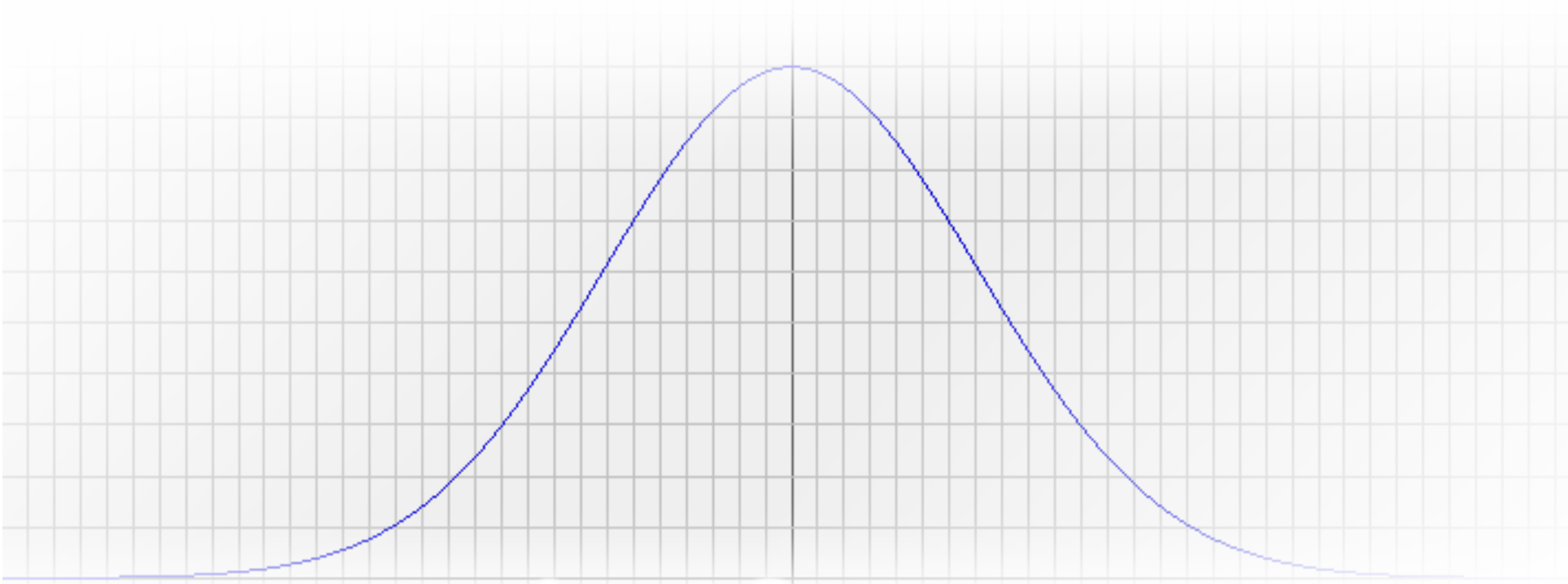
Statistical Geometry Processing

Winter Semester 2011/2012



Least-Squares

UNIVERSITÄT DES SAARLANDES

CLUSTER OF EXCELLENCE

max planck institut informatik

Least-Squares Fitting

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:

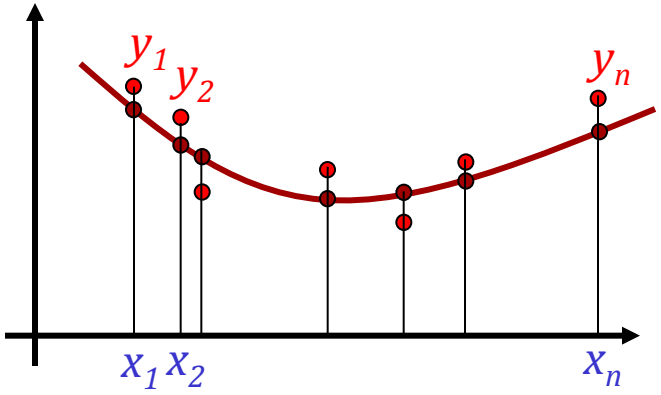
- Least-squares approximation
  (and why/when this makes sense)
- Total least-squares linear approximation
  (get rid of the coordinate system)
- Iteratively reweighted least-squares
  (for nasty noise distributions)

Least-Squares

We assume the following scenario:

- Given: Function values y_i at positions x_i.
  (1D $\rightarrow$ 1D for now)
- Independent variables x_i known exactly.
- Dependent variables y_i with some error.
- Error Gaussian, i.i.d.
  - *normal distributed*
  - *independent*
  - *same distribution* at every point
- We know the class of functions

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:

- Maximum likelihood estimate minimizes sum of squared errors

Next: Compute optimal coefficients

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

Maximum Likelihood Estimation

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

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

$\Rightarrow$ *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...

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

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

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

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

$$= \arg\min_{\tilde{f}} \sum_{i=1}^{n} \underbrace{\frac{1}{\sigma_i^2}}_{\text{weights}} (\tilde{f}(x_i) - y_i)^2$$

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:

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

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

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

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:

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

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

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

$$\text{compute gradient :}$$

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

- "System of normal equations"

SVD

Problem with normal equations:

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

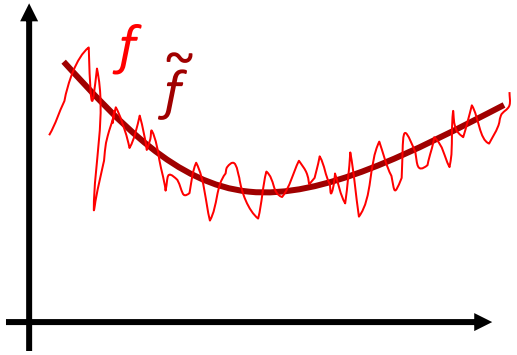
$$\text{SVD}: \mathbf{A} = \mathbf{UDV}$$

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

- 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...

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 $\boldsymbol{\lambda}$?
- 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:

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

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

Galerkin Approximation

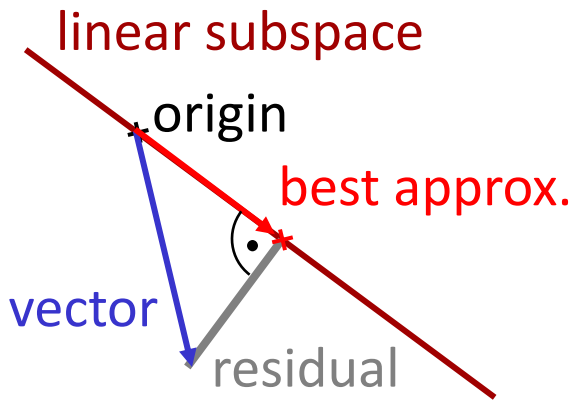
The least-squares criterion is equivalent to:

residual — each basis function

$$\forall i \in \{1..k\} : \left\langle \sum_{j=1}^{k} \lambda_j b_j(x) - f,\ b_i(x) \right\rangle = 0$$

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

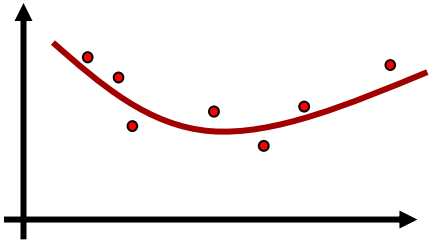
$$\Leftrightarrow \begin{pmatrix} \langle b_1, b_1 \rangle & \cdots & \langle b_n, b_1 \rangle \\ \vdots & \ddots & \vdots \\ \langle b_1, b_n \rangle & \cdots & \langle b_n, b_n \rangle \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \vdots \\ \lambda_n \end{pmatrix} = \begin{pmatrix} \langle b_1(x), f \rangle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

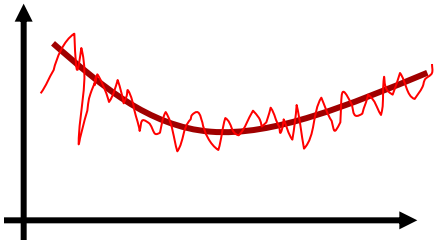


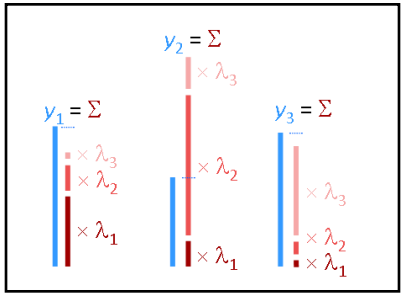
Summary

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:

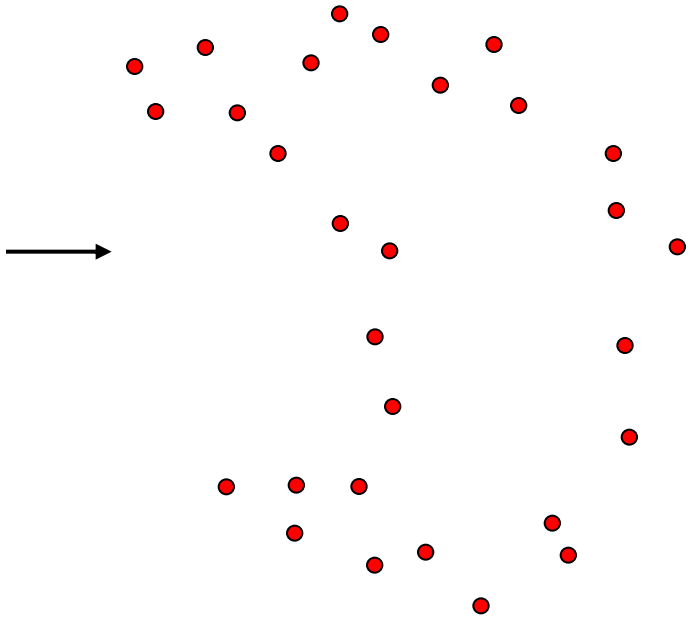
- Any error in **x**-direction is ignored so far (only **y**-direction)
  - We will look at that problem next (total least-squares)...
- Noise must be Gaussian
  - Can be generalized using iteratively reweighted least-squares (M-estimators)

Total Least Squares

Statistical Model

Generative Model:

original curve / surface

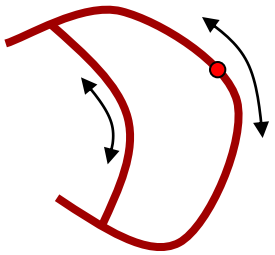


noisy sample points

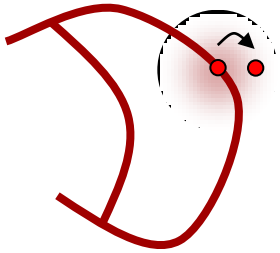
Statistical Model

Generative Model:

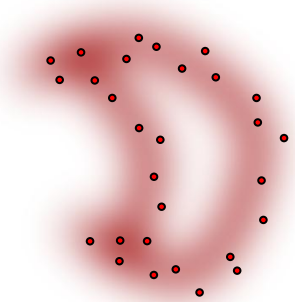
1. Determine sample point (uniform)
2. Add noise (Gaussian)



sampling



Gaussian noise



many samples

distribution (in space)

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

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

- 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 $\mathbf{d}_i \in \mathbb{R}^m$.

Solution: *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

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

(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

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

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

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

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

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

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

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

sample mean:

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

One Dimensional Subspaces...

Next:

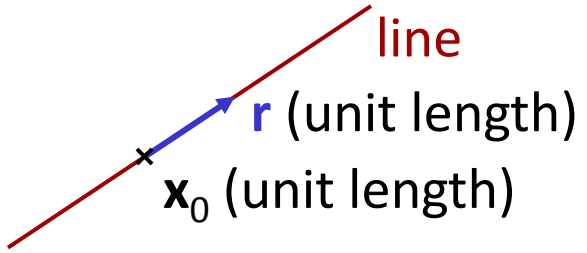
- What is the optimal line (1D subspace) that approximates a set of data points $\mathbf{d}_i$?
- Two questions:
  - Optimum origin (point on the line)?
    - This is still the average (proof omitted)
  - Optimum direction?
    - We will look at that next...
- Parametric line equation:

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

Best Fitting Line

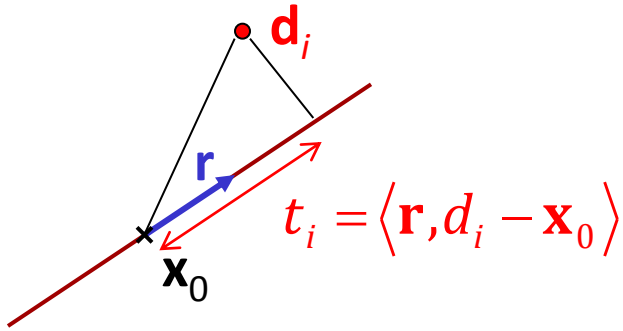
Line equation:

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



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**.

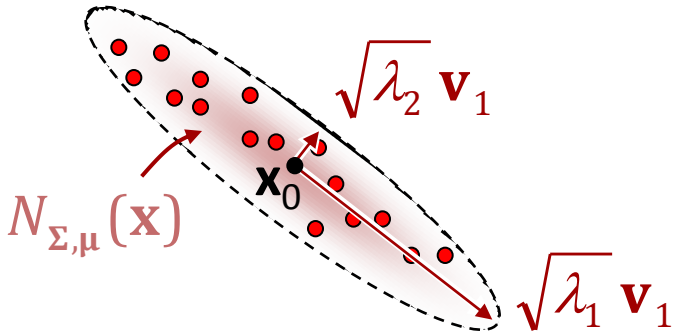
General Case

Procedure: Principal Component Analysis (PCA)

- Compute average $\mathbf{x}_0$ = m(**D**)
- Compute "scatter matrix" $\mathbf{S} = \sum_{i=1}^{n} (d_i - \mathbf{x}_0)(d_i - \mathbf{x}_0)^{\mathrm{T}}$
- Let $(\lambda_1, \mathbf{v}_1)$, ... ,$(\lambda_n, \mathbf{v}_n)$ be sorted eigenvalue/vector pairs of **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



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

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

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

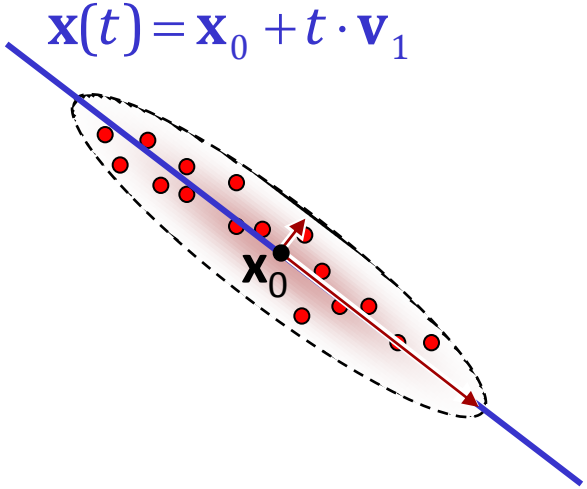
Observation:

- **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

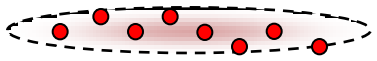
Fitting a line to a point cloud in $\mathbb{R}^2$:

- Sample mean and direction of maximum eigenvalue

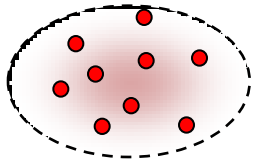


Plane Fitting in $\mathbb{R}^3$:

- 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

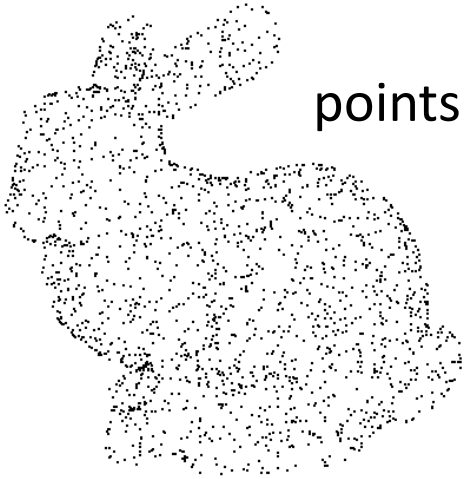
Application: Normal estimation in point clouds

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

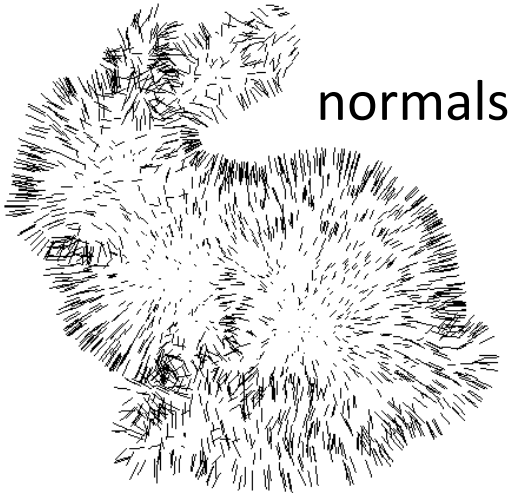
Algorithm:

- 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

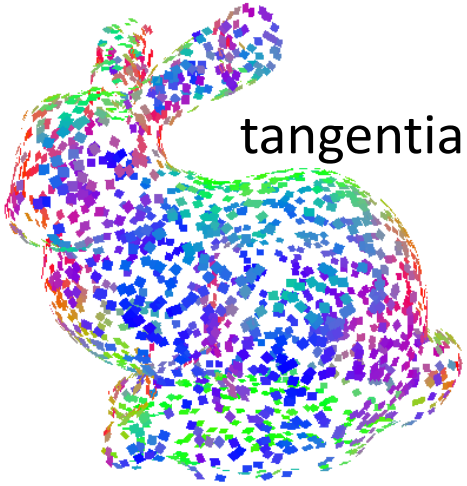
Example



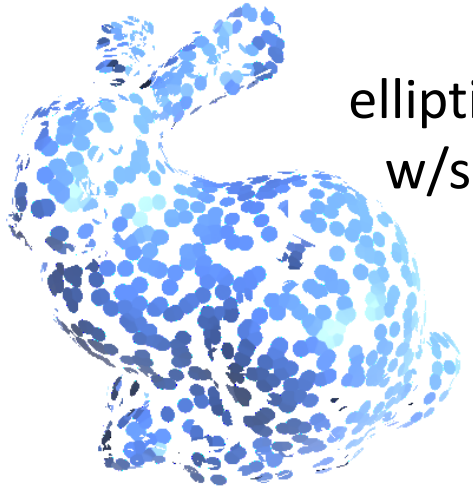
points



normals

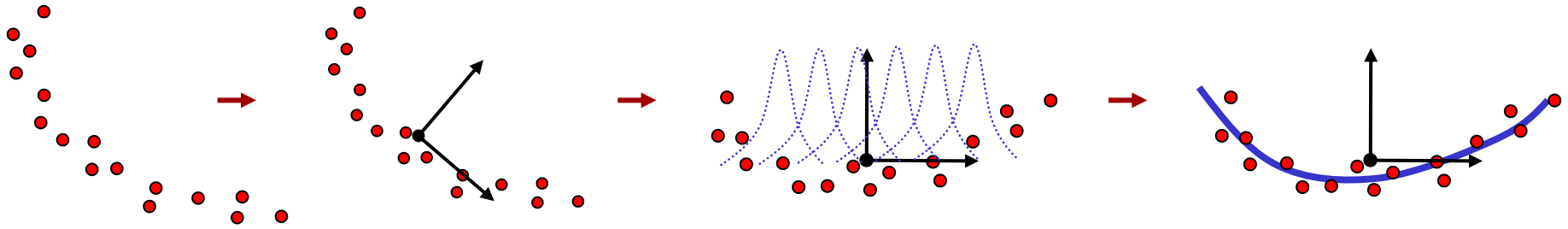


tangential frames



elliptic splats w/shading

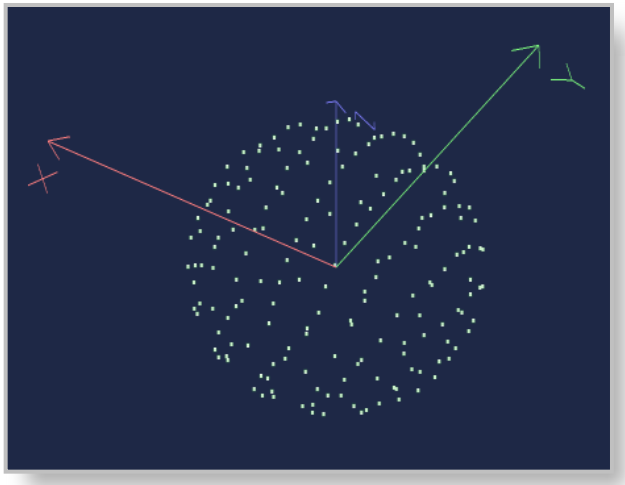
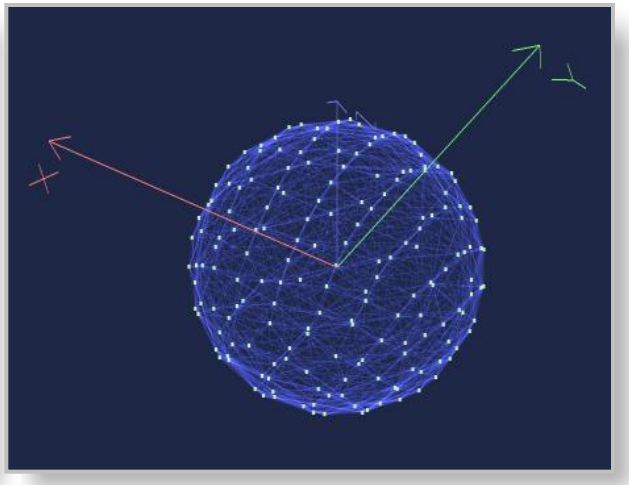
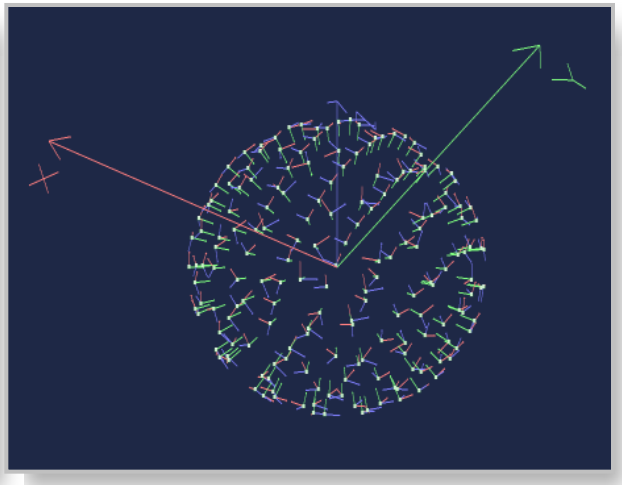
Applications



Another Application: Coordinate frame estimation

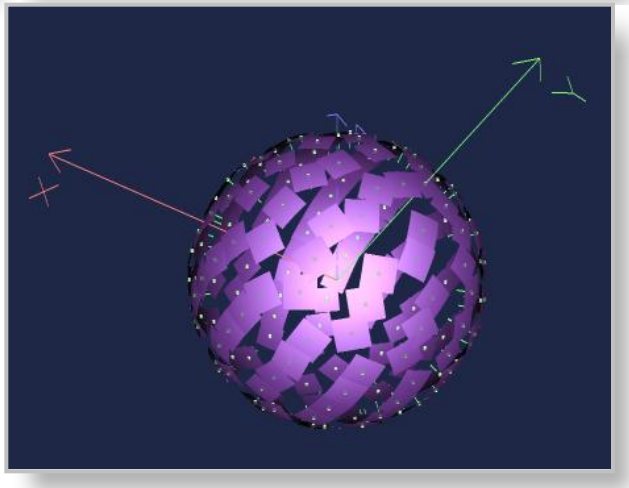
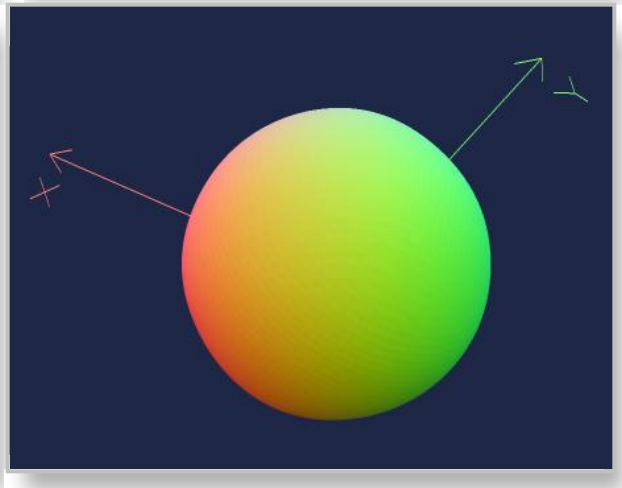
- More general total least-squares (non-linear) is difficult
- However: *We can tweak it*
  - Compute coordinate frames using PCA
  - Smallest eigenvalue = normal direction
  - Form height field in normal direction
  - Then: use ordinary least-squares in this coordinate system

Example

Example:

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

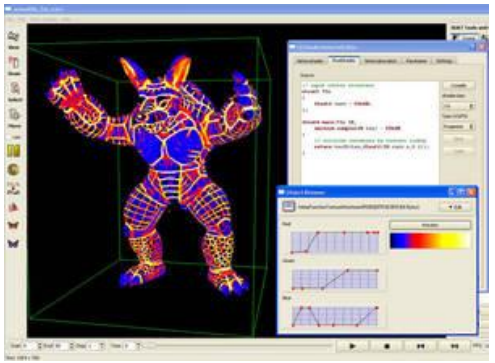



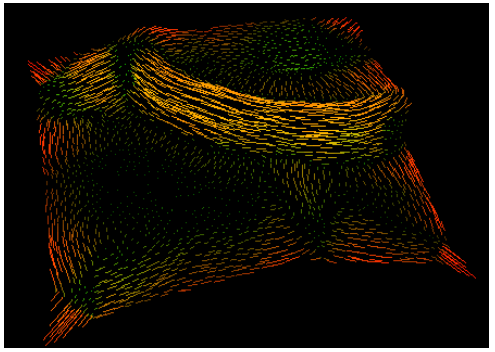
Curvature Estimation

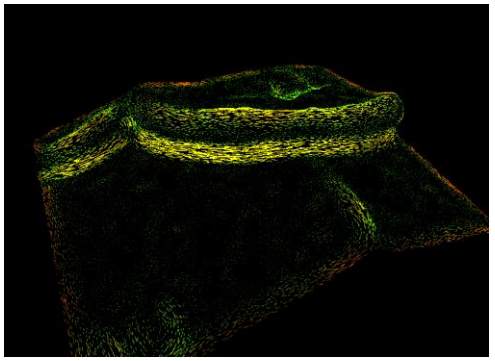
Estimating curvature in point clouds:

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

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

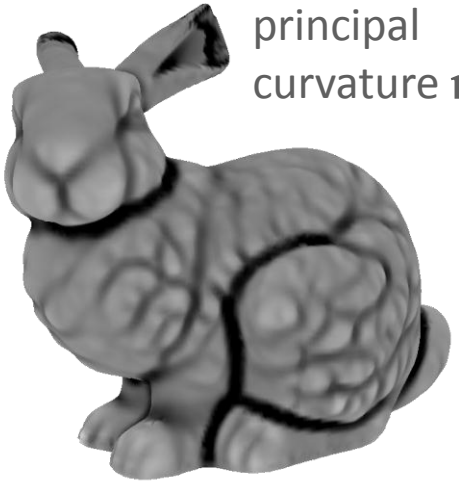




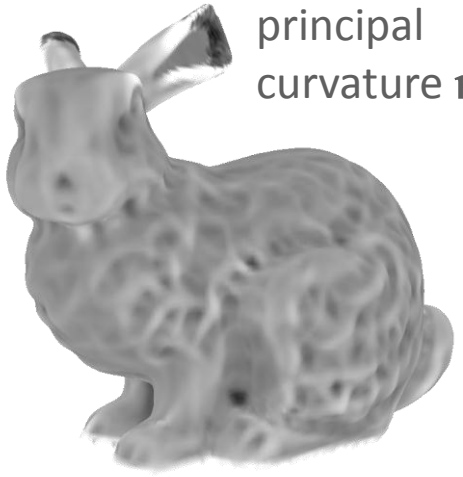


Bunny Curvature

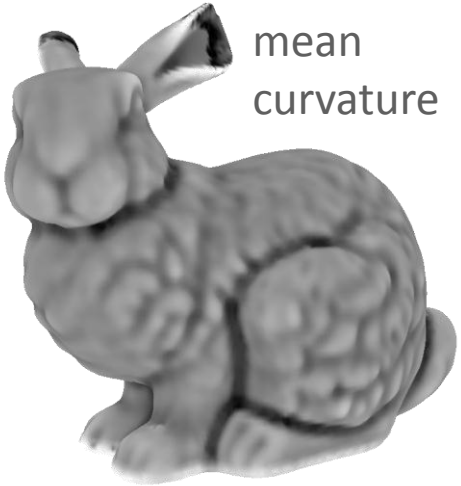
Stanford Bunny
(dense point cloud)



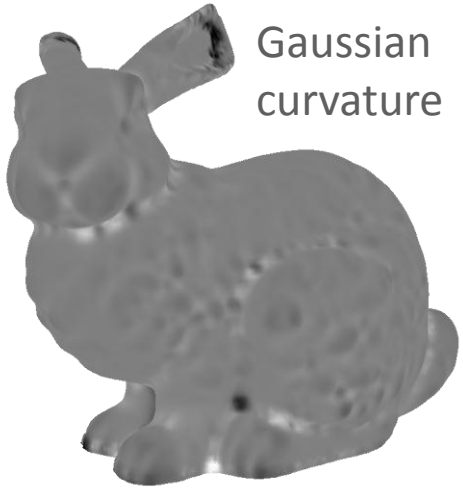
principal curvature κ_1



principal curvature κ_2



mean curvature



Gaussian curvature

[courtesy of Martin Bokeloh]

PCA and MDS

Notation

Data matrix

$$\tilde{\mathbf{X}} = \begin{pmatrix} - & \tilde{x}_1 & - \\ & \vdots & \\ - & \tilde{x}_n & - \end{pmatrix}$$

d-dimensional input vectors $\mathrm{x}_{i,1} \ldots \mathrm{x}_{i,d}$

Centered data matrix

$$\mathbf{X} = \tilde{\mathbf{X}} - \overline{\mathbf{X}} = \tilde{\mathbf{X}} - \begin{pmatrix} \frac{1}{n}\sum_{i=1}^{n} \tilde{x}_i \\ \vdots \\ \frac{1}{n}\sum_{i=1}^{n} \tilde{x}_i \end{pmatrix} = \left(\mathbf{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^{\mathrm{T}}\right)\tilde{\mathbf{X}}$$

MDS

Multi-Dimensional Scaling

- Input: $n \times n$ pairwise distance matrix $\mathbf{D}$
- Compute pairwise scalar product matrix of centered vectors:

$$\tilde{\mathbf{D}} = -\frac{1}{2}\left(d_{ij}^{\,2}\right)_{ij}, \quad \mathbf{K} = \left(\mathbf{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^{\mathrm{T}}\right)\tilde{\mathbf{D}}\left(\mathbf{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^{\mathrm{T}}\right)$$

- By this construction:

$$\mathbf{K} = \mathbf{X}\mathbf{X}^{\mathrm{T}} \quad \left(= \left(\tilde{\mathbf{X}} - \overline{\mathbf{X}}\right)\left(\tilde{\mathbf{X}} - \overline{\mathbf{X}}\right)^{\mathrm{T}} \right)$$

(PCA: $\mathbf{X}^{\mathrm{T}}\mathbf{X}$)

MDS (2)

Multi-Dimensional Scaling

- Next: Compute eigenstructure of $\mathbf{K} = \mathbf{X}\mathbf{X}^{\mathrm{T}}$

  $\mathbf{K} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^{\mathrm{T}}$
- Thus, a candidate reconstruction of $\mathbf{X}$ is given by $\mathbf{X} \equiv \mathbf{U}\boldsymbol{\Lambda}^{1/2}$
- Reminder: $\mathbf{X}$ – data vectors in rows
- Hence: $emb(x_i) \equiv \left(\sqrt{\lambda_1}\,\mathbf{u}_1^{(i)}, \ \ldots \ , \sqrt{\lambda_n}\,\mathbf{u}_n^{(i)}\right)$

$$\approx \left(\sqrt{\lambda_1}\,\mathbf{u}_1^{(i)}, \ldots, \sqrt{\lambda_k}\,\mathbf{u}_k^{(i)}\right)(k \le n)$$

MDS (3)

Properties: (assuming Euclidian distances)

- Recovers points up to global translation and orthogonal mapping
- Reduced version (k-dim.) preserves distances in a least square sense (dimension reduction)
- Result is the same as for PCA embedding on point coordinates (next slide)

MDS is PCA

SVD of Centered Data Matrix

$$\mathbf{X} = \mathbf{U}\Lambda\mathbf{V}^{\mathrm{T}} = \mathbf{U}\begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_{\min(d,n)} \end{pmatrix}\mathbf{V}^{\mathrm{T}}$$

Equivalence of MDS and PCA

PCA: $\mathbf{X}^{\mathrm{T}}\mathbf{X} = \mathbf{V}\Lambda^2\mathbf{V}^{\mathrm{T}}$

MDS: $\mathbf{X}\mathbf{X}^{\mathrm{T}} = \mathbf{U}\Lambda^2\mathbf{U}^{\mathrm{T}}$

PCA: $emb_{PCA}(\mathbf{X}) = \mathbf{X}\mathbf{V}$

MDS: $emb_{MDS}(\mathbf{X}) = \mathbf{U}\Lambda$

$$\mathbf{X} = \mathbf{U}\Lambda\mathbf{V}^{\mathrm{T}} \Rightarrow \mathbf{X}\mathbf{V} = \mathbf{U}\Lambda$$

So Where is the Difference?

	PCA	MDS
input	points	pw. distances / scalar prods.
complexity	$d \times d$ eigenvalue problem (low dim., large data sets)	$n \times n$ eigenvalue problem (high dim., small data sets)
result	data embedding, principal variances, principal axes	data embedding, principal variances, no principal axes
subspace projection	can easily embed additional vectors	not obvious (yes: Nyström method)

Nyström Projection

Embedding further Vectors

- Recompute everything
  - Expensive
  - Inconsistent for some applications (new coordinates)
- "Nyström Formula"
  - Compute embedding by linear combination of computed eigenvectors
  - Uses projections on input data set (scalar products only)
  - Assumes knowledge of point positions (later: measure distances only)

Nyström Projection

Nyström Projection

- Reminder: $\mathbf{X} = \mathbf{U}\Lambda\mathbf{V}^{\mathrm{T}} \qquad \mathbf{K} = \mathbf{X}\mathbf{X}^{\mathrm{T}} = \mathbf{U}\Lambda^2\mathbf{U}^{\mathrm{T}}$

  $emb_{MDS}(\mathbf{X}) = \mathbf{U}\Lambda \quad emb_{PCA}(\mathbf{x}) = \mathbf{V}\mathbf{x}$

- Project vector $\mathbf{x}$ on principal axes:

$$
\begin{aligned}
emb(\mathbf{x}) &= \mathbf{V}\mathbf{x} \\
&= \left(\left[\Lambda^{-1}\mathbf{U}^{\mathrm{T}}\right]\mathbf{X}\right)\mathbf{x} = \left[\Lambda^{-1}\mathbf{U}^{\mathrm{T}}\right]\left[\mathbf{X}\,\mathbf{x}\right] \\
&= \begin{pmatrix} \sum_{i=1}^{n} \frac{1}{\lambda_1} u_i^{(1)} \left\langle \mathbf{x}_i, \mathbf{x} \right\rangle \\ \vdots \\ \sum_{i=1}^{n} \frac{1}{\lambda_n} u_i^{(n)} \left\langle \mathbf{x}_i, \mathbf{x} \right\rangle \end{pmatrix}
\end{aligned}
$$

Kernel PCA

Kernel PCA

"Kernel PCA is classical scaling in feature space" [Williams 2002]

Main Idea:

- MDS can be easily "kernelized" – just replace scalar product matrix $\mathbf{K}$ with kernel matrix
- No need to deal with feature space explicitly (which might be intractable)
- Will yield PCA anyway (but no eigenvectors)

Algorithm

Kernel PCA – Step 1/3

Compute kernel (Gram) matrix

$$\mathbf{K} = \begin{pmatrix} \langle \phi(x_1), \phi(x_1) \rangle & \cdots & \langle \phi(x_n), \phi(x_1) \rangle \\ \vdots & \ddots & \vdots \\ \langle \phi(x_1), \phi(x_n) \rangle & \cdots & \langle \phi(x_n), \phi(x_n) \rangle \end{pmatrix}$$

$$= \begin{pmatrix} k(x_1, x_1) & \cdots & k(x_n, x_1) \\ \vdots & \ddots & \vdots \\ k(x_1, x_n) & \cdots & k(x_n, x_n) \end{pmatrix}$$

Algorithm

Kernel PCA – Step 2/3: Center Data

- Cannot compute $\mathbf{X} = \tilde{\mathbf{X}} - \overline{\mathbf{X}}$ because $\mathbf{X}$ is now in feature space
- Center kernel matrix directly:

$$\mathbf{K}^{(centered)} = \mathbf{K} - \left[\frac{1}{n}\mathbf{1}\mathbf{1}^{\mathrm{T}}\right]\mathbf{K} - \mathbf{K}\left[\frac{1}{n}\mathbf{1}\mathbf{1}^{\mathrm{T}}\right] + \left[\frac{1}{n}\mathbf{1}\mathbf{1}^{\mathrm{T}}\right]\mathbf{K}\left[\frac{1}{n}\mathbf{1}\mathbf{1}^{\mathrm{T}}\right]$$

$$= \left(\mathbf{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^{\mathrm{T}}\right)\mathbf{K}\left(\mathbf{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^{\mathrm{T}}\right) \longleftarrow \text{exactly the same as in MDS}$$

Proof:

$$\mathbf{K}_{i,j} = \left\langle \phi(\mathbf{x}_i) - \sum_{k=1}^{n}\phi(\mathbf{x}_k),\ \phi(\mathbf{x}_j) - \sum_{k=1}^{n}\phi(\mathbf{x}_k) \right\rangle$$

$$= \phi(\mathbf{x}_i)^{\mathrm{T}}\phi(\mathbf{x}_j) - \sum_{k=1}^{n}\phi(\mathbf{x}_k)^{\mathrm{T}}\phi(\mathbf{x}_j) - \sum_{k=1}^{n}\phi(\mathbf{x}_k)^{\mathrm{T}}\phi(\mathbf{x}_i) + \sum_{k=1}^{n}\sum_{l=1}^{n}\phi(\mathbf{x}_k)^{\mathrm{T}}\phi(\mathbf{x}_l)$$

Algorithm

Kernel PCA – Step 3/3 - Compute embedding

- Eigenspace:

$$\mathbf{K}^{(centered)} = \mathbf{U}\Lambda\mathbf{U}^{\mathrm{T}}$$

- Embedding:

$$emb(x_i) = \left(\sqrt{\lambda_1}\mathbf{u}_1^{(i)}, \ldots, \sqrt{\lambda_k}\mathbf{u}_k^{(i)}\right)(k \leq n)$$

Embedding Further Points

Project Additional Points:

- Use Nyström Formula:

$$emb(\mathbf{x}) = \begin{pmatrix} \sum_{i=1}^{n} \frac{1}{\lambda_1} u_i^{(1)} k(\mathbf{x}_i, \mathbf{x}) \\ \vdots \\ \sum_{i=1}^{n} \frac{1}{\lambda_n} u_i^{(n)} k(\mathbf{x}_i, \mathbf{x}) \end{pmatrix}$$

- Uses only kernel computations in feature space

Kernel PCA

Summary:

- Kernel PCA performs PCA/MDS in feature space using dot products (i.e. kernel evaluations) only
- It gives the same result as MDS

Kernel PCA

Remarks:

- Unlike “real” PCA, it does not output principal axes vectors
  - They are in feature space, i.e. usually inaccessible
  - Preimages in (low-dim.) input space do not need to exist (there are approximation techniques)
  - Even if so, they are difficult to compute...
    ...and the space is non-linear anyway
    (so they do not really help)

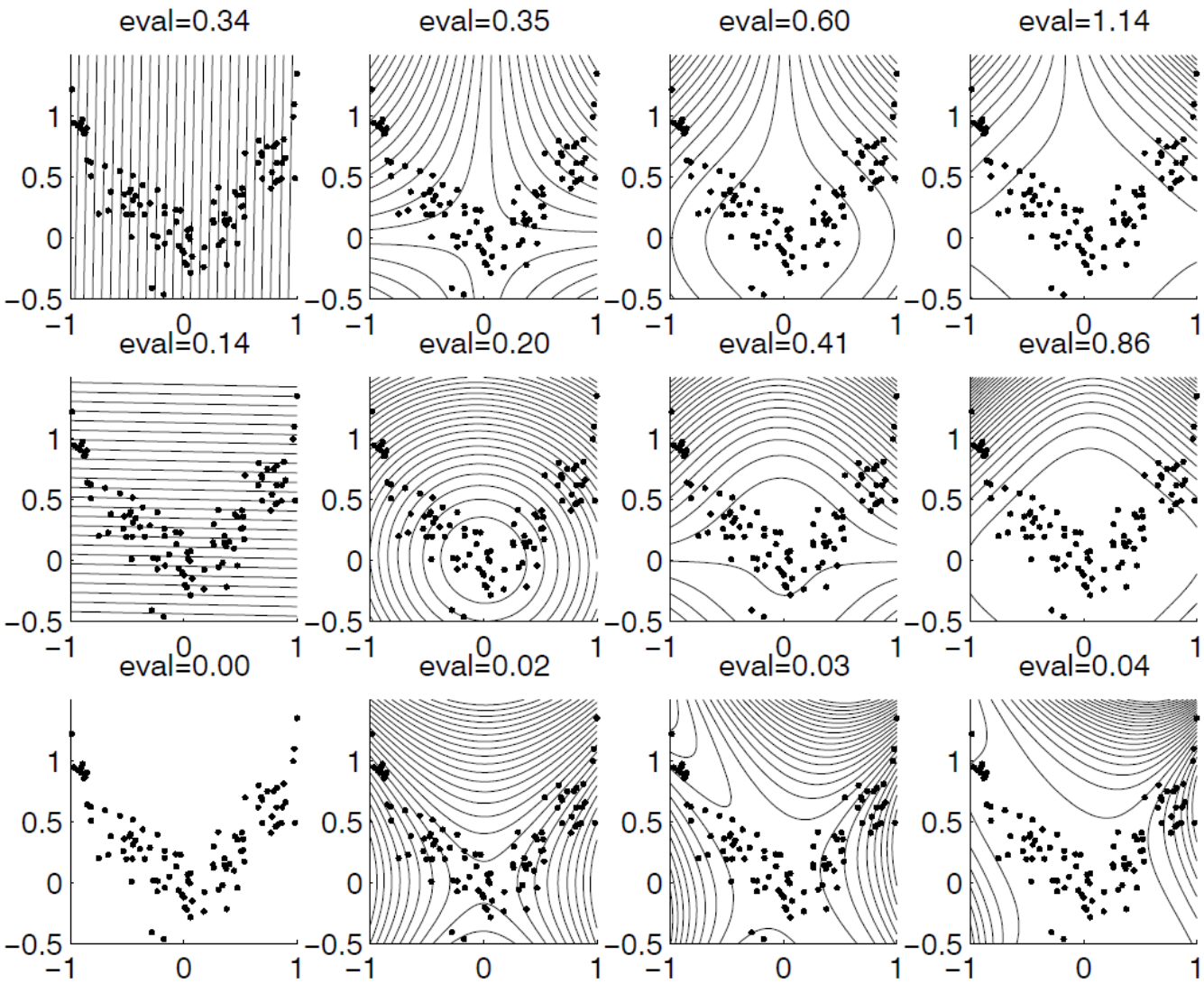
Kernel PCA

Complexity:

- Need to solve $n \times n$ eigenvalue problem
- Memory, Time: $\Omega(n^2)$
- Does not scale for large data sets
  - Can use approximation techniques
  - Idea: Landmark MDS
  - Compute embedding on small subset of landmark points (e.g. random subset)
  - Use Nyström formula to embed other points

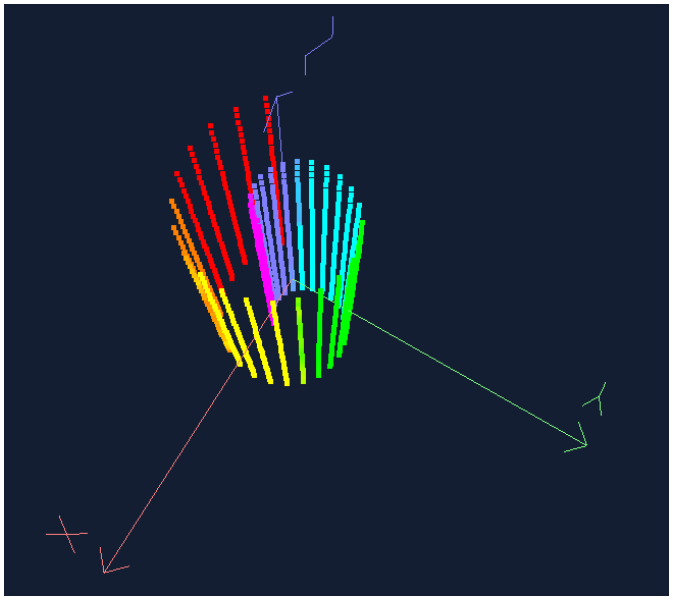
Examples

From the Paper

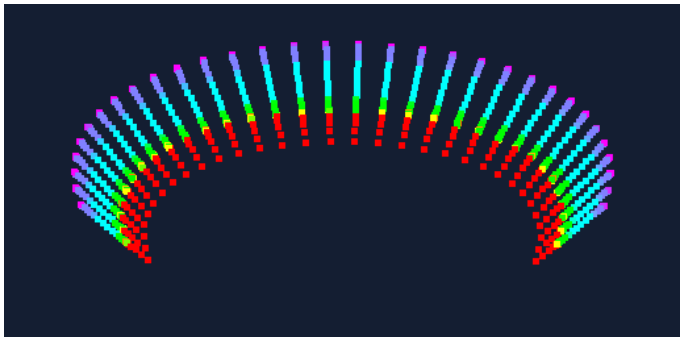


[Schölkopf et al. 1996]

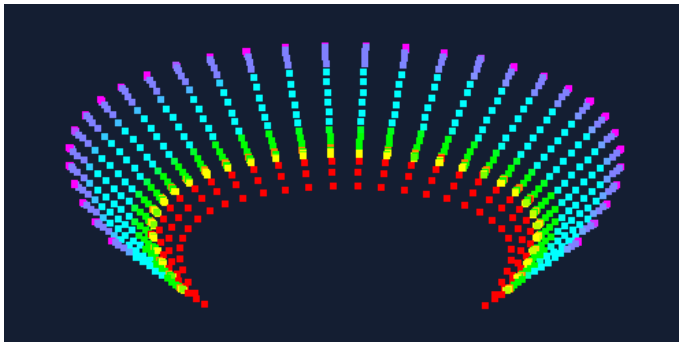
Where is the Swiss Roll?



the roll

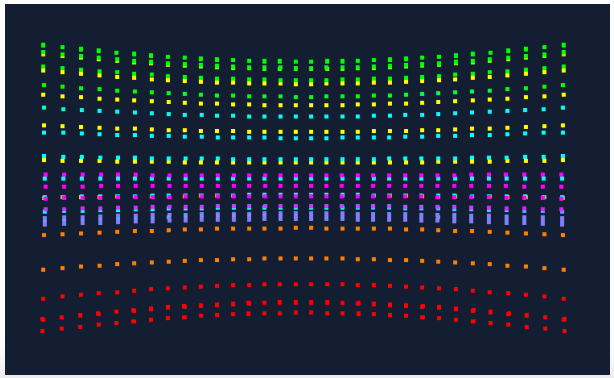


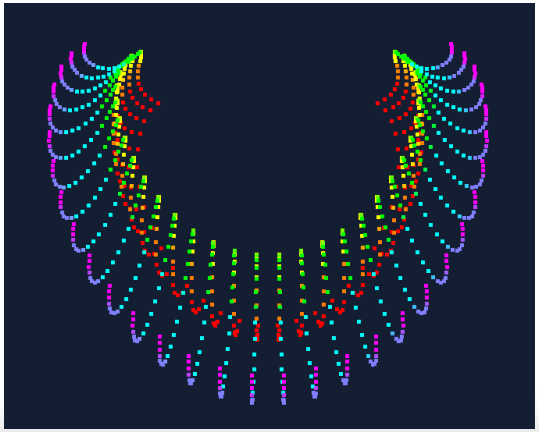
exp. kernel $\sigma = 0.30D$



exp. kernel $\sigma = 0.35D$

poly.-kernel (5th order) $\sigma = 0.35D$

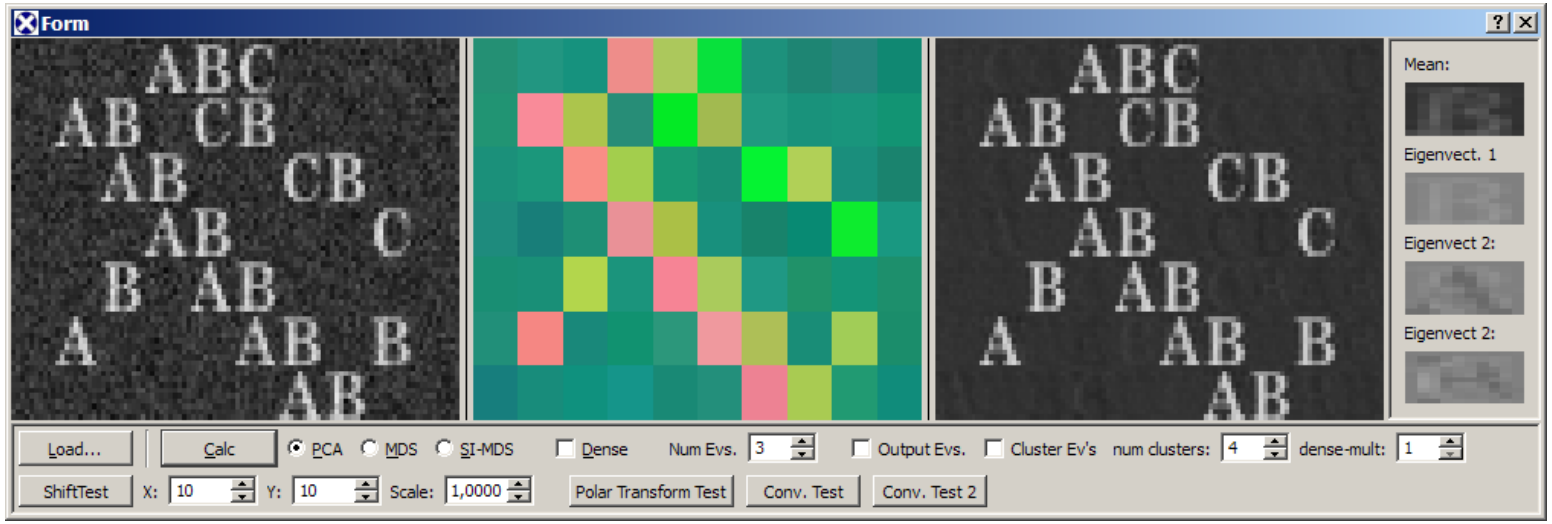


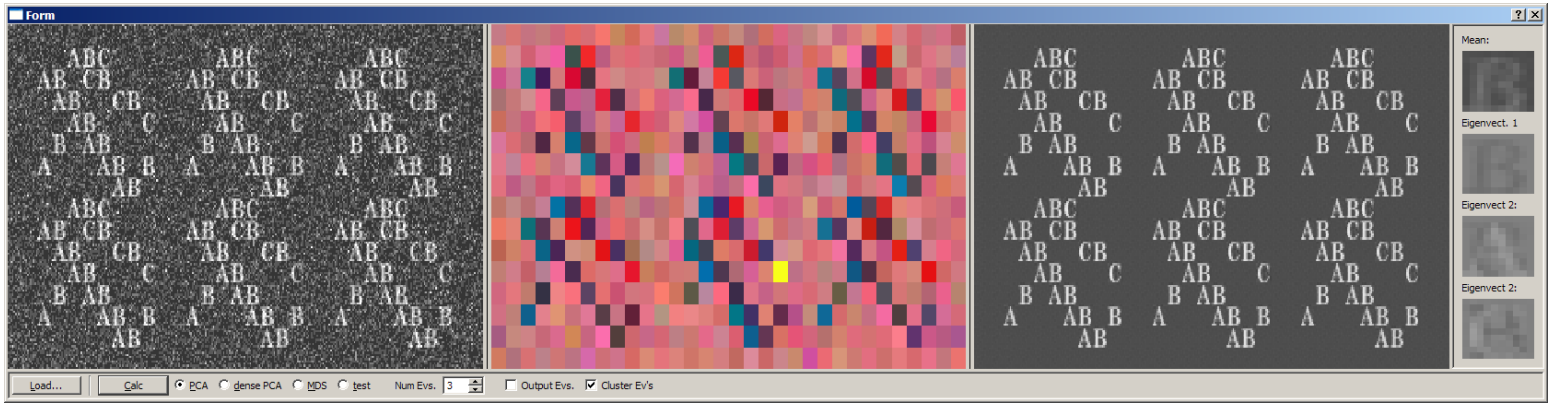


exp. kernel $\sigma = 0.35D$ (centered)

What Else Can You Do?

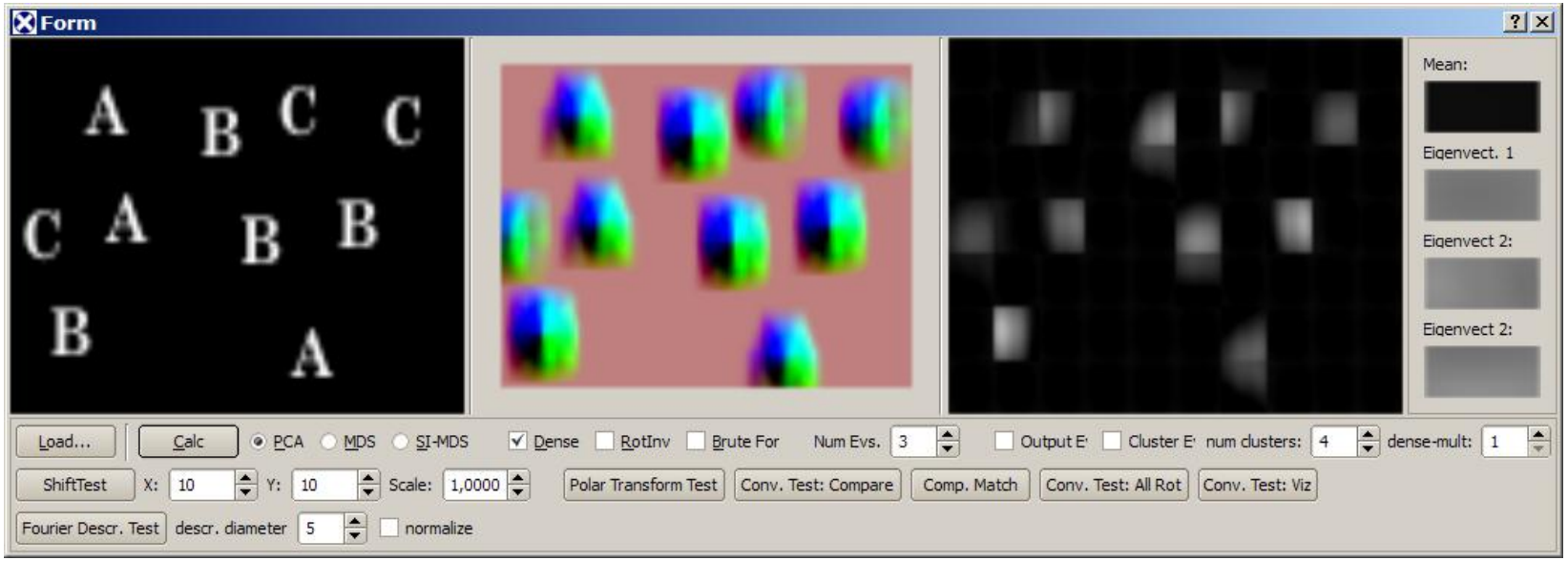
Image Denoising via PCA:





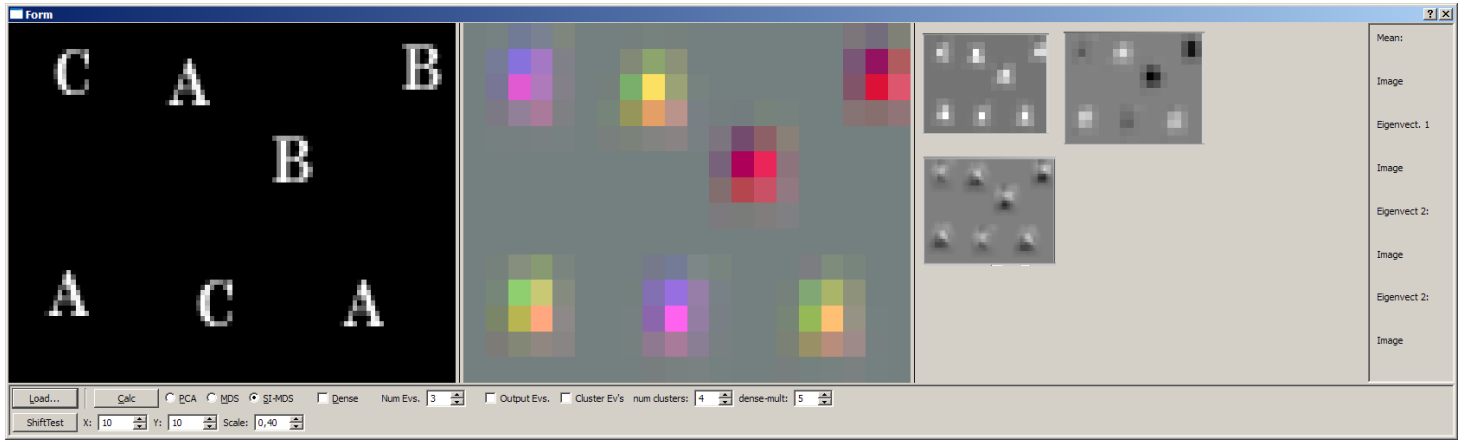
What Else Can You Do?

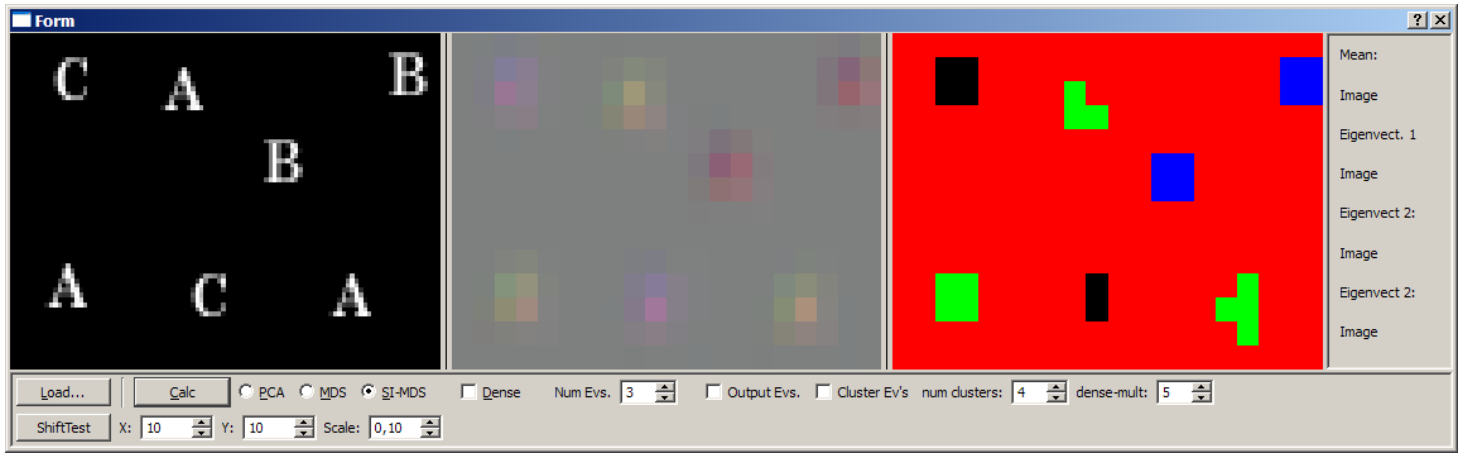
Does not work without correspondences:



What Else Can You Do?

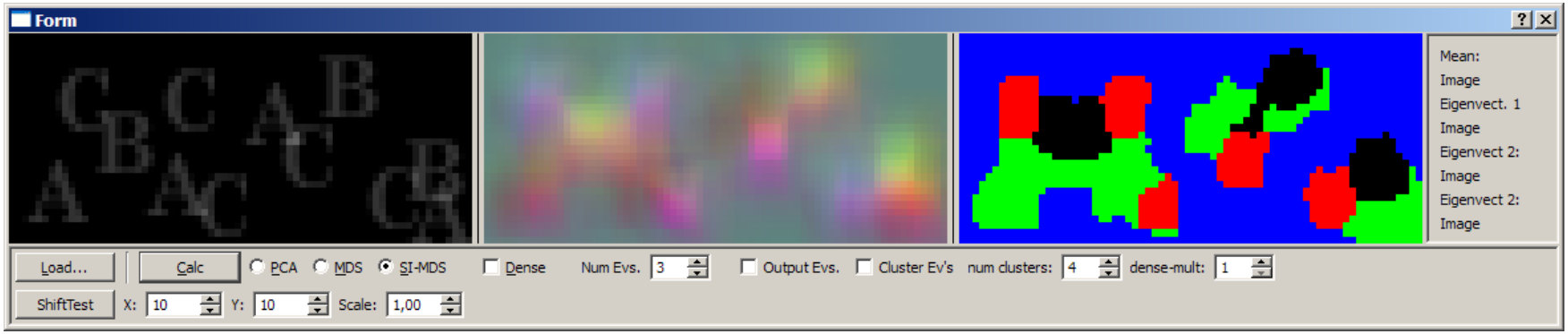
Shift invariant comparison kernel:

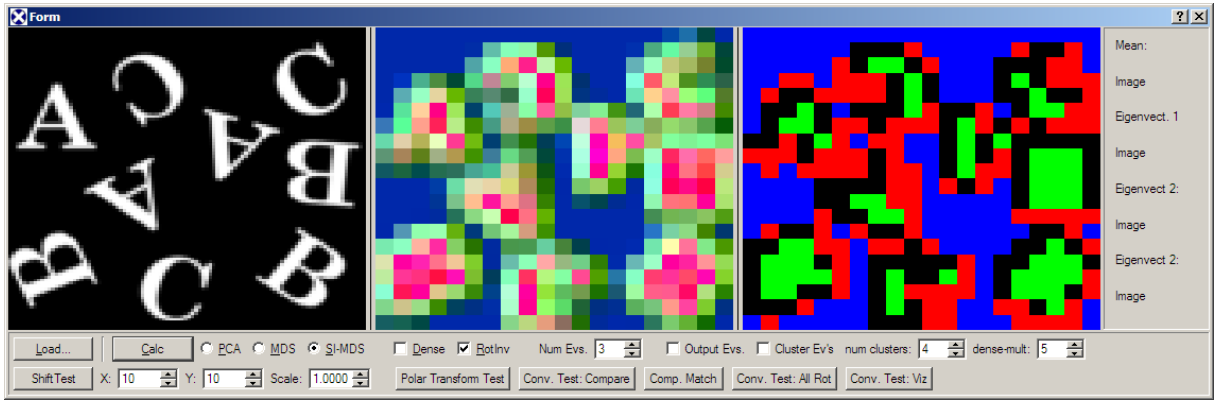




MDS (Kernel PCA)

Shift invariant comparison kernel:





References

B. Schölkopf, A. J. Smola, K.-R. Müller: Nonlinear Component Analysis as a Kernel Eigenvalue Problem. In: Neural Computation, 10:1299-1319, 1998.

B. Schölkopf, S. Mika, C. J. C. Burges, P. Knirsch, K.-R. Müller, G. Ratsch, A. J. Smola: Input Space versus Feature Space in Kernel-Based Methods. In: IEEE Trans. on Neural Networks, 10:1000-1017, 1999.

C. Williams: On a Connection between Kernel PCA and Metric Multidimensional Scaling. In: Machine Learning, 46:11-19, 2002.

K.-R. Müller, S. Mika, G. Ratsch, K. Tsuda, B. Schölkopf: An Introduction to Kernel-Based Learning Algorithms. In: IEEE Trans. on Neural Networks, 12:181-201, 2001.

J. Shawe-Taylor, N. Cristianini: Kernel Methods for Pattern Analysis. Cambridge University Press, 2004.

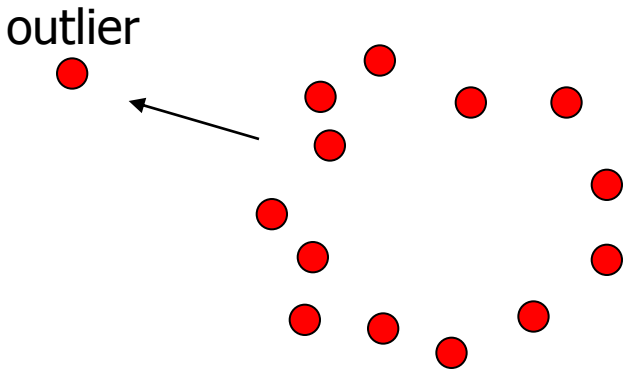
T. Cox, M. Cox: Multi-Dimensional Scaling. Chapman & Hall, 1994.

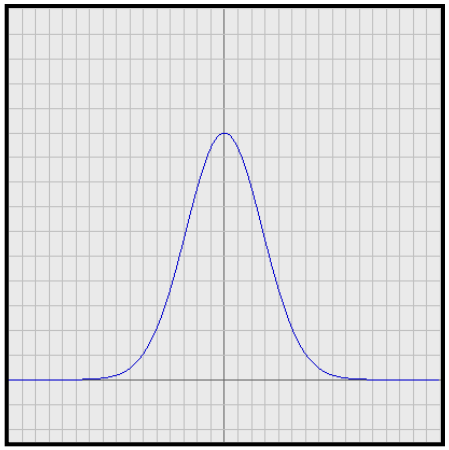
Iteratively Reweighted Least Squares

General Error Distributions

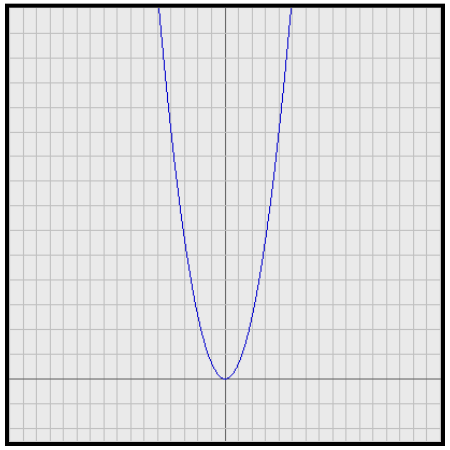
Problem with least-squares

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

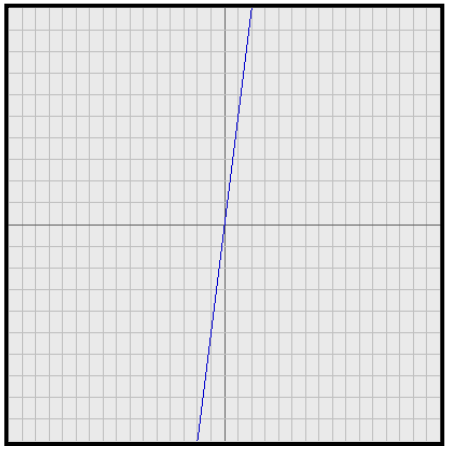




Gaussian likelihood

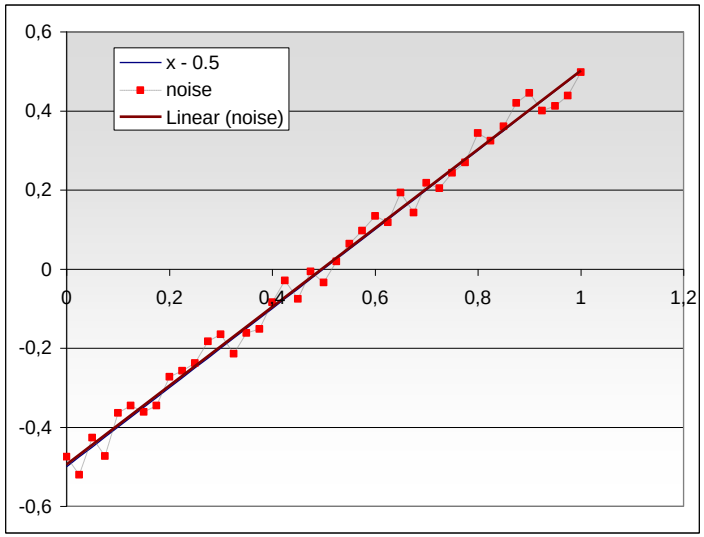


neg. log-likelihood

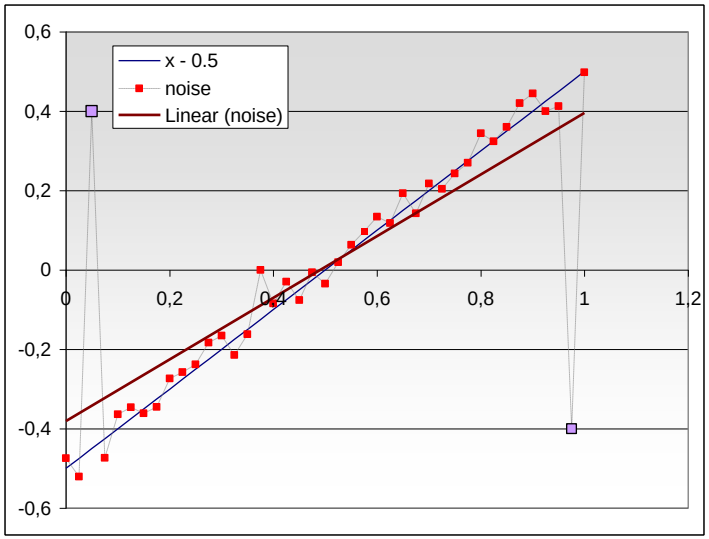


derivative

Outliers



uniform noise

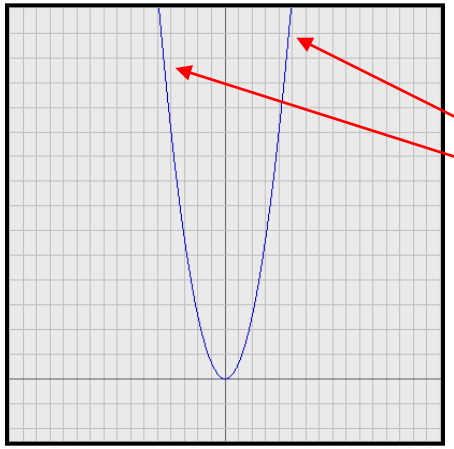


2 outliers (out of 41)

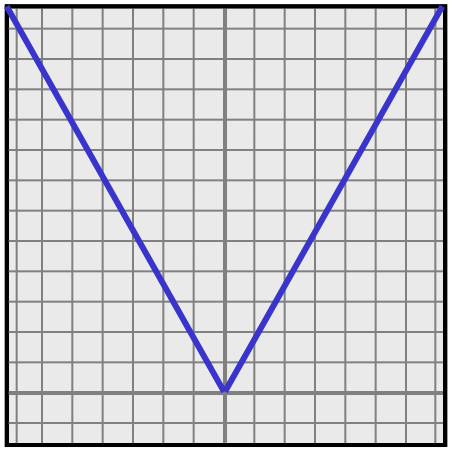
Problem:

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

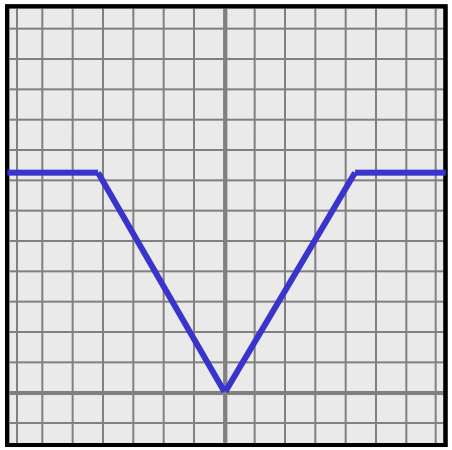
Robust Estimators



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

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

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

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

L1-Norm Fitting

L1-Norm Fitting:

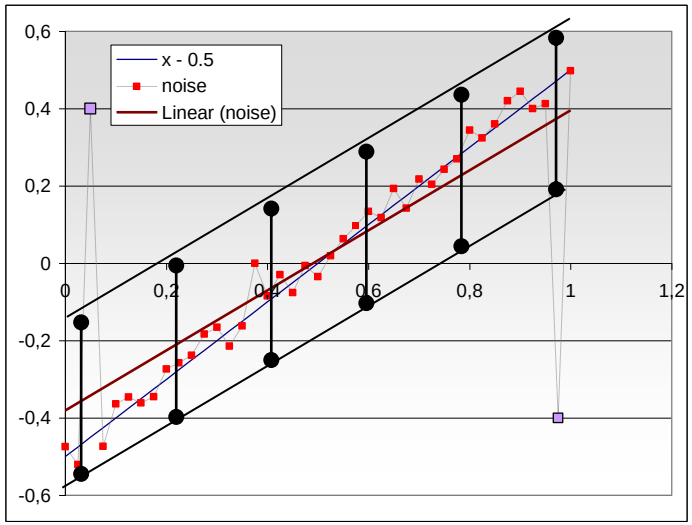
- $\underset{\tilde{f}^{(k)}}{\arg\min} \sum_{i=1}^{n} \frac{1}{\left\| \tilde{f}^{(k-1)}(x_i) - y_i \right\|} (\tilde{f}^{(k)}(x_i) - y_i)^2 \rightarrow \underset{\tilde{f}^{(k)}}{\arg\min} \sum_{i=1}^{n} \left\| \tilde{f}^{(k-1)}(x_i) - y_i \right\|$

- Convergence guaranteed (convex objective function)
- Mixture of l_1 (far) and l_2 (near) norm also works

General M-Estimators

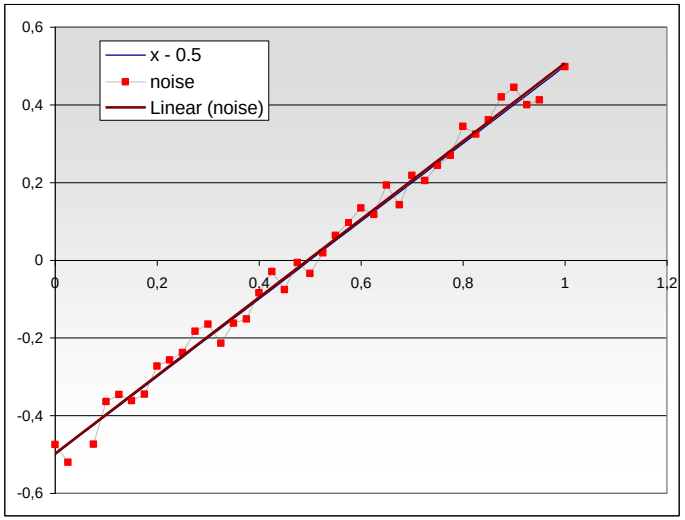
- No convergence guarantees for non-convex potentials
- In particular: Truncated potentials may converge to a local extremum only (depends on initialization)

Example (Schematic)



first iteration:

least squares, then truncation



second iteration:

improved solution