Least Square Solutions

Reporter: Huang Yangyang, Xiao Shuisheng

2018-11-19

Overview

- Least squares
  - Problem && Algorithm
  - Optimality
  - Generalized errors
  - Robust least squares
  - Least-squares solutions to large, sparse matrix

Overview

- Least squares
  - Problem && Algorithm
  - Optimality
  - Generalized errors
  - Robust least squares
  - Least-squares solutions to large, sparse matrix

Problem && Algorithm

- Given a Point set , $P := \{p_i\} \quad (p_i = x_1, x_2, ..., x_d)$ find the best fit hyperplane

$$f(X) = w_1 x_1 + w_2 x_2 + ... + w_d x_d + c$$

$$X = \{\ 1,\quad x_1,\quad x_2,\quad ...\quad x_d\ \}^T$$

$$W^T = \{\ c,\quad w_1,\quad w_2,\quad ...\quad w_d\ \}$$

- Suppose model from which data is observed*: $h(x) = w^T x$*

Problem && Algorithm

- Error measure : squared error

$$E(W) = \frac{1}{N}\sum_{i=1}^{N}(h(x_i) - y_i)$$

how to minimize E(w)?

Matrix Form of $E_{in}(w)$

$$E_{in}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{w}^T\mathbf{x}_n - y_n)^2 = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}_n^T\mathbf{w} - y_n)^2$$

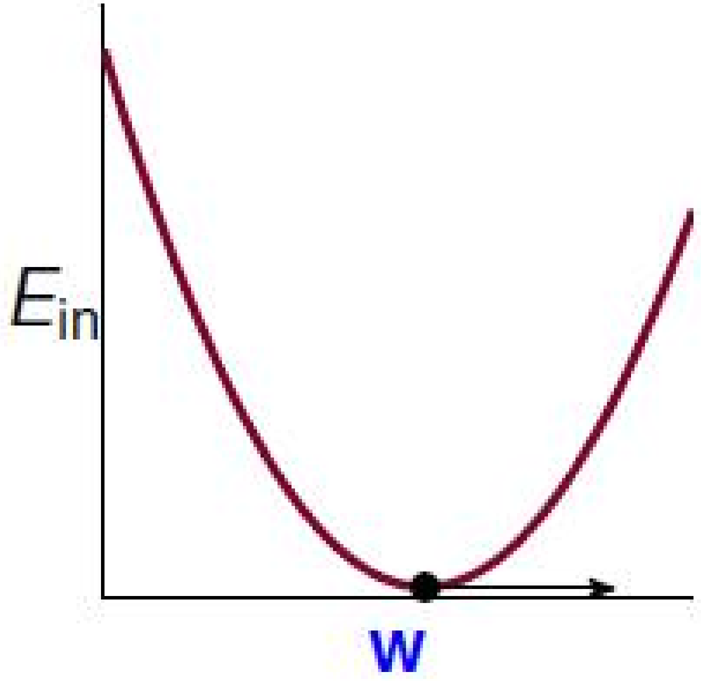
$$= \frac{1}{N}\left\| \begin{matrix} \mathbf{x}_1^T\mathbf{w} - y_1 \\ \mathbf{x}_2^T\mathbf{w} - y_2 \\ \cdots \\ \mathbf{x}_N^T\mathbf{w} - y_N \end{matrix} \right\|^2$$

$$= \frac{1}{N}\left\| \begin{bmatrix} --\mathbf{x}_1^T-- \\ --\mathbf{x}_2^T-- \\ \cdots \\ --\mathbf{x}_N^T-- \end{bmatrix} \mathbf{w} - \begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_N \end{bmatrix} \right\|^2$$

$$= \frac{1}{N}\| \underbrace{\mathrm{X}}_{N\times d+1} \underbrace{\mathbf{w}}_{d+1\times 1} - \underbrace{\mathbf{y}}_{N\times 1} \|^2$$

$$\min E(w) = \frac{1}{N}\|Xw - y\|^2$$

- E(w) : continuous, differentiable, convex
- necessary condition of best **w**:
- $\nabla E(\boldsymbol{w}) = \begin{bmatrix} \frac{\partial E(w)}{\partial w_0} \\ \frac{\partial E(w)}{\partial w_1} \\ \cdots \\ \frac{\partial E(w)}{\partial w_2} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \cdots \\ 0 \end{bmatrix}$



- $\nabla E(\boldsymbol{w}) = \frac{1}{N}(2X^T X w - 2X^T y)$

- Minimizing w yields the equation

$$X^T X w = X^T y$$

- *$X^T X$ is non-singular*, invertible $X^T X$:
  - $W_{LIN} = (X^T X)^{-1} X^T y$
  - $P = X(X^T X)^{-1} X^T$ Symmetric and orthogonal
  - $Py(= Xw)$ is orthogonal to the residual $r = y - Py$
- *$X^T X$ is singular* solution is not unique

Overview

- Least squares
  - Problem && Algorithm
  - Optimality
  - Generalized errors
  - Robust least squares
  - Least-squares solutions to large, sparse matrix

Optimality

- Gauss-Markow theorem

If $\{b_i\}_m$ and $\{x_j\}_n$ are two sets of random variables such that

$$e_i = b_i - a_{i,1}x_1 - \ldots - a_{i,n}x_n$$

and

$$\begin{aligned} &A1: && \{a_{i,j}\} \text{ are not random variables,} \\ &A2: && E(e_i) = 0, \text{ for all } i, \\ &A3: && var(e_i) = \sigma^2, \text{ for all } i, \text{ and} \\ &A4: && cov(e_i, e_j) = 0, \text{ for all } i \text{ and } j, \end{aligned}$$

then the least-squares estimator,

$$\hat{X}_{LSE}(b_1, \ldots, b_m) = \arg\min_x \sum_i e_i^2,$$

is the **best unbiased linear estimator**.

Optimality

- P: $X(X^TX)^{-1}X^T$

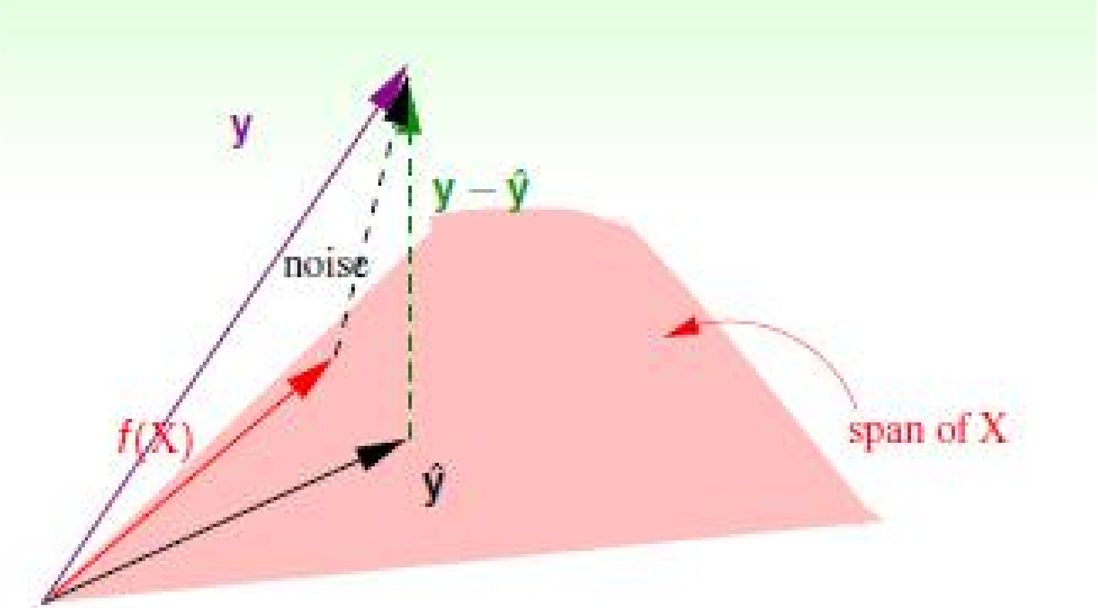
$$E(\boldsymbol{W}_{LIN}) = \frac{1}{N}\|y-\hat{y}\|^2 = \frac{1}{N}\|y-Py\|^2$$

$$= \frac{1}{N}\|(I\ -P)y\|^2$$

$$E(\boldsymbol{W}_{LIN}) = \frac{1}{N}\|y-\hat{y}\|^2$$

$$= \frac{1}{N}\|(I-P)noise\|^2$$

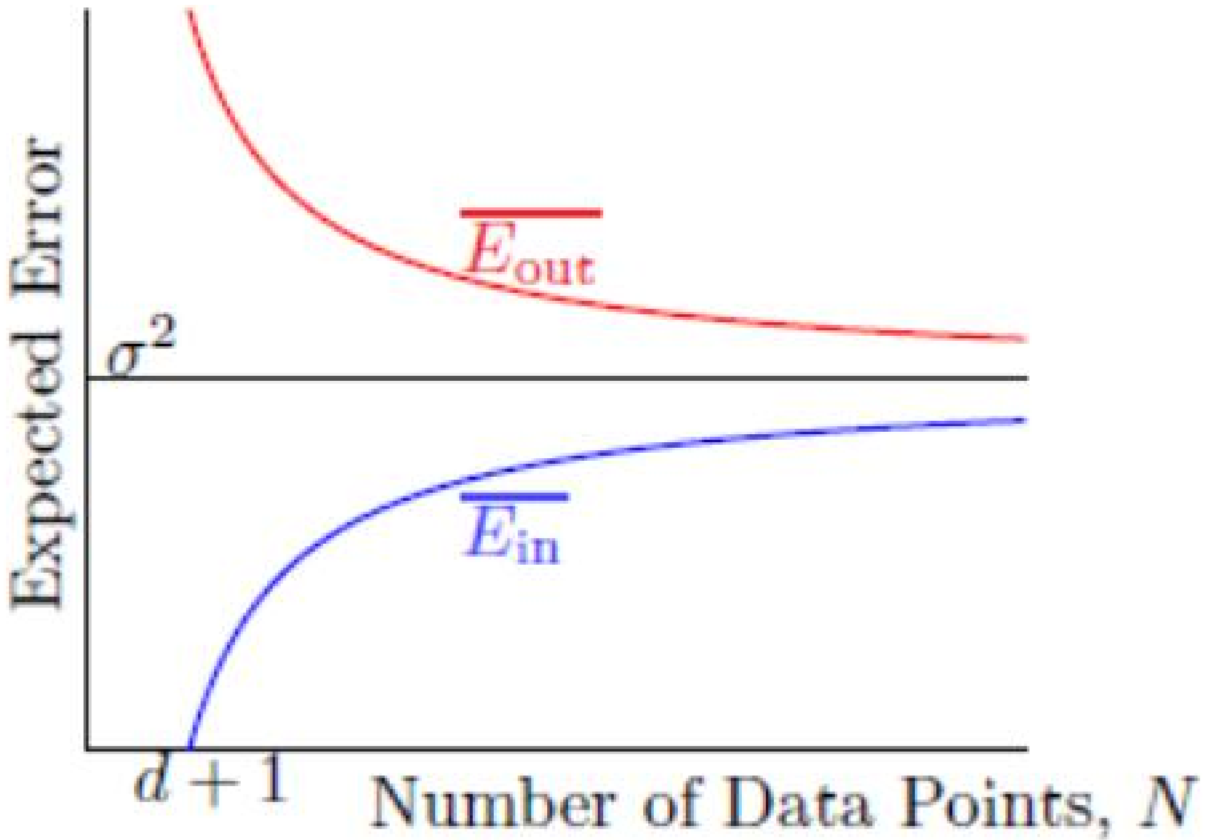
$$= \frac{1}{N}\|(N-(d+1))noise\|^2$$



Optimality

$$\overline{E_{out}} = noise\ level\left(1 + \frac{d+1}{N}\right)$$

$$\overline{E_{in}} = noise\ level\left(1 - \frac{d+1}{N}\right)$$



Overview

- Least squares
  - Problem && Algorithm
  - Optimality
  - Generalized errors
  - Robust least squares
  - Least-squares solutions to large, sparse matrix

Generalized errors

- The error on the data samples have different variance?
  - Assuming that $var(e_i) = \sigma_i^2$
- Weighted least-squares

$$arg\,\min_x \sum_{i=1}^{n} \frac{(y_i - \sum_{j=1}^{m} x_{i,j} w_j)^2}{\sigma_i^2}$$

- Call $V = Var(e)^{-1} = diag(\frac{1}{\sigma_i^2}, ..., \frac{1}{\sigma_n^2})$

$$arg\,\min_x (V(Y - wX))^T (V(Y - wX))$$

Overview

- Least squares
  - Problem && Algorithm
  - Optimality
  - Generalized errors
  - Robust least squares
  - Least-squares solutions to large, sparse matrix

Robust Least Squares

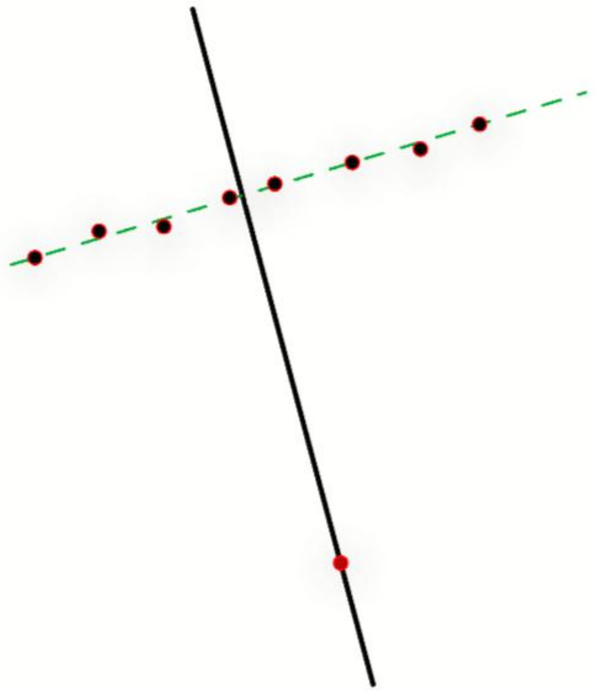
- Outlying point can severely affect least squares estimate

1. Redescending estimators

$$f(x) = log(1 + \frac{1}{2}(\frac{x}{\sigma})^2)$$

$$f'(x) = \frac{2x}{2\sigma^2 + x^2}$$

$$x \to \infty \Rightarrow f'(x) \to 0$$



Robust Least Squares

2. Iteratively reweighted least-squares

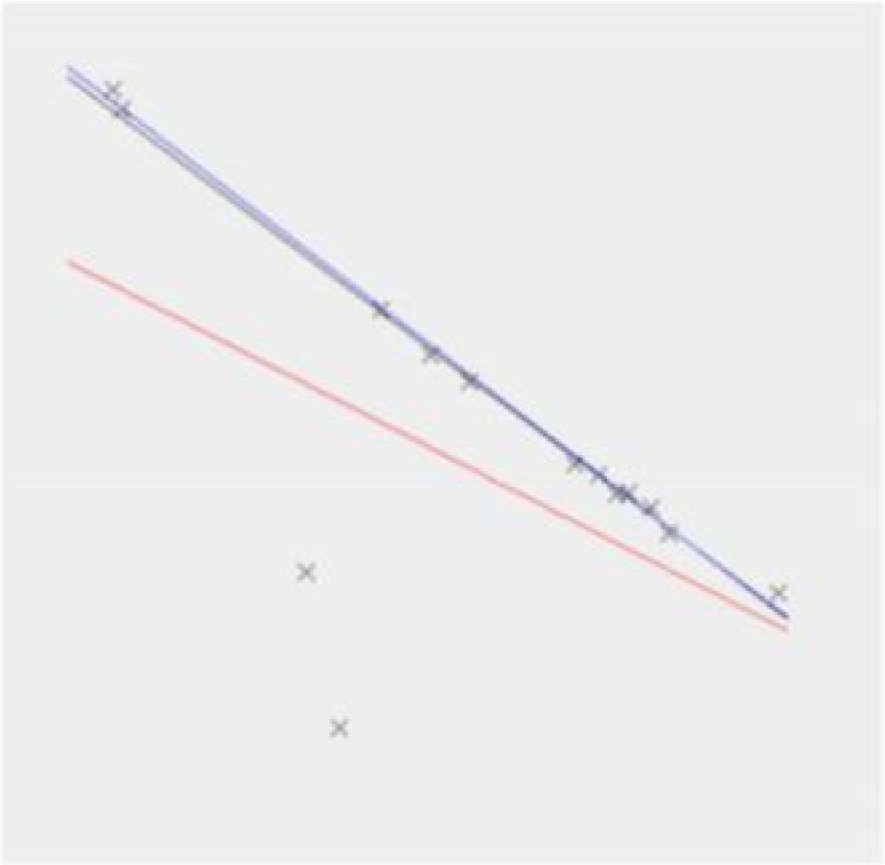
$$V = I$$
$$iterate \quad for \quad i = 1...$$
$$e_i = V(y_i - wx_i)$$
$$V = diag(|e_i|^{p-2}/2)$$
$$w_{k+1} = arg\,min \, \|V(Y - Xw_k)\|$$

- The algorithm converges for $1 < p < 3$

Robust Least Squares

3. Least Median of Squares
   - Randomly select k points
   - Fit the model to these points
   - Evaluate quality of fit on remaining points

$$\min_i \, \mathrm{med} \, e_i^2$$



Overview

- Least squares
  - Problem && Algorithm
  - Optimality
  - Generalized errors
  - Robust least squares
  - Least-squares solutions to large, sparse matrix

Least squares solutions for sparse matrix

- Conjugate Gradient
- LSQR,

Conjugate Gradient

- Equal problem: $min\|Ax-b\|^2 = min \ \ \frac{1}{2}x^TA^TAx - b^TAx$
- A is symmetric, positive-definite and real
- Two non-zero vector x, y are conjugate if $x^TAy = 0$
- Supposed $D = \{d_1, d_2, ..., d_n\}$ which is mutually conjugate respect to A, forms a basis for R^n , each x can be expressed by:

$$x = \sum_{i=1}^{n} a_i d_i$$

Conjugate Gradient

- n iteration to calculate solution

$$\min_{a_1,\dots a_n \in R^n} \frac{1}{2}(\sum_{i=1}^{n} a_i d_i)^T A(\sum_{j=1}^{n} a_j d_j) - b^T(\sum_{i=1}^{n} a_i d_i)$$

$$= \min_{a_1,\dots a_n \in R^n} \sum_{i=1}^{n} (\frac{1}{2} a_i^2 d_i^T A d_i - a_i b^T d_i)$$

$$\min_{a_1,\dots a_n \in R^n} (\frac{1}{2} a_1^2 d_1^T A d_1 - a_1 b^T d_1) + (\frac{1}{2} a_2^2 d_2^T A d_2 - a_2 b^T d_2) + \dots + (\frac{1}{2} a_n^2 d_n^T A d_n - a_n b^T d_n)$$

Conjugate Gradient

- Iterative Method

$$f(x) = \frac{1}{2} x^T A x - x^T b$$

$$f'(x) = Ax - b$$

- Taking $p_0 = b - Ax_0$, enforce p_k to be conjugate to the gradient

$$r_k = b - Ax_k$$

$$d_k = r_k - \sum_{i<k} \frac{d_i^T A r_k}{d_i^T A d_i} d_i$$

Conjugate Gradient

- Following d_k direction, the next optimal location is:

$$x_{k+1} = x_k + \alpha_k d_k$$

$$f'(x_k + \alpha_k d_k) = 0 \Rightarrow \alpha_k = \frac{d_k^T (b - A x_k)}{d_k^T A d_k}$$

Least Square Solutions

- Conjugate Gradient
- LSQR

LSQR

- Let $m \geqslant n$. For each $A \in R_{m\times n}$ there exists a permutation matrix $P \in R_{n\times n}$, an orthogonal matrix $Q \in R_{m\times m}$, and an upper triangular matrix $R \in R_{n\times n}$ such that

$$AP = Q \begin{pmatrix} R \\ 0 \end{pmatrix} \begin{matrix} \} & n \\ \} & m-n \end{matrix} \qquad \text{QR-decomposition.}$$

$$s.t. \quad PP^T = I \quad \|Q^T y\|_2 = \|y\|^2$$

LSQR

- Using properties of Q, let

$$
\begin{aligned}
||Ax - b||_2^2 &= ||Q^T(Ax - b)||_2^2 \\
&= ||Q^T(APP^Tx - b)||_2^2 \\
&= ||(Q^TAP)P^Tx - Q^Tb||_2^2 \\
&= ||\begin{pmatrix} R \\ 0 \end{pmatrix} P^Tx - Q^Tb||_2^2
\end{aligned}
$$

LSQR

- Putting $y = P^T x$ we get

$$\|Ax - b\|_2^2 = \left\| \begin{pmatrix} R \\ 0 \end{pmatrix} y - \begin{pmatrix} c \\ d \end{pmatrix} \right\|_2^2 = \left\| \begin{pmatrix} Ry - c \\ -d \end{pmatrix} \right\|_2^2$$
$$= \|Ry - c\|_2^2 + \|d\|_2^2.$$

$$\min_x \|Ax - b\|_2^2 \Longleftrightarrow \min_y \|Ry - c\|_2^2 + \|d\|_2^2$$

- The solution $x = py = PR^{-1}c$

LSQR

- If A has low column rank. LSQR returns the solution of minimum length

$$\min_{x} \|Ax - b\|^2 + \lambda^2 \|x\|^2$$

- LSQR is recommended for compatible system Ax = b, but it should not be used for symmetric matrix.

Implementation

SciPy.org Sponsored By ENTHOUGHT

Scipy.org | Docs | SciPy v0.14.0 Reference Guide | index | modules | next | previous

Sparse linear algebra (scipy.sparse.linalg)

Abstract linear operators

LinearOperator(shape, matvec[, rmatvec, ...])	Common interface for performing matrix vector products
aslinearoperator(A)	Return A as a LinearOperator.

Matrix Operations

inv(A)	Compute the inverse of a sparse matrix
expm(A)	Compute the matrix exponential using Pade approximation.
expm_multiply(A, B[, start, stop, num, endpoint])	Compute the action of the matrix exponential of A on B.

Matrix norms

onenormest(A[, t, itmax, compute_v, compute_w])	Compute a lower bound of the 1-norm of a sparse matrix.

Solving linear problems

Direct methods for linear equation systems:

spsolve(A, b[, permc_spec, use_umfpack])	Solve the sparse linear system Ax=b, where b may be a vector or a matrix.
factorized(A)	Return a fuction for solving a sparse linear system, with A pre-factorized.

Iterative methods for linear equation systems:

bicg(A, b[, x0, tol, maxiter, xtype, M, ...])	Use BIConjugate Gradient iteration to solve A x = b
bicgstab(A, b[, x0, tol, maxiter, xtype, M, ...])	Use BIConjugate Gradient STABilized iteration to solve A x = b
cg(A, b[, x0, tol, maxiter, xtype, M, callback])	Use Conjugate Gradient iteration to solve A x = b
cgs(A, b[, x0, tol, maxiter, xtype, M, callback])	Use Conjugate Gradient Squared iteration to solve A x = b
gmres(A, b[, x0, tol, restart, maxiter, ...])	Use Generalized Minimal RESidual iteration to solve A x = b.
lgmres(A, b[, x0, tol, maxiter, M, ...])	Solve a matrix equation using the LGMRES algorithm.
minres(A, b[, x0, shift, tol, maxiter, ...])	Use MINimum RESidual iteration to solve Ax=b
qmr(A, b[, x0, tol, maxiter, xtype, M1, M2, ...])	Use Quasi-Minimal Residual iteration to solve A x = b

Iterative methods for least-squares problems:

lsqr(A, b[, damp, atol, btol, conlim, ...])	Find the least-squares solution to a large, sparse, linear system of equations.

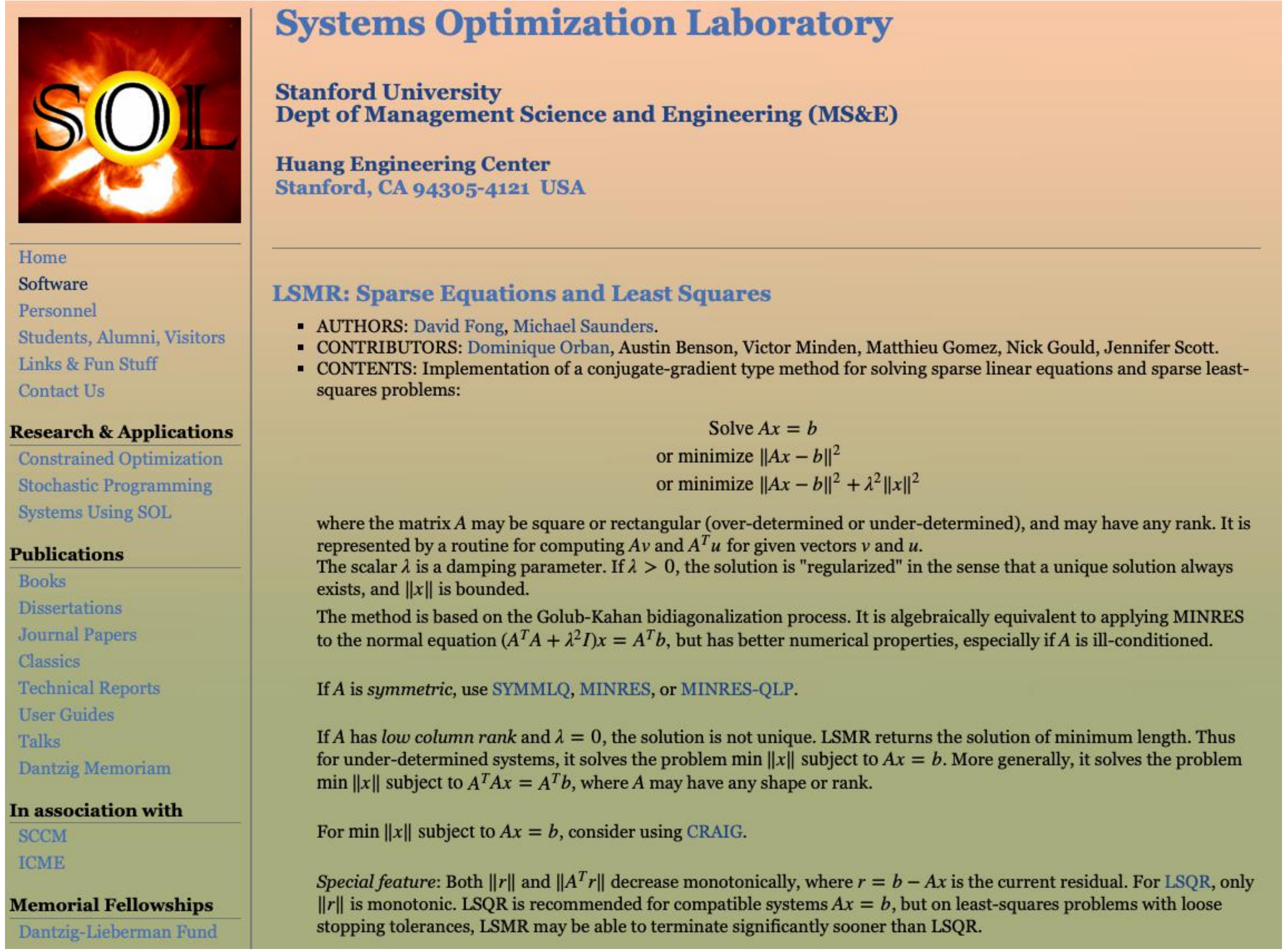
Table Of Contents

Previous topic

scipy.sparse.SparseWarning

Next topic

scipy.sparse.linalg.LinearOperator



Systems Optimization Laboratory

Stanford University
Dept of Management Science and Engineering (MS&E)

Huang Engineering Center
Stanford, CA 94305-4121 USA

Home
Software
Personnel
Students, Alumni, Visitors
Links & Fun Stuff
Contact Us

Research & Applications
Constrained Optimization
Stochastic Programming
Systems Using SOL

Publications
Books
Dissertations
Journal Papers
Classics
Technical Reports
User Guides
Talks
Dantzig Memoriam

In association with
SCCM
ICME

Memorial Fellowships
Dantzig-Lieberman Fund

LSMR: Sparse Equations and Least Squares

- AUTHORS: David Fong, Michael Saunders.
- CONTRIBUTORS: Dominique Orban, Austin Benson, Victor Minden, Matthieu Gomez, Nick Gould, Jennifer Scott.
- CONTENTS: Implementation of a conjugate-gradient type method for solving sparse linear equations and sparse least-squares problems:

$$\text{Solve } Ax = b$$
$$\text{or minimize } \|Ax - b\|^2$$
$$\text{or minimize } \|Ax - b\|^2 + \lambda^2\|x\|^2$$

where the matrix A may be square or rectangular (over-determined or under-determined), and may have any rank. It is represented by a routine for computing Av and $A^T u$ for given vectors v and u.
The scalar λ is a damping parameter. If $\lambda > 0$, the solution is "regularized" in the sense that a unique solution always exists, and $\|x\|$ is bounded.
The method is based on the Golub-Kahan bidiagonalization process. It is algebraically equivalent to applying MINRES to the normal equation $(A^TA + \lambda^2 I)x = A^Tb$, but has better numerical properties, especially if A is ill-conditioned.

If A is *symmetric*, use SYMMLQ, MINRES, or MINRES-QLP.

If A has *low column rank* and $\lambda = 0$, the solution is not unique. LSMR returns the solution of minimum length. Thus for under-determined systems, it solves the problem min $\|x\|$ subject to $Ax = b$. More generally, it solves the problem min $\|x\|$ subject to $A^TAx = A^Tb$, where A may have any shape or rank.

For min $\|x\|$ subject to $Ax = b$, consider using CRAIG.

Special feature: Both $\|r\|$ and $\|A^Tr\|$ decrease monotonically, where $r = b - Ax$ is the current residual. For LSQR, only $\|r\|$ is monotonic. LSQR is recommended for compatible systems $Ax = b$, but on least-squares problems with loose stopping tolerances, LSMR may be able to terminate significantly sooner than LSQR.

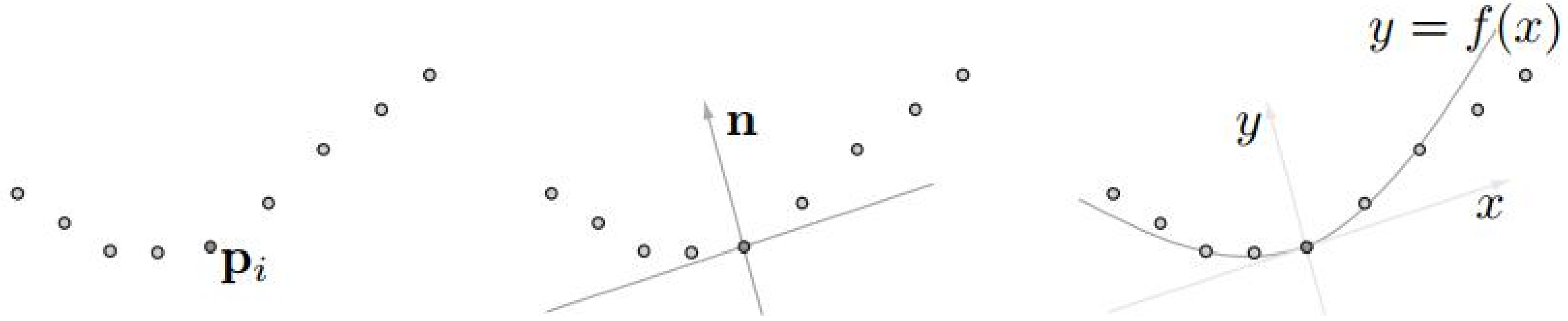
Least Squares Application

- Local surface fitting to 3D points
- Mesh reconstruction
- Skin weights computation from examples

- Local surface fitting to 3D points
- Mesh reconstruction
- Skin weights computation from examples

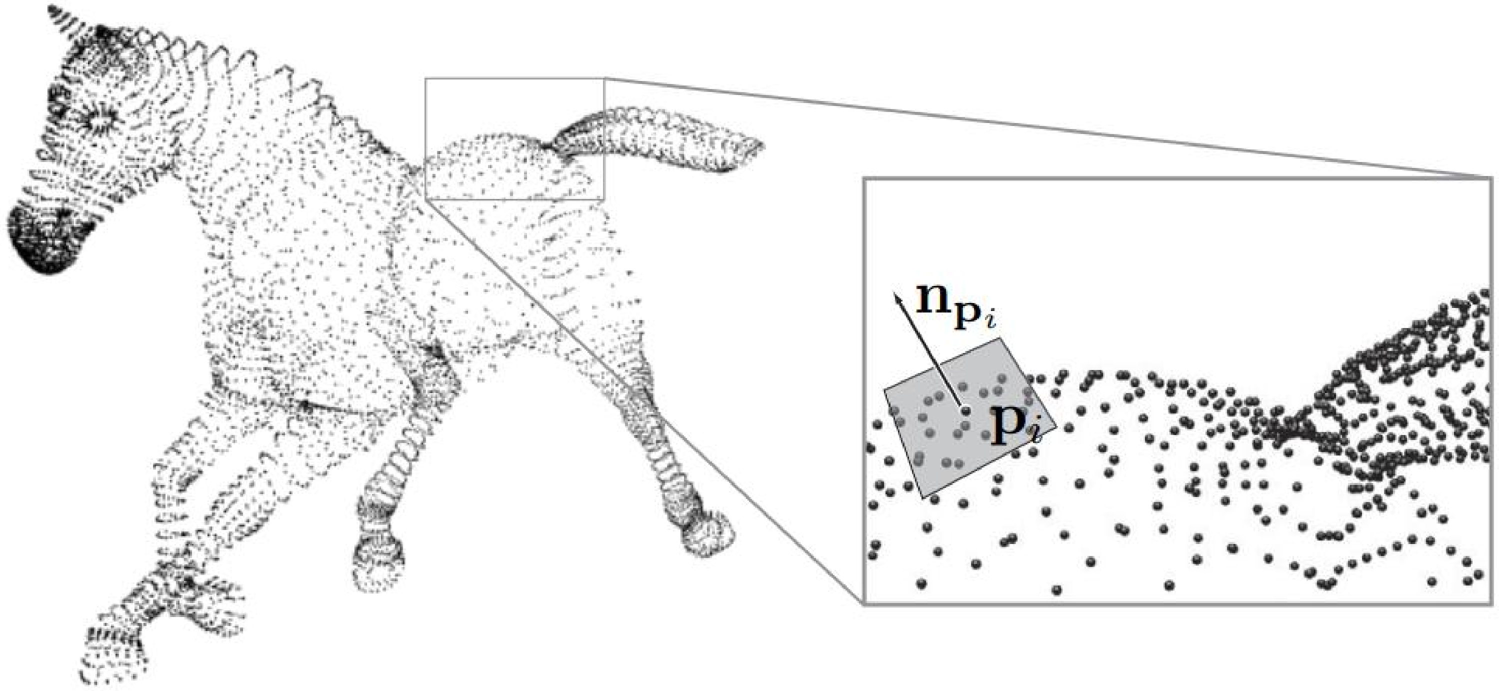
Local surface fitting to 3D points

- LS is useful to fit a polynomial to a set of points coming from a curve.



- It is also important to locally fit a polynomial surface to a set of points in 3D.

Local surface fitting to 3D points



Local surface fitting to 3D points

Local surface fitting to 3D points

- For the problem: $$\min_{\mathbf{n},d} E(\mathbf{n},d) = \sum_{i=1}^{n} (\mathbf{n}^\top \mathbf{p}_i + d)^2, \quad \text{s.t. } \|\mathbf{n}\| = 1$$

- $\partial E(\mathbf{n},d)/\partial d = 0$:
$$\begin{aligned} & \partial E(\mathbf{n},d)/\partial d = 0 \\ \Rightarrow \quad & 2\sum_i (\mathbf{n}^\top \mathbf{p}_i + d) = 0 \\ \Rightarrow \quad & nd = -\mathbf{n}^\top \sum_i \mathbf{p}_i \\ \Rightarrow \quad & d = -\mathbf{n}^\top \bar{\mathbf{p}}\,, \end{aligned}$$

- $$\min_{\mathbf{n}} E(\mathbf{n}) = \sum_{i=1}^{n} (\mathbf{n}^\top \mathbf{p}_i - \mathbf{n}^\top \bar{\mathbf{p}})^2 = \sum_{i=1}^{n} (\mathbf{n}^\top \tilde{\mathbf{p}}_i)^2, \quad \text{s.t. } \|\mathbf{n}\| = 1$$
$$\min_{\mathbf{n}} \Big(\sum_i (\mathbf{n}^\top \tilde{\mathbf{p}}_i)^2 + \lambda (1 - \mathbf{n}^\top \mathbf{n}) \Big) = \min_{\mathbf{n}} (\mathbf{n}^\top C \mathbf{n} + \lambda (1 - \mathbf{n}^\top \mathbf{n})) \qquad C = \textstyle\sum_i \tilde{\mathbf{p}}_i \tilde{\mathbf{p}}_i^\top$$

Local surface fitting to 3D points

$$\min_{\mathbf{n}}\left(\sum_i (\mathbf{n}^\top \tilde{\mathbf{p}}_i)^2 + \lambda(1-\mathbf{n}^\top\mathbf{n})\right) = \min_{\mathbf{n}}(\mathbf{n}^\top C\mathbf{n} + \lambda(1-\mathbf{n}^\top\mathbf{n})) \qquad C = \sum_i \tilde{\mathbf{p}}_i \tilde{\mathbf{p}}_i^\top$$

$$C\mathbf{n} = \lambda\mathbf{n}$$

Local surface fitting to 3D points

- Local surface fitting to 3D points
- Mesh reconstruction
- Skin weights computation from examples

Mesh reconstruction

- Mesh reconstruction: construction of a mesh from a set of samples.

- Given a planar graph with arbitrary connectivity and a sparse set of control points with geometry, reconstruct the geometry of the rest of the mesh vertices.

Mesh reconstruction

$$\mathbf{v}_i - \frac{1}{d_i} \sum_{j:(i,j)\in E} \mathbf{v}_j = 0$$

$$L_{ij} = \begin{cases} 1 & i = j \\ -\frac{1}{d_i} & (i,j) \in E \\ 0 & \text{otherwise.} \end{cases}$$

$$L\mathbf{x} = 0, \;\; L\mathbf{y} = 0, \;\; L\mathbf{z} = 0$$

Mesh reconstruction

- In order to keep sharp features of the mesh, consider we have some control points:

$$\mathbf{v}_s = (x_s, y_s, z_s), \quad s \in C, \qquad C = \{s_1, s_2, ..., s_m\}$$

- The system becomes:

$$A\mathbf{x} = \mathbf{b}$$

$$A = \begin{pmatrix} L \\ F \end{pmatrix}, \quad F_{ij} = \begin{cases} 1 & j = s_i \in C \\ 0 & otherwise \end{cases}$$

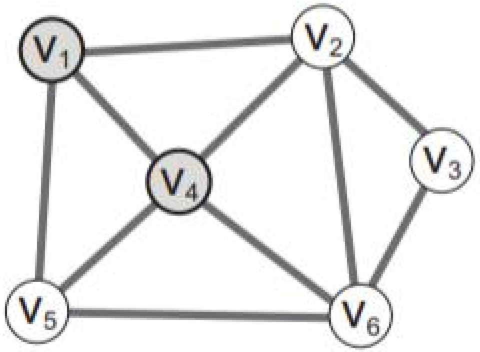
$$b_k = \begin{cases} 0 & k \leq n \\ x_{s_{k-n}} & n < k \leq n+m \end{cases}$$

Mesh reconstruction

- To reconstruct the mesh, we find x that minimizes:

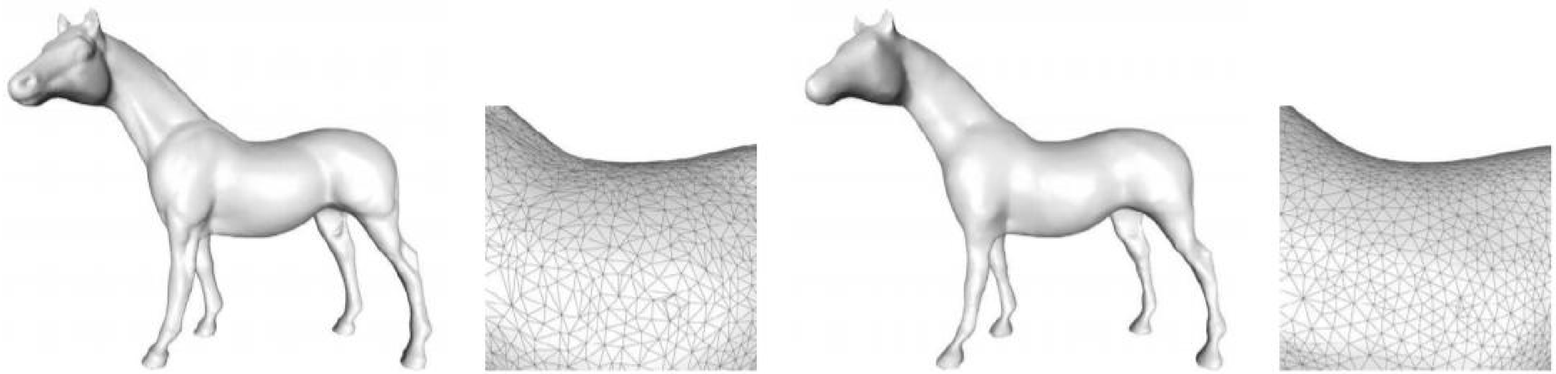
$$\|A\mathbf{x}-\mathbf{b}\|^2 = \|L\mathbf{x}\|^2 + \sum_{s\in C} |x_s - \mathbf{v}_s^{(x)}|^2.$$

in least-squares sense.

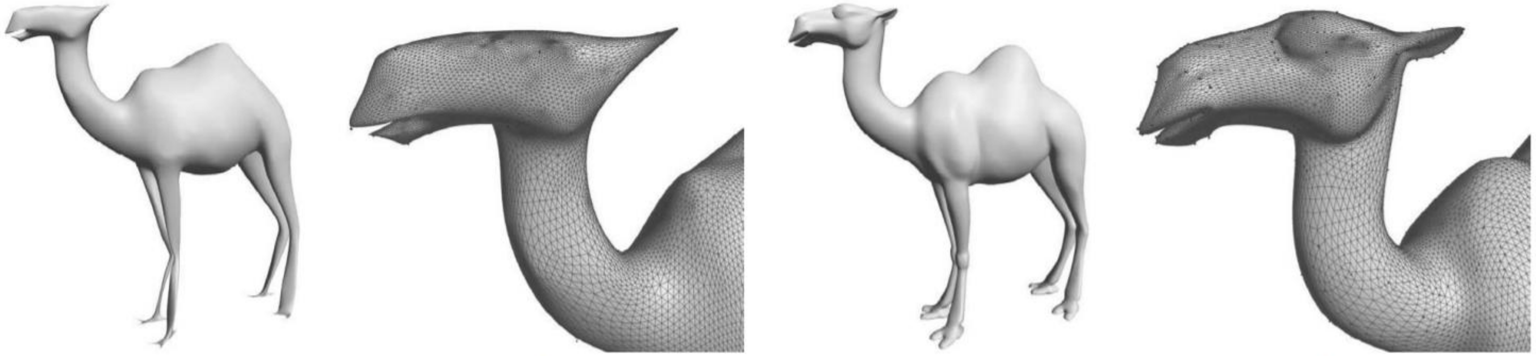


$$\begin{bmatrix} 1 & -\frac{1}{3} & 0 & -\frac{1}{3} & -\frac{1}{3} & 0 \\ -\frac{1}{4} & 1 & -\frac{1}{4} & -\frac{1}{4} & 0 & -\frac{1}{4} \\ 0 & -\frac{1}{2} & 1 & 0 & 0 & -\frac{1}{2} \\ -\frac{1}{4} & -\frac{1}{4} & 0 & 1 & -\frac{1}{4} & -\frac{1}{4} \\ -\frac{1}{3} & 0 & 0 & -\frac{1}{3} & 1 & -\frac{1}{3} \\ 0 & -\frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Mesh reconstruction



(a) (b) (c) (d)



(a) (b) (c) (d)

O. Sorkine and D. Cohen-Or.
Least-squares meshes.

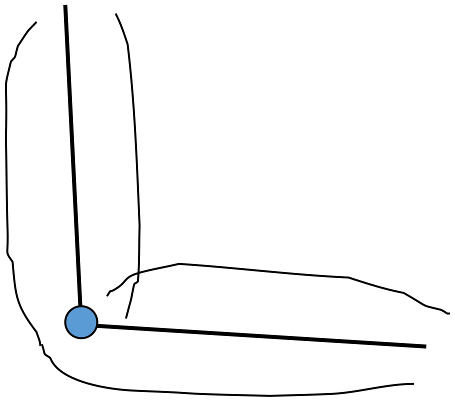
- Local surface fitting to 3D points
- Mesh reconstruction
- Skin weights computation from examples

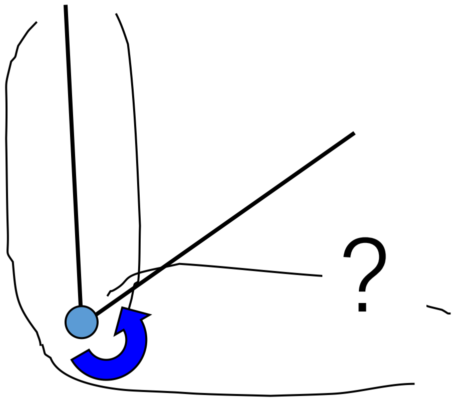
Skin weights computation from examples

- Animating articulated characters such as virtual humans is a fundamental operation in computer graphics and interactive applications.

- Techniques for rigging character skins by weighting vertices to an associated skeleton are widely used in video games and the computer animation industry.

Skin weights computation from examples





D. L. James and C. D. Twigg.
Skinning mesh animations

Skin weights computation from examples

$$\mathsf{p}^t \approx \mathsf{T}^t \tilde{\mathsf{p}}, \quad t = 1 \ldots S$$

$$\mathsf{T}_i^t = \sum_{b \in \mathscr{B}_i} w_{ib} \bar{\mathsf{T}}_b^t$$

Skin weights computation from examples

$$\gamma_{bi} = \sum_{t=1...S} \|\mathsf{p}_i^t - \bar{\mathsf{T}}_b^t \tilde{\mathsf{p}}_i\|_2^2, \quad b = 1 \dots B.$$

Skin weights computation from examples

- Estimating vertex weights:
- Given vertex-bone influence sets, $\{\mathscr{B}_i\}$ the associated weights are computed using a least squares approach.
- Weights are constrained by the mesh sequence approximation equations:

$$\sum_{b \in \mathscr{B}_i} (\bar{\mathsf{T}}^t_b \tilde{\mathsf{p}}_i) w_{ib} = \mathsf{p}^t_i, \qquad t = 1 \ldots S, \qquad \sum_b w_{ib} = 1$$

- It is of the form: $\mathbf{A}^{(i)} \mathbf{w}^{(i)} = \mathbf{b}^{(i)}, \quad i = 1 \ldots N$

Skin weights computation from examples

- We consider the augment system:

$$\begin{bmatrix} c\mathbf{A}^{(i)} \\ 1\ldots1 \end{bmatrix} \mathbf{w}^{(i)} = \begin{pmatrix} c\mathbf{b}^{(i)} \\ 1 \end{pmatrix} \quad \Leftrightarrow \quad \tilde{\mathbf{A}}^{(i)}\mathbf{w}^{(i)} = \tilde{\mathbf{b}}^{(i)}$$

- Then solve the over-constrained system by least squares.

Skin weights computation from examples

- Over-fitting:
- The solution can result in weights with large positive and negative values.
- TSVD, NNLS

TSVD(Truncated SVD)

- TSVD(Truncated SVD)
- Used to handle ill-conditioning: $\kappa(A) = \frac{\sigma_{max}}{\sigma_{min}}$
- How singular value affect the problem solution.

TSVD(Truncated SVD)

- For system:

$$\boldsymbol{Ax} = \boldsymbol{b},$$

$$\boldsymbol{A} = \boldsymbol{UDV}^T$$

$$\boldsymbol{UDV}^T = \left[\begin{array}{c|c|c} \boldsymbol{u}_1 & \dots & \boldsymbol{u}_n \end{array}\right] \begin{bmatrix} \sigma_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \sigma_n \end{bmatrix} \left[\begin{array}{c|c|c} \boldsymbol{v}_1 & \dots & \boldsymbol{v}_n \end{array}\right]^T$$

$$\boldsymbol{A} = \sum_{i=1}^{n} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^T$$

TSVD(Truncated SVD)

Conclusion

- Local surface fitting to 3D points
- Mesh reconstruction
- Skin weights computation from examples

Resources

- Course notes on least squares
  - Fred Pighin
  - http://graphics.stanford.edu/~jplewis/lscourse/
- Learn from data
  - Yaser S. Abu-Mostafa
- Leature notes on least squares
  - Dmitriy Leykekhman
  - http://www.math.uconn.edu/~leykekhman/courses/MATH3795/Lectures/Lecture_8_Linear_least_squares_orthogonal_matrices.pdf

Resources

- Sorkine O, Cohen-Or D. Least-squares meshes[C]//Shape Modeling Applications, 2004. Proceedings. IEEE, 2004: 191-199.
- James D L, Twigg C D. Skinning mesh animations[C]//ACM Transactions on Graphics (TOG). ACM, 2005, 24(3): 399-407.

Thanks!