsymbol{Y}_t - \boldsymbol{X}_t\boldsymbol{A})^\top (\boldsymbol{Y}_t - \boldsymbol{X}_t\boldsymbol{A})$$

It is a **frequentist approach**: it assumes that there are enough measurements to say something meaningful about $\boldsymbol{A}$.

In the Bayesian approach, the parameters are supplemented with additional information in the form of a prior probability distribution. The prior belief about the parameters is combined with the data likelihood function to yield a posterior belief about $\boldsymbol{A}$.

In the computation of the least squares solution, the matrix $\boldsymbol{X}^\top \boldsymbol{X}$ in the pseudoinverse is known as the **Gramian matrix** of $\boldsymbol{X}$.

It is usually inefficient to invert this Gramian matrix directly, but it can be decomposed in ways that either speed up computation or provide numerically more stable estimates.

Least squares with Cholesky decomposition

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

If the matrix $\boldsymbol{S} = \boldsymbol{X}^\top \boldsymbol{X}$ is positive definite and well-conditioned ($\rightarrow$ full rank), the problem can be solved directly by using the **Cholesky decomposition** $\boldsymbol{S} = \boldsymbol{R}^\top \boldsymbol{R}$, where $\boldsymbol{R}$ is an upper triangular matrix

$$\boldsymbol{R}^\top \boldsymbol{R}\, \boldsymbol{A} = \boldsymbol{X}^\top \boldsymbol{Y}$$

The solution is obtained in two stages, a forward substitution step solving for $\boldsymbol{Z}$

$$\boldsymbol{R}^\top \boldsymbol{Z} = \boldsymbol{X}^\top \boldsymbol{Y}$$

followed by a backward substitution solving for $\boldsymbol{A}$

$$\boldsymbol{R}\boldsymbol{A} = \boldsymbol{Z}$$

where both substitutions can exploit the triangular nature of $\boldsymbol{R}$.

When using *Matlab*, $\hat{\boldsymbol{A}}$ and $\hat{\boldsymbol{\Sigma}}^{\boldsymbol{A}}$ can for example be computed with the `lscov` function.

Singular value decomposition (SVD)

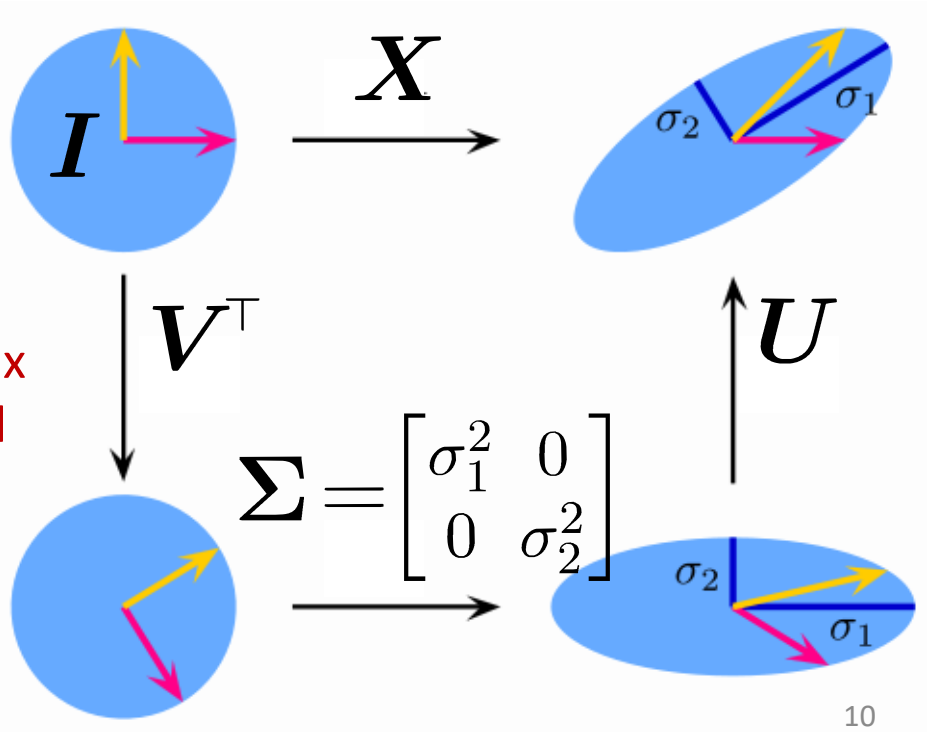
$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 2 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 \end{bmatrix}}_{\boldsymbol{X} \in \mathbb{R}^{N \times D^{\mathcal{I}}}} = \underbrace{\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 \end{bmatrix}}_{\boldsymbol{U}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

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

Unitary matrix (orthonormal bases)

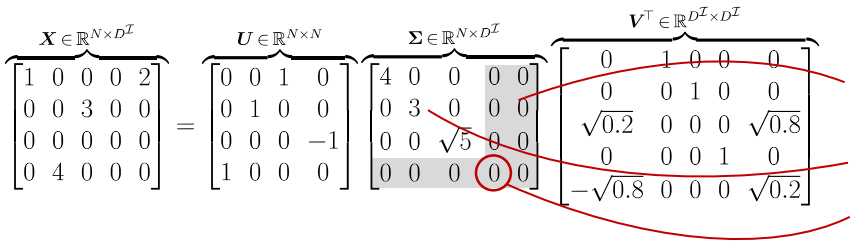
Unitary matrix (orthonormal bases)

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



Singular value decomposition (SVD)

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



SVD provides an explicit representation of the **range** and **null space** of a matrix $\boldsymbol{X}$.

The right-singular vectors corresponding to vanishing singular values of $\boldsymbol{X}$ span the **null space** of $\boldsymbol{X}$ (see gray area above).

The left-singular vectors corresponding to the non-zero singular values of $\boldsymbol{X}$ span the **range** of $\boldsymbol{X}$.

As a consequence, the **rank** of $\boldsymbol{X}$ corresponds to the number of non-zero diagonal elements in $\boldsymbol{\Sigma}$.

In numerical linear algebra, the singular values can be used to determine the effective **rank of a matrix**, as rounding error may lead to small but non-zero singular values in a rank deficient matrix.

Singular value decomposition (SVD) $\boldsymbol{X = U\Sigma V^\top}$

SVD is very general in the sense that it can be applied to any $m \times n$ matrix whereas **eigenvalue decomposition** can only be applied to certain classes of square matrices. Nevertheless, the two decompositions are related. Given an SVD of $\boldsymbol{X}$, we have

$$\mathbf{X}^\top\mathbf{X} = \mathbf{V}\boldsymbol{\Sigma}^\top\mathbf{U}^\top\,\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\top = \mathbf{V}(\boldsymbol{\Sigma}^\top\boldsymbol{\Sigma})\mathbf{V}^\top$$
$$\mathbf{X}\mathbf{X}^\top = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\top\,\mathbf{V}\boldsymbol{\Sigma}^\top\mathbf{U}^\top = \mathbf{U}(\boldsymbol{\Sigma}\boldsymbol{\Sigma}^\top)\mathbf{U}^\top$$

The right-hand sides of these relations describe the eigenvalue decompositions of the left-hand sides. Consequently:

- Columns of $\mathbf{V}$ (right-singular vec.) are eigenvectors of $\mathbf{X}^\top\mathbf{X}$.
- Columns of $\mathbf{U}$ (left-singular vec.) are eigenvectors of $\mathbf{X}\mathbf{X}^\top$.
- The non-zero elements of $\boldsymbol{\Sigma}$ (non-zero singular values) are the square roots of the non-zero eigenvalues of $\mathbf{X}^\top\mathbf{X}$ or $\mathbf{X}\mathbf{X}^\top$.

Singular value decomposition (SVD) $X = U\Sigma V^{\top}$

- Applications employing SVD include pseudoinverse computation, least squares, multivariable control, matrix approximation, as well as the determination of the **rank, range and null space of a matrix**.
- The SVD can also be thought as the decomposition of a matrix into a weighted, ordered sum of separable matrices (e.g., decomposition of an image processing filter into separable horizontal and vertical filters).
- It is possible to use the SVD of a square matrix **A** to determine the orthogonal matrix **O** closest to **A**. The closeness of fit is measured by the Frobenius norm of **O**−**A**. The solution is the product $\mathbf{UV}^{\mathrm{T}}$.
- A similar problem, with interesting applications in shape analysis, is the orthogonal **Procrustes problem**, which consists of finding an orthogonal matrix **O** which most closely maps **A** to **B**.
- Extensions to higher order arrays exist, generalizing SVD to a multi-way analysis of the data (tensor methods, multilinear algebra).

Condition number of a matrix with SVD

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

Using the pseudoinverse and a matrix norm, one can define the **condition number** of a matrix $\boldsymbol{X}$ as

$$\text{cond}(\boldsymbol{X}) = \|\boldsymbol{X}\|\|\boldsymbol{X}^\dagger\|$$

A large condition number implies that the problem of finding least-squares solutions to the corresponding system of linear equations is **ill-conditioned** in the sense that small errors in the entries of $\boldsymbol{X}$ can lead to huge errors in the entries of the solution.

The condition number can also be computed from the singular value decomposition $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$, where σ_i are the singular values in the diagonal of $\boldsymbol{\Sigma}$, with

$$\text{cond}(\boldsymbol{X}) = \frac{\max \boldsymbol{\sigma}}{\min \boldsymbol{\sigma}}$$

Least squares with SVD

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

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

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

where $\boldsymbol{U}$ and $\boldsymbol{V}$ are $m \times m$ and $n \times n$ orthogonal matrices, and $\boldsymbol{\Sigma}$ is an $m \times n$ 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. This can for example be computed in Matlab/GNU Octave with

```
[U,S,V] = svd(X)
S(S>0) = S(S>0).^-1
A = V*S'*U' * Y
```

Least squares with SVD

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

This method is computationally intensive (dominated by the cost of computing the SVD).

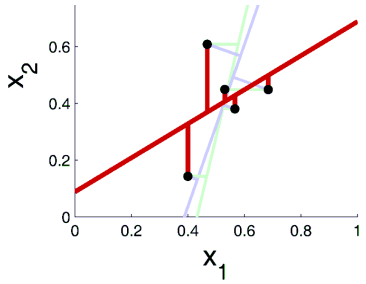
However, it is particularly useful if $\boldsymbol{X}^{\top}\boldsymbol{X}$ is **ill-conditioned** (i.e. if its condition number is large).

In that case, including the smallest singular values in the inversion merely adds numerical noise to the solution.

This can be cured with the truncated SVD approach, giving a more stable and exact answer, by explicitly setting to zero all singular values below a certain threshold (a process closely related to factor analysis).

Data fitting with linear least squares

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



- Data fitting with linear least squares does not mean that we are restricted to fitting lines models.

- For instance, we could have chosen a quadratic model $Y=X^2A$, and this model is still linear in the A parameter.

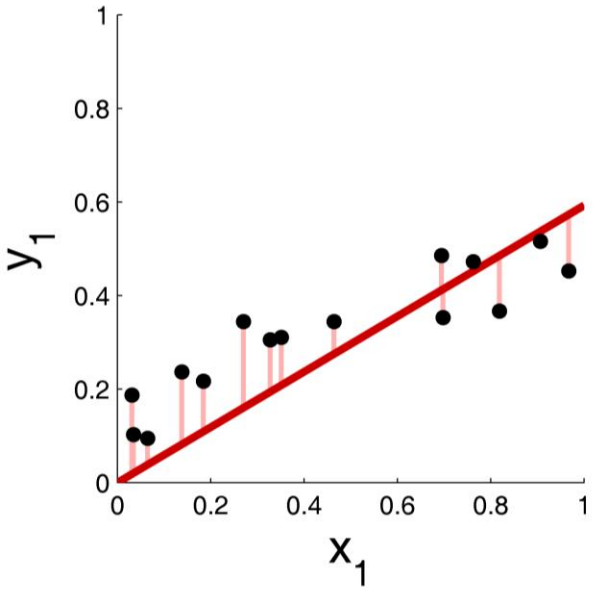
  → **The function does not need to be linear in the argument**:
  only in the parameters that are determined to give the best fit.

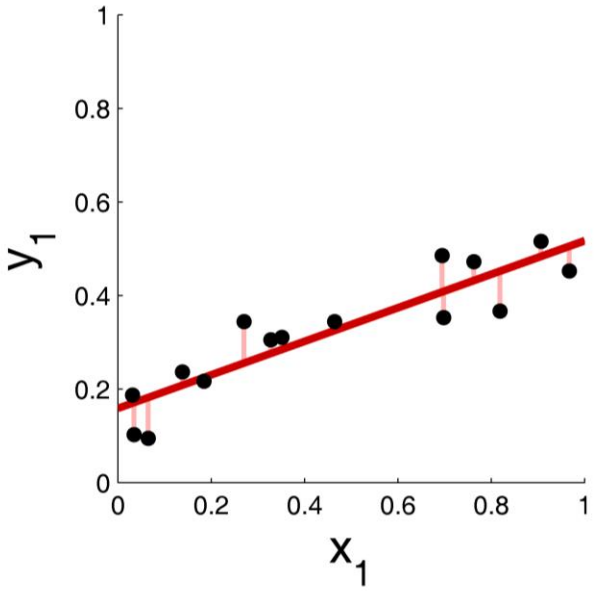
- Also, not all of X contains information about the datapoints: the first/last column can for example be populated with ones, so that an offset is learned

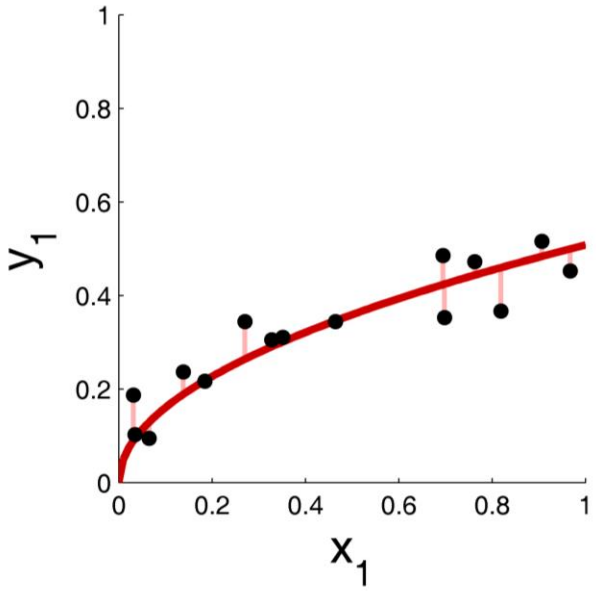
- This can be used for **polynomial fitting** by treating x, x2, ... as being distinct independent variables in a multiple regression model.

Data fitting with linear least squares

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







$$\boldsymbol{X} = \begin{bmatrix} x_{1,1} & \cdots & x_{1,D^{\mathcal{I}}} \\ \vdots & \ddots & \vdots \\ x_{N,1} & \cdots & x_{N,D^{\mathcal{I}}} \end{bmatrix} \qquad \boldsymbol{X} = \begin{bmatrix} x_{1,1} & \cdots & x_{1,D^{\mathcal{I}}} & 1 \\ \vdots & \ddots & \vdots & \vdots \\ x_{N,1} & \cdots & x_{N,D^{\mathcal{I}}} & 1 \end{bmatrix} \qquad \boldsymbol{X} = \begin{bmatrix} \sqrt{x_{1,1}} & \cdots & \sqrt{x_{1,D^{\mathcal{I}}}} \\ \vdots & \ddots & \vdots \\ \sqrt{x_{N,1}} & \cdots & \sqrt{x_{N,D^{\mathcal{I}}}} \end{bmatrix}$$

Data fitting with linear least squares

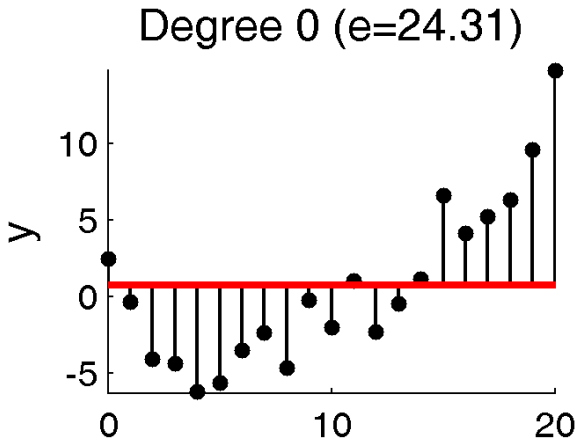
$$\hat{A} = X^{\dagger} Y$$

- Polynomial regression is an example of regression analysis using **basis functions** to model a functional relationship between two quantities. Specifically, it replaces x in linear regression with polynomial basis [**1**, **x**, $\mathbf{x}^2$, … , $\mathbf{x}^d$].
- A drawback of polynomial bases is that the basis functions are "non-local".
- It is for this reason that the polynomial basis functions are often used along with other forms of basis functions, such as splines, radial basis functions, and wavelets.

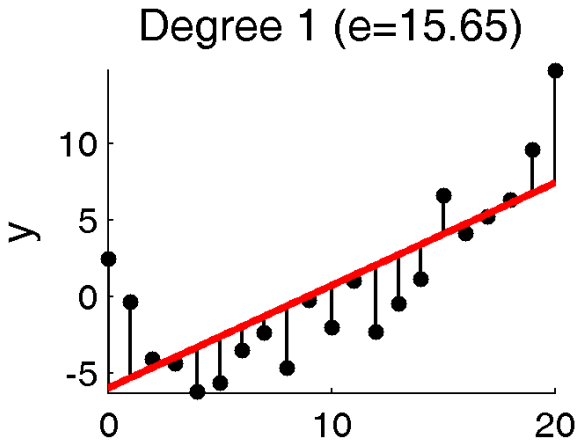
  (We will learn more about this in the next lecture…)

Polynomial fitting with least squares

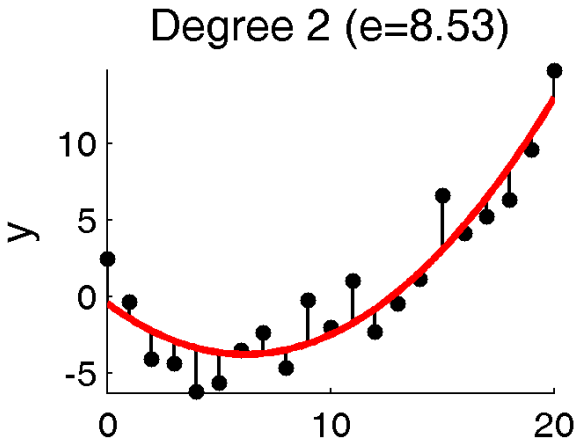
$\hat{A} = X^{\dagger} Y$



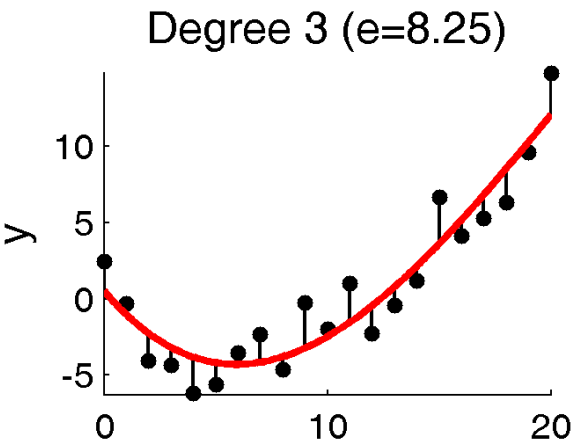
$$X = 1$$



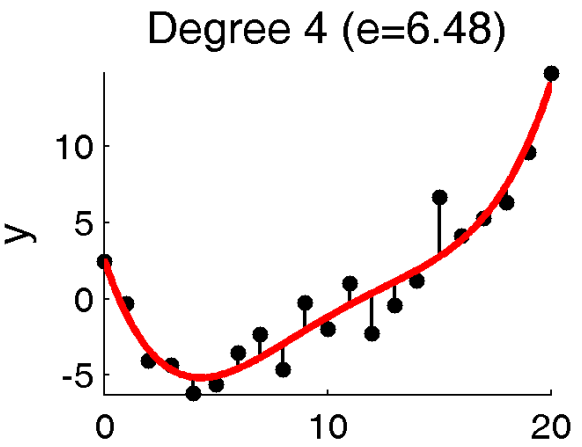
$$X = [x, 1]$$



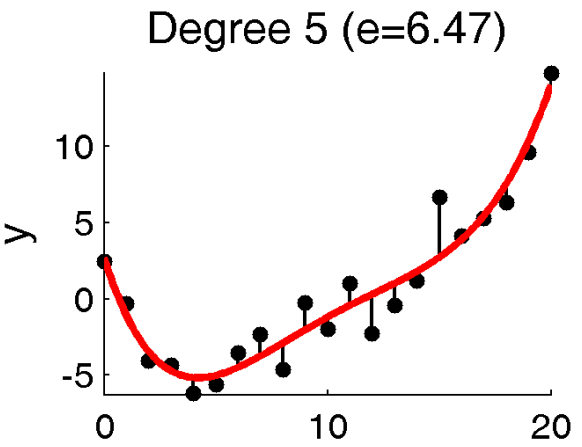
$$X = [x^2, x, 1]$$



$$X = [x^3, x^2, x, 1]$$



$$X = [x^4, x^3, \ldots, 1]$$



$$X = [x^5, x^4, \ldots, 1]$$

Kernels in least squares (nullspace projection)

Matlab code:
demo_LS_polFit_nullspace01.m

Kernels in least squares (nullspace)

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

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

In the above, the solution is unique if and only if $\hat{\boldsymbol{A}}$ has full column rank, in which case $\boldsymbol{N}$ is a zero matrix.

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

The nullspace projection guarantees that $\|\boldsymbol{Y} - \boldsymbol{X}\hat{\boldsymbol{A}}\|_2$ is still minimized.

Kernels in least squares (nullspace)

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

Indeed, we can see that for any vector/matrix $\boldsymbol{V}$, the nullspace projection ensures that it does not affect the minimization

$$\begin{aligned}
\boldsymbol{E} &= \boldsymbol{Y} - \boldsymbol{X}\hat{\boldsymbol{A}} \\
&= \boldsymbol{Y} - \boldsymbol{X}\Big(\boldsymbol{X}^\top(\boldsymbol{X}\boldsymbol{X}^\top)^{-1}\boldsymbol{Y} + \overbrace{\big(\boldsymbol{I} - \boldsymbol{X}^\top(\boldsymbol{X}\boldsymbol{X}^\top)^{-1}\boldsymbol{X}\big)}^{\boldsymbol{N}}\boldsymbol{V}\Big) \\
&= \boldsymbol{Y} - \boldsymbol{X}\boldsymbol{X}^\top(\boldsymbol{X}\boldsymbol{X}^\top)^{-1}\boldsymbol{Y} + \boldsymbol{X}\big(\boldsymbol{I} - \boldsymbol{X}^\top(\boldsymbol{X}\boldsymbol{X}^\top)^{-1}\boldsymbol{X}\big)\boldsymbol{V} \\
&= \boldsymbol{Y} - \boldsymbol{Y} + \big(\boldsymbol{X} - \boldsymbol{X}\boldsymbol{X}^\top(\boldsymbol{X}\boldsymbol{X}^\top)^{-1}\boldsymbol{X}\big)\boldsymbol{V} \\
&= (\boldsymbol{X} - \boldsymbol{X})\boldsymbol{V} \\
&= \boldsymbol{0}\,\boldsymbol{V}
\end{aligned}$$

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

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

to compute

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

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

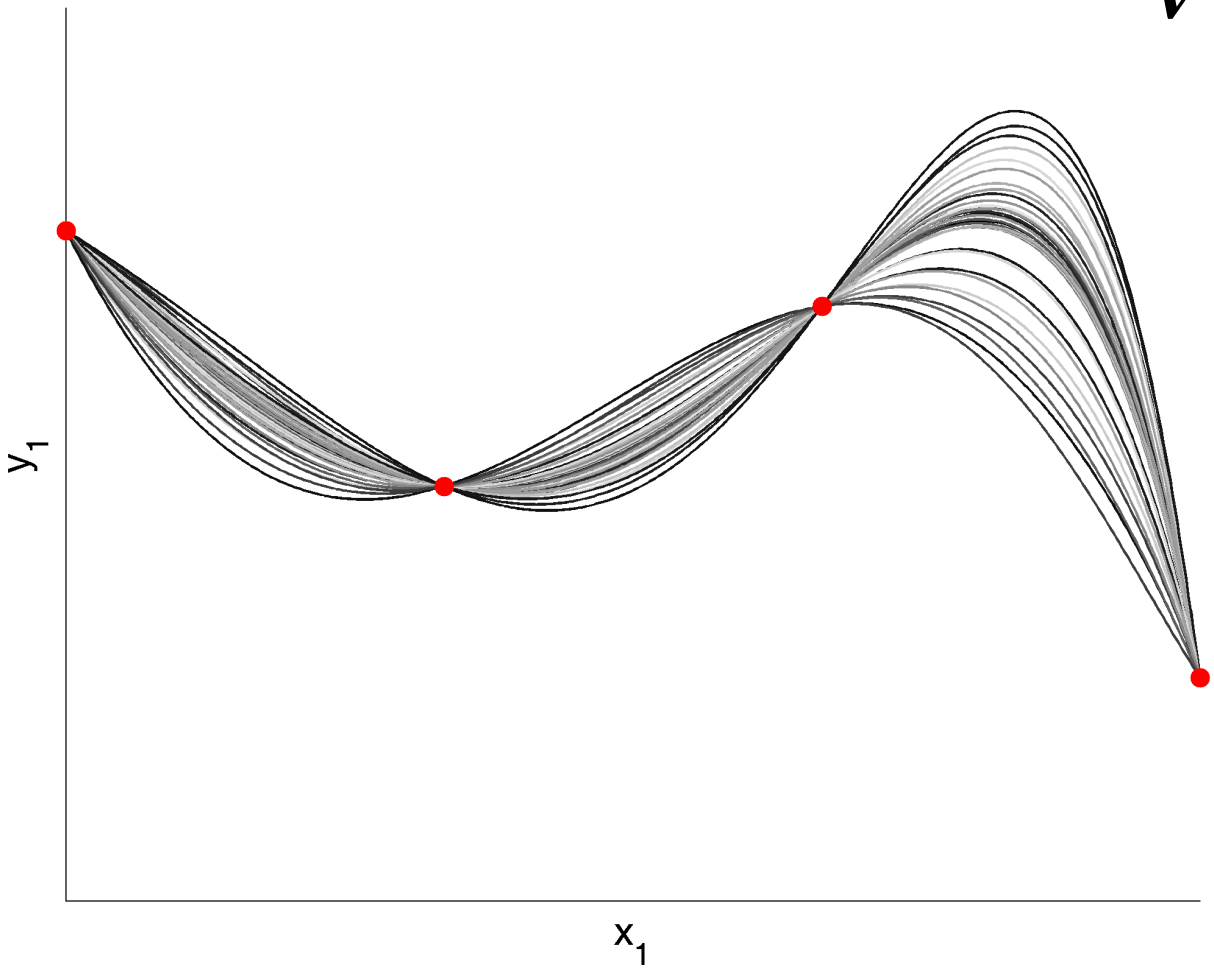
where $\tilde{\boldsymbol{U}}$ is a matrix formed by the columns of $\boldsymbol{U}$ that span for the corresponding zero rows in $\boldsymbol{\Sigma}$.

This can for example be implemented in Matlab/Octave with

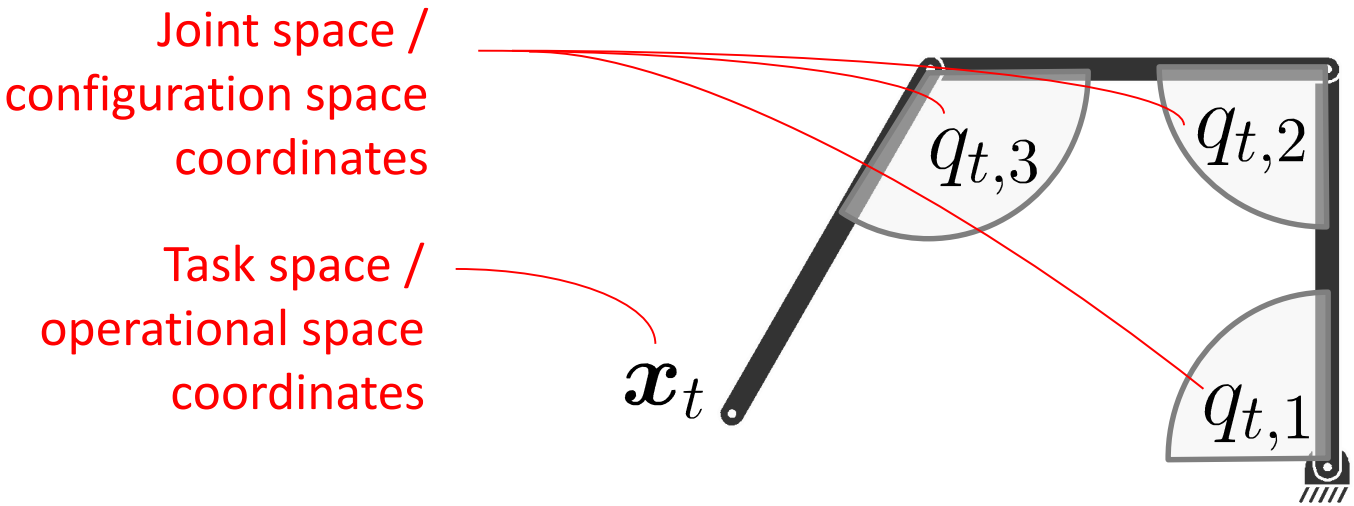
```
[U,S,V] = svd(pinv(X))
sp = sum(S,2) < 1E-1
N = U(:,sp) * U(:,sp)'
```

Example with polynomial fitting

$$\hat{\boldsymbol{A}} = \boldsymbol{X}^{\dagger}\boldsymbol{Y} + \boldsymbol{N}\boldsymbol{V} \quad \text{with} \quad \boldsymbol{X} = [\boldsymbol{x}^6, \boldsymbol{x}^5, \ldots, \mathbf{1}]$$
$$\boldsymbol{V} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$$



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

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

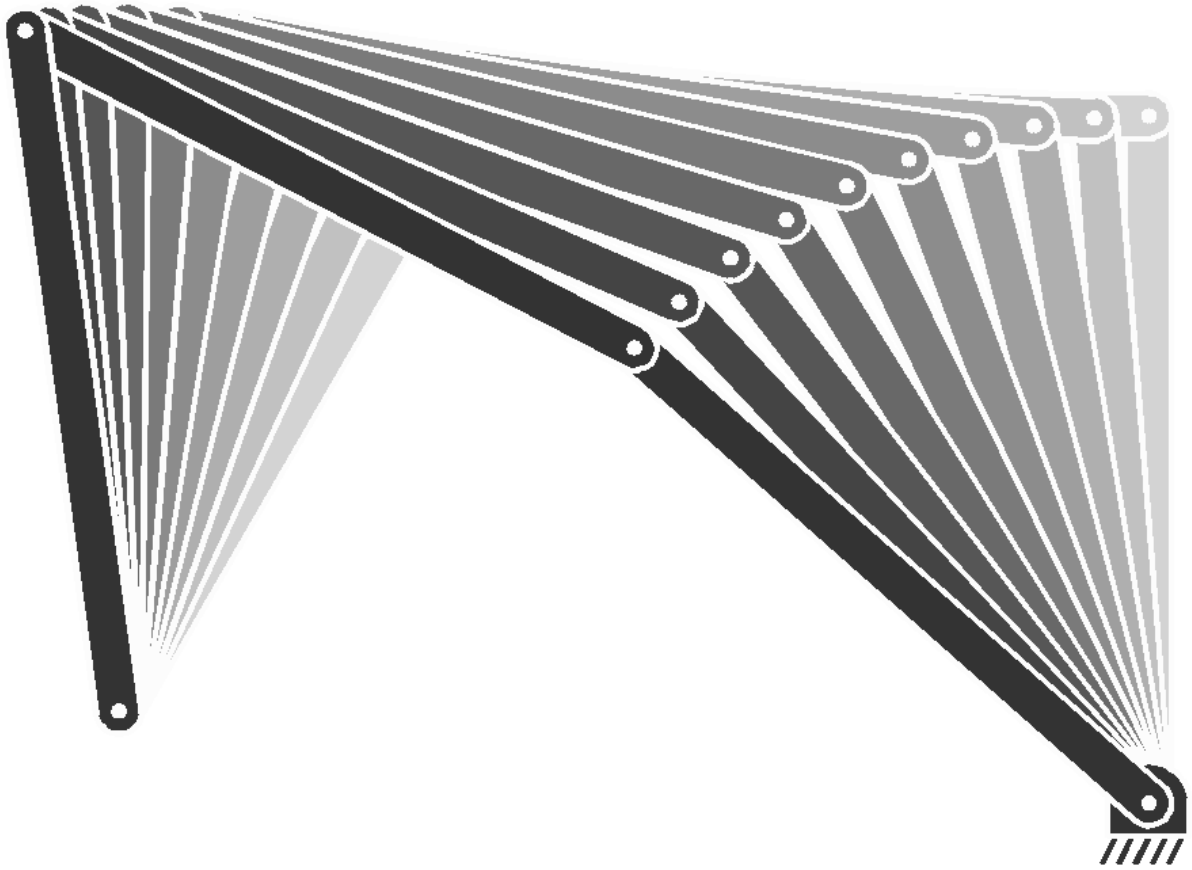
Example with robot inverse kinematics

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

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

→ **Primary constraint:** keeping the tip of the robot still

→ **Secondary constraint:** trying to move the first joint



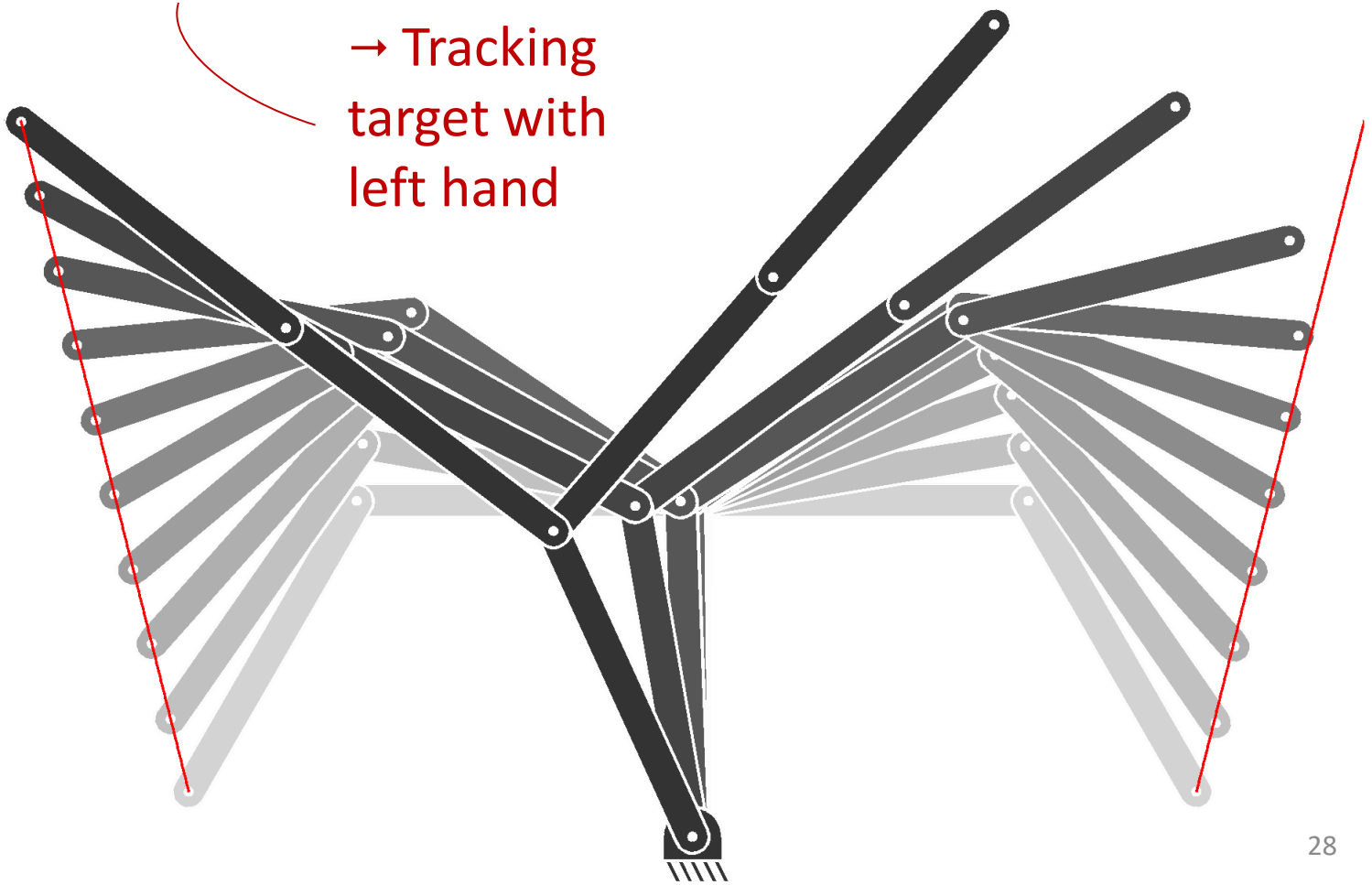
Example with robot inverse kinematics

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

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

→ Tracking target with right hand, if possible

→ Tracking target with left hand



Ridge regression (Tikhonov regularization, penalized least squares)

Matlab example: demo_LS_polFit02.m

Ridge regression (Tikhonov regularization)

The classical frequentist linear least squares solution is to estimate the matrix of regression coefficients using the Moore-Penrose pseudoinverse.

In the Bayesian approach, additional information are given to the parameters in the form of a **prior probability distribution**.

The prior can take different functional forms depending on the domain and the information that is available *a priori*.

After specifying the prior, the posterior distribution can be expressed by using the prior mean $\boldsymbol{\mu}$, with the strength of the prior indicated by the prior precision matrix $\boldsymbol{\Gamma}$.

The special case with $\boldsymbol{\mu} = \mathbf{0}$ and $\boldsymbol{\Gamma} = \lambda \boldsymbol{I}$ is called **ridge regression**.

Ridge regression (Tikhonov regularization)

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

$$\begin{aligned}\hat{\boldsymbol{A}} &= \arg\min_{\boldsymbol{A}} \|\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{A}\|_2 + \|\boldsymbol{\Gamma}\boldsymbol{A}\|_2 \\ &= \arg\min_{\boldsymbol{A}} (\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{A})^\top (\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{A}) + (\boldsymbol{\Gamma}\boldsymbol{A})^\top \boldsymbol{\Gamma}\boldsymbol{A} \\ &= \arg\min_{\boldsymbol{A}} (\boldsymbol{Y}^\top\boldsymbol{Y} - 2\boldsymbol{A}^\top\boldsymbol{X}^\top\boldsymbol{Y} + \boldsymbol{A}^\top\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{A} + \boldsymbol{A}^\top\boldsymbol{\Gamma}^\top\boldsymbol{\Gamma}\boldsymbol{A})\end{aligned}$$

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 $\lambda \ll 1$ (i.e., giving preference to solutions with smaller norms), the process is known as $\boldsymbol{L_2}$ **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

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

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

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)

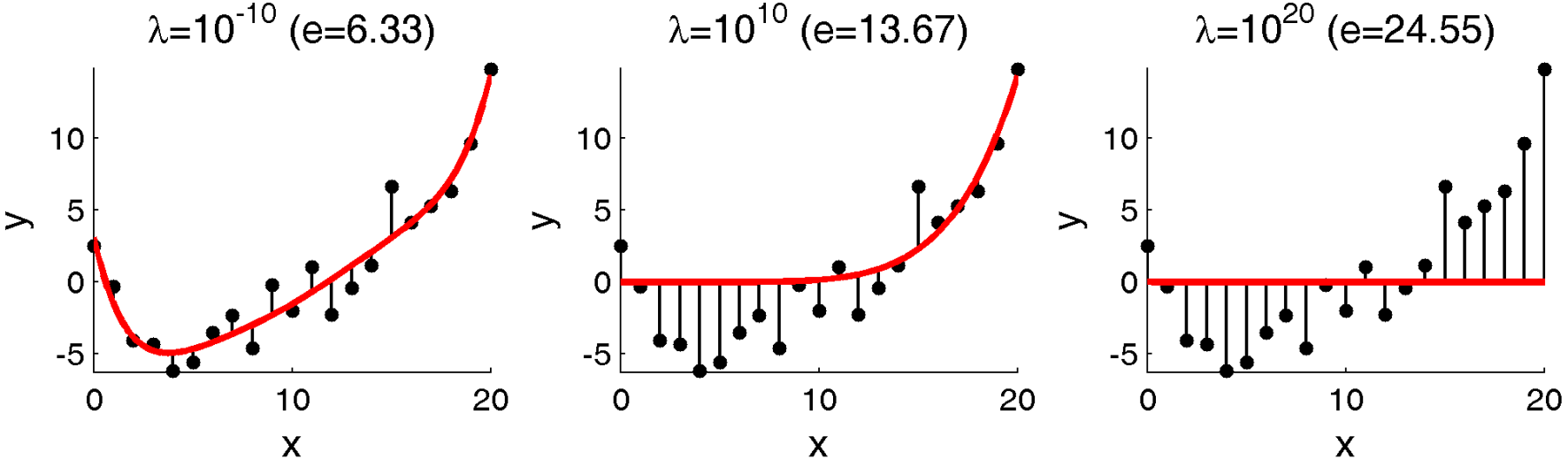
Although the choice of the solution to this regularized problem with a matrix $\mathbf{\Gamma}$ may appear rather arbitrary, the process can be justified from a **Bayesian point of view**.

Statistically, the prior probability distribution of $\boldsymbol{A}$ can be taken as a multivariate normal distribution. The assumptions that the means are zero and their components are independent can for example be considered, with the components having the same covariance $\mathbf{\Sigma}^{\boldsymbol{A}}$. The data are also subject to errors, with the errors in $\boldsymbol{Y}$ also assumed to be independent with zero mean and covariance $\mathbf{\Sigma}^{\boldsymbol{Y}}$.

Under these assumptions, the Tikhonov-regularized solution is the most probable solution given the data and the *a priori* distribution of $\boldsymbol{A}$, by following Bayes theorem.

Ridge regression (Tikhonov regularization)

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



LASSO (L_1 regularization)

- An alternative regularized version of least squares is **LASSO (least absolute shrinkage and selection operator)** using the constraint that the L_1-norm $|A|_1$ is smaller than a given value.
- The **L_1-regularized formulation** is useful due to its tendency to prefer solutions with fewer nonzero parameter values, effectively reducing the number of variables upon which the given solution is dependent.
- The increase of the penalty term in ridge regression will reduce all parameters while still remaining non-zero, while in LASSO, it will cause more and more of the **parameters to be driven toward zero**. This is a potential advantage of Lasso over ridge regression, as driving the parameters to zero **deselects the features from the regression**.

LASSO (L_1 regularization)

- Thus, LASSO automatically selects more relevant features and discards the others, whereas ridge regression never fully discards any features.

- LASSO is equivalent to an unconstrained minimization of the least-squares penalty with $|A|_1$ added.

- In a Bayesian context, this is equivalent to placing a zero-mean Laplace prior distribution on the parameter vector.

- The optimization problem may be solved using quadratic programming or more general convex optimization methods, as well as by specific algorithms such as the **least angle regression algorithm**.

Weighted least squares (Generalized least squares)

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 $\boldsymbol{Y} \in \mathbb{R}^{N \times D^{\mathcal{O}}}$, we want to minimize

$$\begin{aligned}\hat{\boldsymbol{A}} &= \arg\min_{\boldsymbol{A}} (\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{A})^{\top} \boldsymbol{W} (\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{A}) \\ &= \arg\min_{\boldsymbol{A}} (\boldsymbol{Y}^{\top}\boldsymbol{W}\boldsymbol{Y} - 2\boldsymbol{A}^{\top}\boldsymbol{X}^{\top}\boldsymbol{W}\boldsymbol{Y} + \boldsymbol{A}^{\top}\boldsymbol{X}^{\top}\boldsymbol{W}\boldsymbol{X}\boldsymbol{A})\end{aligned}$$

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

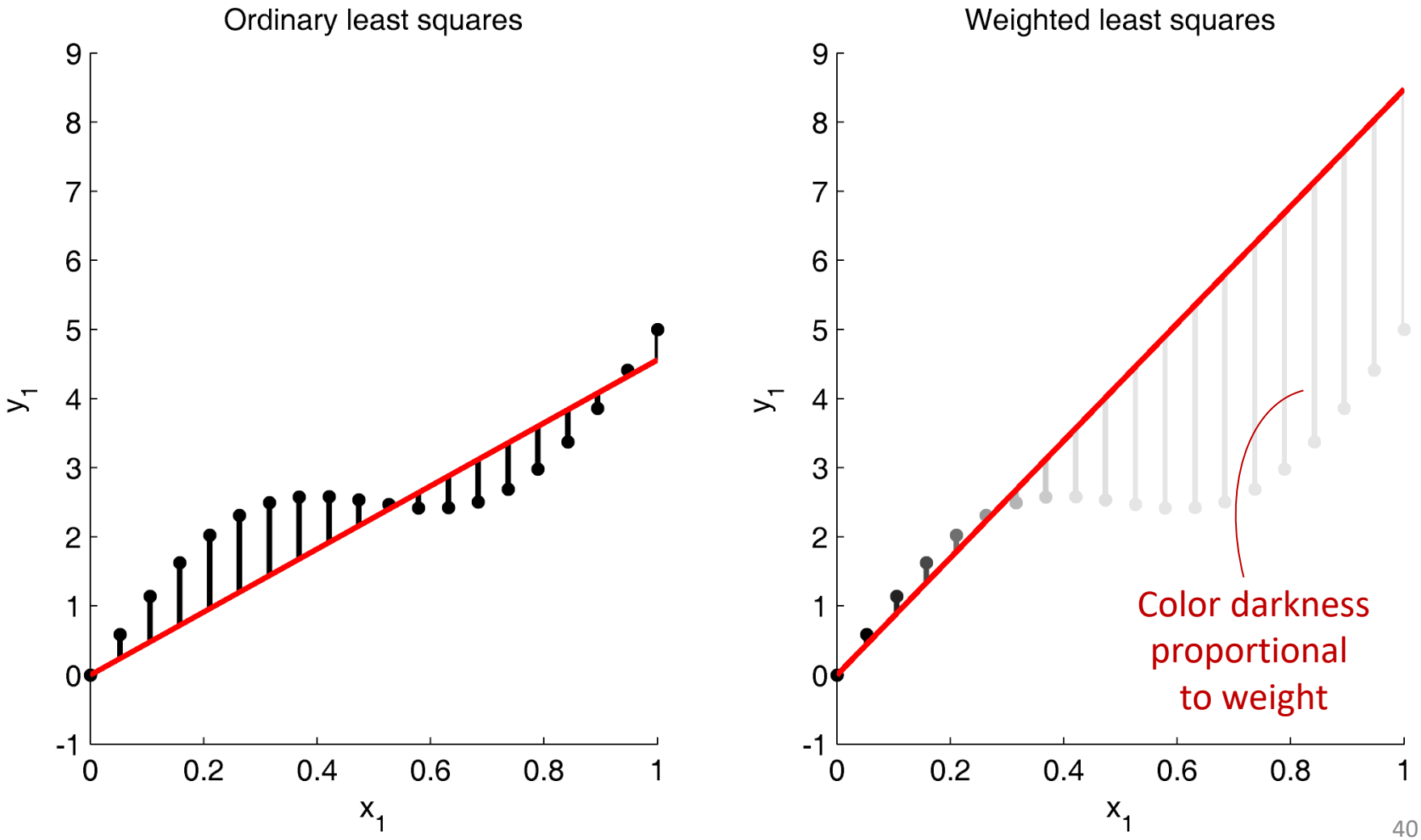
$$-2\boldsymbol{X}^{\top}\boldsymbol{W}\boldsymbol{Y} + 2\boldsymbol{X}^{\top}\boldsymbol{W}\boldsymbol{X}\boldsymbol{A} = \boldsymbol{0} \iff \hat{\boldsymbol{A}} = (\boldsymbol{X}^{\top}\boldsymbol{W}\boldsymbol{X})^{-1}\boldsymbol{X}^{\top}\boldsymbol{W}\,\boldsymbol{Y}$$

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

with residuals (parameters errors) given by $\hat{\boldsymbol{\Sigma}}^{\boldsymbol{A}} = (\boldsymbol{X}^{\top}\boldsymbol{W}\boldsymbol{X})^{-1}$

Weighted least squares

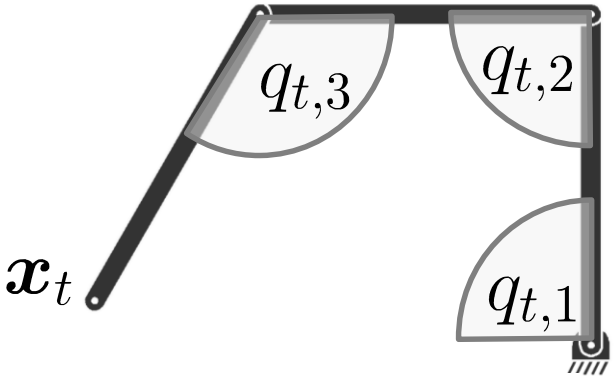
$$\hat{A} = (X^\top W X)^{-1} X^\top W Y$$

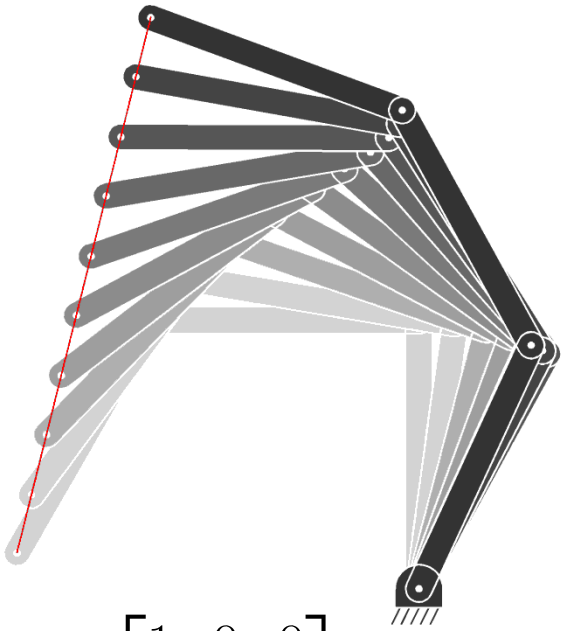


Weighted least squares – Example I

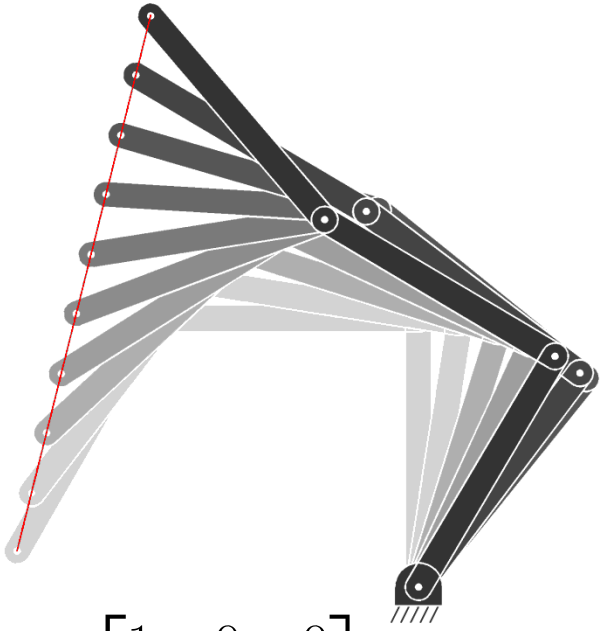
$$\hat{\boldsymbol{A}} = \boldsymbol{W}\boldsymbol{X}^{\top}(\boldsymbol{X}\boldsymbol{W}\boldsymbol{X}^{\top})^{-1}\boldsymbol{Y}$$

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

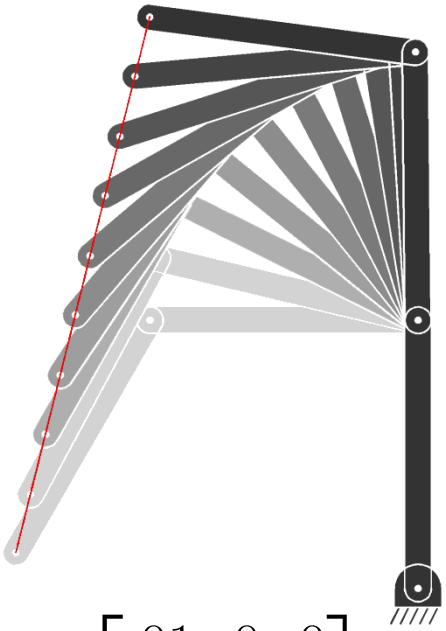




$$\boldsymbol{W}^{\mathcal{Q}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



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

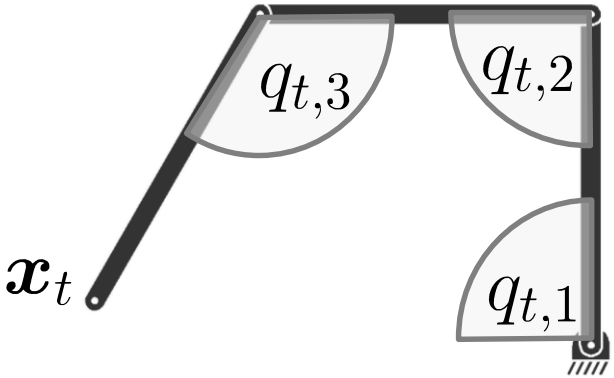


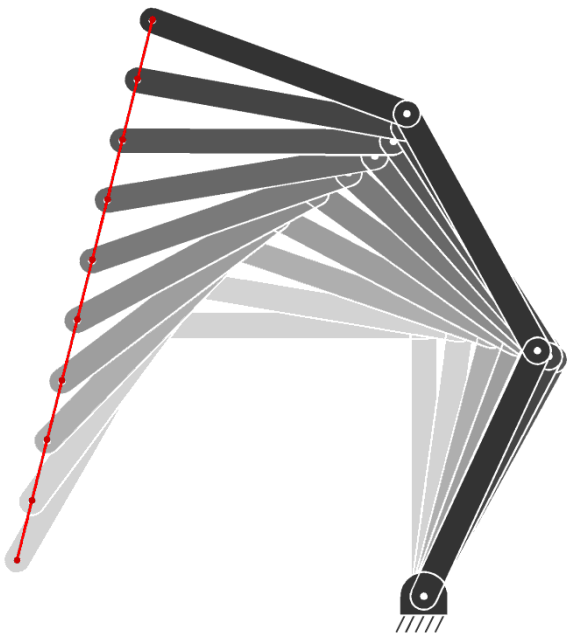
$$\boldsymbol{W}^{\mathcal{Q}} = \begin{bmatrix} .01 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Weighted least squares – Example II

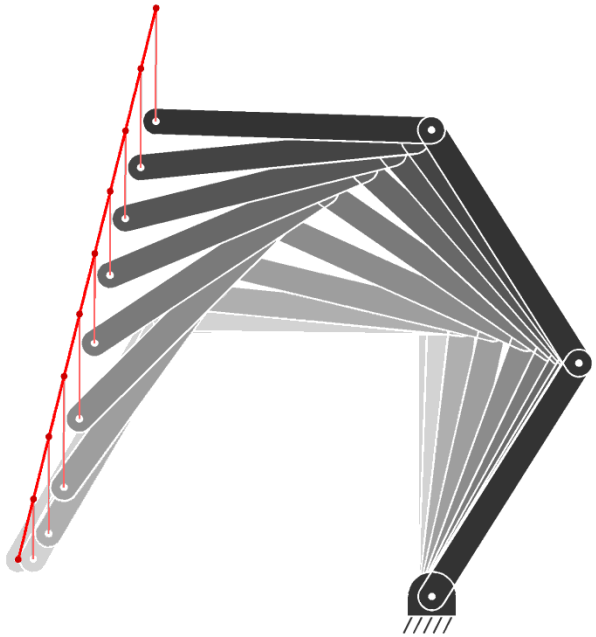
$$\hat{\boldsymbol{A}} = (\boldsymbol{X}^\top \boldsymbol{W} \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{W} \boldsymbol{Y}$$

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

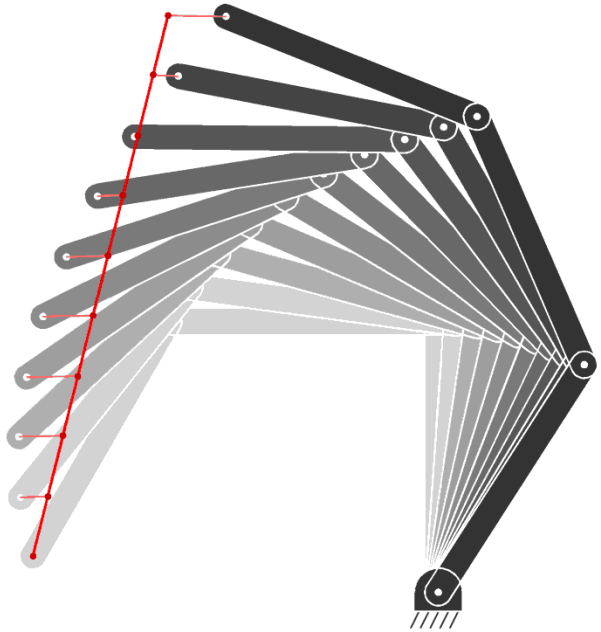




$$\boldsymbol{W}^x = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$



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



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

Iteratively reweighted least squares (IRLS)

Matlab code: demo_LS_IRLS01.m

Robust regression: Overview

- Regression analysis seeks to find the relationship between one or more independent variables and a dependent variable.

- Methods such as ordinary least squares have favorable properties if their underlying assumptions are true, but can give misleading results if those assumptions are not true.

  → **Least squares estimates are highly sensitive to outliers**.

- Robust regression methods are designed to be only mildly affected by violations of assumptions through the underlying data-generating process. It down-weights the influence of outliers, which makes their residuals larger and easier to identify.

Robust regression: Methods

- First doing an ordinary least squares fit, then identifying the k data points with the largest residuals, omitting them, and performing the fit on the remaining data

- Assuming that the residuals follow a **mixture of normal distributions**:

  → A contaminated normal distribution in which the majority of observations are from a specified normal distribution, but a small proportion are from an other normal distribution.

- Replacing the normal distribution with a **heavy-tailed distribution** (e.g., t-distributions)

  → Bayesian robust regression often relies on such distributions.

- Using **least absolute deviations** criterion (see next slide)

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 simply “least squares”
- IRLS can be used to find the maximum likelihood estimates of a generalized linear model, and find an M-estimator as a way of mitigating the influence of outliers in an otherwise normally-distributed data set. For example, **by minimizing the least absolute error rather than the least square error**.
- The strategy is that an error $|\mathbf{e}|^p$ can be rewritten as $|\mathbf{e}|^p = |\mathbf{e}|^{p-2}\,\mathbf{e}^2$. Then, $|\mathbf{e}|^{p-2}$ 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)

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

The resulting algorithm is simple to implement.

For an L_p norm objective defined by

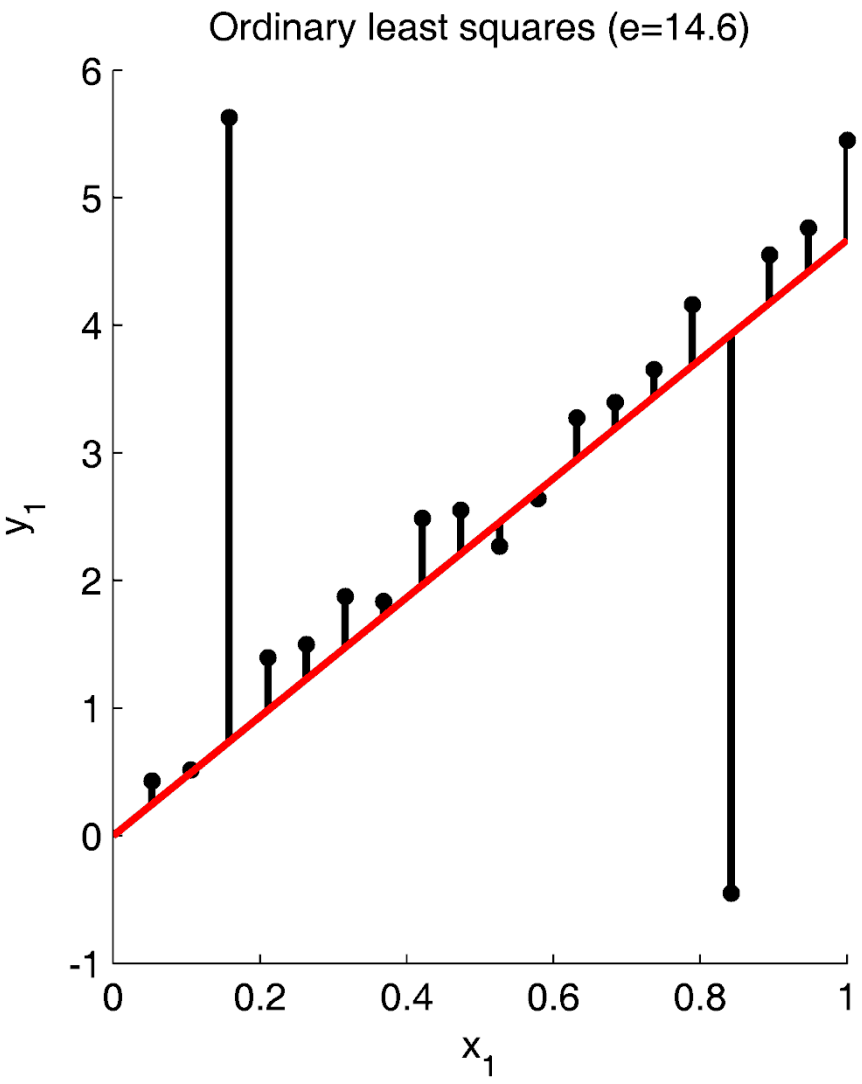
$$\hat{\boldsymbol{A}} = \arg\min_{\boldsymbol{A}} \|\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{A}\|_p$$

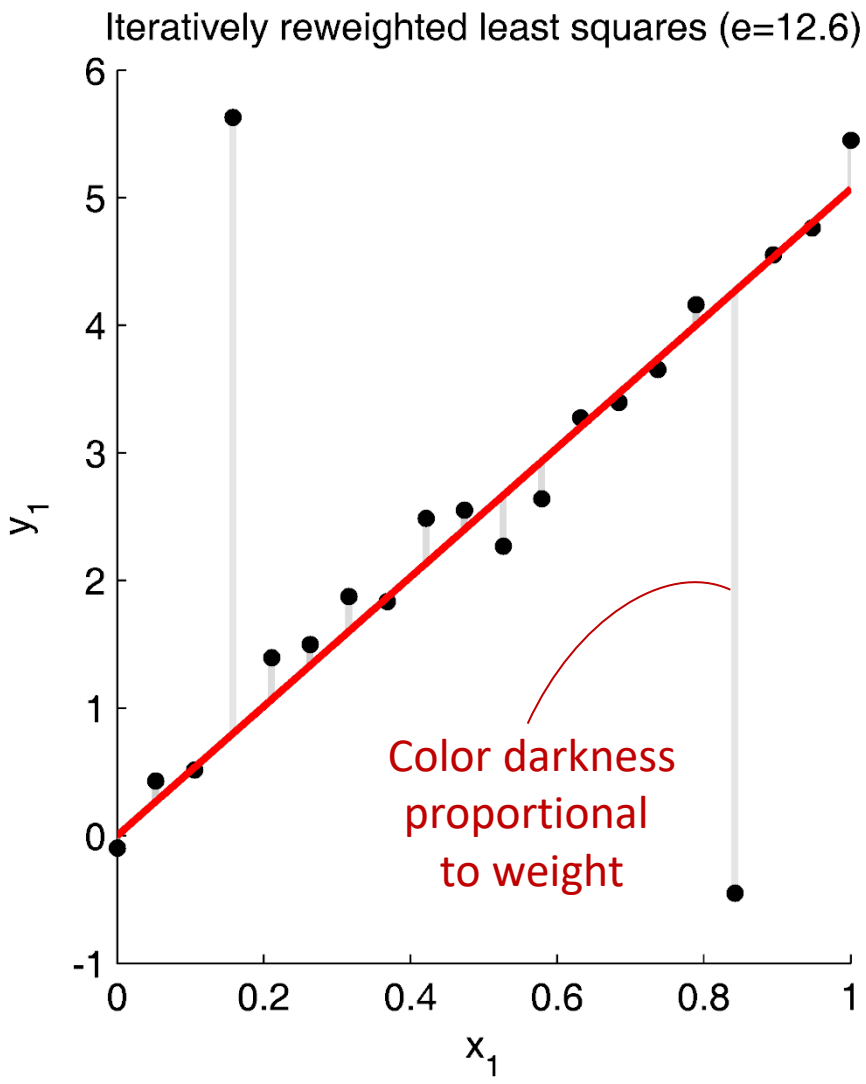
$\hat{\boldsymbol{A}}$ is estimated by starting from $\boldsymbol{W} = \boldsymbol{I}$ and iteratively computing until convergence

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

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

Iteratively reweighted least squares (IRLS)





Recursive least squares

Matlab code: demo_LS_recursive01.m

Recursive least squares

If new datapoints become available after computation of the pseudoinverse, the solution can be updated efficiently by exploiting the **Sherman-Morrison-Woodbury formulas** that relate the inverse of a matrix after a small-rank perturbation to the inverse of the original matrix, namely

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

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 $\boldsymbol{B}+\boldsymbol{U}\boldsymbol{V}$.

Recursive least squares

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

The relationship can be proven by showing that multiplying both sides of the equation by $\boldsymbol{B}+\boldsymbol{UV}$ gives the identity:

$$\begin{aligned}
&(\boldsymbol{B}+\boldsymbol{UV})\left(\boldsymbol{B}^{-1} - \boldsymbol{B}^{-1}\boldsymbol{U}\left(\boldsymbol{I}+\boldsymbol{VB}^{-1}\boldsymbol{U}\right)^{-1}\boldsymbol{VB}^{-1}\right) \\
&\quad = \boldsymbol{I}+\boldsymbol{UVB}^{-1} - (\boldsymbol{U}+\boldsymbol{UVB}^{-1}\boldsymbol{U})(\boldsymbol{I}+\boldsymbol{VB}^{-1}\boldsymbol{U})^{-1}\boldsymbol{VB}^{-1} \\
&\quad = \boldsymbol{I}+\boldsymbol{UVB}^{-1} - \boldsymbol{U}(\boldsymbol{I}+\boldsymbol{VB}^{-1}\boldsymbol{U})(\boldsymbol{I}+\boldsymbol{VB}^{-1}\boldsymbol{U})^{-1}\boldsymbol{VB}^{-1} \\
&\quad = \boldsymbol{I}+\boldsymbol{UVB}^{-1} - \boldsymbol{UVB}^{-1} = \boldsymbol{I}
\end{aligned}$$

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

Recursive least squares

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

Indeed, with $\boldsymbol{B}=\boldsymbol{X}^\top\boldsymbol{X}$, 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

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

Recursive least squares

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

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

This is exploited to estimate the update as

$$X_{\text{new}} = \begin{bmatrix} X \\ V \end{bmatrix}, \quad Y_{\text{new}} = \begin{bmatrix} Y \\ C \end{bmatrix}$$

$$\begin{aligned}
\hat{A}_{\text{new}} &= B_{\text{new}}^{-1} X_{\text{new}}^\top Y_{\text{new}} \\
&= B_{\text{new}}^{-1}\left(X^\top Y + V^\top C\right) \\
&\quad \hat{A} = B^{-1}X^\top Y \\
&= B_{\text{new}}^{-1}\left(B\hat{A} + V^\top C\right) \\
&\quad B_{\text{new}} = B + V^\top V \\
&= B_{\text{new}}^{-1}\left((B_{\text{new}} - V^\top V)\hat{A} + V^\top C\right) \\
&= B_{\text{new}}^{-1}\left(B_{\text{new}}\hat{A} - V^\top V\hat{A} + V^\top C\right) \\
&= \hat{A} + B_{\text{new}}^{-1}V^\top\left(C - V\hat{A}\right) \\
&= \hat{A} + K\left(C - V\hat{A}\right)
\end{aligned}$$

Recursive least squares

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

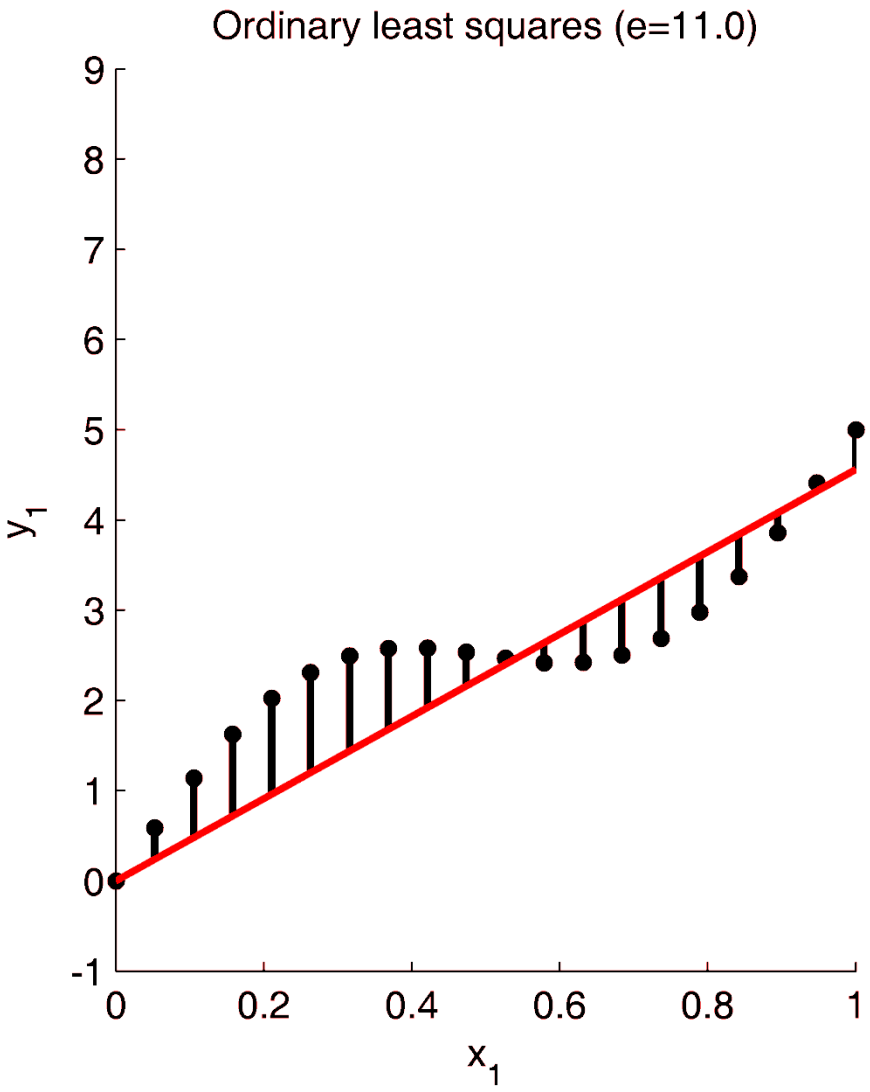
$$\boldsymbol{B}_{\text{new}}^{-1} = \boldsymbol{B}^{-1} - \boldsymbol{B}^{-1}\boldsymbol{V}^{\top}\left(\boldsymbol{I} + \boldsymbol{V}\boldsymbol{B}^{-1}\boldsymbol{V}^{\top}\right)^{-1}\boldsymbol{V}\boldsymbol{B}^{-1}$$

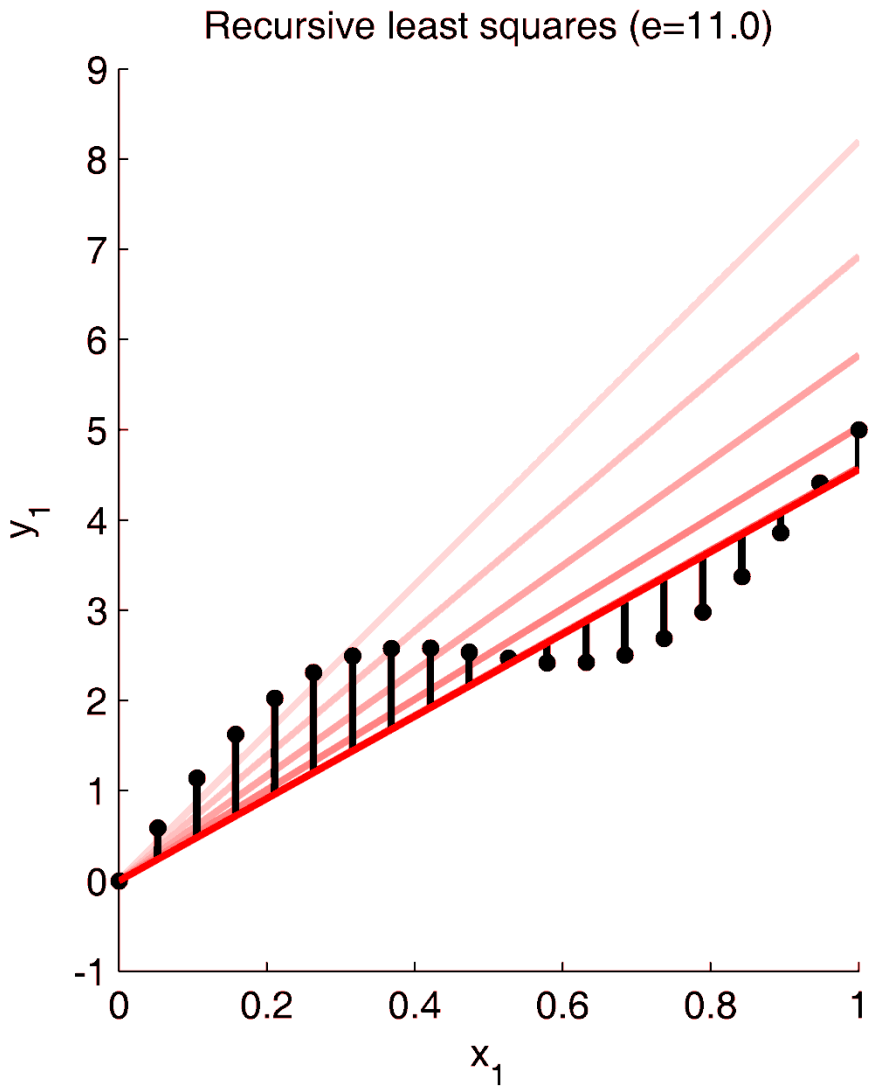
$$\boldsymbol{X}_{\text{new}} = \begin{bmatrix}\boldsymbol{X}\\ \boldsymbol{V}\end{bmatrix}, \quad \boldsymbol{Y}_{\text{new}} = \begin{bmatrix}\boldsymbol{Y}\\ \boldsymbol{C}\end{bmatrix}$$

with Kalman gain efficiently computed as

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

Recursive least squares





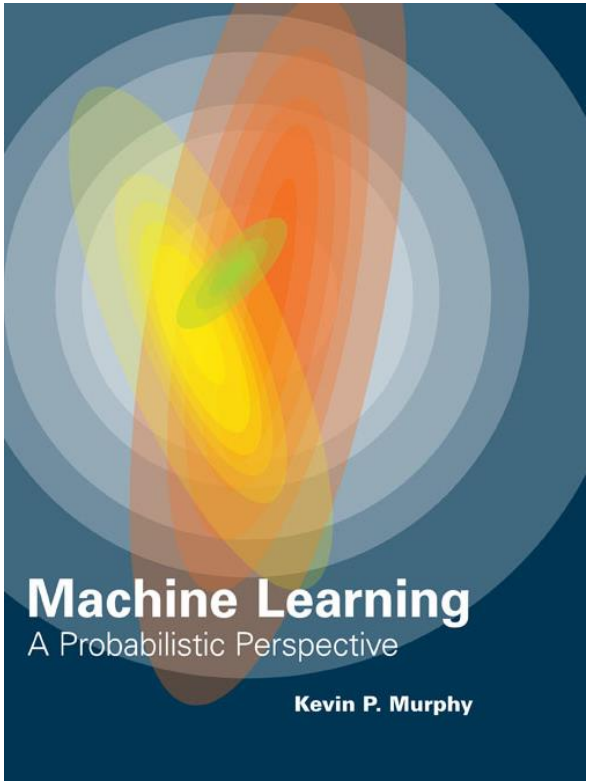
Main references

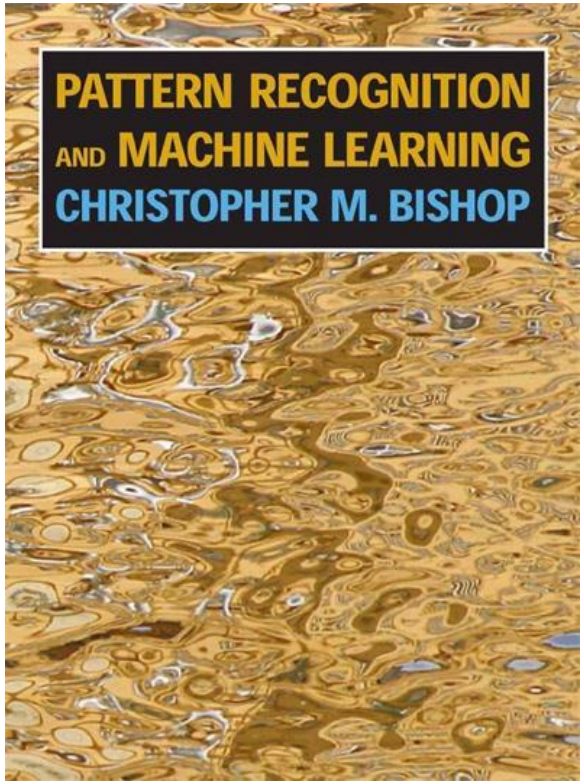
Regression

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