Geometric Modeling

Summer Semester 2010

Interpolation and Approximation

Interpolation · Least-Squares Techniques

UNIVERSITÄT DES SAARLANDES

CLUSTER OF EXCELLENCE

max planck institut informatik

Today...

Topics:

- Introduction
- Mathematical Background
- Interpolation & Approximation
  - Interpolation
  - Least-Squares Approach
  - Total-Least-Squares (PCA)

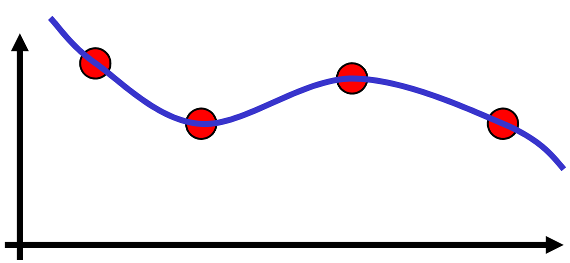
Interpolation

General & Polynomial Interpolation

Interpolation Problem

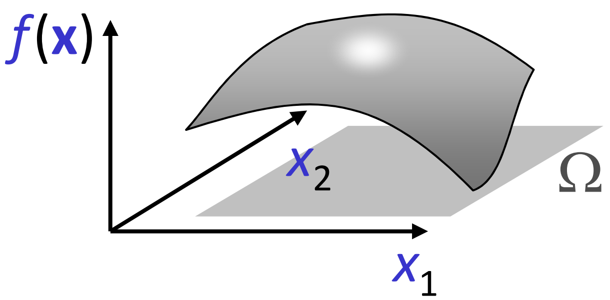
First approach to modeling smooth objects:

- Given a set of points along a curve or surface
- Choose basis functions that span a suitable function space
  - Smooth basis functions
  - Any linear combination will be smooth, too
- Find a linear combination such that the curve/surface interpolates the given points



General Formulation

Settings:

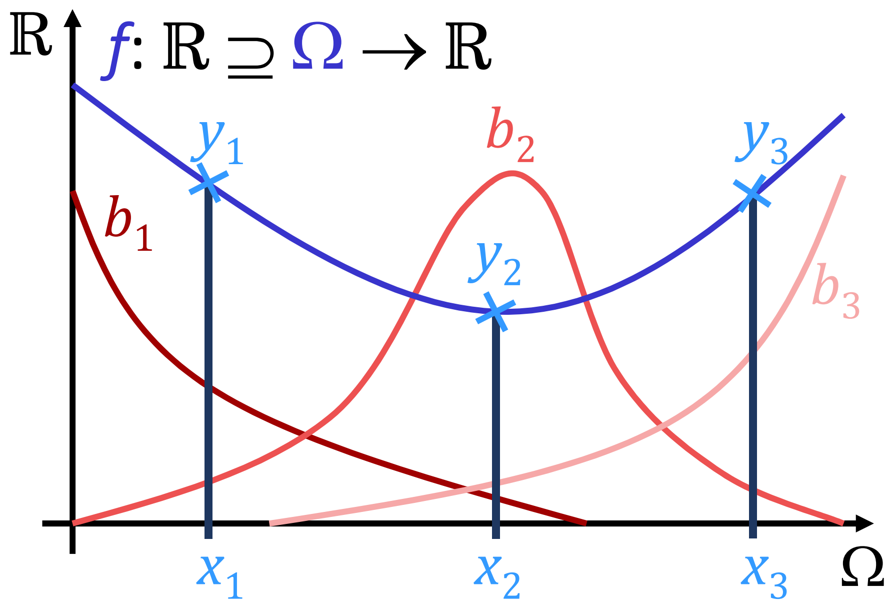


- Domain $\Omega \subseteq \mathbb{R}^{d_s}$, mapping to $\mathbb{R}$.
- Looking for a function f: $\Omega \to \mathbb{R}$.
- Basis set: $B = \{b_1,\ldots,b_n\}$, b_i: $\Omega \to \mathbb{R}$.
- Represent f as linear combination of basis functions:

$$f_{\boldsymbol{\lambda}}(\mathbf{x}) = \sum_{i=1}^{n} \lambda_i b_i(\mathbf{x}), \text{ i.e. } f \text{ is just determined by } \boldsymbol{\lambda} = \begin{pmatrix} \lambda_1 \\ \vdots \\ \lambda_n \end{pmatrix}$$

- Function values: $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, $(\mathbf{x}_i, y_i) \in \mathbb{R}^{d_s} \times \mathbb{R}$
- We want to find $\boldsymbol{\lambda}$ such that: $\forall i \in \{1,\ldots,n\} : f_{\boldsymbol{\lambda}}(\mathbf{x}_i) = y_i$

Illustration



1D Example

Solving the Interpolation Problem

Solution: linear system of equations

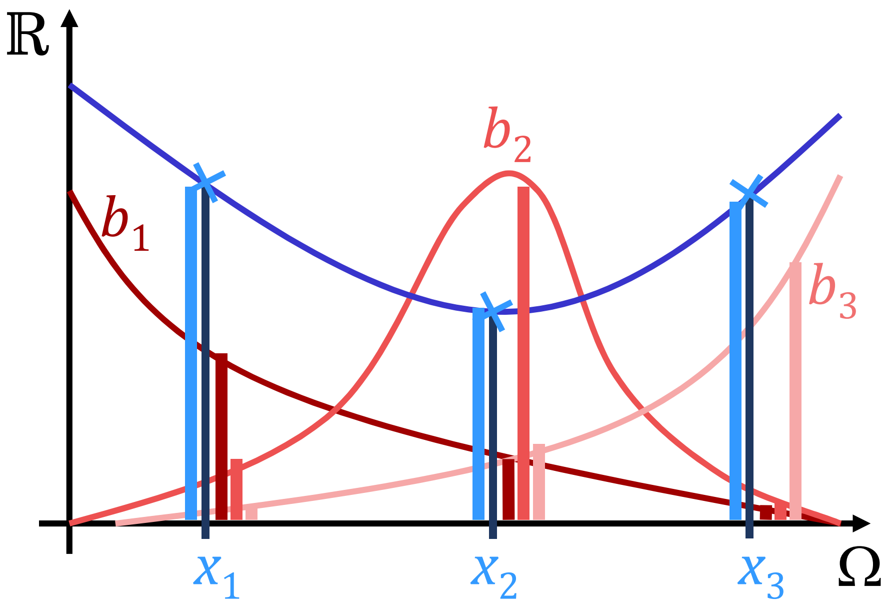
- Evaluate basis functions at points $\mathbf{x}_i$:

$$\forall i \in \{1,\ldots,n\} : \sum_{i=1}^{n} \lambda_i b_i(\mathbf{x}_i) = y_i$$

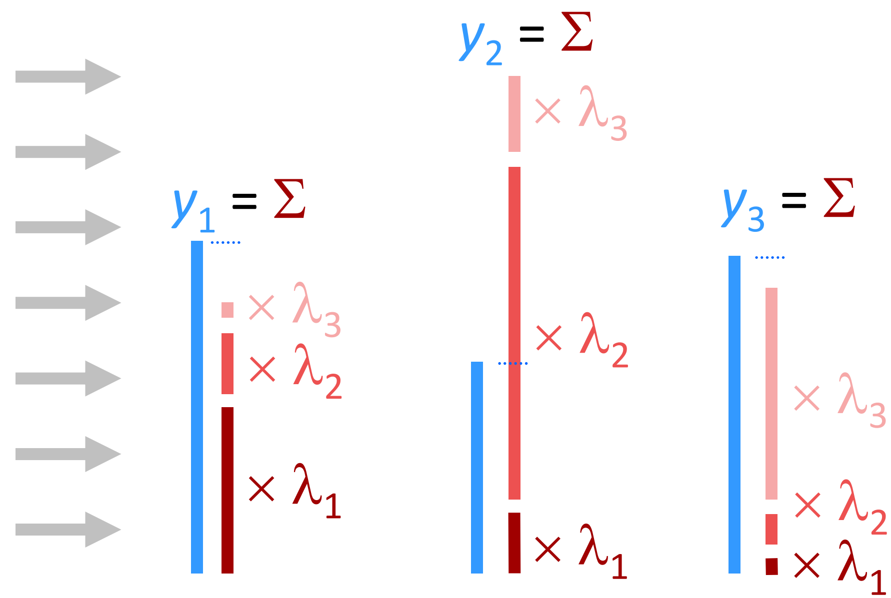
- Matrix form:

$$\begin{pmatrix} b_1(\mathbf{x}_1) & \cdots & b_n(\mathbf{x}_1) \\ \vdots & & \vdots \\ b_1(\mathbf{x}_n) & \cdots & b_n(\mathbf{x}_n) \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \vdots \\ \lambda_n \end{pmatrix} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$$

Illustration

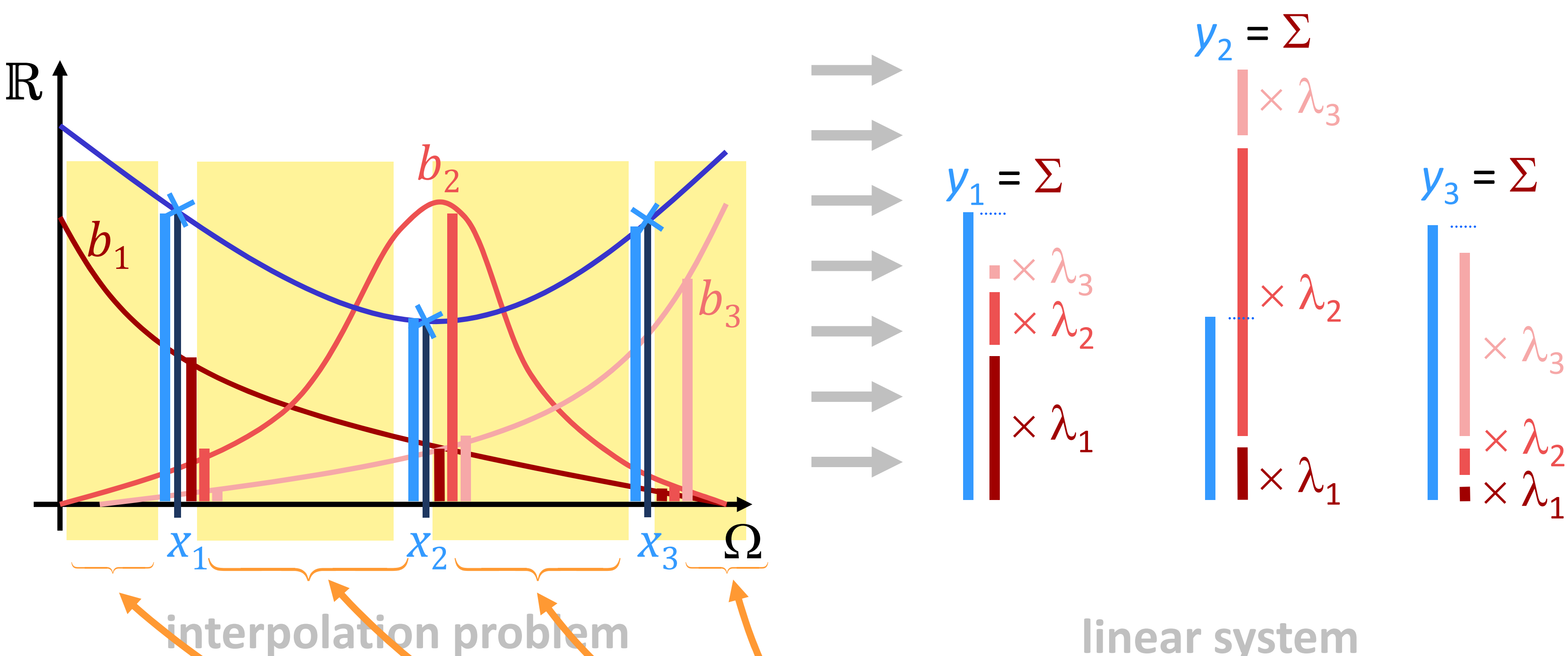


interpolation problem



linear system

Illustration



anything in between does not matter
(determined by basis only)

Example

Example: Polynomial Interpolation

- Monomial basis B = $\{1, x, x^2, x^3, \ldots, x^{n-1}\}$
- Linear system to solve:

$$\begin{pmatrix} 1 & x_1 & \cdots & x_1^{n-1} \\ 1 & x_2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & \cdots & x_n^{n-1} \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \vdots \\ \lambda_n \end{pmatrix} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$$

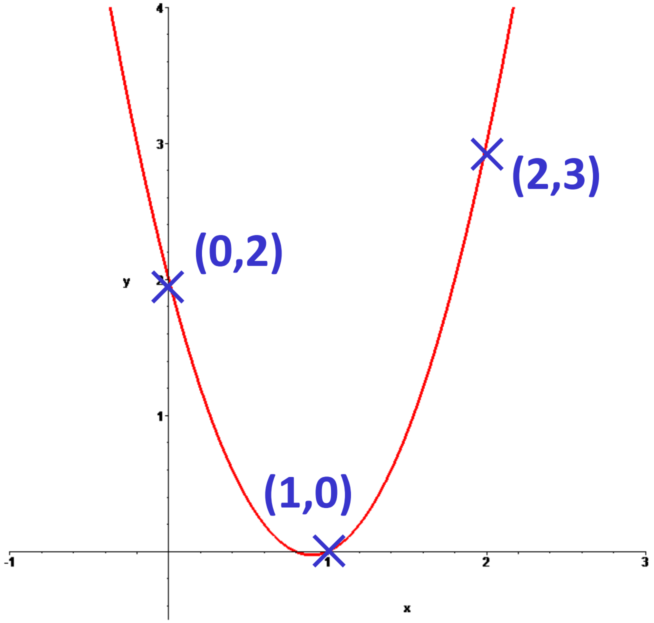
"Vandermonde Matrix"

Example with Numbers

Example with numbers

- Quadratic monomial basis B = $\{1, x, x^2\}$
- Function values: $\{(0,2), (1,0), (2,3)\}$ $[(x, y)]$
- Linear system to solve:

$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 4 \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \\ 3 \end{pmatrix}$$

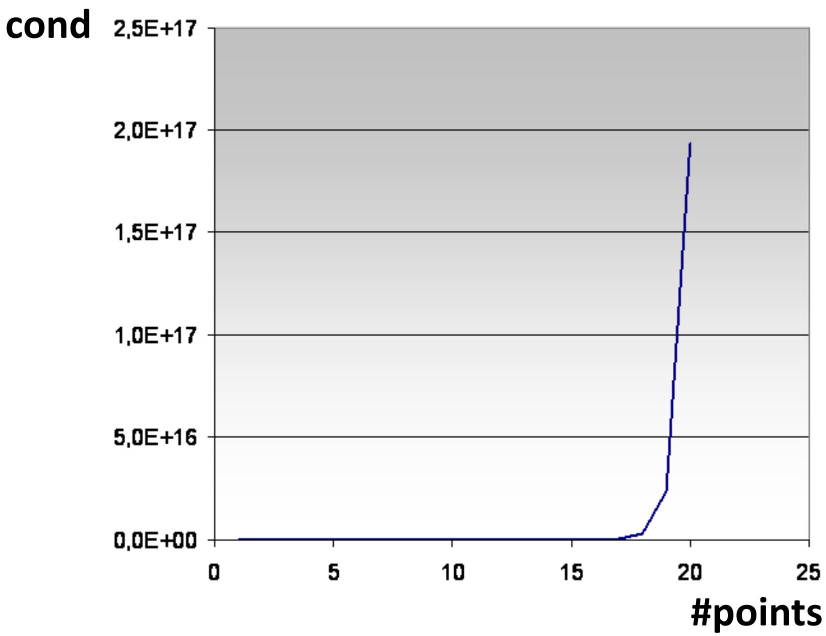


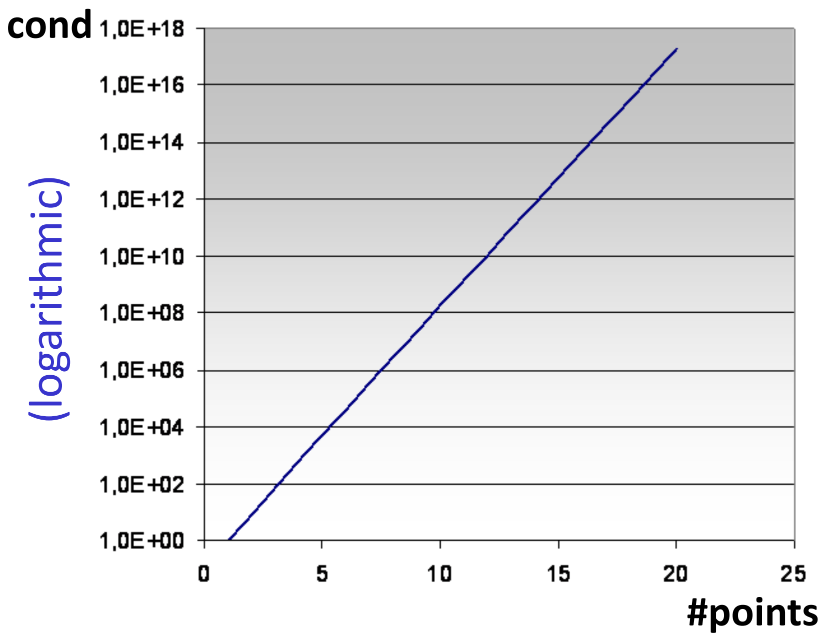
- Result: $\lambda_1 = 2$, $\lambda_2 = -9/2$, $\lambda_3 = 5/2$

Condition Number...

The interpolation problem is ill conditioned:

- For equidistant x_i, the condition number of the Vandermode matrix grows exponentially with n (maximum degree+1 = number of points to interpolate)

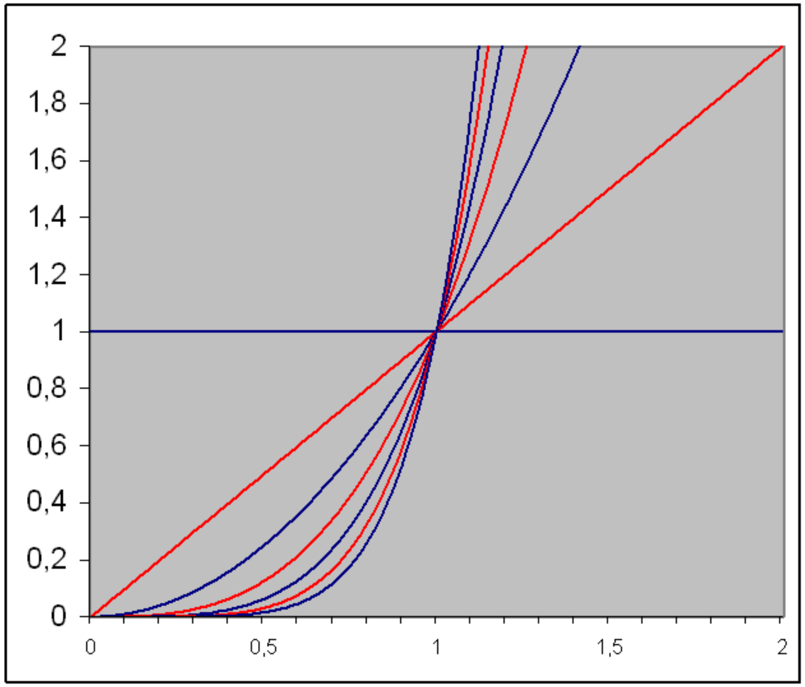




Why is that?

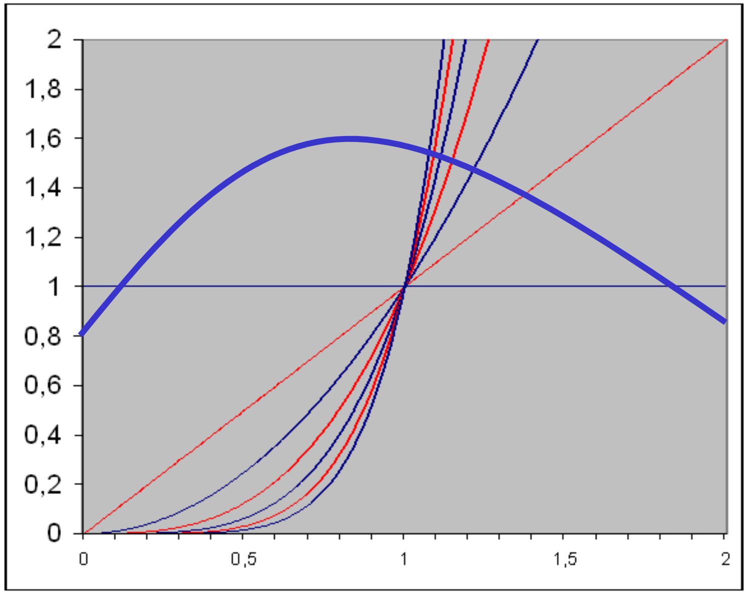
Monomial Basis:

- Functions become increasingly indistinguishable with degree
- Only differ in growing rate (x^i growth faster than x^{i-1})



Monomial basis

Cancellation



Monomials:

- From left to right in x-direction...
- First 1 dominates
- Then x grows faster
- Then x^2 grows faster
- Then x^3 grows faster
- ...

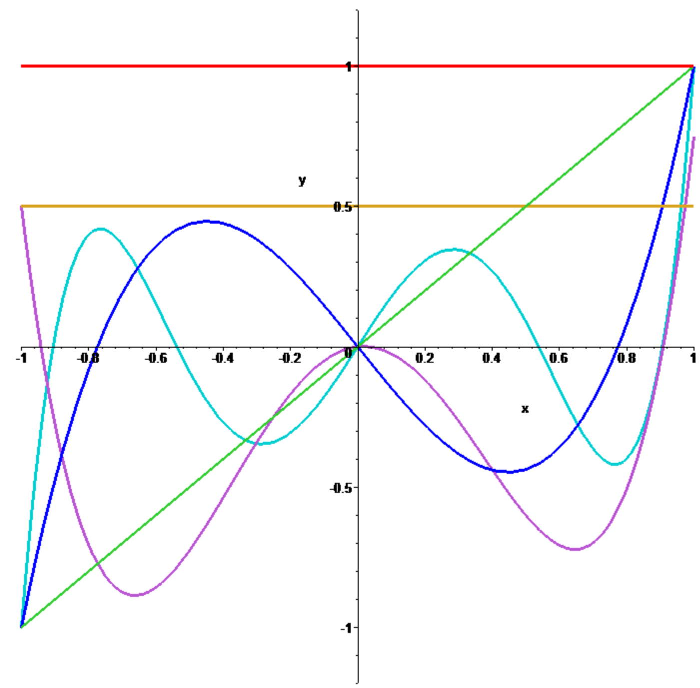
Tendency:

- Well behaved functions often require alternating sequence of coefficients (left turn, right turn, left turn,...)
- *Cancellation* problems

The Cure...

This problem can be fixed:

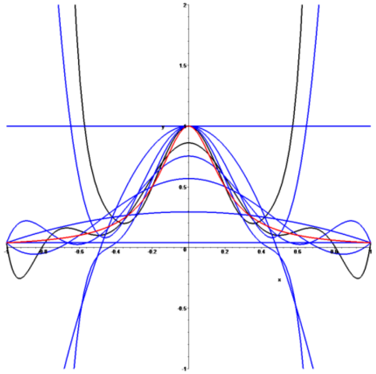
- Use orthogonal polynomial basis
- How to get one? $\rightarrow$ e.g. Gram-Schmidt orthogonalization (see assignment sheet 1)
- Legendre polynomials – orthonormal on [-1..1]
- Much better condition of the linear system (converges to 1)



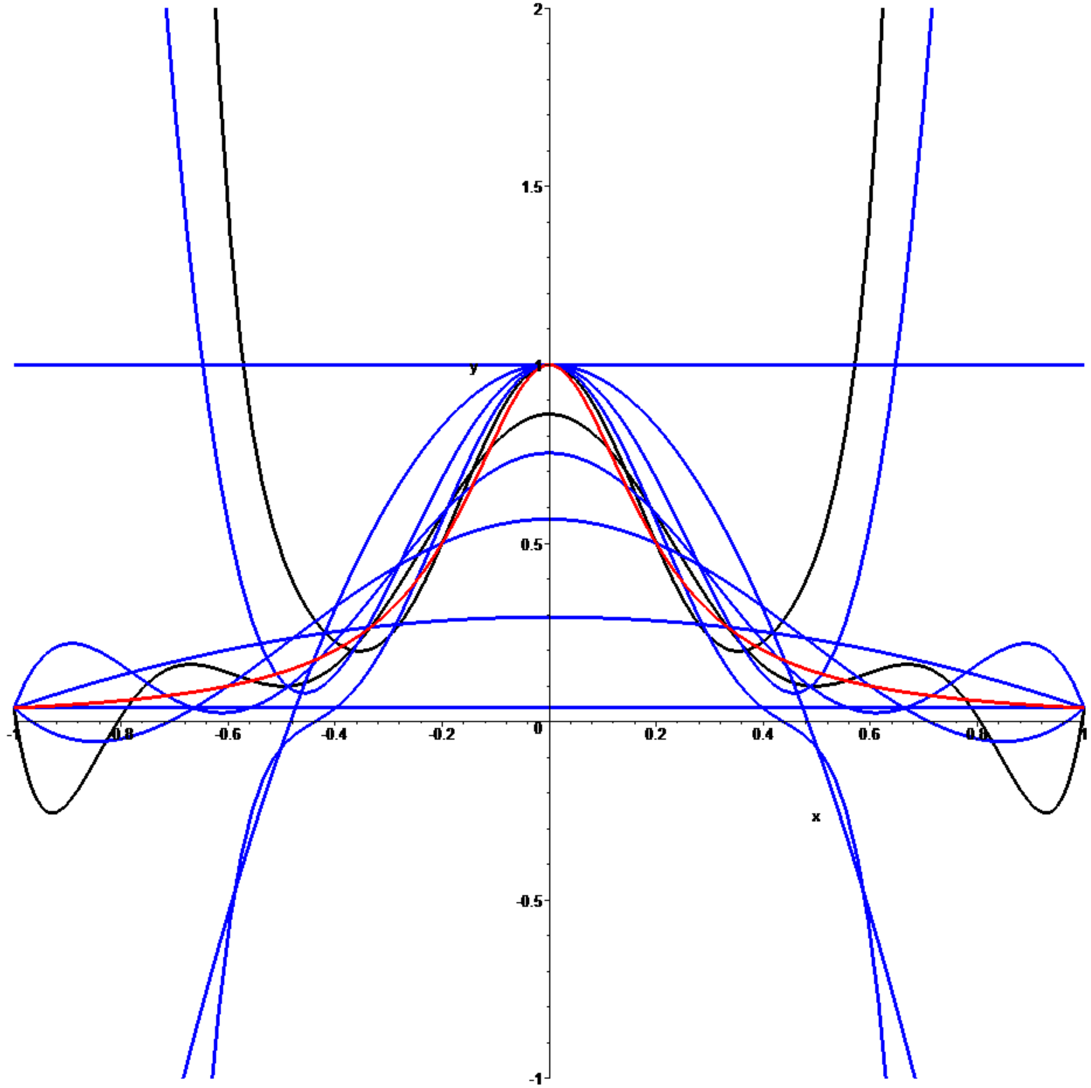
However...

This does not fix all problems:

- Polynomial interpolation is instable
  - "Runge's phenomenon": *Oscillating behavior*
  - Small changes in control points can lead to very different result. $\mathbf{x}_i$ sequence important.
- Weierstraß approximation theorem:
  - Smooth functions (C^0) can be approximated arbitrarily well with polynomials
  - However: Need carefully chosen construction for convergence
  - Not useful in practice



Runge's Phenomenon



Conclusion

Conclusion: Need a better basis for interpolation

For example, piecewise polynomials will work much better

Generalizations

Differential Constraints

Constraints on differential properties:

- For clarity: looking at the one-dimensional case
- We can also specify values of derivatives

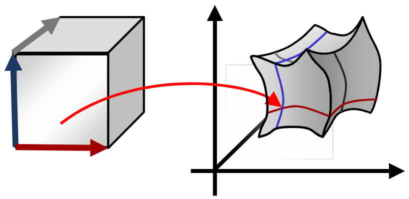
$$\forall i \in \{1,\dots,n\}: \frac{d^{k_i}}{dx^{k_i}} f(x_i) = y_i \;\longrightarrow\; \forall i \in \{1,\dots,n\}: \sum_{i=1}^{n} \lambda_i \frac{d^{k_i}}{dx^{k_i}} b_i(x_i) = y_i$$

$$\begin{pmatrix} \frac{d^{k_1}}{dx^{k_1}} b_1(x_1) & \cdots & \frac{d^{k_1}}{dx^{k_1}} b_n(x_1) \\ \vdots & \ddots & \vdots \\ \frac{d^{k_n}}{dx^{k_n}} b_1(x_n) & \cdots & \frac{d^{k_n}}{dx^{k_n}} b_n(x_n) \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \vdots \\ \lambda_n \end{pmatrix} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$$

- Need to evaluate derivatives of basis functions only (linearity)

Multi-Dimensional Targets

Settings:



- Domain $\Omega \subseteq \mathbb{R}^{d_s}$, mapping to $\mathbb{R}^{d_t}$.
- Looking for a function f: $\Omega \to \mathbb{R}^{d_t}$.
- Basis set: $B = \{b_1,\ldots,b_n\}$, b_i: $\Omega \to \mathbb{R}^{d_t}$.
- Represent f as linear combination of basis functions:

$$f_\lambda(\mathbf{x}) = \sum_{i=1}^{n} \lambda_i^{(k)} b_i^{(k)}(\mathbf{x}), \; k = 1 \ldots d_t \quad \text{(i.e.: } \lambda \text{ is a tupel of vectors)}$$

- Function values: $\{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)\}$, $(\mathbf{x}_i, \mathbf{y}_i) \in \mathbb{R}^{d_s} \times \mathbb{R}^{d_t}$
- Solve for λ by solving d_t linear systems
  (each target space coordinate separately)

Solving the Interpolation Problem

Solution: linear system of equations

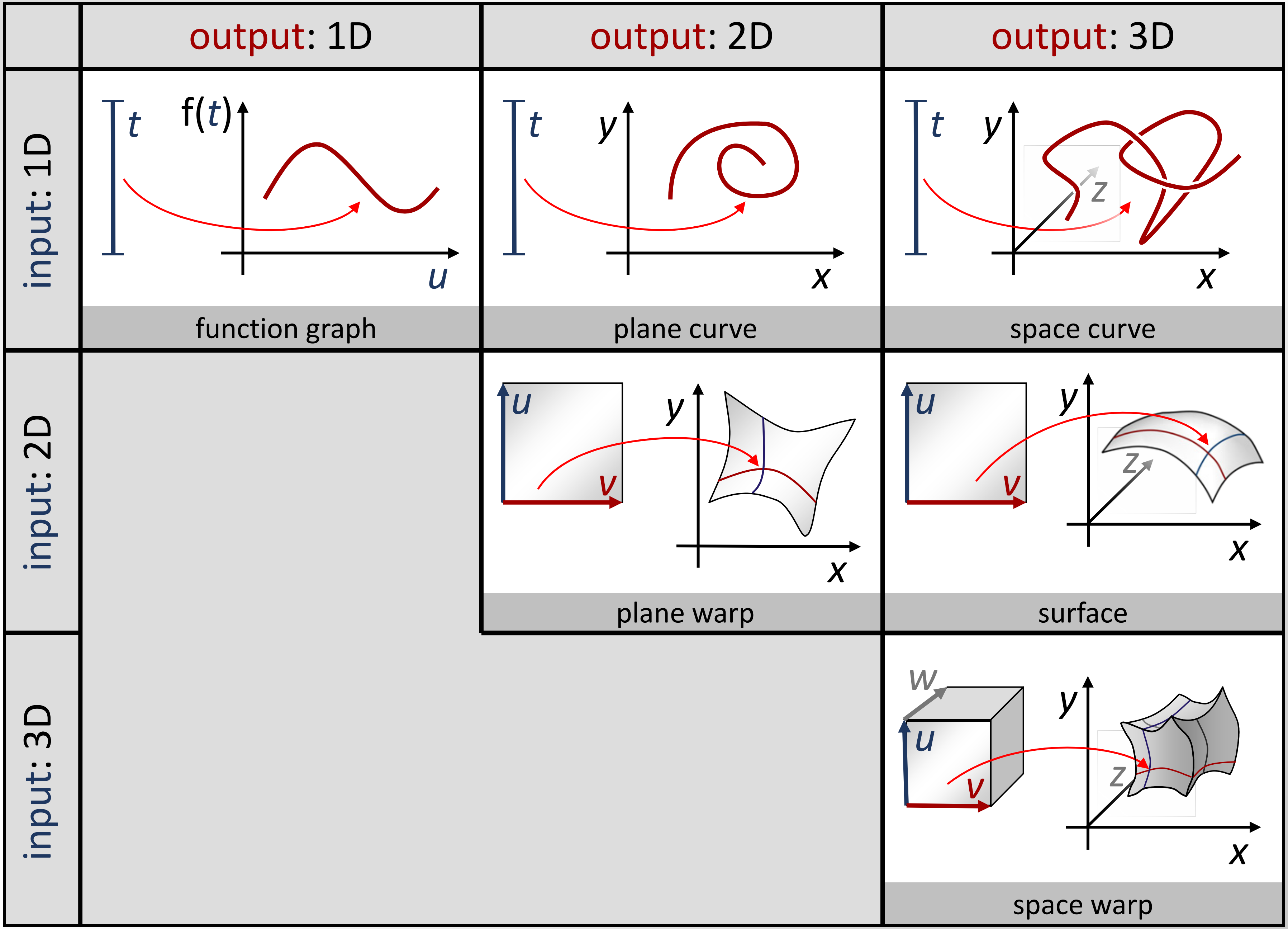
- For every target coordinate k = 1...d_t: Evaluate basis functions at points $\mathbf{x}_i$:

$$\forall k \in \{1,\ldots,d_t\} \forall i \in \{1,\ldots,n\} : \sum_{i=1}^{n} \lambda_i^{(k)} b_i^{(k)}(\mathbf{x}_i) = y_i^{(k)}$$

- Yields d_t linear systems of the form:

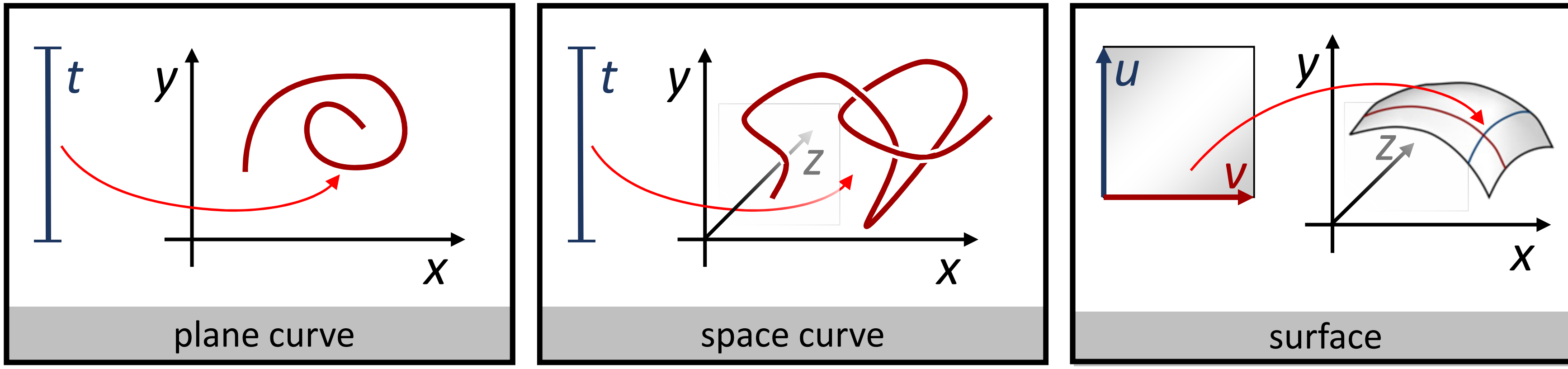
$$\begin{pmatrix} b_1^{(k)}(\mathbf{x}_1) & \cdots & b_n^{(k)}(\mathbf{x}_1) \\ \vdots & & \vdots \\ b_1^{(k)}(\mathbf{x}_n) & \cdots & b_n^{(k)}(\mathbf{x}_n) \end{pmatrix} \begin{pmatrix} \lambda_1^{(k)} \\ \vdots \\ \lambda_n^{(k)} \end{pmatrix} = \begin{pmatrix} y_1^{(k)} \\ \vdots \\ y_n^{(k)} \end{pmatrix}$$

- In other words: solve a separate interpolation problem for each target dimension



output: 1D
output: 2D
output: 3D
input: 1D
input: 2D
input: 3D
t
f(t)
u
function graph
t
y
x
plane curve
t
y
z
x
space curve
u
v
y
x
plane warp
u
v
y
z
x
surface
w
u
v
y
z
x
space warp

Parametric Interpolation



Parametric Interpolation

- Problem is defined for function graphs, height fields, etc...
- For fitting parametric objects:
  - We need to specify parameter for points
  - Then solve the standard interpolation problem
- Same problem for parametric approximation problems

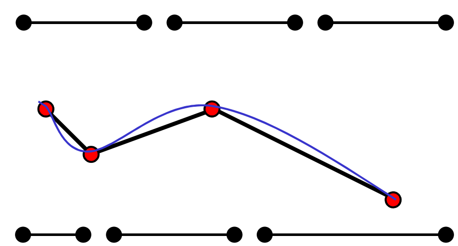
How to Make up Parameter Values

Estimating parameter values:

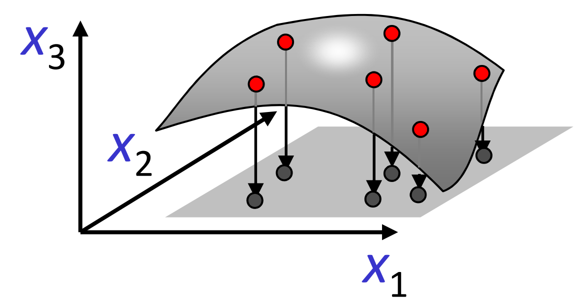
- Not a well-defined problem per se

Standard heuristics:

- Curves (sorted input):
  - Uniform spacing $(1,2,3,\ldots,n)$
  - Chord length $\left(0,\|\mathbf{y}_2-\mathbf{y}_1\|,\|\mathbf{y}_3-\mathbf{y}_2\|,\ldots,\|\mathbf{y}_n-\mathbf{y}_{n-1}\|\right)$



- Surfaces:
  - Height-field like data: project in "normal" direction (points onto a plane)
  - In general: point cloud parametrization problem



Approximation

(Reweighted) Least-squares, Scattered Data

Approximation

Common Situation:

- We have many data points, they might be noisy
- Example: Scanned data
- Want to approximate the data with a smooth curve / surface

What we need:

- Criterion – what is a good approximation?
- Methods to compute this approximation

Approximation Techniques

Agenda:

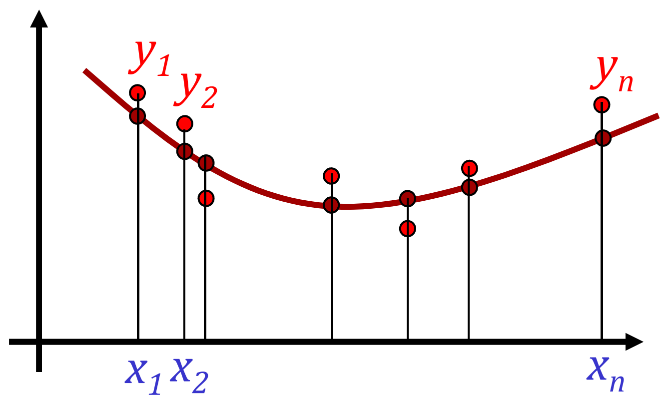
- Least-squares approximation
  (and why/when this makes sense)
- Total least-squares linear approximation
  (get rid of the coordinate system)
- Iteratively reweighted least-squares
  (for nasty noise distributions)

Least-Squares

We assume the following scenario:

- We have a set of function values y_i at positions x_i.
  (1D $\rightarrow$ 1D for now)
- The independent variables x_i are known exactly.
- The dependent variables y_i are known approximately, with some error.
- The error is *normal distributed*, *independent*, and with the *same distribution* at every point (normal iid noise).
- We know the class of functions from which the noisy samples were taken.

Situation



Situation:

- Original sample points taken at x_i from original f.
- Unknown Gaussian iid noise added to each y_i.
- Want to estimated reconstructed $\tilde{f}$.

Summary

Statistical model yields least-squares criterion:

$$\arg\min_{\tilde{f}} \sum_{i=1}^{n} (\tilde{f}(x_i) - y_i)^2$$

Linear function space leads to quadratic objective:

$$\tilde{f}(x) := \sum_{j=1}^{k} \lambda_j b_j(x) \quad \longrightarrow \quad \arg\min_{\boldsymbol{\lambda}} \sum_{i=1}^{n} \left[\left(\sum_{j=1}^{k} \lambda_j b_j(x_i) \right) - y_i \right]^2$$

Critical point: linear system

$$\begin{pmatrix} \langle \mathbf{b}_1, \mathbf{b}_1 \rangle & \cdots & \langle \mathbf{b}_1, \mathbf{b}_k \rangle \\ \vdots & \ddots & \vdots \\ \langle \mathbf{b}_k, \mathbf{b}_1 \rangle & \cdots & \langle \mathbf{b}_k, \mathbf{b}_k \rangle \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \vdots \\ \lambda_k \end{pmatrix} = \begin{pmatrix} \langle \mathbf{y}, \mathbf{b}_1 \rangle \\ \vdots \\ \langle \mathbf{y}, \mathbf{b}_k \rangle \end{pmatrix}$$

with:

$$\langle \mathbf{b}_i, \mathbf{b}_j \rangle := \sum_{t=1}^{n} b_i(x_t) \cdot b_j(x_t)$$

$$\langle \mathbf{y}, \mathbf{b}_i \rangle := \sum_{t=1}^{n} b_i(x_t) \cdot y_t$$

Maximum Likelihood Estimation

Goal:

- Maximize the probability that the data originated from the reconstructed curve $\tilde{f}$.
- "Maximum likelihood estimation"

$$p_{\mu,\sigma}(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

Gaussian normal distribution

Maximum Likelihood Estimation

$$\arg\max_{\tilde{f}} \prod_{i=1}^{n} N_{0,\sigma}(\tilde{f}(x_i) - y_i)$$

Maximum Likelihood Estimation

$$\arg\max_{\tilde{f}} \prod_{i=1}^{n} N_{0,\sigma}(\tilde{f}(x_i) - y_i) = \arg\max_{\tilde{f}} \prod_{i=1}^{n} \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(\tilde{f}(x_i) - y_i)^2}{2\sigma^2} \right)$$

$$= \arg\max_{\tilde{f}} \ln \prod_{i=1}^{n} \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(\tilde{f}(x_i) - y_i)^2}{2\sigma^2} \right)$$

$$= \arg\max_{\tilde{f}} \sum_{i=1}^{n} \left[\left(\ln \frac{1}{\sigma\sqrt{2\pi}} \right) - \frac{(\tilde{f}(x_i) - y_i)^2}{2\sigma^2} \right]$$

$$= \arg\min_{\tilde{f}} \sum_{i=1}^{n} \frac{(\tilde{f}(x_i) - y_i)^2}{2\sigma^2}$$

$$= \arg\min_{\tilde{f}} \sum_{i=1}^{n} (\tilde{f}(x_i) - y_i)^2$$

Least-Squares Approximation

This shows:

- The solution with maximum likelihood in the considered scenario (y-direction, iid Gaussian noise) minimizes the sum of squared errors.

Next: Compute optimal coefficients

- Linear ansatz: $\tilde{f}(x) := \sum_{j=1}^{k} \lambda_j b_j(x)$
- Task: determine optimal λ_i

Maximum Likelihood Estimation

$$\boldsymbol{\lambda} := \left.\begin{pmatrix} \lambda_1 \\ \vdots \\ \lambda_k \end{pmatrix}\right\} k \text{ entries}, \quad \mathbf{b}(x) := \left.\begin{pmatrix} b_1(x) \\ \vdots \\ b_k(x) \end{pmatrix}\right\} k \text{ entries}, \quad \mathbf{b}_i := \left.\begin{pmatrix} b_i(x_1) \\ \vdots \\ b_i(x_n) \end{pmatrix}\right\} n \text{ entries}, \quad \mathbf{y} := \left.\begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}\right\} n \text{ entries}$$

$$\begin{aligned}
\arg\min_{\boldsymbol{\lambda}} \sum_{i=1}^{n} (\tilde{f}(x_i) - y_i)^2 &= \arg\min_{\boldsymbol{\lambda}} \sum_{i=1}^{n} \left[\left(\sum_{j=1}^{k} \lambda_j b_j(x_i) \right) - y_i \right]^2 \\
&= \arg\min_{\boldsymbol{\lambda}} \sum_{i=1}^{n} \left[\boldsymbol{\lambda}^{\mathrm{T}} \mathbf{b}(x_i) - y_i \right]^2 \\
&= \arg\min_{\boldsymbol{\lambda}} \left(\underbrace{\boldsymbol{\lambda}^{\mathrm{T}} \left[\sum_{i=1}^{n} \mathbf{b}(x_i) \mathbf{b}^{\mathrm{T}}(x_i) \right] \boldsymbol{\lambda}}_{\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x}} \underbrace{- 2 \sum_{i=1}^{n} y_i \boldsymbol{\lambda}^{\mathrm{T}} \mathbf{b}(x_i)}_{\mathbf{b}\mathbf{x}} + \underbrace{\sum_{i=1}^{n} y_i^2}_{c} \right)
\end{aligned}$$

$\Rightarrow$ *Quadratic optimization* problem

Critical Point

$$\boldsymbol{\lambda} := \left.\begin{pmatrix} \lambda_1 \\ \vdots \\ \lambda_k \end{pmatrix}\right\} k \text{ entries}, \quad \mathbf{b}(x) := \left.\begin{pmatrix} b_1(x) \\ \vdots \\ b_k(x) \end{pmatrix}\right\} k \text{ entries}, \quad \mathbf{b}_i := \left.\begin{pmatrix} b_i(x_1) \\ \vdots \\ b_i(x_n) \end{pmatrix}\right\} n \text{ entries}, \quad \mathbf{y} := \left.\begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}\right\} n \text{ entries}$$

$$\nabla_{\boldsymbol{\lambda}} \left(\boldsymbol{\lambda}^{\mathrm{T}} \left[\sum_{i=1}^{n} \mathbf{b}(x_i) \mathbf{b}^{\mathrm{T}}(x_i) \right] \boldsymbol{\lambda} - 2 \sum_{i=1}^{n} y_i \boldsymbol{\lambda}^{\mathrm{T}} \mathbf{b}(x_i) + \sum_{i=1}^{n} y_i^2 \right)$$

$$= 2 \left[\sum_{i=1}^{n} \mathbf{b}(x_i) \mathbf{b}^{\mathrm{T}}(x_i) \right] \boldsymbol{\lambda} - 2 \begin{pmatrix} \mathbf{y}^{\mathrm{T}} \mathbf{b}_1 \\ \vdots \\ \mathbf{y}^{\mathrm{T}} \mathbf{b}_k \end{pmatrix}$$

We obtain a linear system of equations:

$$\left[\sum_{i=1}^{n} \mathbf{b}(x_i) \mathbf{b}^{\mathrm{T}}(x_i) \right] \boldsymbol{\lambda} = \begin{pmatrix} \mathbf{y}^{\mathrm{T}} \mathbf{b}_1 \\ \vdots \\ \mathbf{y}^{\mathrm{T}} \mathbf{b}_k \end{pmatrix}$$

Critical Point

This can also be written as:

$$\begin{pmatrix} \langle \mathbf{b}_1, \mathbf{b}_1 \rangle & \cdots & \langle \mathbf{b}_1, \mathbf{b}_k \rangle \\ \vdots & \ddots & \vdots \\ \langle \mathbf{b}_k, \mathbf{b}_1 \rangle & \cdots & \langle \mathbf{b}_k, \mathbf{b}_k \rangle \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \vdots \\ \lambda_k \end{pmatrix} = \begin{pmatrix} \langle \mathbf{y}, \mathbf{b}_1 \rangle \\ \vdots \\ \langle \mathbf{y}, \mathbf{b}_k \rangle \end{pmatrix}$$

with:

$$\langle \mathbf{b}_i, \mathbf{b}_j \rangle := \sum_{t=1}^{n} b_i(x_t) \cdot b_j(x_t)$$

$$\langle \mathbf{y}, \mathbf{b}_i \rangle := \sum_{t=1}^{n} b_i(x_t) \cdot y_t$$

Summary (again)

Statistical model yields least-squares criterion:

$$\arg\min_{\tilde{f}} \sum_{i=1}^{n} (\tilde{f}(x_i) - y_i)^2$$

Linear function space leads to quadratic objective:

$$\tilde{f}(x) := \sum_{j=1}^{k} \lambda_j b_j(x) \quad \longrightarrow \quad \arg\min_{\boldsymbol{\lambda}} \sum_{i=1}^{n} \left[\left(\sum_{j=1}^{k} \lambda_j b_j(x_i) \right) - y_i \right]^2$$

Critical point: linear system

$$\begin{pmatrix} \langle \mathbf{b}_1, \mathbf{b}_1 \rangle & \cdots & \langle \mathbf{b}_1, \mathbf{b}_k \rangle \\ \vdots & \ddots & \vdots \\ \langle \mathbf{b}_k, \mathbf{b}_1 \rangle & \cdots & \langle \mathbf{b}_k, \mathbf{b}_k \rangle \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \vdots \\ \lambda_k \end{pmatrix} = \begin{pmatrix} \langle \mathbf{y}, \mathbf{b}_1 \rangle \\ \vdots \\ \langle \mathbf{y}, \mathbf{b}_k \rangle \end{pmatrix}$$

with:

$$\langle \mathbf{b}_i, \mathbf{b}_j \rangle := \sum_{t=1}^{n} b_i(x_t) \cdot b_j(x_t)$$

$$\langle \mathbf{y}, \mathbf{b}_i \rangle := \sum_{t=1}^{n} b_i(x_t) \cdot y_t$$

Variants

Weighted least squares:

- In case the data point's noise has different standard deviations σ at the different data points
- This gives a weighted least squares problem
- Noisier points have smaller influence

Same procedure as prev. slides...

$$
\begin{aligned}
\arg\max_{\tilde{f}} \prod_{i=1}^{n} N_\sigma(\tilde{f}(x_i) - y_i) &= \arg\max_{\tilde{f}} \prod_{i=1}^{n} \frac{1}{\sigma_i\sqrt{2\pi}} \exp\left(-\frac{(\tilde{f}(x_i)-y_i)^2}{2\sigma_i^2}\right) \\
&= \arg\max_{\tilde{f}} \log \prod_{i=1}^{n} \frac{1}{\sigma_i\sqrt{2\pi}} \exp\left(-\frac{(\tilde{f}(x_i)-y_i)^2}{2\sigma_i^2}\right) \\
&= \arg\max_{\tilde{f}} \sum_{i=1}^{n} \left[\left(\log \frac{1}{\sigma_i\sqrt{2\pi}}\right) - \frac{(\tilde{f}(x_i)-y_i)^2}{2\sigma_i^2}\right] \\
&= \arg\min_{\tilde{f}} \sum_{i=1}^{n} \frac{(\tilde{f}(x_i)-y_i)^2}{2\sigma_i^2} \\
&= \arg\min_{\tilde{f}} \sum_{i=1}^{n} \underbrace{\frac{1}{\sigma_i^2}}_{weights} (\tilde{f}(x_i)-y_i)^2
\end{aligned}
$$

Result

Linear system for the general case:

$$\begin{pmatrix} \langle \mathbf{b}_1, \mathbf{b}_1 \rangle & \cdots & \langle \mathbf{b}_1, \mathbf{b}_n \rangle \\ \vdots & \ddots & \vdots \\ \langle \mathbf{b}_n, \mathbf{b}_1 \rangle & \cdots & \langle \mathbf{b}_n, \mathbf{b}_n \rangle \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \vdots \\ \lambda_n \end{pmatrix} = \begin{pmatrix} \langle \mathbf{y}, \mathbf{b}_1 \rangle \\ \vdots \\ \langle \mathbf{y}, \mathbf{b}_n \rangle \end{pmatrix}$$

with:

$$\langle \mathbf{b}_i, \mathbf{b}_j \rangle := \sum_{l=1}^{n} b_i(x_i) \cdot b_j(x_i) \cdot \boxed{\omega^2(x_i)}$$

$$\langle \mathbf{y}, \mathbf{b}_i \rangle := \sum_{l=1}^{n} b_i(x_i) \cdot y_i \cdot \boxed{\omega^2(x_i)}$$

$$\omega^2(x_i) = \frac{1}{\sigma_i^2}, \quad \text{i.e. } \omega(x_i) = \frac{1}{\sigma_i}$$

Larger $\omega \rightarrow$ larger influence of data point

Least-Squares Linear Systems

Remark:

- We get the same result, if we solve an overdetermined system for the interpolation problem in a least squares sense
- Least-squares solution to linear system:

$$\mathbf{Ax} = \mathbf{b}$$

$$\rightarrow \arg\min_{\mathbf{x}} (\mathbf{Ax} - \mathbf{b})^2$$

$$= \arg\min_{\mathbf{x}} \left(\mathbf{x}^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}\mathbf{Ax} - 2\mathbf{Ax}\cdot\mathbf{b} + \mathbf{b}^{\mathrm{T}}\mathbf{b}\right)$$

$$\text{compute gradient :}$$

$$\rightarrow 2\mathbf{A}^{\mathrm{T}}\mathbf{Ax} = 2\mathbf{A}^{\mathrm{T}}\mathbf{b},\ \text{i.e.:}\ \mathbf{A}^{\mathrm{T}}\mathbf{Ax} = \mathbf{A}^{\mathrm{T}}\mathbf{b}$$

- “System of normal equations”

SVD

Problem with normal equations:

- Condition number of normal equations is square of that of A itself
- Proof:

$$\text{SVD}: \ \mathbf{A} = \mathbf{UDV}$$

$$\mathbf{A}^{\mathrm{T}}\mathbf{A} = \mathbf{V}^{\mathrm{T}}\mathbf{D}\mathbf{U}^{\mathrm{T}}\mathbf{U}\mathbf{D}\mathbf{V} = \mathbf{V}^{\mathrm{T}}\mathbf{D}^{2}\mathbf{V}$$

- For "evil" (i.e. ill conditioned) problems, normal equations are not the best way to solve the problem
- In that case, we can use the SVD to solve the problem...

Least-Squares with SVD

Compute singular value decomposition, then:

$$\mathbf{A} = \mathbf{UDV}$$

$$\mathbf{A}^{\mathrm{T}}\mathbf{Ax} = \mathbf{A}^{\mathrm{T}}\mathbf{b}$$

$$\mathbf{V}^{\mathrm{T}}\mathbf{DU}^{\mathrm{T}}\mathbf{UDVx} = \mathbf{V}^{\mathrm{T}}\mathbf{DU}^{\mathrm{T}}\mathbf{b}$$

$$\Leftrightarrow \mathbf{V}^{\mathrm{T}}\mathbf{D}^{2}\mathbf{Vx} = \mathbf{V}^{\mathrm{T}}\mathbf{DU}^{\mathrm{T}}\mathbf{b}$$

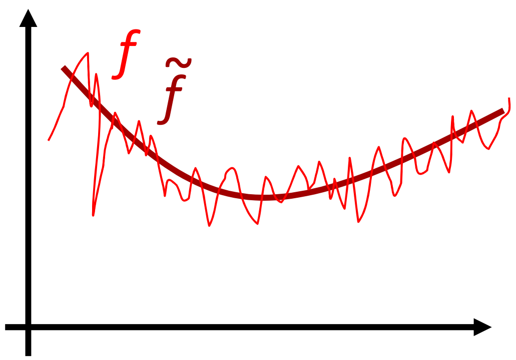
$$\Leftrightarrow \mathbf{D}^{2}\mathbf{Vx} = \mathbf{DU}^{\mathrm{T}}\mathbf{b}$$

$$\Leftrightarrow \mathbf{DVx} = \mathbf{U}^{\mathrm{T}}\mathbf{b}$$

$$\Leftrightarrow \mathbf{x} = \mathbf{V}^{T}\mathbf{D}^{-1}\mathbf{U}^{\mathrm{T}}\mathbf{b}$$

If $\mathbf{D}$ is not invertible (not full rank), inverting the non-zero entries only yields the least-squares solution of minimal norm (critical point with $\|\mathbf{x}\|$ minimal).

One more Variant...



Function Approximation

- Given the following problem:
  - We know a function f: $\Omega \supseteq \mathbb{R}^n \to \mathbb{R}$
  - We want to approximate f in a linear subspace: $\tilde{f}(x) := \sum_{j=1}^{k} \lambda_j b_j(x)$
  - How to choose λ?
- Difference: Continuous function as “data” to be matched.
- Solution: Almost the same as before...

Function Approximation

Objective function:

- $\left\| \tilde{f}(x) - f \right\|^2 \to \min$
- We obtain:

$$\left\| \sum_{j=1}^{k} \lambda_j b_j(x) - f \right\|^2 = \left\langle \sum_{j=1}^{k} \lambda_j b_j(x) - f, \sum_{j=1}^{k} \lambda_j b_j(x) - f \right\rangle$$

$$= \boldsymbol{\lambda}^{\mathrm{T}} \begin{pmatrix} \langle b_1, b_1 \rangle & \cdots & \langle b_n, b_1 \rangle \\ \vdots & \ddots & \vdots \\ \langle b_1, b_n \rangle & \cdots & \langle b_n, b_n \rangle \end{pmatrix} \boldsymbol{\lambda} - 2 \sum_{j=1}^{k} \lambda_j \langle b_j(x), f \rangle + \langle f, f \rangle$$

Function Approximation

Critical point (i.e., solution):

$$\begin{pmatrix} \langle b_1, b_1 \rangle & \cdots & \langle b_k, b_1 \rangle \\ \vdots & \ddots & \vdots \\ \langle b_1, b_k \rangle & \cdots & \langle b_k, b_k \rangle \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \vdots \\ \lambda_k \end{pmatrix} = \begin{pmatrix} \langle b_1(x), f \rangle \\ \vdots \\ \langle b_k(x), f \rangle \end{pmatrix}$$

with:

$\langle f, g \rangle = \int_\Omega f(x) g(x) dx$ (unweighted version)

$\langle f, g \rangle_\omega = \int_\Omega f(x) g(x) \omega^2(x) dx$ (weighted version)

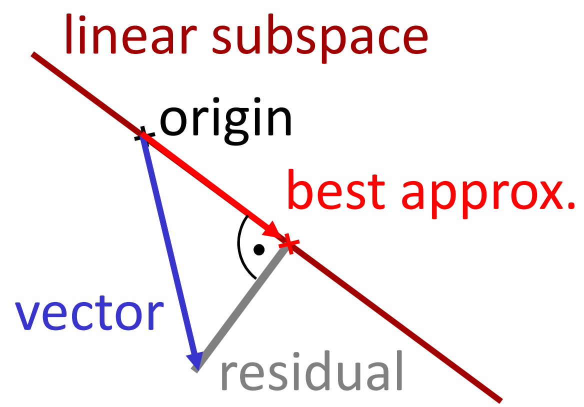
Galerkin Approximation

The least-squares criterion is equivalent to:

$$\forall i \in \{1..k\}: \Big\langle \underbrace{\sum_{j=1}^{k} \lambda_j b_j(x) - f}_{\text{residual}}, \underbrace{b_i(x)}_{\text{each basis function}} \Big\rangle = 0$$

$$\Leftrightarrow \forall i \in \{1..k\}: \Big\langle \sum_{j=1}^{k} \lambda_j b_j(x), b_i(x) \Big\rangle = \langle f, b_i(x) \rangle$$

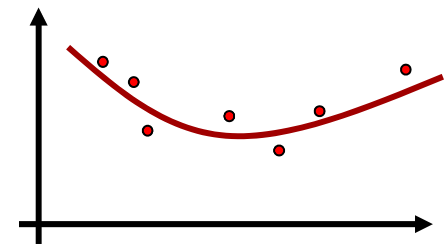
$$\Leftrightarrow \begin{pmatrix} \langle b_1, b_1 \rangle & \cdots & \langle b_n, b_1 \rangle \\ \vdots & \ddots & \vdots \\ \langle b_1, b_n \rangle & \cdots & \langle b_n, b_n \rangle \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \vdots \\ \lambda_n \end{pmatrix} = \begin{pmatrix} \langle b_1(x), f \rangle \\ \vdots \\ \langle b_n(x), f \rangle \end{pmatrix}$$



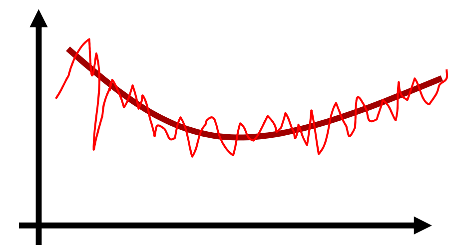
Summary

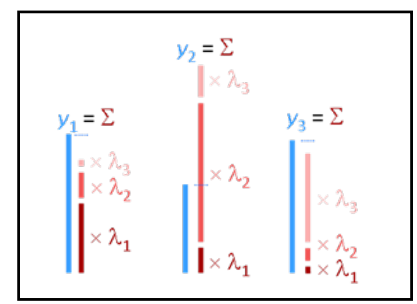
What we can do so far:

- Least-squares approximation:
  - Given more data points than basis functions, we can fit an approximate function from a basis function set to the data



- Variants:
  - We can solve linear systems in a least-squares sense
  - Given a function, we can fit the most similar approximation from a subspace





- Extensions:
  - Any known uncertainty in the data can be modeled by weights
  - The multi-dimensional case is similar

Remaining problems

What is missing:

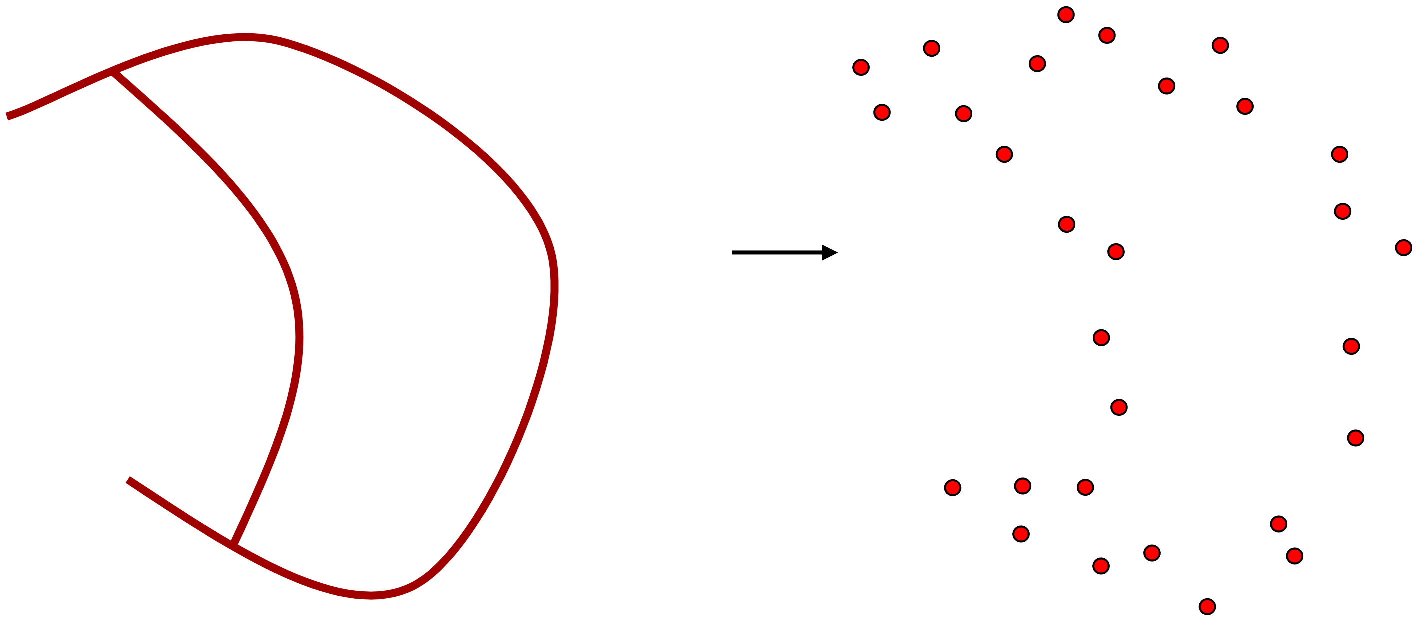
- Any error in **x**-direction is ignored so far (only **y**-direction)
  - We will look at that problem next (total least-squares)...
- Noise must be Gaussian
  - Can be generalized using iteratively reweighted least-squares (M-estimators)

Approximation

Total Least Squares

Statistical Model

Generative Model:



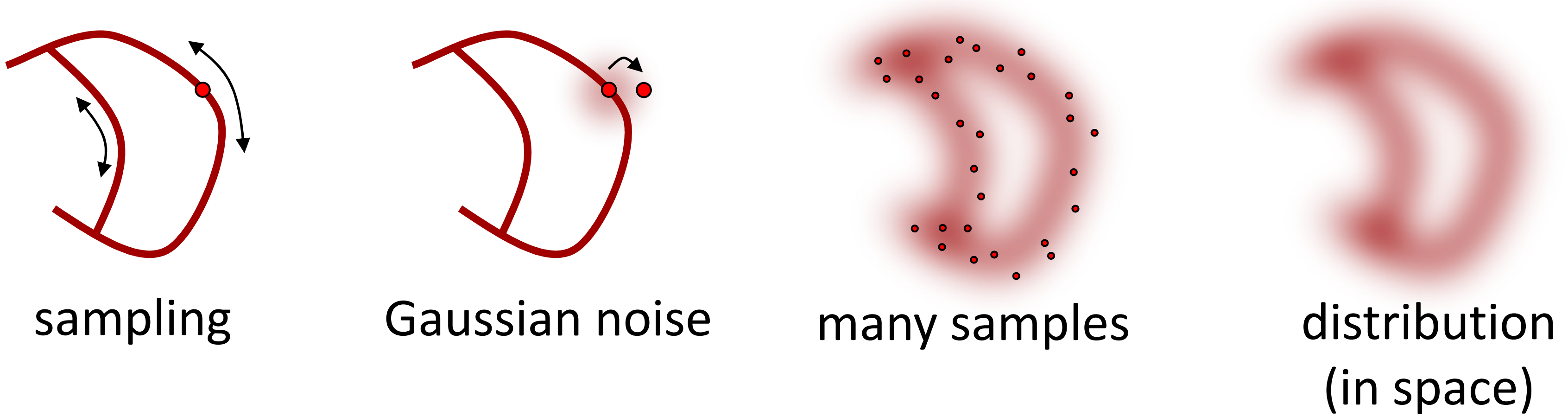
original curve / surface

noisy sample points

Statistical Model

Generative Model:

1. Determine sample point (uniform)
2. Add noise (Gaussian)



Squared Distance Function

Result:

- Gaussian distribution convolved with object
- No analytical density

Approximation:

- 1D Gaussian $\rightarrow$ minimize squared residual
- This case $\rightarrow$ minimize squared distance function

General Total Least Squares

General Total Least Squares:

- Given a class of objects *obj* with parameters $\lambda \in \mathbb{R}^k$.
- A set of *n* sample points (Gaussian, iid, isotropic covariance) $d_i \in \mathbb{R}^m$.
- Total least squares solution minimizes:

$$\underset{\boldsymbol{\lambda} \in \mathbb{R}^k}{\arg\min} \sum_{i=1}^{n} dist\left(obj_{\boldsymbol{\lambda}}, d_i\right)^2$$

- In general: Non-linear, possibly constrained (restrictions on admissible λs) optimization problem
- Special cases can be solved exactly

Fitting Affine Subspaces

The following problem can be solved exactly:

- Best fitting line to a set of 2D, 3D points
- Best fitting plane to a set of 3D points
- In general: Affine subspace of $\mathbb{R}^m$, with dimension $d \leq m$ that best approximates a set of data points $\mathbf{d}_i \in \mathbb{R}^m$.

This will lead to the - famous - ***principle component analysis (PCA)*****.**

Start: 0-dim Subspaces

Easy Start: The optimal 0-dimensional affine subspace

- Given a set **D** of n data points $\mathbf{d}_i \in \mathbb{R}^m$, what is the point $\mathbf{x}_0$ with minimum least square error to all data points?
- Answer: just the sample mean (average)...:

$$\mathbf{x}_0^{(opt)} = \mathrm{m}(\mathbf{D}) := \frac{1}{n}\sum_{i=1}^{n}\mathbf{d}_i$$

- Proof: minimize

$$E(\mathbf{x}_0) = \sum_{i=1}^{n}\left\|\mathbf{x}_0 - \mathbf{d}_i\right\|^2$$

(next slide...)

Proof

$$E(\mathbf{x}_0) = \sum_{i=1}^{n} \left\| \mathbf{x}_0 - \mathbf{d}_i \right\|^2$$

sample mean:

$$\mathrm{m}(\mathbf{D}) := \frac{1}{n} \sum_{i=1}^{n} \mathbf{d}_i$$

Proof

sample mean:

$$\mathrm{m}(\mathbf{D}) := \frac{1}{n}\sum_{i=1}^{n}\mathbf{d}_i$$

$$E(\mathbf{x}_0) = \sum_{i=1}^{n}\left\|\mathbf{x}_0 - \mathbf{d}_i\right\|^2$$

$$= \sum_{i=1}^{n}\left\|\mathbf{x}_0 - \mathrm{m}(\mathbf{D}) + \mathrm{m}(\mathbf{D}) - \mathbf{d}_i\right\|^2$$

$$= \sum_{i=1}^{n}\left\|\mathbf{x}_0 - \mathrm{m}(\mathbf{D})\right\|^2 - 2\sum_{i=1}^{n}\left\langle \mathbf{x}_0 - \mathrm{m}(\mathbf{D}), \mathrm{m}(\mathbf{D}) - \mathbf{d}_i \right\rangle + \sum_{i=1}^{n}\left\|\mathrm{m}(\mathbf{D}) - \mathbf{d}_i\right\|^2$$

$$= \sum_{i=1}^{n}\left\|\mathbf{x}_0 - \mathrm{m}(\mathbf{D})\right\|^2 - 2\left\langle \mathbf{x}_0 - \mathrm{m}(\mathbf{D}), \sum_{i=1}^{n}\mathrm{m}(\mathbf{D}) - \mathbf{d}_i \right\rangle + \sum_{i=1}^{n}\left\|\mathrm{m}(\mathbf{D}) - \mathbf{d}_i\right\|^2$$

$$= \sum_{i=1}^{n}\left\|\mathbf{x}_0 - \mathrm{m}(\mathbf{D})\right\|^2 - 2\left\langle \mathbf{x}_0 - \mathrm{m}(\mathbf{D}), n \cdot \mathrm{m}(\mathbf{D}) - \sum_{i=1}^{n}\mathbf{d}_i \right\rangle + \sum_{i=1}^{n}\left\|\mathrm{m}(\mathbf{D}) - \mathbf{d}_i\right\|^2$$

$$= \sum_{i=1}^{n}\left\|\mathbf{x}_0 - \mathrm{m}(\mathbf{D})\right\|^2 - 2\left\langle \mathbf{x}_0 - \mathrm{m}(\mathbf{D}), \sum_{i=1}^{n}\mathbf{d}_i - \sum_{i=1}^{n}\mathbf{d}_i \right\rangle + \sum_{i=1}^{n}\left\|\mathrm{m}(\mathbf{D}) - \mathbf{d}_i\right\|^2$$

$$= \sum_{i=1}^{n}\underbrace{\left\|\mathbf{x}_0 - \mathrm{m}(\mathbf{D})\right\|^2}_{\text{minimum for } \mathbf{x}_0 = \mathrm{m}(\mathbf{D})} + \underbrace{\sum_{i=1}^{n}\left\|\mathrm{m}(\mathbf{D}) - \mathbf{d}_i\right\|^2}_{\text{independent of } \mathbf{x}_0}$$

One Dimensional Subspaces...

Next:

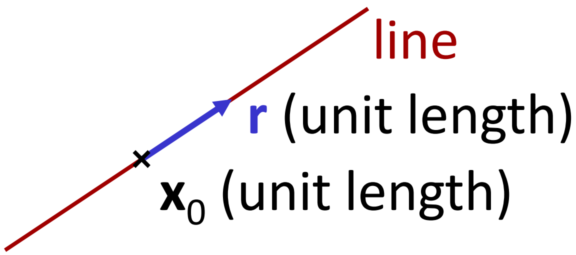
- What is the optimal line (1D subspace) that approximates a set of data points $\mathbf{d}_i$?
- Two questions:
  - Optimum origin (point on the line)?
    - This is still the average (proof omitted)
  - Optimum direction?
    - We will look at that next...
- Parametric line equation:

$$\mathbf{x}(t) = \mathbf{x}_0 + t \cdot \mathbf{r} \quad (\mathbf{x}_0 \in \mathbb{R}^m, \mathbf{r} \in \mathbb{R}^m, \|\mathbf{r}\| = 1)$$

Best Fitting Line

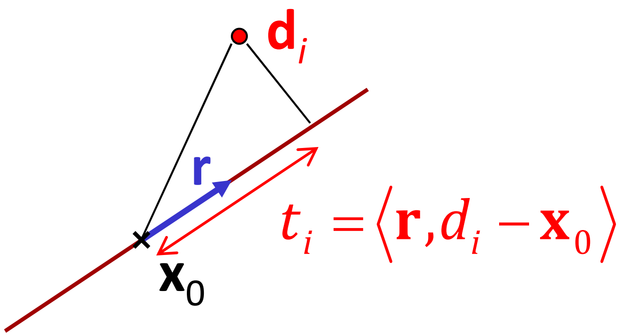
Line equation:

$$\mathbf{x}(t) = \mathbf{x}_0 + t \cdot \mathbf{r} \quad (\mathbf{x}_0 \in \mathbb{R}^m, \mathbf{r} \in \mathbb{R}^m, \|\mathbf{r}\| = 1)$$



Best projection on any line:

$$t_i = \langle \mathbf{r}, d_i - \mathbf{x}_0 \rangle$$



Objective Function:

$$\sum_{i=1}^{n} dist(line, \mathbf{d}_i)^2 = \sum_{i=1}^{n} \left([\mathbf{x}_0 + t_i \mathbf{r}] - \mathbf{d}_i \right)^2$$

Best Fitting Line

Objective Function:

$$\sum_{i=1}^{n} dist(line, \mathbf{d}_i)^2 = \sum_{i=1}^{n} \left([\mathbf{x}_0 + t_i \mathbf{r}] - \mathbf{d}_i \right)^2$$

optimal parameters t_i:

$$t_i = \langle \mathbf{r}, d_i - \mathbf{x}_0 \rangle$$

Best Fitting Line

Objective Function:

optimal parameters t_i:

$$t_i = \langle \mathbf{r}, d_i - \mathbf{x}_0 \rangle$$

$$\sum_{i=1}^{n} dist(line, \mathbf{d}_i)^2 = \sum_{i=1}^{n} \left([\mathbf{x}_0 + t_i \mathbf{r}] - \mathbf{d}_i \right)^2 = \sum_{i=1}^{n} \left(t_i \mathbf{r} - [\mathbf{d}_i - \mathbf{x}_0] \right)^2$$

$$= \sum_{i=1}^{n} t_i^2 \mathbf{r}^2 - 2 \sum_{i=1}^{n} t_i \langle \mathbf{r}, d_i - \mathbf{x}_0 \rangle + \sum_{i=1}^{n} [\mathbf{d}_i - \mathbf{x}_0]^2$$

$$= \sum_{i=1}^{n} \langle \mathbf{r}, d_i - \mathbf{x}_0 \rangle^2 - 2 \sum_{i=1}^{n} \langle \mathbf{r}, d_i - \mathbf{x}_0 \rangle^2 + \sum_{i=1}^{n} [\mathbf{d}_i - \mathbf{x}_0]^2$$

$$= -\sum_{i=1}^{n} \left(\mathbf{r}^{\mathrm{T}} (d_i - \mathbf{x}_0) \right)^2 + \sum_{i=1}^{n} [\mathbf{d}_i - \mathbf{x}_0]^2$$

$$= -\mathbf{r}^{\mathrm{T}} \underbrace{\left[\sum_{i=1}^{n} (d_i - \mathbf{x}_0)(d_i - \mathbf{x}_0)^{\mathrm{T}} \right]}_{\text{Matrix},=:\mathbf{S}} \mathbf{r} + \underbrace{\sum_{i=1}^{n} [\mathbf{d}_i - \mathbf{x}_0]^2}_{\text{const. w.r.t.}\,\mathbf{r}}$$

Best Fitting Line

Result:

$$\sum_{i=1}^{n} dist(line,\mathbf{d}_i)^2 = -\mathbf{r}^{\mathrm{T}}\mathbf{S}\mathbf{r} + const.$$

$$\text{with}: \mathbf{S} = \sum_{i=1}^{n} (d_i - \mathbf{x}_0)(d_i - \mathbf{x}_0)^{\mathrm{T}}, \ \|\mathbf{r}\| = 1$$

Eigenvalue Problem:

- $\mathbf{r}^{\mathrm{T}}\mathbf{S}\mathbf{r}$ is a Rayleigh quotient
- Minimizing the energy: maximum quotient
- Solution: eigenvector with *largest* eigenvalue

General Case

Fitting a *d*-dimensional affine subspace:

- $d = 1$: line
- $d = 2$: plane
- $d = 3$: 3D subspace
- ...

Simple rule:

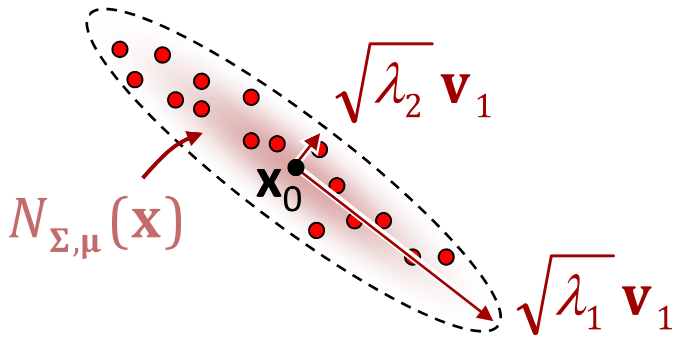
- Use the d eigenvectors with the *largest eigenvalues* from the spectrum of S.
- Gives the (total) least-squares optimal subspace that approximates the data set **D**.

General Case

Procedure: Principal Component Analysis (PCA)

- Compute average $\mathbf{x}_0 = \mathrm{m}(\mathbf{D})$
- Compute "scatter matrix" $\mathbf{S} = \sum_{i=1}^{n} (d_i - \mathbf{x}_0)(d_i - \mathbf{x}_0)^{\mathrm{T}}$
- Let $(\lambda_1, \mathbf{v}_1), \ldots, (\lambda_n, \mathbf{v}_n)$ be sorted eigenvalue/vector pairs of $\mathbf{S}$, where λ_1 is the largest, and the $\mathbf{v}_i$ are of unit length.
- The subspace spanned by $p(t_1, \ldots, t_d) = x_0 + \sum_{i=1}^{d} (t_i \mathbf{v}_i)$ approximates the data optimally in terms of squared distances to a point in the subspace.
- Stronger: projecting the data into this subspace is the best d-dimensional (affine subspace) data approximation.

Statistical Interpretation



$$\boldsymbol{\mu} = \mathbf{x}_0 = \frac{1}{n}\sum_{i=1}^{n} d_i$$

$$\tilde{\boldsymbol{\Sigma}} = \frac{1}{n-1}\mathbf{S} = \frac{1}{n-1}\sum_{i=1}^{n}(\mathbf{d}_i - \mathbf{x}_0)(\mathbf{d}_i - \mathbf{x}_0)^{\mathrm{T}}$$

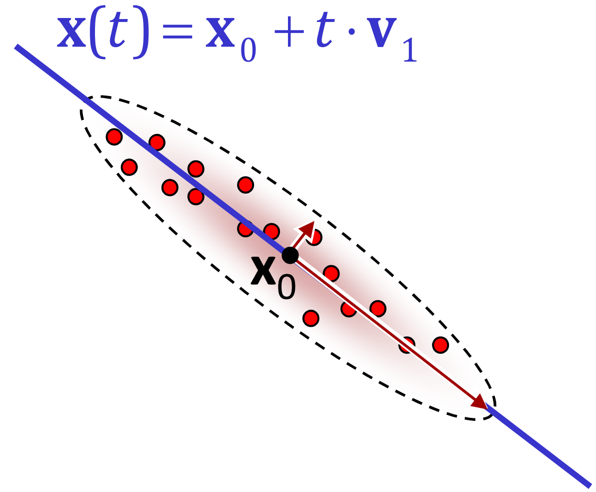
$$N_{\boldsymbol{\Sigma},\boldsymbol{\mu}}(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}\det(\boldsymbol{\Sigma})^{1/2}}\exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right)$$

Observation:

- **S**/(n-1) is the covariance matrix of the data set **D** = {$\mathbf{d}_i$}$_i$
- PCA can be interpreted as fitting a Gaussian distribution and computing the main axes

Applications

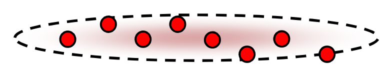
Fitting a line to a point cloud in $\mathbb{R}^2$:



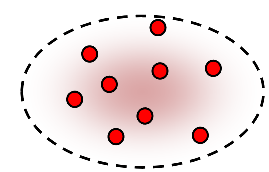
- Sample mean and direction of maximum eigenvalue

Plane Fitting in $\mathbb{R}^3$:

- Sample mean and the two directions of maximum eigenvalues
- Smallest eigenvalue
  - Eigenvector points in normal direction
  - Aspect ratio (λ_3 / λ_2) is a measure of “flatness” (quality of fit)



(λ_2 / λ_1) small



(λ_2 / λ_1) larger

Applications

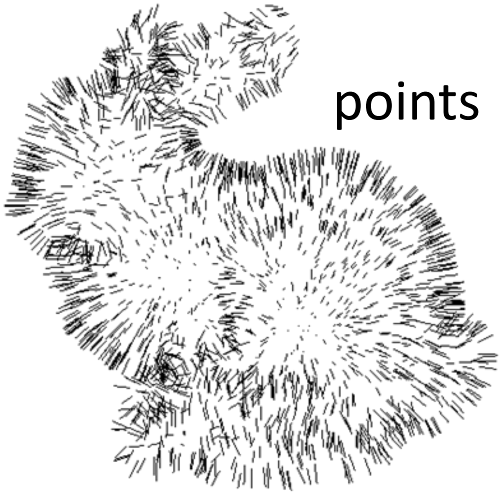
Application: Normal estimation in point clouds

- Given a set of points $p_i \in \mathbb{R}^3$ that form a smooth surface.
- We want to estimate:
  - Surface normals
  - Sampling spacing

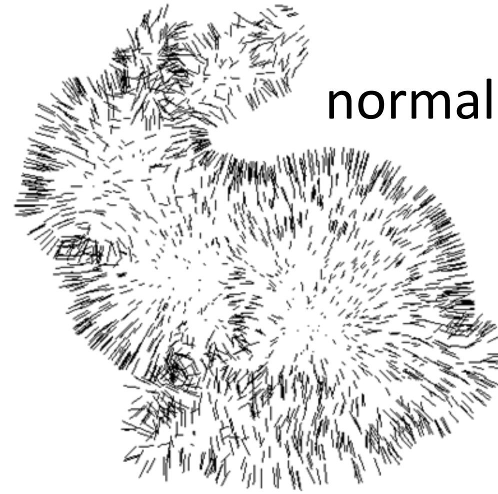
Algorithm:

- For each point, compute the k nearest neighbors ($k \approx 20$)
- Compute a PCA (average, main axes) of these points
  - Eigenvector with smallest eigenvalue $\rightarrow$ normal direction
  - The other two eigenvectors $\rightarrow$ tangent vectors
  - Tangent eigenvalues give sample spacing estimate

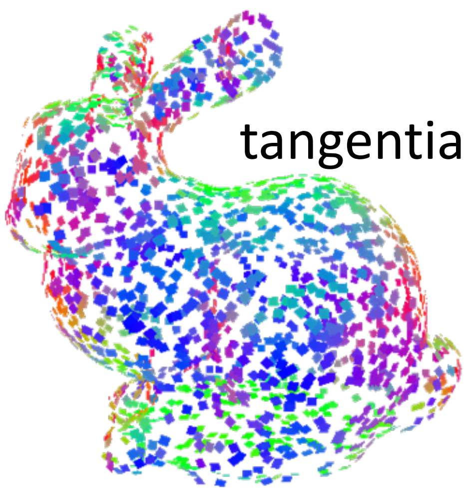
Example



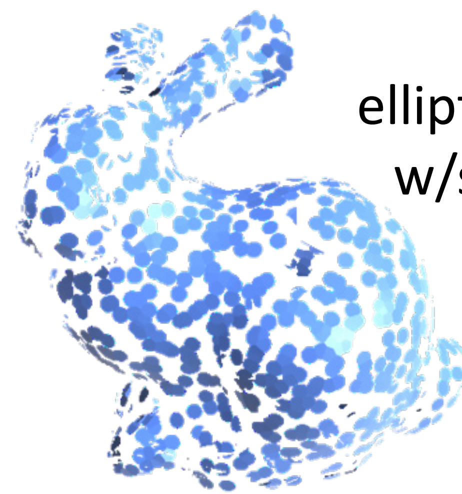
points



normals

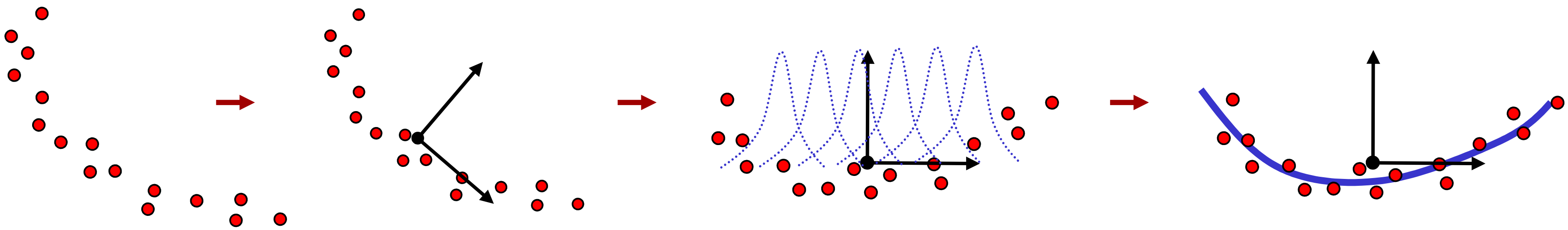


tangential frames



elliptic splats
w/shading

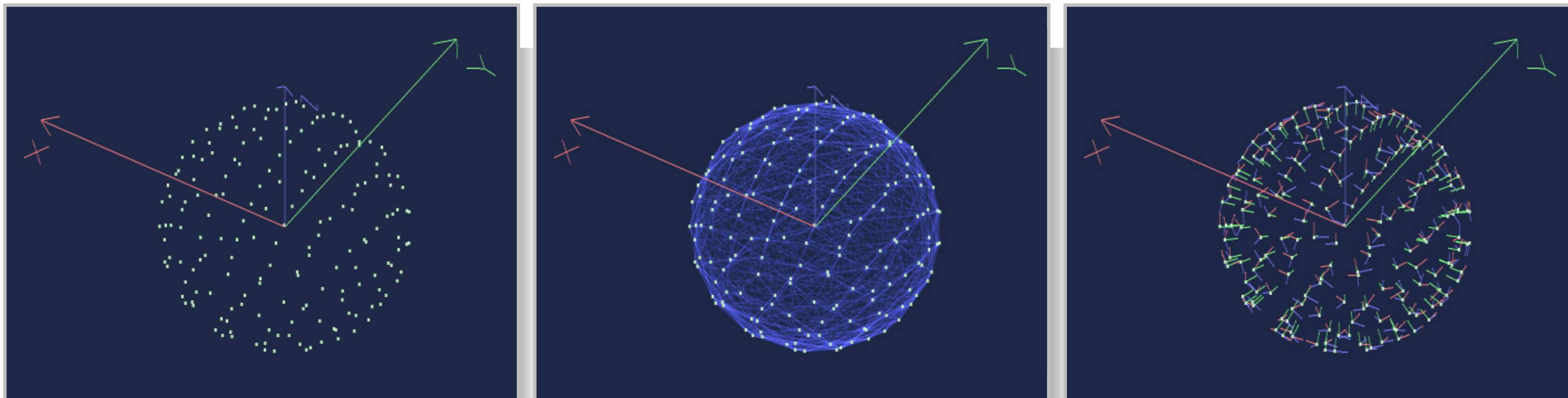
Applications



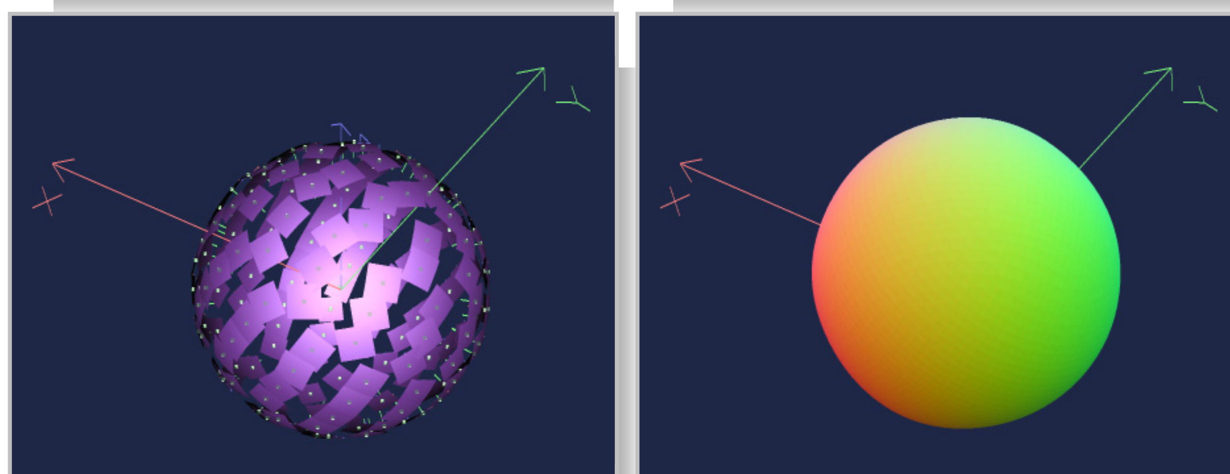
Another Application: Coordinate frame estimation

- More general total least-squares (non-linear) is difficult
- However: *We can tweak it*
    - Compute coordinate frames using PCA
    - Smallest eigenvalue = normal direction
    - Form height field in normal direction
    - Then: use ordinary least-squares in this coordinate system

Example



Example:
- *k*-nearest neighbors
- PCA coordinate frames at *each* point
- Quadratic monomials (bivariate, local coords.)
- Least squares fit

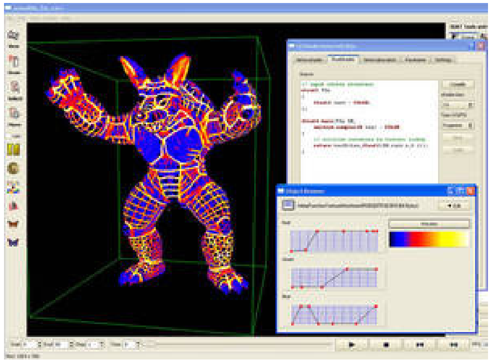


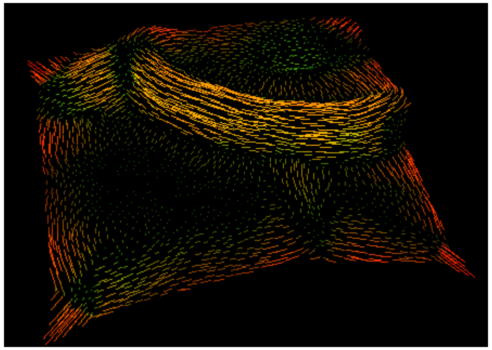
Curvature Estimation

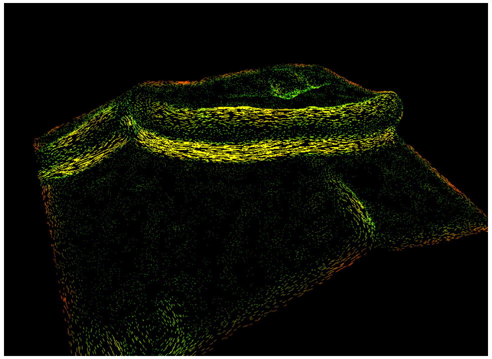
Estimating curvature in point clouds:

- Quadratic fitting (as in the example)
  - PCA coordinate frame (u, v, n)
  - Height field in normal direction $(u, v) \to n$
  - Fit quadratic polynomials $\{1, u, v, uv, u^2, v^2\}$
- Eigenanalysis of the quadratic terms
  - Hessian matrix from fitted polynomial

    $$2\lambda_{uv}uv + \lambda_{uu}u^2 + \lambda_{vv}v^2 \to \begin{pmatrix} \lambda_{uu} & \lambda_{uv} \\ \lambda_{uv} & \lambda_{vv} \end{pmatrix}$$
  - Compute principal directions
  - Mean / Gauss / Normal curvature

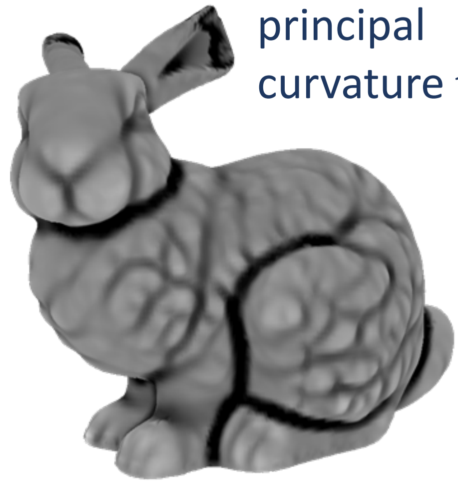




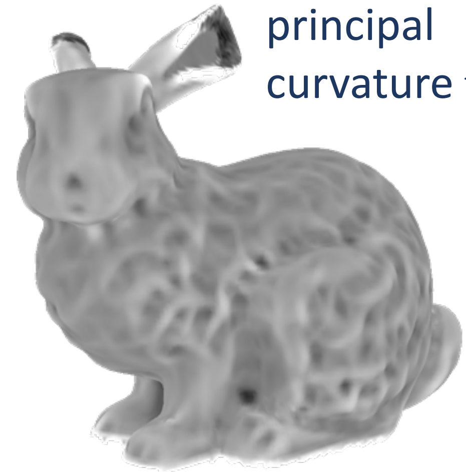


Bunny Curvature

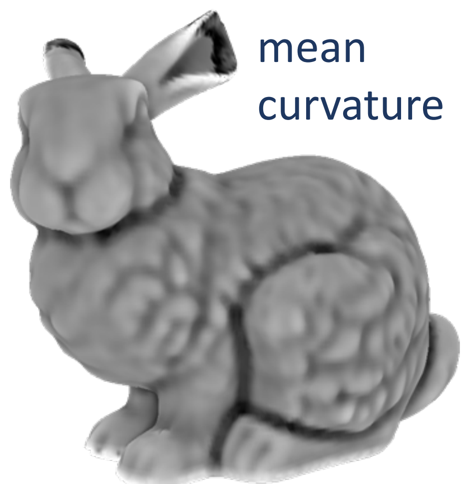
Stanford Bunny
(dense point cloud)



principal
curvature κ_1



principal
curvature κ_2



mean
curvature

Gaussian
curvature

[courtesy of Martin Bokeloh]

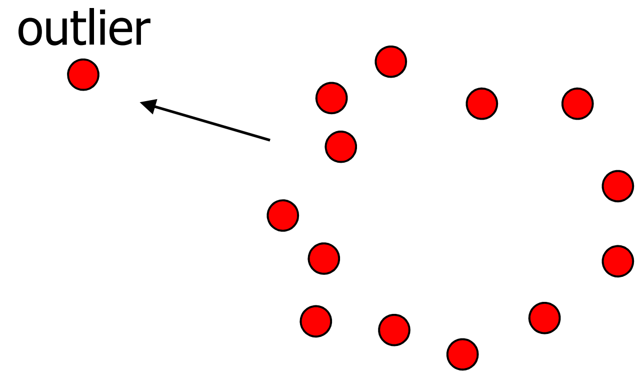
Approximation

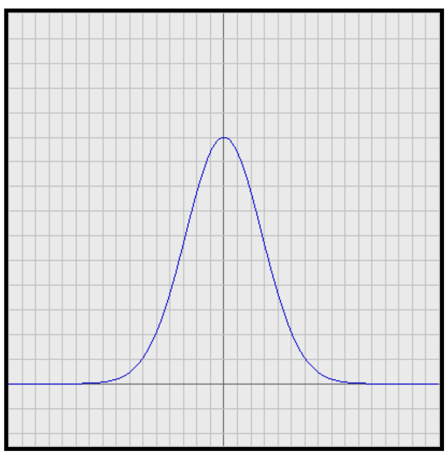
Iteratively Reweighted Least Squares

General Error Distributions

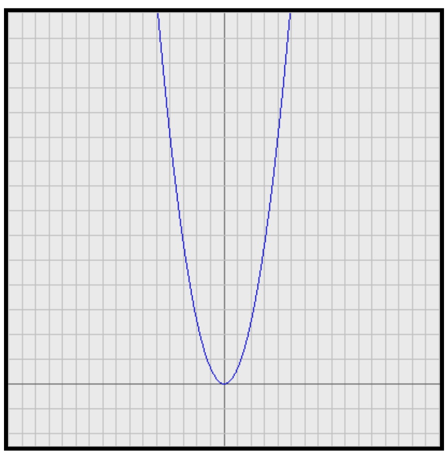
Problem with least-squares

- Quadratic error measure
- The farther away, the stronger the attraction force
- Outliers are disastrous

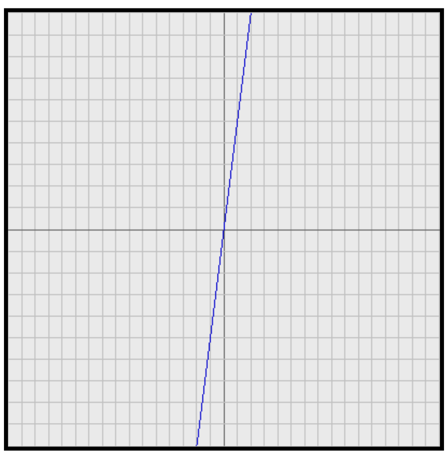




Gaussian likelihood

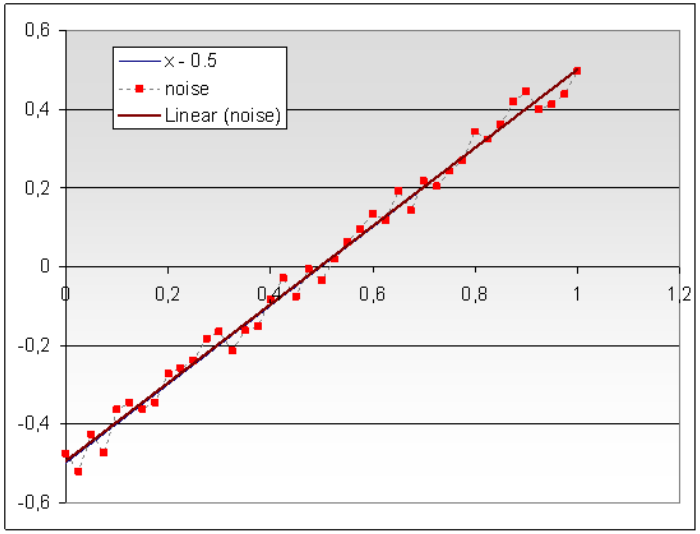


neg. log-likelihood

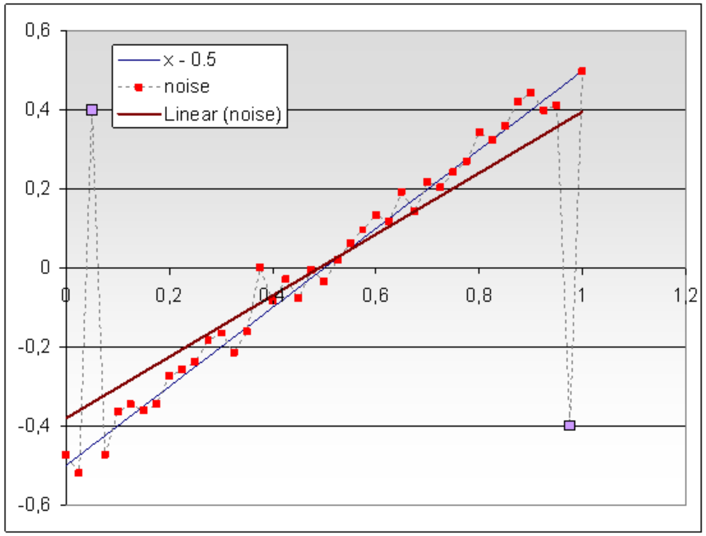


derivative

Outliers



uniform noise

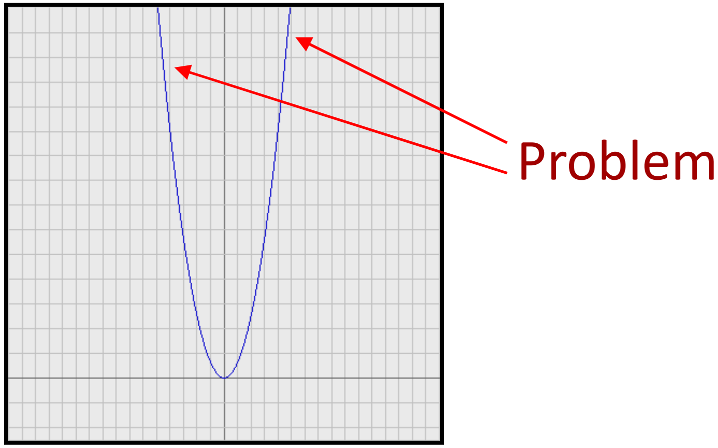


2 outliers (out of 41)

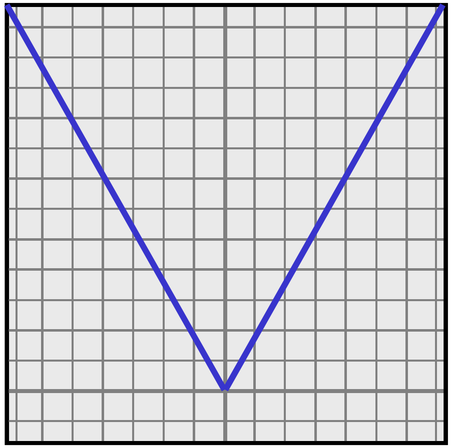
Problem:

- Outliers are rather common
  - 3D scanners: Weird outlier points at random locations (Shiny surfaces, transmission errors, etc...)
- Least squares does not work well

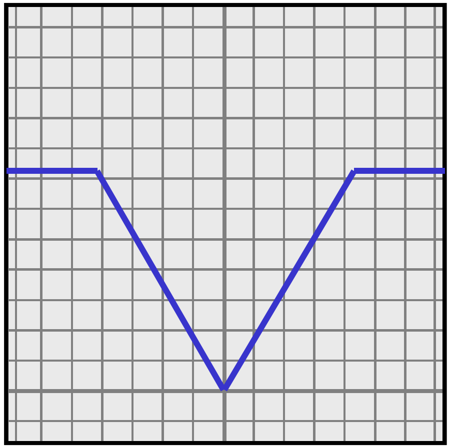
Robust Estimators



squared distance
("l_2-norm error")



absolute distance
("l_1-norm error")



truncated

Problem:

- Least squares criterion too strict
- More robust: absolute distance ("l_1-norm")
- "M-estimator": Truncated squared/absolute distance

Implementation

Implementation: Iteratively reweighted least-squares

- Reminder: weighted least-squares

$$\underset{\tilde{f}}{\arg\min} \sum_{i=1}^{n} \underbrace{\omega_i}_{\text{weights}} (\tilde{f}(x_i) - y_i)^2$$

- Iteratively reweighted least-squares:
  - Compute least-squares fit
  - Compute weights (dependent on solution)
  - Recompute / iterate until convergence
- L1 norm (absolute distance) weights:

$$\boxed{\omega_i = \frac{1}{\left\| \tilde{f}^{(k-1)}(x_i) - y_i \right\|}} \rightarrow \underset{\tilde{f}^{(k)}}{\arg\min} \sum_{i=1}^{n} \underbrace{\omega_i}_{\text{weights}} (\tilde{f}^{(k)}(x_i) - y_i)^2 \quad \text{(iteration k)}$$

L1-Norm Fitting

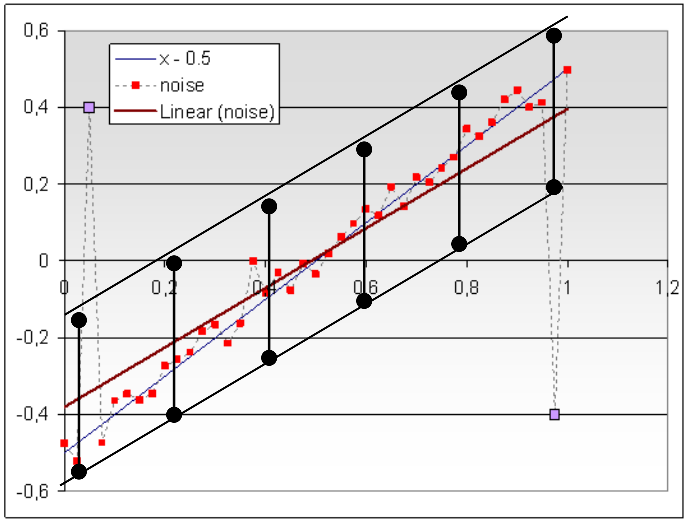
L1-Norm Fitting:

- $$\arg\min_{\tilde{f}^{(k)}} \sum_{i=1}^{n} \frac{1}{\left\| \tilde{f}^{(k-1)}(x_i) - y_i \right\|} \left(\tilde{f}^{(k)}(x_i) - y_i \right)^2 \rightarrow \arg\min_{\tilde{f}^{(k)}} \sum_{i=1}^{n} \left\| \tilde{f}^{(k-1)}(x_i) - y_i \right\|$$
- Convergence guaranteed (convex objective function)
- Mixture of l_1 (far) and l_2 (near) norm also works

General M-Estimators

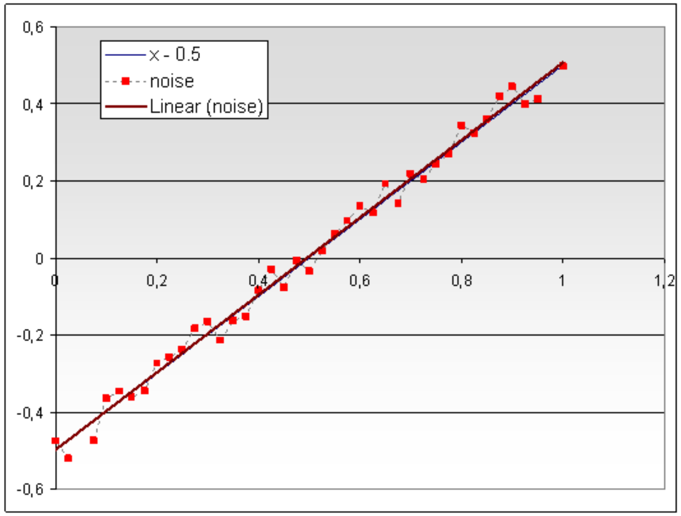
- No convergence guarantees for non-convex potentials
- In particular: Truncated potentials may converge to a local extremum only (depends on initialization)

Example (Schematic)



first iteration:

least squares, then truncation



second iteration:

improved solution