Computer Graphics 1

IV.1 Optimization I

Basics

Singular Value Decomposition

Solving Linear Least Squares Problems

Plane Fitting

Registration of 3D Data Sets

Introduction to Mathematical Optimization

BASICS

Optimization Problems

Parameterization

- vector of n optimization variables $\boldsymbol{x} = (x_1,\ldots,x_n)$

Goal

- find optimum $\boldsymbol{x}^*$ that minimizes objective function f
- subject to

  m equality constraints c_i and

  M inequality constraints h_j

  over all $\boldsymbol{x} \in \boldsymbol{R}^n$

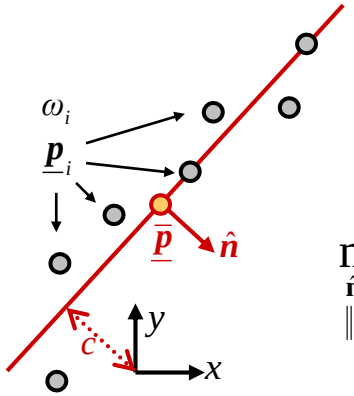
$$\mathbf{x}^* = \arg\min_{\substack{\mathbf{x}\in\mathbf{R}^n \\ c_i(\mathbf{x})=0, i=1\ldots m \\ h_j(\mathbf{x})\le 0, j=1\ldots M}} f(\mathbf{x})$$

- maximization problems can be cast into minimization problems:

$$\max f(\mathbf{x}) = \min - f(\mathbf{x})$$

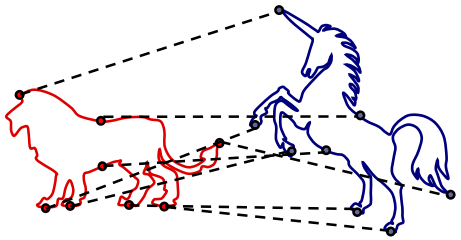
Examples from CG

- fit plane to set of points by minimizing sum of possibly weighted squared plane distances



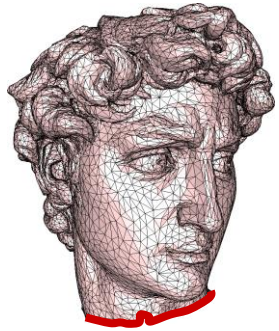
$$\min_{\substack{\hat{\mathbf{n}},c \\ \|\hat{\mathbf{n}}\|=1}} \sum_i \omega_i \left(\langle \hat{\mathbf{n}}, \underline{\mathbf{p}}_i \rangle + c \right)^2$$

- rigidly align shapes based on point correspondences

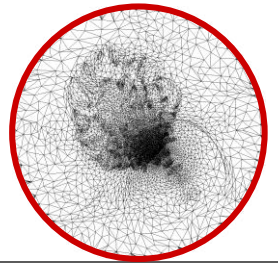


$$\min_{\substack{\mathbf{R},\vec{\mathbf{t}} \\ \mathbf{R}^T\mathbf{R}=\mathbf{I}}} \sum_i \left\| \underline{\mathbf{p}}_i - \left(\mathbf{R}\underline{\mathbf{q}}_i + \vec{\mathbf{t}} \right) \right\|^2$$

- embed 3d surface patches with low distortion into 2d plane for texture mapping



$$\min_{\substack{\underline{\boldsymbol{\tau}}_i \\ \text{boundary constraints, no overlaps}}} \sum_{ij \in E} \omega_{ij} \left\| \underline{\boldsymbol{\tau}}_i - \underline{\boldsymbol{\tau}}_j \right\|^2$$



Properties of Problems

- optimization variables are continuous or discrete or a mixture of both
- optimization variables are constrained or unconstrained
- objective function / constraints are convex or not convex
- objective function is differentiable (pointwise function, gradient and or Hessian evaluation is possible)
- instances of mathematical programming
  (= solution strategies, term was chosen already 1940)
  - linear least squares and non linear least squares
  - linear programming
  - quadratic programming
  - convex optimization
  - (mixed) integer [linear] programming,…

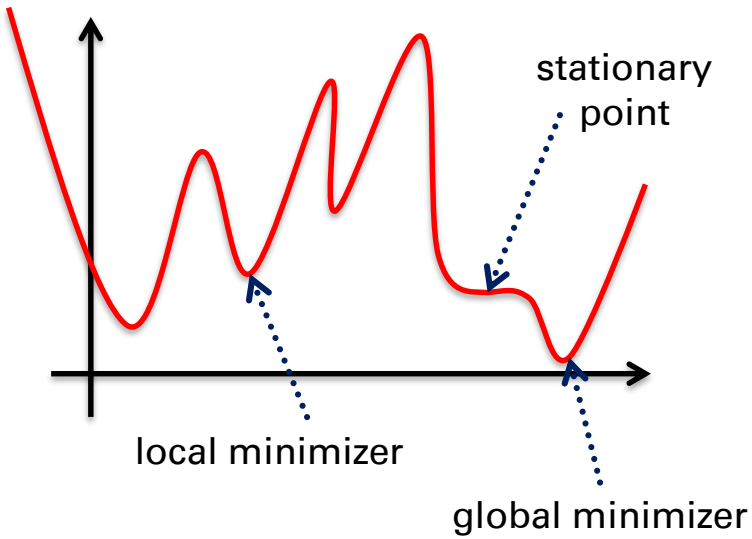
Properties of Optimization methods

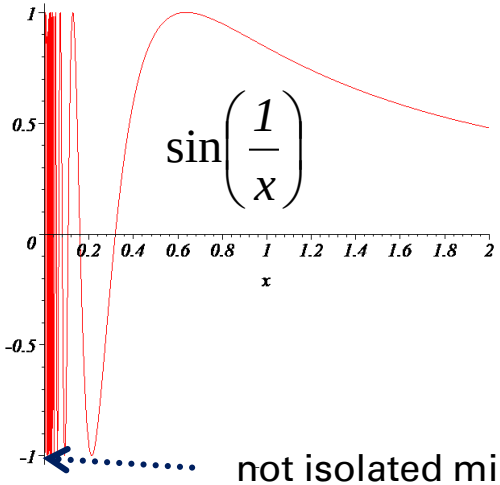
- deterministic (always return same result) vs. stochastic
- local vs. global (are able to find local/global optimum (sometimes))
- closed form solution vs. iterative methods (start at initial guess and improve until (local) optimum is found
- derivative-free methods vs. methods which rely on the pointwise evaluation of first or second derivatives of objective functions with respect to objective variables

Different Minima

one distinguishes minima

- (strict if not equal) global minimizer:
  $$\forall \boldsymbol{x}: f(\boldsymbol{x}^*) < f(\boldsymbol{x})$$
- (strict if not equal) local minimizer:
  $\forall \boldsymbol{x} \in N(\boldsymbol{x}^*): f(\boldsymbol{x}^*) < f(\boldsymbol{x})$
- isolated if local minimizer is unique in arbitrarily small neighborhood

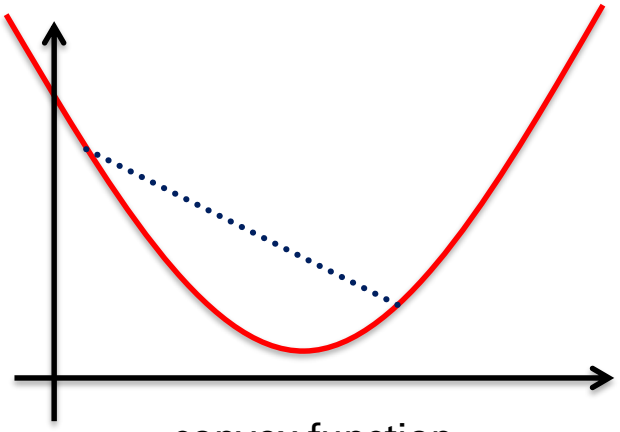




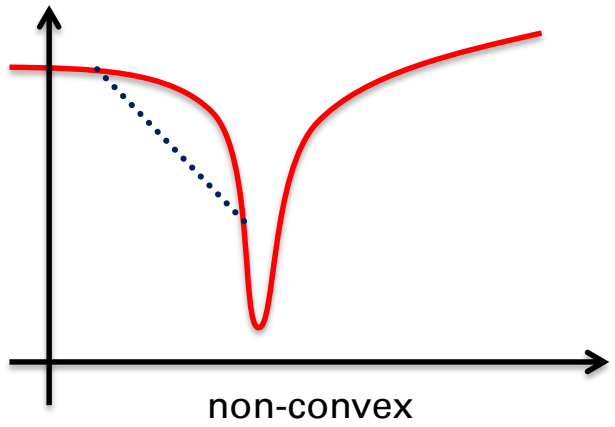
Convexity

- a function is convex iff for all α, $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$:

$$f\big((1-\alpha)\cdot\boldsymbol{x}_1+\alpha\cdot\boldsymbol{x}_2\big) < (1-\alpha)\cdot f(\boldsymbol{x}_1)+\alpha\cdot f(\boldsymbol{x}_2)$$



- An optimization is convex iff the objective function is convex, all equality constraints are linear and all inequality constraints are concave
- Theorem: in a convex optimization problem, a stationary point is always a global minimizer



Derivatives

- The gradient vector of f is denoted by:

$$f_x(\boldsymbol{x}) = \nabla f(\boldsymbol{x}) = \left(\frac{\partial f}{\partial x_1} \quad \frac{\partial f}{\partial x_2} \quad \cdots \quad \frac{\partial f}{\partial x_n} \right)^T$$

- The Hessian of second derivatives is denoted by:

$$f_{xx}(\boldsymbol{x}) = \nabla^2 f(\boldsymbol{x}) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_j} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & \ddots & & & \vdots \\ \frac{\partial^2 f}{\partial x_i \partial x_1} & & \frac{\partial^2 f}{\partial x_i \partial x_j} & & \frac{\partial^2 f}{\partial x_i \partial x_n} \\ \vdots & & & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_j} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{pmatrix}$$

- For twice continuously differentiable functions the Hessian is symmetric

Computing derivatives

- Analytically (symbolic toolboxes can do that for you!)
- finite differences with epsilon related to machine precision π.

$$\frac{\partial f}{\partial x}(x) = \frac{f(x+\varepsilon) - f(x)}{\varepsilon}$$

forward differences

$$\varepsilon = \sqrt{\pi}$$

$$\frac{\partial f}{\partial x}(x) = \frac{f(x+\varepsilon) - f(x-\varepsilon)}{2\varepsilon}$$

central differences

$$\varepsilon = \sqrt[3]{\pi}$$

`std::numeric_limits<float>::epsilon()`

- automatic differentiation
  - overload number type to jointly compute $[f(\boldsymbol{x}), \nabla f(\boldsymbol{x})]$
  - all functions and operators are overloaded to jointly compute their derivatives
  - backward substitution mode can reuse computations in $f(\boldsymbol{x})$ for $\nabla f(\boldsymbol{x})$ allowing for computation of $\nabla f(\boldsymbol{x})$ without significant overhead

Positive Definitness

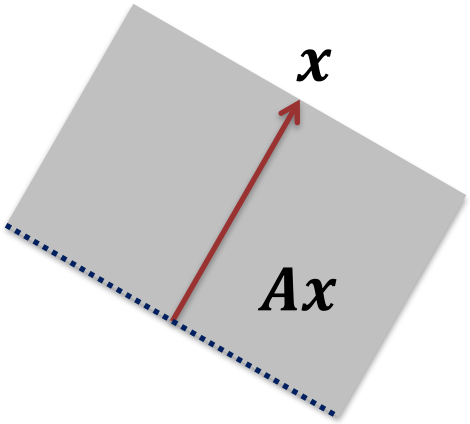
- a matrix $\boldsymbol{A}$ is called positive semi-definite / definite iff

$$\forall \boldsymbol{x}: \boldsymbol{x}^T \boldsymbol{A}\boldsymbol{x} \geq 0 / \boldsymbol{x}^T \boldsymbol{A}\boldsymbol{x} > 0$$

- we denote briefly:

$$\boldsymbol{A} \geq 0 / \boldsymbol{A} > 0$$

- geometrically this implies that no vector is mapped to a direction opposite to itself



Detecting Minima using Derivatives

- Use Taylor expansion in case of continuously differentiable functions

$$f(\boldsymbol{x}+\boldsymbol{h}) = f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^T \boldsymbol{h} + \frac{1}{2}\boldsymbol{h}^T \nabla^2 f(\boldsymbol{x})\boldsymbol{h} + \cdots$$

- necessary conditions: 1st order 2nd order

$$\boldsymbol{x}^* \text{ minimizer} \Rightarrow \nabla f(\boldsymbol{x}^*) = \boldsymbol{0} \text{ and } \nabla^2 f(\boldsymbol{x}^*) \geq 0$$

- A point $\boldsymbol{x}$ with $\nabla f(\boldsymbol{x}) = \boldsymbol{0}$ is called stationary. Stationary points can also be saddle points
- 2nd order sufficient condition

$$\nabla f(\boldsymbol{x}^*) = \boldsymbol{0} \text{ and } \nabla^2 f(\boldsymbol{x}^*) > 0 \Rightarrow \boldsymbol{x}* \text{ is strict minimizer}$$

opposite direction
not true for example for
$f(x) = x^4$

Mathematical Optimization

SINGULAR VALUE DECOMPOSITION

Linear Transformations

- a linear transformation can be represented by a matrix $\boldsymbol{A}$
- a vector $\boldsymbol{x}$ can be transformed to $\boldsymbol{y}$ by matrix vector multiplication

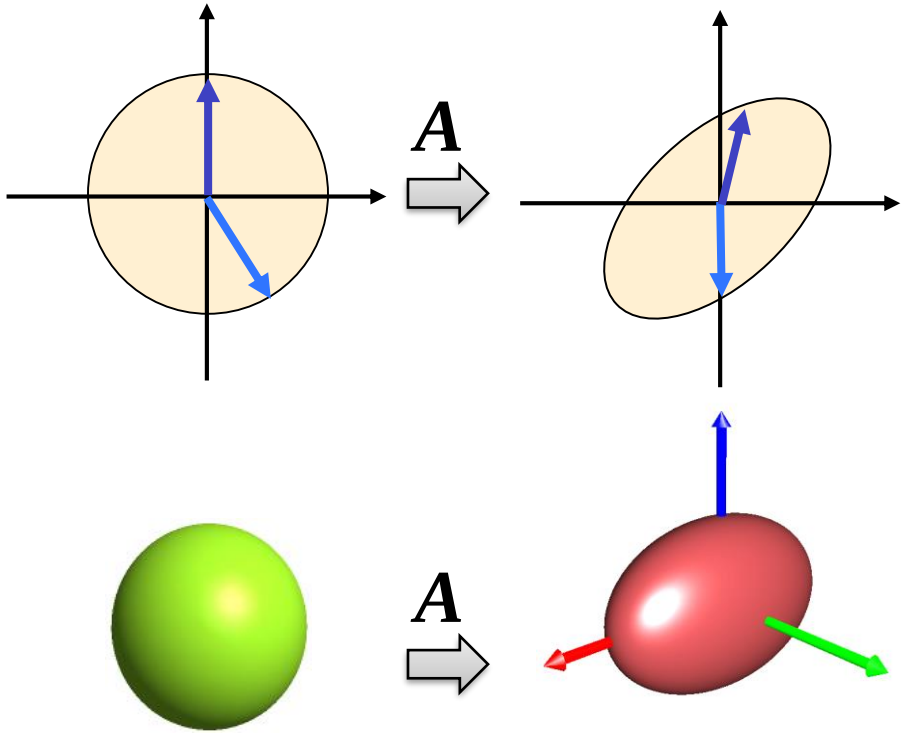
$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$$

Geometric Interpretation

- restriction to unit vectors as $\boldsymbol{A}(\alpha\boldsymbol{x}) = \alpha(\boldsymbol{A}\boldsymbol{x})$
- any matrix maps unit sphere to hyper-ellipsoid

Understanding

- find vectors that map to major axes of ellipsoid

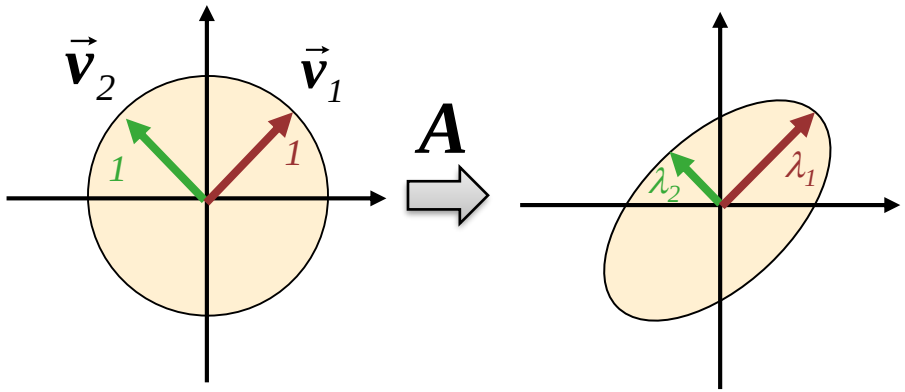


Symmetric Matrices

- In case of symmetric matrices, the eigenvectors of $\boldsymbol{A}$ are the axes of the ellipsoid
- the Eigen-decomposition $\boldsymbol{V}$, $\boldsymbol{\Lambda}$ of $\boldsymbol{A}$ tells us in the columns of $\boldsymbol{V}$, which orthogonal axes it scales, and the entries of the diagonal matrix $\boldsymbol{\Lambda}$ by how much.

$$\mathbf{A} = \mathbf{V}\boldsymbol{\Lambda}\,\mathbf{V}^T$$

$$\boldsymbol{A} = \begin{pmatrix} \vec{\boldsymbol{v}}_1 & \dots & \vec{\boldsymbol{v}}_n \end{pmatrix} \begin{pmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_n \end{pmatrix} \begin{pmatrix} \vec{\boldsymbol{v}}_1^T \\ \vdots \\ \vec{\boldsymbol{v}}_n^T \end{pmatrix}$$



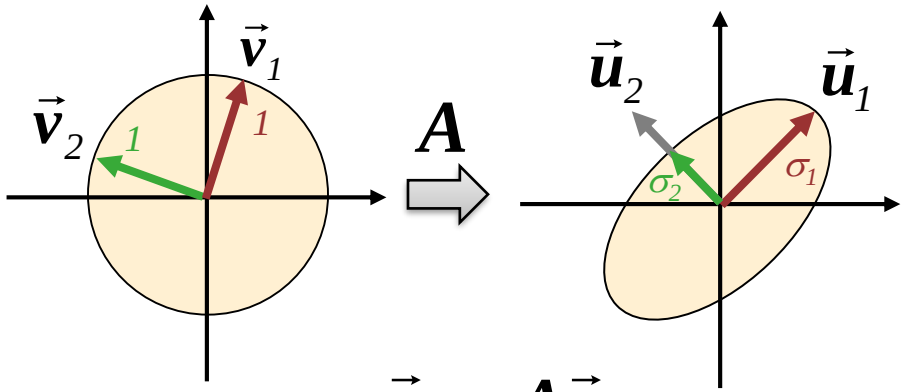
$$\lambda_i \vec{\boldsymbol{v}}_i = \boldsymbol{A}\vec{\boldsymbol{v}}_i$$

General Quadratic Case

- In the general case $\boldsymbol{A}$ also contains a rotation and maps the right singular vectors in columns of $\boldsymbol{V}$ to the left singular vectors $\boldsymbol{U}$ that also correspond to the axes of the ellipsoid, whose lengths are called singular values
- The change of bases between $\boldsymbol{V}$ and $\boldsymbol{U}$ can be used to ensure that all singular values are positive and to permute the major axes such that the singular values decrease in size.

$$\mathbf{A} = \mathbf{U}\boldsymbol{\Sigma}\,\mathbf{V}^T$$

$$\boldsymbol{A} = \begin{pmatrix} \vec{\boldsymbol{u}}_1 & \dots & \vec{\boldsymbol{u}}_n \end{pmatrix} \begin{pmatrix} \sigma_1 & & 0 \\ & \ddots & \\ 0 & & \sigma_n \end{pmatrix} \begin{pmatrix} \vec{\boldsymbol{v}}_1^T \\ \vdots \\ \vec{\boldsymbol{v}}_n^T \end{pmatrix}$$



$$\sigma_i \vec{\boldsymbol{u}}_i = \boldsymbol{A}\vec{\boldsymbol{v}}_i$$

$$\sigma_1 \geq \dots \geq \sigma_n \geq 0$$

General Rectangular Case

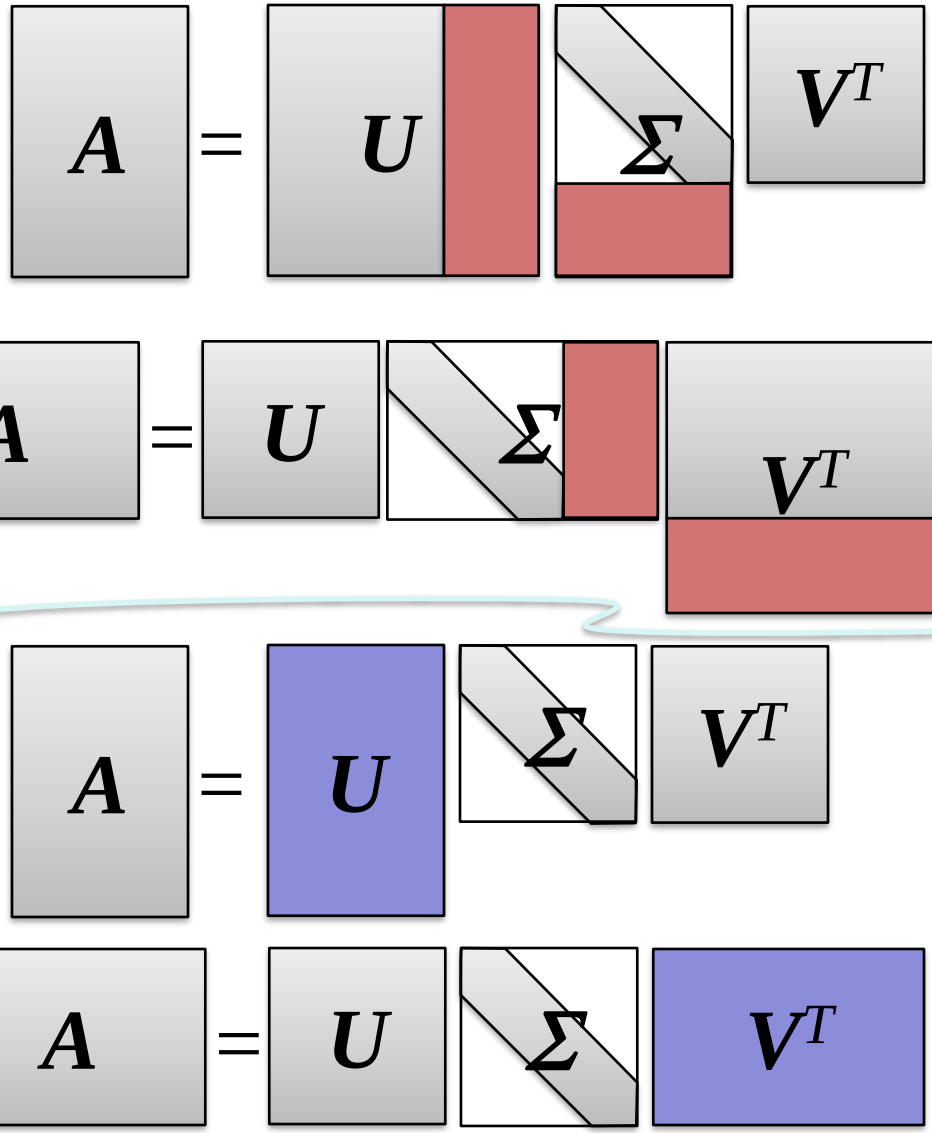
- for rectangular matrices the dimensions of unit sphere and ellipsoid can differ

additional dimensions

- singular values are zero
- singular vectors give not unique orthogonalization

reduced forms

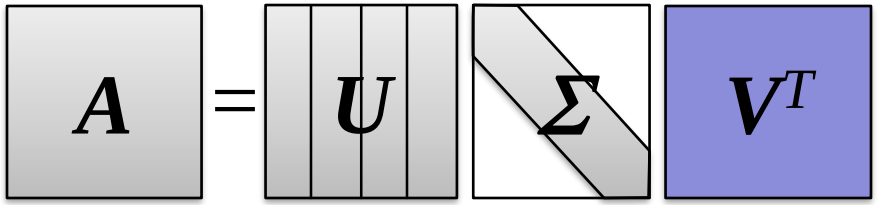
- $\boldsymbol{U}$ or $\boldsymbol{V}$ is rectangular
- costs less memory and is faster to compute



Matrix Approximation

- Rearranging the SVD, one can decompose a matrix into components of decreasing singular / eigenvalues:

$$\boldsymbol{A} = \sigma_1 \vec{\boldsymbol{u}}_1 \vec{\boldsymbol{v}}_1^T + \sigma_2 \vec{\boldsymbol{u}}_2 \vec{\boldsymbol{v}}_2^T + \ldots + \sigma_n \vec{\boldsymbol{u}}_n \vec{\boldsymbol{v}}_n^T$$



- cutting the decomposition after m terms yields the approximation $\tilde{\boldsymbol{A}}_m = \sigma_1 \vec{\boldsymbol{u}}_1 \vec{\boldsymbol{v}}_1^T + \ldots + \sigma_m \vec{\boldsymbol{u}}_m \vec{\boldsymbol{v}}_m^T$
- The error of the approximation with respect to the Frobenius norm can be computed from the singular values: $\left\| \boldsymbol{A} - \tilde{\boldsymbol{A}}_m \right\|_2^2 = \left\| \sigma_{m+1} \vec{\boldsymbol{u}}_{m+1} \vec{\boldsymbol{v}}_{m+1}^T + \ldots + \sigma_n \vec{\boldsymbol{u}}_n \vec{\boldsymbol{v}}_n^T \right\|_2^2 = \sigma_{m+1}^2 + \ldots + \sigma_n^2$

Discussion

- For symmetric matrices, the SVD can be used to compute the eigenvalue decomposition:

$$\boldsymbol{A} = \boldsymbol{A}^T \Rightarrow \boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^T$$

- Numerical SVD is an expensive operation
- We always need to pay attention to the dimensions of the matrix we're applying SVD to.
- In some applications one needs only a very limited set of the largest or smallest singular/Eigen value and vectors. More efficient iterative algorithms exist to extract this.

Mathematical Optimization

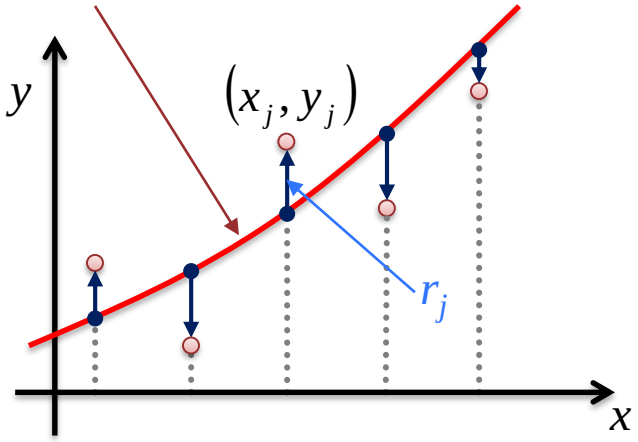
SOLVING LINEAR LEAST SQUARES PROBLEMS

Least Squares Fitting

- Given a model function $g(\boldsymbol{p}; \boldsymbol{x})$, and a set of m data points $(\boldsymbol{x}_j, y_j)$
- residuals r_j are signed distances between model function and data points
- Least squares: minimize sum of squared residuals to find best model parameters $\boldsymbol{p}^*$
- linear least squares (LLS) residuals depend linearly on optimization variables p_i, such that basis $\phi_i(\boldsymbol{x})$ exists

Example polynomial fitting:

$$g(\boldsymbol{p}; x) = p_1 + p_2 x + p_3 x^2$$



residual: $r_j(\boldsymbol{p}) = g(\boldsymbol{p}; \boldsymbol{x}_j) - y_j$

$$r_j(\boldsymbol{p}) = p_1 + p_2 x + p_3 x^2 - y_j$$

least squares fit:

$$\boldsymbol{p}^* = \underset{\boldsymbol{p}}{\operatorname{argmin}} \sum_{j=1}^{m} r_j^2(\boldsymbol{p})$$

linear least squares fit:

$$g(\boldsymbol{p}; \boldsymbol{x}) = \sum_{i=1}^{n} p_i \cdot \phi_i(\boldsymbol{x})$$

Matrix Form of LLS

- Given the following linear least squares (LLS) problem:

$$\boldsymbol{p}^* = \underset{\boldsymbol{p}}{\operatorname{argmin}} \sum_{j=1}^{m} r_j^2(\boldsymbol{p}), \; r_j(\boldsymbol{p}) = g(\boldsymbol{p}; \boldsymbol{x}_j) - y_j, g(\boldsymbol{p}; \boldsymbol{x}) = \sum_{i=1}^{n} p_i \cdot \phi_i(\boldsymbol{x})$$

$f(\boldsymbol{p})$

- one introduces the matrix notation:
  - matrix $\boldsymbol{A} = \begin{pmatrix} \phi_1(\boldsymbol{x}_1) & \cdots & \phi_n(\boldsymbol{x}_1) \\ \vdots & \ddots & \vdots \\ \phi_1(\boldsymbol{x}_m) & \cdots & \phi_n(\boldsymbol{x}_m) \end{pmatrix} \in \boldsymbol{R}^{m \times n}$ with m rows and n columns

$$\boldsymbol{A} = \begin{pmatrix} 1 & x_1 & x_1^2 \\ \vdots & \vdots & \vdots \\ 1 & x_m & x_m^2 \end{pmatrix}$$

  - result vector $\boldsymbol{b} = (y_1, y_2, \ldots y_m)$
  - residual vector $\boldsymbol{r} = \big(r_1(\boldsymbol{p}), \ldots, r_m(\boldsymbol{p})\big) = \boldsymbol{Ap} - \boldsymbol{b}$
- The objective function can be written as

$$f(\boldsymbol{p}) = \boldsymbol{r}^T \boldsymbol{r} = \|\boldsymbol{Ap} - \boldsymbol{b}\|_2^2$$

- The LLS objective function is convex and has a globally unique minimum if $\operatorname{rank}(\boldsymbol{A}) = n = \dim(\boldsymbol{p})$

Normal Equations

- Setting the gradient of the objective function to zero yields the normal equations:

$$f(\boldsymbol{p}) = \boldsymbol{r}^T\boldsymbol{r} = \langle \boldsymbol{Ap} - \boldsymbol{b}, \boldsymbol{Ap} - \boldsymbol{b} \rangle$$

$$\mathbf{0} = f_{\boldsymbol{p}}(\boldsymbol{p}) = \langle \boldsymbol{Ap} - \boldsymbol{b}, \boldsymbol{Ap} - \boldsymbol{b} \rangle_{\boldsymbol{p}}$$

$$= \langle \boldsymbol{Ap}, \boldsymbol{Ap} \rangle_{\boldsymbol{p}} - 2\langle \boldsymbol{Ap}, \boldsymbol{b} \rangle_{\boldsymbol{p}}$$

$$= 2\boldsymbol{A}^T\boldsymbol{Ap} - 2\boldsymbol{A}^T\boldsymbol{b}$$

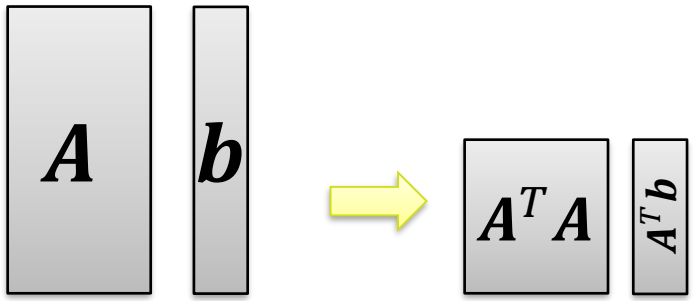
$$\boldsymbol{A}^T\boldsymbol{Ap} = \boldsymbol{A}^T\boldsymbol{b}$$

- If $\mathrm{rank}(\boldsymbol{A}) = n$, it is possible to use the pseudo inverse $\boldsymbol{A}^+$ to compute the least squares solution in a closed form

$$\boldsymbol{A}^+ := (\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T \in R^{n \times n}$$

$$\boldsymbol{p} = \boldsymbol{A}^+\boldsymbol{b}$$

- the sizes of $\boldsymbol{A}^T\boldsymbol{A}$ and $\boldsymbol{A}^T\boldsymbol{b}$ only depend on the number of parameters n and not on the number of data points!



Overdetermined case

- $\boldsymbol{A}$ has more rows than columns
- LLS $\min \boldsymbol{r}^T\boldsymbol{r}$ minimizes squared length of residual vector $\boldsymbol{r} = \boldsymbol{Ap} - \boldsymbol{b}$, for $\boldsymbol{p}$.
- Split $\boldsymbol{A} = (\boldsymbol{a}_{.1} \ldots \boldsymbol{a}_{.n})$ into n columns $\boldsymbol{a}_{.i}$ and find linear combination that best approximates $\boldsymbol{b}$.
- the shortest length residual $\boldsymbol{r}$ is orthogonal to all columns $\boldsymbol{a}_{.i}$.
- This again yields the normal equations

$m > n$

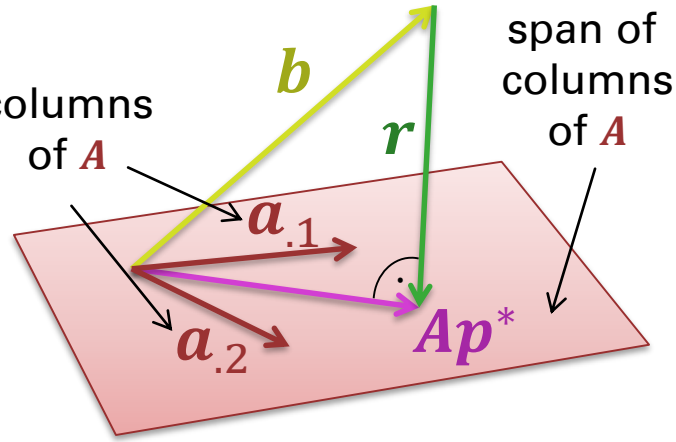


illustration for $n = 2$ of abstract column space and result vector $\boldsymbol{b}$ outside of column span

$$0 = \boldsymbol{a}_1^T\boldsymbol{r} \quad 0 = \boldsymbol{a}_2^T\boldsymbol{r} \Rightarrow \mathbf{0} = \boldsymbol{A}^T\boldsymbol{r} \xrightarrow{\boldsymbol{r} = \boldsymbol{Ap} - \boldsymbol{b}} \mathbf{0} = \boldsymbol{A}^T(\boldsymbol{Ap} - \boldsymbol{b})$$

$$\boldsymbol{A}^T\boldsymbol{Ap} = \boldsymbol{A}^T\boldsymbol{b}$$

Underdetermined Case

- if $\text{rank}(\boldsymbol{A}) < n$ the pseudo inverse does not exist.
- This can happen if
  1. $m < n$, i.e. there are less data points than parameters, or
  2. columns of $\boldsymbol{A}$ are dependent
- 2nd case can be detected and handled with SVD:
  - global epsilon to check whether σ_1 is zero
  - relative epsilon to check whether σ_i / σ_1 is zero
- define rank-k pseudo inverse $\boldsymbol{A}_k^+$ and solve problem with this

$$k = \text{rank}(\boldsymbol{A}) < n$$

$$\boldsymbol{A}$$

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$$

$$\Rightarrow \sigma_1 > \sigma_2 > \cdots > \sigma_k > 0 \approx \sigma_{k+1} \approx \cdots \approx \sigma_n$$

$$\Rightarrow \boldsymbol{A} \approx \boldsymbol{U}_k \cdot \boldsymbol{\Sigma}_k \cdot \boldsymbol{V}_k^T$$

$$\boldsymbol{A}_k^+ := \boldsymbol{V}_k \cdot \boldsymbol{\Sigma}_k^{-1} \cdot \boldsymbol{U}_k^T \Rightarrow \boldsymbol{p}^* = \boldsymbol{A}_k^+ \boldsymbol{b}$$

Illustration of Underdetermined case

- Let $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_m$ be the rows of $\boldsymbol{A}$. Each pair of row and result value $(\boldsymbol{a}_i, b_i)$ corresponds to a plane equation in parameter space:

$$\boldsymbol{a}_1^T \boldsymbol{p} - b_1 = 0$$
$$\boldsymbol{a}_2^T \boldsymbol{p} - b_2 = 0$$

- In the underdetermined case the intersection of all planes forms a space of solutions with $n - \operatorname{rank} \boldsymbol{A}$ dimensions.

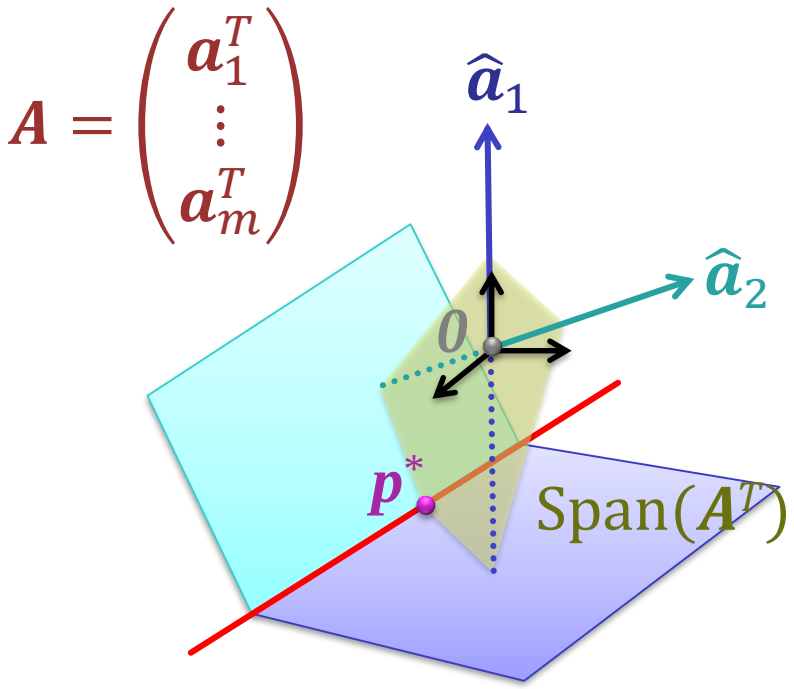
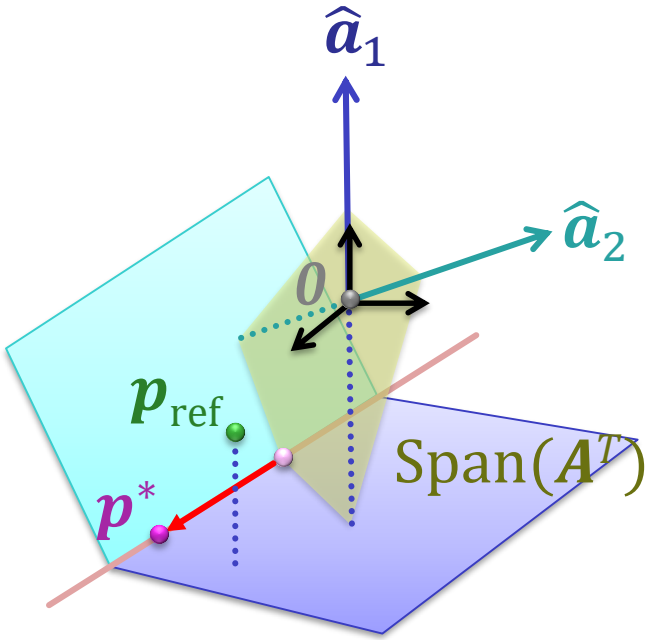


Figure shows the n-dimensional parameter space, with coordinate frame in the middle (origin 0 is grey point). Two linear constraints are visualized by blue and cyan planes. Rows of A are the normal vectors and span of A is yellow plane through origin spanned by normals. All points on red line (plane intersection) are minima, of which the one in the span of A (magenta point) is selected.

Reference Point

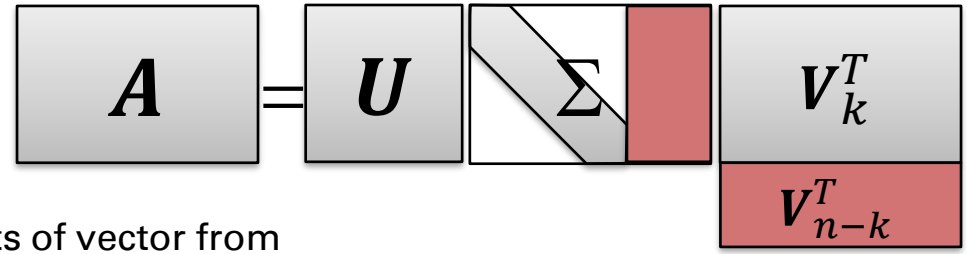
- To disambiguate the underdetermined case one can specify a reference point $\boldsymbol{p}_{\text{ref}}$.
- Then the solution closest to the reference point is wanted, which can be computed from the rest of the $\boldsymbol{V}$-matrix of the complete SVD:

$$\boldsymbol{p}^* = \boldsymbol{A}_k^+ \boldsymbol{b} + \boldsymbol{V}_{n-k} \boldsymbol{V}_{n-k}^T \boldsymbol{p}_{\text{ref}}$$



red difference vector moves old solution to the one closest to reference point

components of vector from origin to $\boldsymbol{p}_{\text{ref}}$ that does not change objective function



Pseudo Code to solve any LLS with SVD

- Robust least squares solver using SVD and reference point for ambiguous cases:

```
solve_ls_svd(A, b, p_ref, ε_glob, ε_loc) -> p
   [U, σ_i, V] = svd(A, sort σ_i=decreasing);
   if σ_1 < ε_glob then return p_ref;
   p = 0;
   for i=1 to A.nr_cols() do
      if σ_i/σ_1 > ε_loc then
         p += V.col(i) * (dot(U.col(i), b)/σ_i);
      else
         p += V.col(i) *  dot(V.col(i), p_ref);
   return p;
```

$$\boldsymbol{p}^* = \overbrace{\boldsymbol{V}_k \boldsymbol{\Sigma}_k^{-1} \boldsymbol{U}_k^T}^{\boldsymbol{A}_k^+} \boldsymbol{b} + \boldsymbol{V}_{n-k} \boldsymbol{V}_{n-k}^T \boldsymbol{p}_{\text{ref}}$$

- Function can be called with $\boldsymbol{A}^T\boldsymbol{A}$ and $\boldsymbol{A}^T\boldsymbol{b}$ as well and yields in both cases the least squares solution corresponding to $\boldsymbol{A}\boldsymbol{p} = \boldsymbol{b}$ which is closest to $\mathrm{p}_{\mathrm{ref}}$.

Weighted Least Squares Fitting

- In a lot of applications, one can express the validity of each data point $(\boldsymbol{x}_j, \boldsymbol{y}_j)$ through a weight $\omega_j > 0$.
- Extension of least squares to weighted least squares:

$$f(\boldsymbol{p}) = \sum_{j=1}^{m} \omega_j r_j^2(\boldsymbol{p})$$

- One can also extend the matrix form of LLS to weighted linear least squares (WLLS) by introducing the diagonal matrix $\boldsymbol{W} = \mathrm{diag}(\omega_1, \ldots, \omega_m)$:

$$f(\boldsymbol{p}) = \boldsymbol{r}^T \boldsymbol{W} \boldsymbol{r} = \left\| \sqrt{\boldsymbol{W}} (\boldsymbol{A}\boldsymbol{p} - \boldsymbol{b}) \right\|_2^2$$

- And the normal equations become: $\boldsymbol{A}^T \boldsymbol{W} \boldsymbol{A} \boldsymbol{p} = \boldsymbol{A}^T \boldsymbol{W} \boldsymbol{b}$
- If $\mathrm{rank}(\boldsymbol{A}) = n$, one can define weighted pseudo inverse:

$$\boldsymbol{A}_{\boldsymbol{W}}^{+} := (\boldsymbol{A}^T \boldsymbol{W} \boldsymbol{A})^{-1} \boldsymbol{A}^T \boldsymbol{W} \in R^{n \times n} \Rightarrow \boldsymbol{p}^* = \boldsymbol{A}_{\boldsymbol{W}}^{+} \boldsymbol{b}$$

Some Remarks to Linear Least Squares

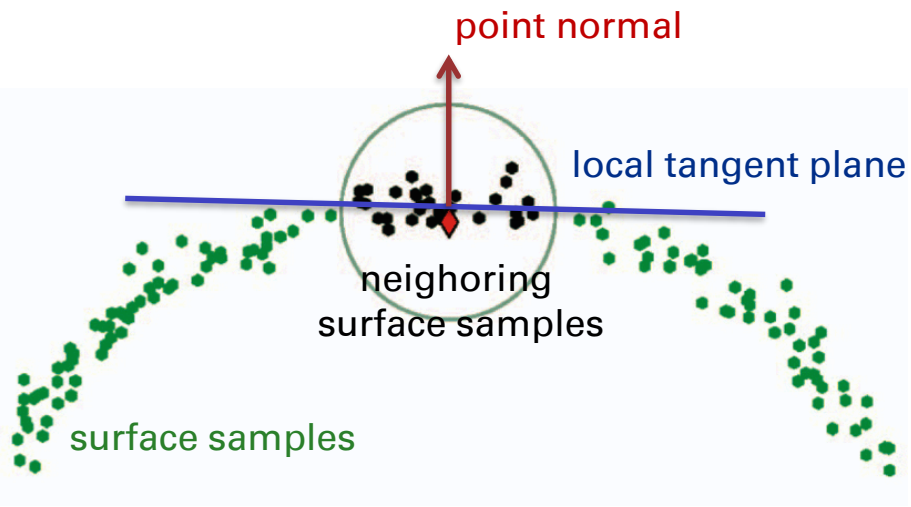
- Linear Least squares is a useful technique to solve overdetermined systems of linear equations
- If residuals correspond to measurements with errors distributed according to normal distribution, LLS is maximum likelihood estimator.

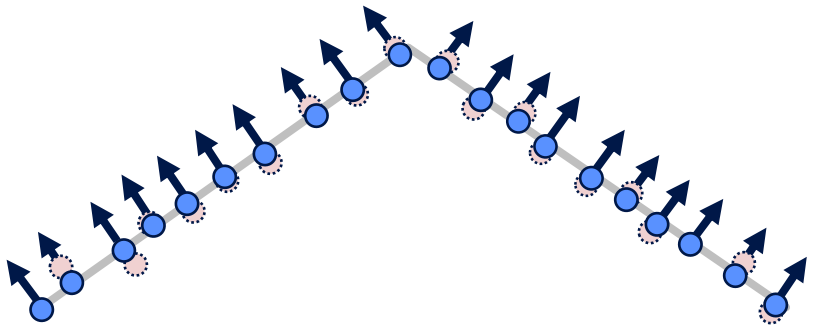
Surface Denoising and Surface Normal Estimation

PLANE FITTING

Normal Estimation in 3D Scans

- Input: set of 3D points sampled from surface
- Output: set of denoised 3D points with normal
- Approach:
  - for each point collect neighborhood
  - define distance-based weight function
  - estimate local tangent plane from weighted least squares problem
  - orthogonally project input point onto local tanget plane
  - assign tangent plane normal to output point





integrated normal estimation and denoising

Weighted Plane Fitting Problem

- for each point $\boldsymbol{x}$ of point cloud collect m points $\boldsymbol{x}_j$ with knn-query sorted by distance, typically $10 < m < 50$.
- estimate reference radius: $h = \|\boldsymbol{x}_{10} - \boldsymbol{x}\|$
- assign pre-weights: $\omega_j' = \exp\left(-\frac{\|x_j - x\|^2}{h^2}\right)$ and normalize: $\omega_j = \frac{\omega_j'}{\Omega'}, \quad \Omega' = \sum_{j=1}^{m} \omega_j{}'$.
- residuals: $r_j = \hat{\boldsymbol{n}}^T \boldsymbol{x}_j + d = \tilde{\boldsymbol{p}}^T \tilde{\boldsymbol{x}}_j$ with $\tilde{\boldsymbol{x}}_j = (\boldsymbol{x}_j, 1)$
- parameters: $\tilde{\boldsymbol{p}} = (\hat{\boldsymbol{n}}, d)$ with $\|\hat{\boldsymbol{n}}\| = 1$
- As $\boldsymbol{b} = \boldsymbol{0}$ the normalization constraint is essential to avoid the trivial solution $\tilde{\boldsymbol{p}}^* = \boldsymbol{A}_{\boldsymbol{W}}^{+}\boldsymbol{b} = \boldsymbol{0}$. The constrained LLS is:

$$\underset{\tilde{\boldsymbol{p}}=(\hat{\boldsymbol{n}},d)|\|\hat{\boldsymbol{n}}\|=1}{\operatorname{minarg}} \quad f(\tilde{\boldsymbol{p}}), f(\tilde{\boldsymbol{p}}) = \left\|\sqrt{\boldsymbol{W}}\boldsymbol{A}\tilde{\boldsymbol{p}}\right\|_2^2 = \tilde{\boldsymbol{p}}^T \boldsymbol{A}^T \boldsymbol{W} \boldsymbol{A} \tilde{\boldsymbol{p}} = \tilde{\boldsymbol{p}}^T \boldsymbol{M}_{\boldsymbol{W}} \tilde{\boldsymbol{p}}$$

- For brevity we define a weighted cov. matrix $\boldsymbol{M}_{\boldsymbol{W}}$

Reduced Plane Fitting Problem

- Theorem: if $\tilde{\boldsymbol{p}}^* = (\hat{\boldsymbol{n}}^*, d^*)$ is the solution to the plane fitting problem, then the weighted center of mass

$$\bar{\boldsymbol{x}} = \sum_{j=1}^{m} \omega_j \boldsymbol{x}_j$$

  is on $\tilde{\boldsymbol{p}}^*$, i.e. $\hat{\boldsymbol{n}}^{*T} \bar{\boldsymbol{x}} + d^* = 0$. [Proof by setting $\partial_d f(\tilde{\boldsymbol{p}}) = 0$]

- We can enforce $d = 0$ by considering a coordinate system with $\bar{\boldsymbol{x}}$ in the origin, a reduced parameter vector $\boldsymbol{p}'$ and reduced weighted cov. matrix:

$$\boldsymbol{p}' = \hat{\boldsymbol{n}}, \qquad \boldsymbol{x}'_j = \boldsymbol{x}_j - \bar{\boldsymbol{x}}, \qquad \boldsymbol{M}'_{\boldsymbol{W}} := \boldsymbol{A}'^T \boldsymbol{W} \boldsymbol{A}' = \sum_{j=1}^{m} \omega_j \boldsymbol{x}'_j \boldsymbol{x}'^T_j$$

- The plane fitting problem reduces to

$$\hat{\boldsymbol{n}}^* = \underset{\hat{\boldsymbol{n}} \mid \|\hat{\boldsymbol{n}}\| = 1}{\operatorname{minarg}} \hat{\boldsymbol{n}}^T \boldsymbol{M}'_{\boldsymbol{W}} \hat{\boldsymbol{n}}$$

Reduced Plane Fitting Problem

- Eigenvalue decomposition of the symmetric matrix $\boldsymbol{M}'_{\boldsymbol{W}}$

$$\boldsymbol{M}'_{\boldsymbol{W}} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^T = \lambda_1 \boldsymbol{v}_1 \boldsymbol{v}_1^T + \lambda_2 \boldsymbol{v}_2 \boldsymbol{v}_2^T + \lambda_3 \boldsymbol{v}_3 \boldsymbol{v}_3^T, \lambda_1 \leq \lambda_2 \leq \lambda_3$$

- plugged into the objective function yields

$$\hat{\boldsymbol{n}}^T \boldsymbol{M}'_{\boldsymbol{W}} \hat{\boldsymbol{n}} = \lambda_1 (\boldsymbol{v}_1^T \hat{\boldsymbol{n}})^2 + \lambda_2 (\boldsymbol{v}_2^T \hat{\boldsymbol{n}})^2 + \lambda_3 (\boldsymbol{v}_3^T \hat{\boldsymbol{n}})^2$$

- From the increasing ordering of the λ_i it follows that the optimal normal is given by

$$\hat{\boldsymbol{n}}^* = \pm \boldsymbol{v}_1$$

- Note that the sign of the normal direction (plane orientation) is not unique. A globally consistent orientation is typically achieved by
  - knowledge of an exterior point (scanner location)
  - a region growing normal orientation algorithm

Summary of Plane Fitting Problem

- Input: set of m weighted points $\boldsymbol{x}_j, \omega_j'$.

1. normalize weights: $\omega_j = \frac{\omega_j'}{\Omega'}, \quad \Omega' = \sum_{j=1}^{m} \omega_j'$
2. compute weighted center of mass $\bar{\boldsymbol{x}} = \sum_{j=1}^{m} \omega_j \boldsymbol{x}_j$
3. transform points: $\boldsymbol{x}_j' = \boldsymbol{x}_j - \bar{\boldsymbol{x}}$
4. compute weighted cov. matrix: $\boldsymbol{M}_{\boldsymbol{W}}' = \sum_{j=1}^{m} \omega_j \boldsymbol{x}_j' \boldsymbol{x}_j'^{T}$
5. compute Eigenvector $\boldsymbol{v}_1$ of smallest Eigenvalue of $\boldsymbol{M}_{\boldsymbol{W}}'$

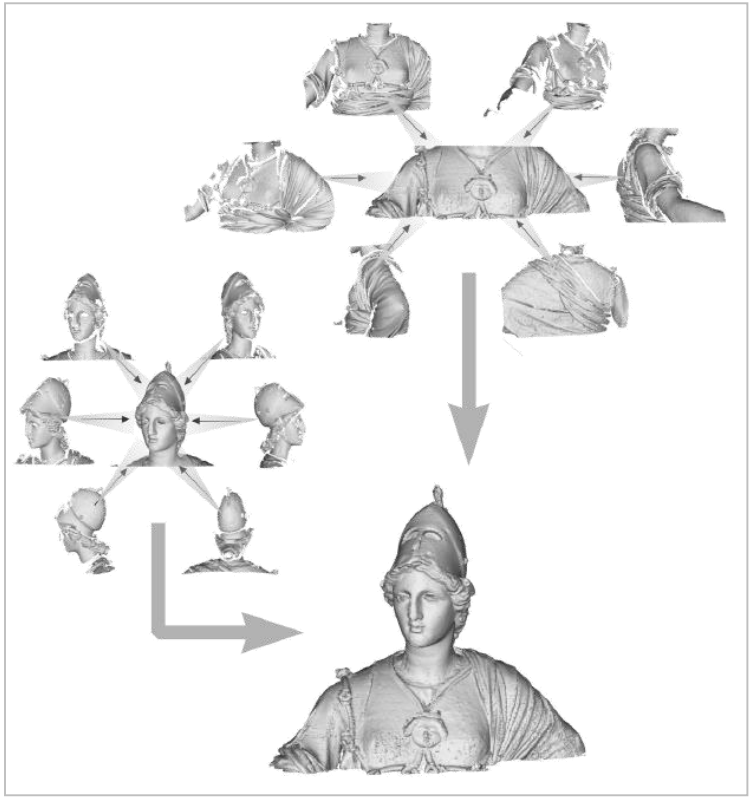
- Output: return plane through $\bar{\boldsymbol{x}}$ orthogonal to $\hat{\boldsymbol{n}}^* = \pm \boldsymbol{v}_1$.

Fitting rigid transformations to a set of point-to-point correspondences

REGISTRATION OF 3D DATA SETS

3D Dataset Registration

- Registration is the process of bringing two data sets into a joint coordinate system based on feature correspondences.
  - common features are points, lines and planes
  - common correspondences are point-to-point, point-to-plane or line-to-line
  - one distinguishes between rigid and non-rigid registration
- For registration of 3D scans we consider rigid registration with point-to-point or point-to-plane correspondences.
- Given two scans A & B we want to find a rigid transformation T s.t. $A = T(B)$
- Standard approach: ICP algorithm



for acquisition of a 3D Model several 3D-Scans from different view points need to be transformed into a common coordinate system and then fused

Iterative Closest Points (ICP) Algorithm

- Input: two point clouds and coarse initial alignment
- ICP alternates between generation of correspondences based on closeness according to some distance function and the alignment according to correspondences.
- algorithm is iterative and assumes a coarse initial alignment of the 3D scans, as well as an overlap of the scans
- Pseudo-Code:

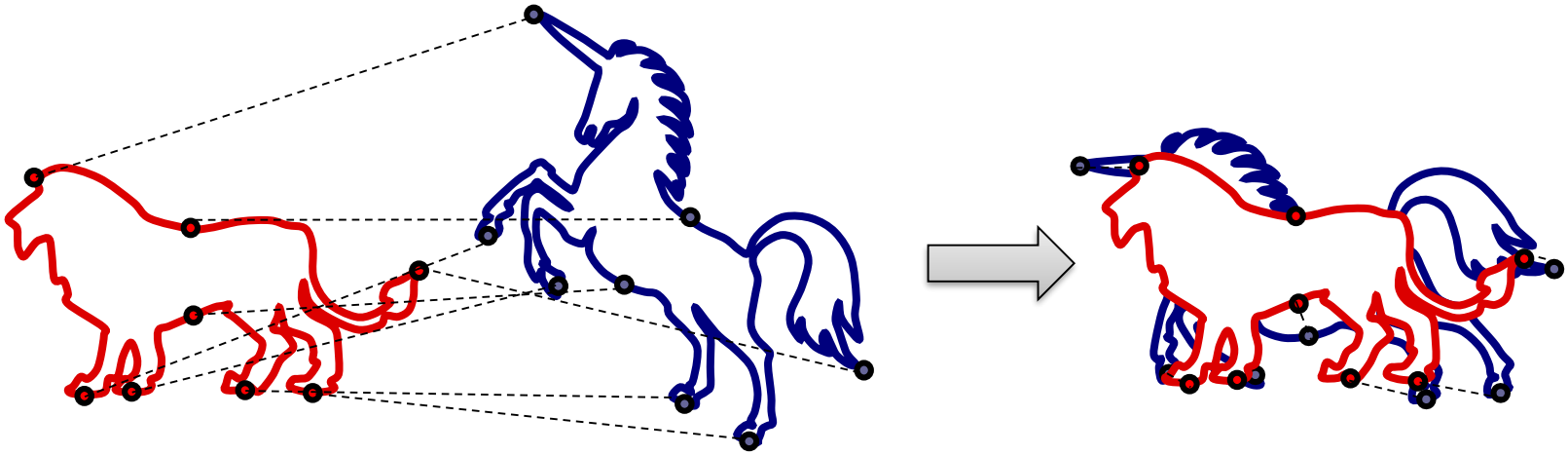
1. find coarse initial alignment T_0
   (you can use markers, geometry features or do it manually)
2. find correspondences: subsample A and B to S_A and S_B and find $\forall a \in S_A$ closest point $b_a \in B$ and define (a, b_a) as correspondence (similarly $\forall b \in S_B$). Filter correspondences, for example only symmetric ones where a is closest point to b_a.
3. compute T_{i+1} such that squared distance of all corresponding point pairs with respect to T_i is minimized (Kabsch Algorithm)

until convergence: $T_{i+1} \approx T_i$

Kabsch Algorithm

- Input: n correspondences $\{(\boldsymbol{p}_1, \boldsymbol{q}_1), \ldots, (\boldsymbol{p}_n, \boldsymbol{q}_n)\}$ on two different shapes A and B.
- Goal: find rigid transformation $(\boldsymbol{R}, \vec{\boldsymbol{t}})$ (rotation matrix $\boldsymbol{R}$ and translation vector $\vec{\boldsymbol{t}}$) that minimizes the squared distances between all point-to-point correspondences:

$$(\boldsymbol{R}^*, \vec{\boldsymbol{t}}^*) = \underset{\boldsymbol{R}, \vec{\boldsymbol{t}} | \boldsymbol{R}\boldsymbol{R}^T = \boldsymbol{1}}{\operatorname{minarg}} \sum_{i=1}^{n} \left\| \boldsymbol{p}_i - (\boldsymbol{R}\boldsymbol{q}_i + \vec{\boldsymbol{t}}) \right\|^2$$



Kabsch Algorithm – optimal translation

- Similarly to plane fitting it turns out that we can solve the translation separately.
- Theorem: if $(\boldsymbol{R}^*, \vec{\boldsymbol{t}}^*)$ is the optimal transformation, then the points $\{\boldsymbol{p}_i\}$ and $\{\boldsymbol{R}^*\boldsymbol{q}_i + \vec{\boldsymbol{t}}^*\}$ have the same centers of mass.
- Centers of mass:

$$\overline{\boldsymbol{p}} = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{p}_i\,, \qquad \overline{\boldsymbol{q}} = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{q}_i$$

- Plugging into theorem yields:

$$\overline{\boldsymbol{p}} = \frac{1}{n}\sum_{i=1}^{n} \left(\boldsymbol{R}^*\boldsymbol{q}_i + \vec{\boldsymbol{t}}^*\right) = \boldsymbol{R}^*\overline{\boldsymbol{q}} + \vec{\boldsymbol{t}}^*$$

- Finally, solving for translation: $\vec{\boldsymbol{t}}^* = \overline{\boldsymbol{p}} - \boldsymbol{R}^*\overline{\boldsymbol{q}}$

Kabsch Algorithm – optimal rotation

- Translate the input points to the centroids:

$$\boldsymbol{p}_i' = \boldsymbol{p}_i - \overline{\boldsymbol{p}}, \qquad \boldsymbol{q}_i' = \boldsymbol{q}_i - \overline{\boldsymbol{q}}$$

- Compute the "covariance matrix"

$$\boldsymbol{H} = \sum_{i=1}^{n} \boldsymbol{q}_i' \boldsymbol{p}_i'^{T}$$

more details and proofs can be found in Olga Sorkine's note: igl.ethz.ch/projects/ARAP/svd_rot.pdf

- Compute the SVD of $\boldsymbol{H}$:

$$\boldsymbol{H} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$$

- The optimal rotation is

$$\boldsymbol{R}^* = \boldsymbol{V} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \det(\boldsymbol{V}\boldsymbol{U}^T) \end{pmatrix} \boldsymbol{U}^T$$

(avoid reflections)

- The translation vector is: $\vec{\boldsymbol{t}}^* = \overline{\boldsymbol{p}} - \boldsymbol{R}^* \overline{\boldsymbol{q}}$