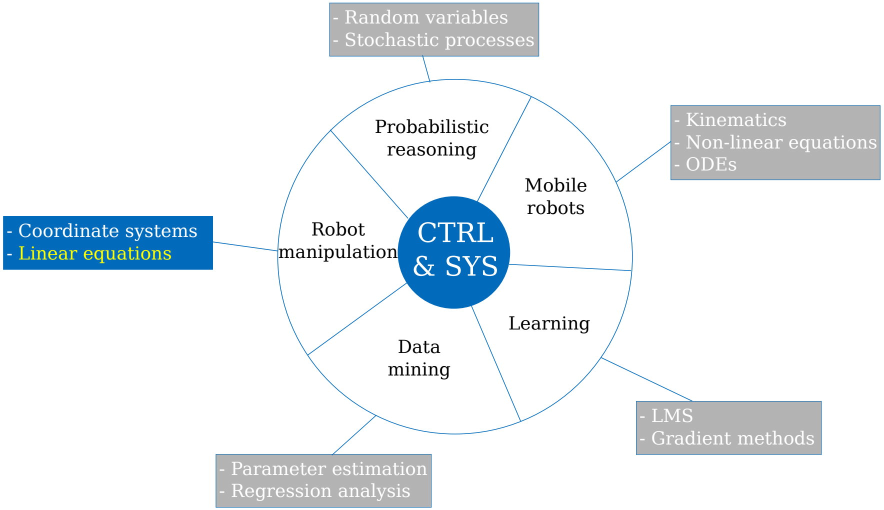
Singular Value Decomposition (SVD)



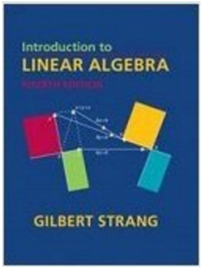
Today's Topics

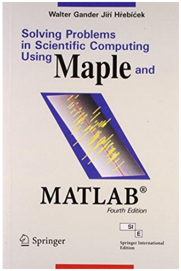
$\rightarrow$ Matrix Decompositions

LU Decomposition

Singular Value Decomposition (SVD)

Polar Decomposition




Walter Gander Jiří Hřebíček
Solving Problems in Scientific Computing Using Maple and MATLAB®
Fourth Edition
Springer
Springer International Edition

Matrix Decomposition

Definition

A matrix decomposition is the transformation of a given matrix into some desired canonical form

Objectives:

- computational convenience
- analytical simplicity

Motivation:
It is in general difficult to perform matrix computations, such as matrix inversions, matrix determinants, solving linear systems or least square fits, in an optimal explicit way

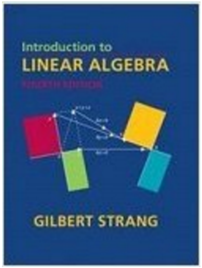
Some of these problems may be converted into several easier tasks such as e.g. solving triangular or diagonal systems

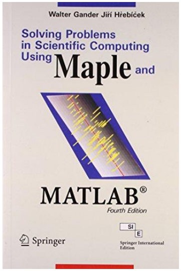
Matrix Decompositions

→ LU Decomposition

Singular Value Decomposition (SVD)

Polar Decomposition




Walter Gander Jiří Hřebíček
Solving Problems in Scientific Computing Using Maple and MATLAB®
Fourth Edition
Springer
SI E
Springer International Edition

LU Factorization

Assume a decomposition $A = LU$ is given, where U is an **U**pper diagonal and L a **L**ower diagonal matrix

$$A = \begin{pmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{pmatrix} = LU$$

The solution of $A\mathbf{x} = \mathbf{b}$ is then an easy two-step procedure:

1. Solve $L\mathbf{y} = \mathbf{b}$ for an unknown $\mathbf{y}$
   Get $\mathbf{y}$ by the easy forward elimination $L\mathbf{y} = \mathbf{b}$
2. Use $\mathbf{y}$ and solve $U\mathbf{x} = \mathbf{y}$ for the unknown $\mathbf{x}$
   Get $\mathbf{x}$ by the easy backward elimination $U\mathbf{x} = \mathbf{y}$

Calculate LU **once**, then repeat the solution by changing $\mathbf{b}$, but keeping L and U!

LU Factorization

Example: matrix inversion

Repeatedly solve:

$$A \begin{pmatrix} | & | & | & | \\ \mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_3 & \mathbf{x}_4 \\ | & | & | & | \end{pmatrix} = \overbrace{\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}}^{\mathbf{e}_1}, \overbrace{\begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}}^{\mathbf{e}_2}, \overbrace{\begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}}^{\mathbf{e}_3}, \overbrace{\begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}}^{\mathbf{e}_4}$$

Idea

1. decompose $A = LU$ once
2. solve $L\mathbf{y}_i = \mathbf{e}_i$
3. get $\mathbf{x}_i$ by solving $U\mathbf{x}_i = \mathbf{y}_i$
4. finally

$$\begin{pmatrix} | & | & | & | \\ \mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_3 & \mathbf{x}_4 \\ | & | & | & | \end{pmatrix} = A^{-1}$$

How to calculate L and U?

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = \begin{pmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{pmatrix}$$

$$LU = \begin{pmatrix} l_{11}u_{11} & l_{11}u_{12} & l_{11}u_{13} \\ l_{21}u_{11} & l_{21}u_{12} + l_{22}u_{22} & l_{21}u_{13} + l_{22}u_{23} \\ l_{31}u_{11} & l_{31}u_{12} + l_{32}u_{22} & l_{31}u_{13} + l_{32}u_{23} + l_{33}u_{33} \end{pmatrix}$$

Six ls, six us, nine equations

too many unknowns

set $l_{11} = l_{22} = l_{33} = 1$

Solution: Stepwise

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = \begin{pmatrix} u_{11} & u_{12} & u_{13} \\ l_{21}u_{11} & l_{21}u_{12} + u_{22} & l_{21}u_{13} + u_{23} \\ l_{31}u_{11} & l_{31}u_{12} + l_{32}u_{22} & l_{31}u_{13} + l_{32}u_{23} + u_{33} \end{pmatrix}$$

$$\Longrightarrow \mathbf{u_{11}} = a_{11}, \quad \mathbf{u_{12}} = a_{12}, \quad \mathbf{u_{13}} = a_{13}$$

$$\Longrightarrow \mathbf{l_{21}} = \frac{a_{21}}{u_{11}} = \frac{a_{21}}{a_{11}}, \quad \mathbf{l_{31}} = \frac{a_{31}}{a_{11}}$$

$$\Longrightarrow \mathbf{u_{22}} = a_{22} - l_{21}u_{12} = a_{22} - \frac{a_{21}}{a_{11}}a_{12}$$

$$\mathbf{u_{23}} = a_{23} - l_{21}u_{13} = a_{23} - \frac{a_{21}}{a_{11}}a_{13}$$

$$\Longrightarrow \mathbf{l_{32}} = \frac{a_{32} - \frac{a_{31}}{a_{11}}a_{12}}{a_{22} - \frac{a_{21}}{a_{11}}a_{12}}$$

$$\Longrightarrow \mathbf{u_{33}} = a_{33} - \frac{a_{31}}{a_{11}}a_{13} - \left(\frac{a_{32} - \frac{a_{31}}{a_{11}}a_{12}}{a_{22} - \frac{a_{21}}{a_{11}}a_{12}}\right)\left(a_{23} - \frac{a_{21}}{a_{11}}a_{13}\right)$$

LU Decomposition - Example (1/2)

U is the result of Gaussian elimination; L are the pivot elements!

```
import numpy as np
A = np.array([[2., 3., -5.], [4., 8., -3.], [-6., 1., 4.]])
B = np.array([A[0,:], A[1,:] - 2*A[0,:], A[2,:] -(-3)*A[0,:]])
U = np.array([B[0,:], B[1,:], B[2,:] - 5*B[1,:]])
L = np.array([[1., 0., 0.], [2., 1., 0.], [-3., 5., 1.]])
np.allclose(A, L.dot(U))
```

Program output

$$A = \begin{pmatrix} 2 & 3 & -5 \\ 4 & 8 & -3 \\ -6 & 1 & 4 \end{pmatrix} \quad B = \begin{pmatrix} 2 & 3 & -5 \\ 0 & 2 & 7 \\ 0 & 10 & -11 \end{pmatrix} \quad U = \begin{pmatrix} 2 & 3 & -5 \\ 0 & 2 & 7 \\ 0 & 0 & -46 \end{pmatrix} \quad L = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -3 & 5 & 1 \end{pmatrix}$$

$$\text{Joint storage: } LU = \begin{pmatrix} 2 & 3 & -5 \\ \underline{2} & 2 & 7 \\ -\underline{3} & \underline{5} & -46 \end{pmatrix}$$

LU Decomposition - Example (2/2)

Finding the LU decomposition using Python

```
import numpy as np
from scipy.linalg import lu
A = np.array([[2, 3, -5], [4, 8, -3], [-6, 1, 4]])
p, l, u = lu(A)
```

Wrap-up LU Decomposition

Any matrix A can be decomposed into an **upper-triangular** matrix U and a **lower-triangular** matrix L

Gaussian elimination is equivalent to building an LU decomposition

After finishing the elimination, U is given by the coefficients on and above the diagonal of the final matrix, while L is generated from an identity matrix which is successively filled up with the pivot elements used during the elimination phase

Both L and U can be stored in place of the original matrix; NO additional memory is needed

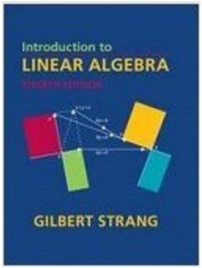
Both matrices express **a stage of Gaussian elimination**: forward elimination by U and backward elimination by L

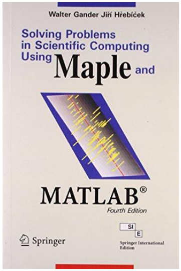
Matrix Decompositions

LU Decomposition

→ Singular Value Decomposition (SVD)

Polar Decomposition




Walter Gander Jiří Hřebíček
Solving Problems in Scientific Computing Using Maple and MATLAB®
Fourth Edition
Springer

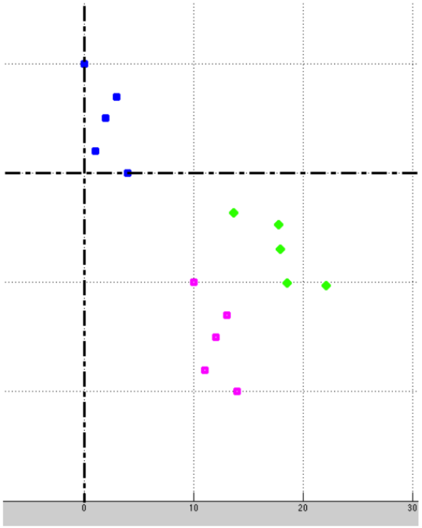
SVD Motivation: Find Q and $\mathbf{t}$

Given:

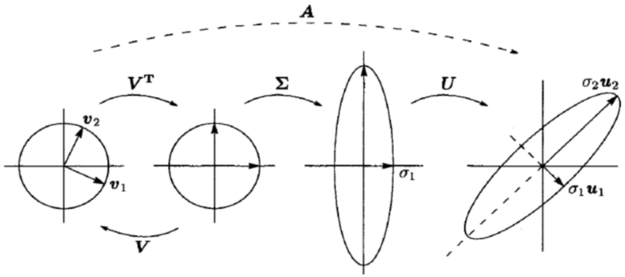
- A map of fixed known nominal landmarks (blue) $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \mathbb{R}^n$
- Measured observations of these landmarks in a robot coordinate system (green) $\xi_1, \ldots, \xi_m$

Find:

- Q (rotation) and $\mathbf{t}$ (translation) such that $\xi_i = Q\mathbf{x}_i + \mathbf{t}$



Singular Value Decomposition - Informal



Decompose **any** matrix A into a sequence of

- an orthogonal V
- an anisotropic scaling Σ and
- another orthogonal U, such that

$$A = U \Sigma V^T$$

Why is this ***SO*** good?

Since V and U are orthogonal, we know: $V^T V = I$ and $U^T U = I$, so $V^{-1} = V^T$ and $U^{-1} = U^T$

In addition

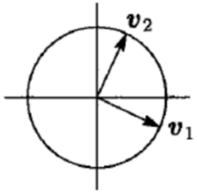
$$\begin{pmatrix} \sigma_1 & 0 & \dots & 0 \\ 0 & \ddots & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \dots & 0 & \sigma_n \end{pmatrix}^{-1} = \begin{pmatrix} \frac{1}{\sigma_1} & 0 & \dots & 0 \\ 0 & \ddots & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \dots & 0 & \frac{1}{\sigma_n} \end{pmatrix}$$

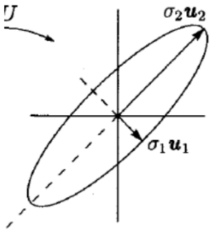
$$\implies A^{-1} = V \Sigma^{-1} U^T$$

Orthogonal maps do not change condition numbers $\implies$ numerically nice

It even works for **singular** and **non-square** A!

How to Calculate V, U, and Σ?





Wanted: $\mathbf{v}_1$ and $\mathbf{v}_2$ such that: $\mathbf{v}_1 \perp \mathbf{v}_2$ and $A\mathbf{v}_1 \perp A\mathbf{v}_2$
Define unit vectors:

$$\mathbf{u}_1 = \frac{A\mathbf{v}_1}{\|A\mathbf{v}_1\|} \text{ and } \mathbf{u}_2 = \frac{A\mathbf{v}_2}{\|A\mathbf{v}_2\|}$$

So, we want $A\mathbf{v}_1 = \sigma_1\mathbf{u}_1$ and $A\mathbf{v}_2 = \sigma_2\mathbf{u}_2$

$$A\begin{pmatrix}\mathbf{v}_1 & \mathbf{v}_2\end{pmatrix} = \begin{pmatrix}\sigma_1\mathbf{u}_1 & \sigma_2\mathbf{u}_2\end{pmatrix} = \begin{pmatrix}\mathbf{u}_1 & \mathbf{u}_2\end{pmatrix}\begin{pmatrix}\sigma_1 & \\ & \sigma_2\end{pmatrix}$$

How to Get V if U is Unknown?

Observe: assuming that $A = U\Sigma V^T$ is already known, then

$$\begin{aligned}
A^T A &= \left(U\Sigma V^T\right)^T \left(U\Sigma V^T\right) \\
&= V\Sigma^T U^T U\Sigma V^T \\
&= V\Sigma^T \Sigma V^T \\
&= V \begin{pmatrix} \sigma_1^2 & \\ & \sigma_2^2 \end{pmatrix} V^T \\
&\implies A^T A V = V \begin{pmatrix} \sigma_1^2 & \\ & \sigma_2^2 \end{pmatrix} \\
&\implies A^T A \begin{pmatrix} \mathbf{v}_1 & \mathbf{v}_2 \end{pmatrix} = \begin{pmatrix} \sigma_1^2 \mathbf{v}_1 & \sigma_2^2 \mathbf{v}_2 \end{pmatrix}
\end{aligned}$$

How to Get V if U is Unknown?

Observe: assuming that $A = U\Sigma V^T$ is already known, then

$$\begin{aligned}
A^T A &= \left(U\Sigma V^T\right)^T \left(U\Sigma V^T\right) \\
&= V\Sigma^T U^T U \Sigma V^T \\
&= V\Sigma^T \Sigma V^T \\
&= V \begin{pmatrix} \sigma_1^2 & \\ & \sigma_2^2 \end{pmatrix} V^T \\
\implies A^T A V &= V \begin{pmatrix} \sigma_1^2 & \\ & \sigma_2^2 \end{pmatrix} \\
\implies A^T A \begin{pmatrix} \mathbf{v}_1 & \mathbf{v}_2 \end{pmatrix} &= \begin{pmatrix} \sigma_1^2 \mathbf{v}_1 & \sigma_2^2 \mathbf{v}_2 \end{pmatrix}
\end{aligned}$$

Observe: Eigenvectors of symmetric matrices are $\perp$ to one another (and $A^T A$ is of course symmetric - a theorem from the outside world ;))

So, the eigenvectors $\mathbf{v}_1, \mathbf{v}_2$ of $A^T A$ will do the job!

$\implies$ **Find the eigenvectors $\mathbf{v_1}, \mathbf{v_2}$!**

SVD - Example 1 (1/3)

$$A = \begin{pmatrix} 2 & 2 \\ -1 & 1 \end{pmatrix}$$

$$A^T A = \begin{pmatrix} 2 & -1 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} 2 & 2 \\ -1 & 1 \end{pmatrix} = \begin{pmatrix} 5 & 3 \\ 3 & 5 \end{pmatrix}$$

Eigenvalues

$$(5-\lambda)^2 - 9 = 0$$

$$\iff 5-\lambda = \pm 3$$

$$\iff \lambda_{1/2} = 8 \text{ or } 2$$

SVD - Example 1 (2/3)

$\lambda_1 = 8$ (take larger first)

$$\begin{pmatrix} 5-8 & 3 \\ 3 & 5-8 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \implies \left.\begin{array}{r} -3v_1 + 3v_2 = 0 \\ 3v_1 - 3v_2 = 0 \end{array}\right\} \implies v_1 = v_2$$

$$\text{Eigenvector } = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \text{ Unit eigenvector} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}$$

$\lambda_2 = 2$ (then by heart)

$$v_1 = -v_2$$

$$\text{Eigenvector } = \begin{pmatrix} -1 \\ 1 \end{pmatrix} \text{ Unit eigenvector} = \begin{pmatrix} -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}$$

$$\implies V = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}$$

SVD - Example 1 (3/3)

$$A\mathbf{v}_1 = \begin{pmatrix} 2 & 2 \\ -1 & 1 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} = \begin{pmatrix} 2\sqrt{2} \\ 0 \end{pmatrix} \text{ unit vector } \mathbf{u}_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$A\mathbf{v}_2 = \begin{pmatrix} 2 & 2 \\ -1 & 1 \end{pmatrix} \begin{pmatrix} -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} = \begin{pmatrix} 0 \\ \sqrt{2} \end{pmatrix} \text{ unit vector } \mathbf{u}_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

$$\sigma_1 = 2\sqrt{2}\ (\sigma_1^2 = 8 \implies \text{OK})$$

$$\sigma_2 = \sqrt{2}\ (\sigma_2^2 = 2 \implies \text{OK})$$

$$A = U\Sigma V^T$$

$$\begin{pmatrix} 2 & 2 \\ -1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 2\sqrt{2} & 0 \\ 0 & \sqrt{2} \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}$$

SVD - Example 2 (1/3)

$$A = \begin{pmatrix} 2 & 2 \\ 1 & 1 \end{pmatrix}$$

$$A^T A = \begin{pmatrix} 2 & 1 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} 2 & 2 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 5 & 5 \\ 5 & 5 \end{pmatrix}$$

Eigenvalues

$$(5-\lambda)^2 - 25 = 0$$
$$\iff 5 - \lambda = \pm 5$$
$$\iff \lambda_{1/2} = 10 \text{ or } 0$$

$\lambda_1 = 10$

$$\begin{pmatrix} -5 & 5 \\ 5 & -5 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \implies \left. \begin{array}{l} -5v_1 + 5v_2 = 0 \\ 5v_1 - 5v_2 = 0 \end{array} \right\} \implies v_1 = v_2$$

$$\text{Eigenvector } = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \text{ Unit eigenvector} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}$$

SVD - Example 2 (2/3)

A only hits the space along $\begin{pmatrix} 2 \\ 1 \end{pmatrix}$

$$\implies \mathbf{u}_1 = \text{ unit multiple of } \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$

$$\implies \mathbf{u}_1 = \frac{1}{\sqrt{5}} \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$

$$\implies A\mathbf{v}_1 = \sigma_1 \mathbf{u}_1$$

$$\implies \sigma_1 = \sqrt{10}$$

$$\begin{pmatrix} 2 & 2 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} \frac{2}{\sqrt{5}} \\ \frac{1}{\sqrt{5}} \end{pmatrix} \begin{pmatrix} \sqrt{10} \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}$$

Yet, U and V should be square matrices!

SVD - Example 2 (3/3)

So, extend via choosing the second $\mathbf{v}_2 \perp \mathbf{v}_1$ and the second $\mathbf{u}_2 \perp \mathbf{u}_1$

$$\mathbf{v}_2 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$
$$\mathbf{u}_2 = \frac{1}{\sqrt{5}} \begin{pmatrix} 1 \\ -2 \end{pmatrix}$$

Now complete the full SVD:

$$\begin{pmatrix} 2 & 2 \\ 1 & 1 \end{pmatrix} = \frac{1}{\sqrt{5}} \begin{pmatrix} 2 & 1 \\ 1 & -2 \end{pmatrix} \begin{pmatrix} \sqrt{10} & 0 \\ 0 & 0 \end{pmatrix} \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

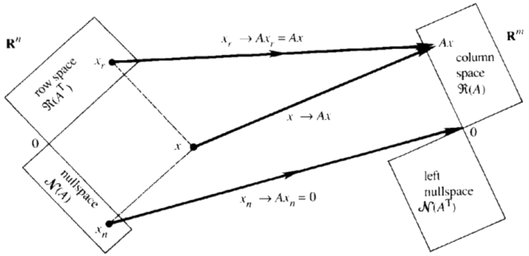
Observe: $\mathbf{v}_2$ and $\mathbf{u}_2$ are only specified as being $\perp$ to $\mathbf{v}_1$ and $\mathbf{u}_1$; they are thus arbitrary to a large degree!

SVD and the Four Matrix Spaces

U and V contain orthonormal basis vectors (i.e. all lengths are one, all vectors are perpendicular)

These vectors span different subspaces:

- first r columns of V: row space of A
- last $n - r$ columns of V: kernel of A
- first r columns of U: column space of A
- last $m - r$ columns of U: kernel of A^T



Remarks

Like with pivot elements, only singular values not equal to zero are counted as such

If we begin with AA^T instead of A^TA we find the $\mathbf{u}_i$ vectors first (the eigenvalues are kept)

SVD Calculation Wrap Up

Find the eigenvalues and eigenvectors of $A^T A$ (or AA^T)

The eigenvectors of $A^T A$ make up the columns of V

The eigenvectors of AA^T make up the columns of U

The singular values in Σ are the square roots of the eigenvalues of $A^T A$

The singular values are the diagonal entries of Σ and are arranged in descending order

SVD - Example 3

```
A = numpy.array([[1, -1, 1], [1, 0, 0], [1, 1,
      1], [1, 2, 4]])
b = numpy.array([[2], [1], [2], [3]])
U, s, VT = numpy.linalg.svd(A)
S = numpy.zeros((U.shape[1], VT.shape[0]))
S[:VT.shape[0],:VT.shape[0]] = numpy.diag(s)
```

$$U = \begin{pmatrix} -0.1517 & 0.8959 & -0.3525 & -0.2236 \\ -0.0612 & 0.4013 & 0.6207 & 0.6708 \\ -0.3215 & 0.0406 & 0.6671 & -0.6708 \\ -0.9327 & -0.1861 & -0.2134 & 0.2236 \end{pmatrix}$$

$$S = \begin{pmatrix} 4.8956 & 0 & 0 \\ 0 & 1.6942 & 0 \\ 0 & 0 & 1.0784 \\ 0 & 0 & 0 \end{pmatrix}$$

$$V = \begin{pmatrix} -0.2997 & 0.6798 & 0.6693 \\ -0.4157 & -0.7245 & 0.5498 \\ -0.8587 & 0.1135 & -0.4997 \end{pmatrix}$$

```
y = numpy.linalg.lstsq(S, U.T.dot(b))
```

$$\mathbf{y} = (-0.7774 \quad 1.0130 \quad 0.5653)^T$$

```
x = VT.T.dot(y[0])
```

$$\mathbf{y} = (1.3000 \quad -0.1000 \quad 0.5000)^T$$

It may numerically pay off to set small σ_i to zero

U and V are orthogonal bases of the null space and range of A, namely those where $\sigma_i \neq 0$

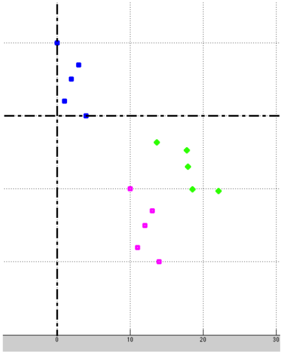
An SVD decomposition is always possible, regardless of whether A is nonsingular

The SVD is unique up to permutations and linear combinations of the column vectors where $\sigma_i = \sigma_j$

Find Unitary Q and Translation $\mathbf{t}$

Given:

- A map of fixed known nominal landmarks (blue) $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \mathbb{R}^n$
- Measured observations of these landmarks in a robot coordinate system (green) $\xi_1, \ldots, \xi_m$



Find: Q and $\mathbf{t}$ such that $\xi_i = Q\mathbf{x}_i + \mathbf{t}$

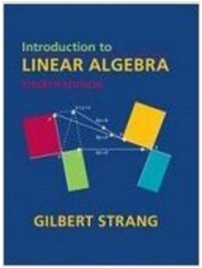
1. $\bar{\xi} = \text{mean}(\xi_i)$, $\bar{\mathbf{x}} = \text{mean}(\mathbf{x}_i)$
2. center: $A = \left(\mathbf{x}_1 - \bar{\mathbf{x}}, \ldots, \mathbf{x}_n - \bar{\mathbf{x}}\right)$, $B = \left(\xi_1 - \bar{\xi}, \ldots, \xi_n - \bar{\xi}\right)$
3. SVD decompose: $AB^T = U\Sigma V^T$
4. $Q = VU^T$ and $\mathbf{t} = \bar{\xi} - Q\bar{\mathbf{x}}$

Matrix Decompositions

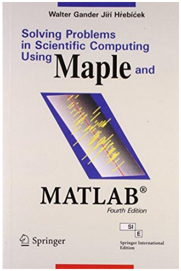
LU Decomposition

Singular Value Decomposition (SVD)

$\rightarrow$ Polar Decomposition


Introduction to LINEAR ALGEBRA

GILBERT STRANG


Walter Gander Jiří Hřebíček

Solving Problems in Scientific Computing Using Maple and MATLAB®

Fourth Edition

Springer

Matrix Polar Decomposition

Motivation: given a matrix A, which rotation W is **closest** to A? (aka: orthogonal Procrustes problem)

Idea: split matrices like imaginary numbers $z = re^{i\theta}$, where r is a **stretching** and $e^{i\theta}$ is a **rotation**

Definition

The polar decomposition of a square complex matrix A is the decomposition $A = WP$, where W is unitary and P is positive-semidefinite if A is real, such that W and P may be taken real as well

Matrix Polar Decomposition

Recall

Unitary matrix

A matrix $W \in \mathbb{R}^{n\times n}$ is called **orthogonal** iff $W^TW = I$.
A matrix $W \in \mathbb{C}^{n\times n}$ is called **unitary** iff $\overline{W}^TW = I$;
note the short hand notation: $W^* := \overline{W}^T$

Rotation

An orthogonal W is called a **rotation** if it is real and $\det(W) = 1$

Positive (semi)definite matrix

A matrix P is a positive (semi)definite matrix iff

$$\mathbf{x}^*P\mathbf{x} > (\geq)\ 0 \quad \forall \mathbf{x} \neq \mathbf{0},\ \text{for } P \in \mathbb{C}^{n\times n} \text{ or}$$

$$\mathbf{x}^TP\mathbf{x} > (\geq)\ 0 \quad \forall \mathbf{x} \neq \mathbf{0},\ \text{for } P \in \mathbb{R}^{n\times n}$$

How to Find W and P

Very easy if the SVD is given:

$$A = U\Sigma V^T = UV^T V\Sigma V^T$$

Now use

$$W = UV^T$$
$$P = V\Sigma V^T$$

$\mathbf{W}$ is a polar factor of A

How to Find W and P

W is unitary since U and V are unitary:

$$W^TW = \left(UV^T\right)^T UV^T = V^{TT}U^TUV^T = VIV^T = I$$

Proof that P is positive definite

$$\mathbf{x}^TP\mathbf{x} = \mathbf{x}^TV\Sigma V^T\mathbf{x} = \left(V^T\mathbf{x}\right)^T\Sigma\left(V^T\mathbf{x}\right)$$

Define $\mathbf{y} = V^T\mathbf{x}$, then rewrite

$$\mathbf{y}^T\Sigma\mathbf{y} = \left(y1, \quad \ldots \quad, y_n\right)\begin{pmatrix} \sigma_1 & 0 & \ldots & 0 \\ \vdots & \ddots & \ldots & 0 \\ 0 & & \sigma_m & 0 \\ 0 & \ldots & \ldots & 0 \end{pmatrix}\begin{pmatrix} y_1 \\ \vdots \\ \\ y_n \end{pmatrix}$$

$$= \sigma_1 y_1^2 + \ldots + \sigma_m y_m^2 \geq 0$$

Matrix Decomposition Summary

LU decomposition

$$A = PLU$$

P = permutation matrix
L = lower triangular matrix
U = upper triangular matrix

SVD

$$A = U\Sigma V^T$$

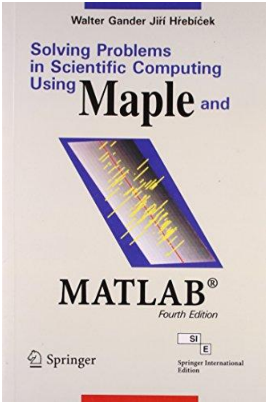
$U \& V$ = orthogonal matrices
Σ = diagonal matrix

Polar decomposition

$$A = WP$$

W = orthogonal matrix
P = positive definite matrix

Least Square Fit of Point Clouds



Walter Gander Jiří Hřebíček

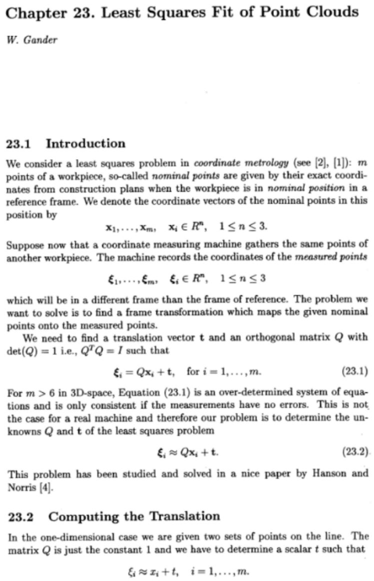
Solving Problems in Scientific Computing Using Maple and MATLAB®

Fourth Edition

Springer

SI E Springer International Edition

Paper chapter 23



Chapter 23. Least Squares Fit of Point Clouds

W. Gander

23.1 Introduction

We consider a least squares problem in *coordinate metrology* (see [2], [1]): m points of a workpiece, so-called *nominal points* are given by their exact coordinates from construction plans when the workpiece is in *nominal position* in a reference frame. We denote the coordinate vectors of the nominal points in this position by

$$\mathbf{x}_1, \dots, \mathbf{x}_m, \quad \mathbf{x}_i \in R^n, \quad 1 \le n \le 3.$$

Suppose now that a coordinate measuring machine gathers the same points of another workpiece. The machine records the coordinates of the *measured points*

$$\boldsymbol{\xi}_1, \dots, \boldsymbol{\xi}_m, \quad \boldsymbol{\xi}_i \in R^n, \quad 1 \le n \le 3$$

which will be in a different frame than the frame of reference. The problem we want to solve is to find a frame transformation which maps the given nominal points onto the measured points.

We need to find a translation vector $\mathbf{t}$ and an orthogonal matrix Q with $\det(Q) = 1$ i.e., $Q^T Q = I$ such that

$$\boldsymbol{\xi}_i = Q\mathbf{x}_i + \mathbf{t}, \quad \text{for } i = 1, \dots, m. \tag{23.1}$$

For $m > 6$ in 3D-space, Equation (23.1) is an over-determined system of equations and is only consistent if the measurements have no errors. This is not the case for a real machine and therefore our problem is to determine the unknowns Q and $\mathbf{t}$ of the least squares problem

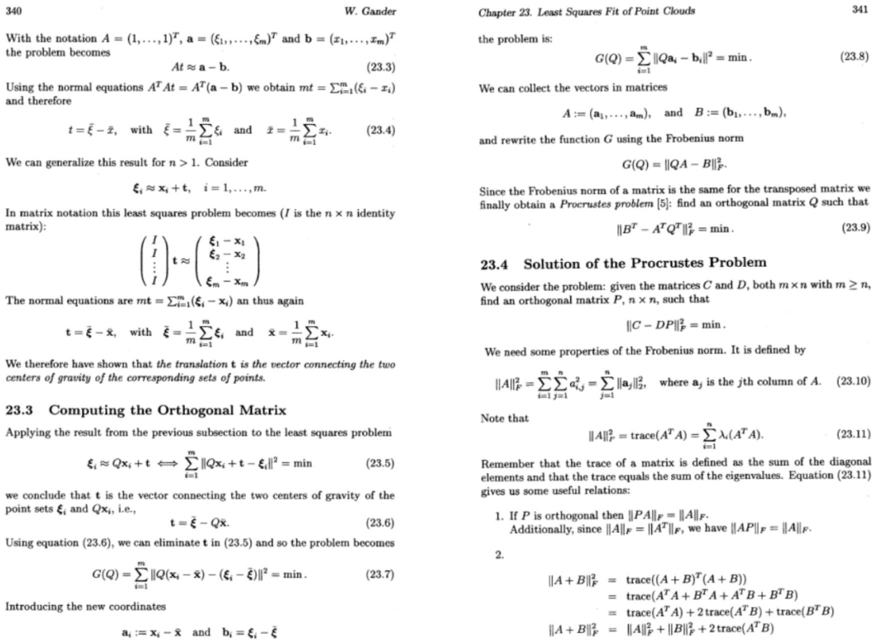
$$\boldsymbol{\xi}_i \approx Q\mathbf{x}_i + \mathbf{t}. \tag{23.2}$$

This problem has been studied and solved in a nice paper by Hanson and Norris [4].

23.2 Computing the Translation

In the one-dimensional case we are given two sets of points on the line. The matrix Q is just the constant 1 and we have to determine a scalar t such that

$$\xi_i \approx x_i + t, \quad i = 1, \dots, m.$$



With the notation $A = (1, \dots, 1)^T$, $\mathbf{a} = (\xi_1, \dots, \xi_m)^T$ and $\mathbf{b} = (x_1, \dots, x_m)^T$ the problem becomes

$$At \approx \mathbf{a} - \mathbf{b}. \tag{23.3}$$

Using the normal equations $A^T At = A^T(\mathbf{a} - \mathbf{b})$ we obtain $mt = \sum_{i=1}^m (\xi_i - x_i)$ and therefore

$$t = \bar{\xi} - \bar{x}, \quad \text{with} \quad \bar{\xi} = \frac{1}{m}\sum_{i=1}^m \xi_i \quad \text{and} \quad \bar{x} = \frac{1}{m}\sum_{i=1}^m x_i. \tag{23.4}$$

We can generalize this result for $n > 1$. Consider

$$\boldsymbol{\xi}_i \approx \mathbf{x}_i + \mathbf{t}, \quad i = 1, \dots, m.$$

In matrix notation this least squares problem becomes (I is the $n \times n$ identity matrix):

$$\begin{pmatrix} I \\ I \\ \vdots \\ I \end{pmatrix} \mathbf{t} \approx \begin{pmatrix} \boldsymbol{\xi}_1 - \mathbf{x}_1 \\ \boldsymbol{\xi}_2 - \mathbf{x}_2 \\ \vdots \\ \boldsymbol{\xi}_m - \mathbf{x}_m \end{pmatrix}$$

The normal equations are $m\mathbf{t} = \sum_{i=1}^m (\boldsymbol{\xi}_i - \mathbf{x}_i)$ an thus again

$$\mathbf{t} = \bar{\boldsymbol{\xi}} - \bar{\mathbf{x}}, \quad \text{with} \quad \bar{\boldsymbol{\xi}} = \frac{1}{m}\sum_{i=1}^m \boldsymbol{\xi}_i \quad \text{and} \quad \bar{\mathbf{x}} = \frac{1}{m}\sum_{i=1}^m \mathbf{x}_i.$$

We therefore have shown that *the translation* $\mathbf{t}$ *is the vector connecting the two centers of gravity of the corresponding sets of points.*

23.3 Computing the Orthogonal Matrix

Applying the result from the previous subsection to the least squares problem

$$\boldsymbol{\xi}_i \approx Q\mathbf{x}_i + \mathbf{t} \iff \sum_{i=1}^m \|Q\mathbf{x}_i + \mathbf{t} - \boldsymbol{\xi}_i\|^2 = \min \tag{23.5}$$

we conclude that $\mathbf{t}$ is the vector connecting the two centers of gravity of the point sets $\boldsymbol{\xi}_i$ and $Q\mathbf{x}_i$, i.e.,

$$\mathbf{t} = \bar{\boldsymbol{\xi}} - Q\bar{\mathbf{x}}. \tag{23.6}$$

Using equation (23.6), we can eliminate $\mathbf{t}$ in (23.5) and so the problem becomes

$$G(Q) = \sum_{i=1}^m \|Q(\mathbf{x}_i - \bar{\mathbf{x}}) - (\boldsymbol{\xi}_i - \bar{\boldsymbol{\xi}})\|^2 = \min. \tag{23.7}$$

Introducing the new coordinates

$$\mathbf{a}_i := \mathbf{x}_i - \bar{\mathbf{x}} \quad \text{and} \quad \mathbf{b}_i = \boldsymbol{\xi}_i - \bar{\boldsymbol{\xi}}$$

the problem is:

$$G(Q) = \sum_{i=1}^m \|Q\mathbf{a}_i - \mathbf{b}_i\|^2 = \min. \tag{23.8}$$

We can collect the vectors in matrices

$$A := (\mathbf{a}_1, \dots, \mathbf{a}_m), \quad \text{and} \quad B := (\mathbf{b}_1, \dots, \mathbf{b}_m),$$

and rewrite the function G using the Frobenius norm

$$G(Q) = \|QA - B\|_F^2.$$

Since the Frobenius norm of a matrix is the same for the transposed matrix we finally obtain a *Procrustes problem* [5]: find an orthogonal matrix Q such that

$$\|B^T - A^T Q^T\|_F^2 = \min. \tag{23.9}$$

23.4 Solution of the Procrustes Problem

We consider the problem: given the matrices C and D, both $m \times n$ with $m \geq n$, find an orthogonal matrix P, $n \times n$, such that

$$\|C - DP\|_F^2 = \min.$$

We need some properties of the Frobenius norm. It is defined by

$$\|A\|_F^2 = \sum_{i=1}^m \sum_{j=1}^n a_{i,j}^2 = \sum_{j=1}^n \|\mathbf{a}_j\|_2^2, \quad \text{where } \mathbf{a}_j \text{ is the } j\text{th column of } A. \tag{23.10}$$

Note that

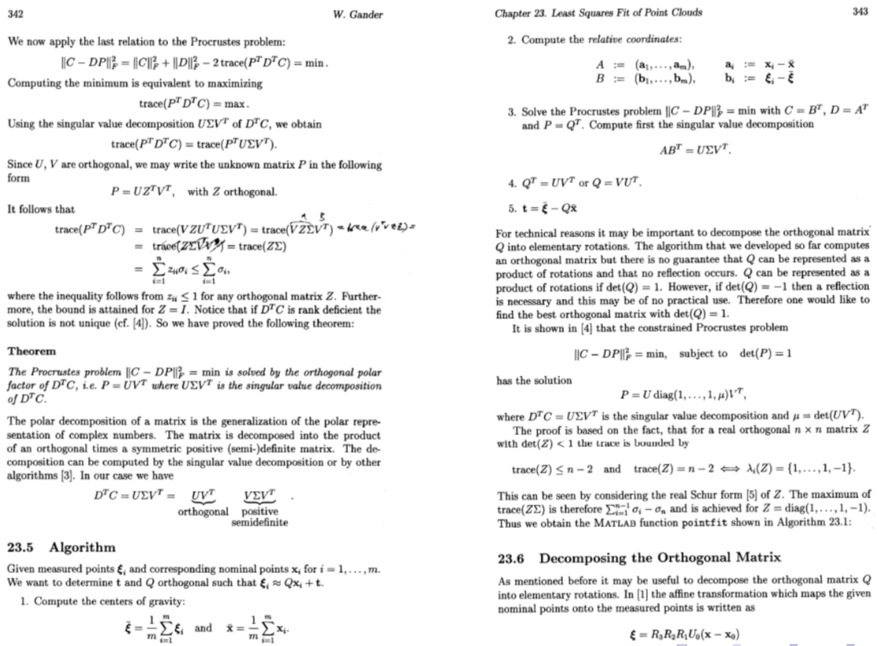
$$\|A\|_F^2 = \operatorname{trace}(A^T A) = \sum_{i=1}^n \lambda_i(A^T A). \tag{23.11}$$

Remember that the trace of a matrix is defined as the sum of the diagonal elements and that the trace equals the sum of the eigenvalues. Equation (23.11) gives us some useful relations:

1. If P is orthogonal then $\|PA\|_F = \|A\|_F$.
   Additionally, since $\|A\|_F = \|A^T\|_F$, we have $\|AP\|_F = \|A\|_F$.

2.

$$\begin{aligned} \|A + B\|_F^2 &= \operatorname{trace}((A+B)^T(A+B)) \\ &= \operatorname{trace}(A^T A + B^T A + A^T B + B^T B) \\ &= \operatorname{trace}(A^T A) + 2\operatorname{trace}(A^T B) + \operatorname{trace}(B^T B) \\ \|A + B\|_F^2 &= \|A\|_F^2 + \|B\|_F^2 + 2\operatorname{trace}(A^T B) \end{aligned}$$



We now apply the last relation to the Procrustes problem:

$$\|C - DP\|_F^2 = \|C\|_F^2 + \|D\|_F^2 - 2\,\text{trace}(P^T D^T C) = \min .$$

Computing the minimum is equivalent to maximizing

$$\text{trace}(P^T D^T C) = \max .$$

Using the singular value decomposition $U\Sigma V^T$ of $D^T C$, we obtain

$$\text{trace}(P^T D^T C) = \text{trace}(P^T U \Sigma V^T).$$

Since U, V are orthogonal, we may write the unknown matrix P in the following form

$$P = UZ^T V^T, \quad \text{with } Z \text{ orthogonal.}$$

It follows that

$$\begin{aligned} \text{trace}(P^T D^T C) &= \text{trace}(VZU^T U \Sigma V^T) = \text{trace}(\overbrace{VZ}^{A}\overbrace{\Sigma V^T}^{B}) = \text{trace}(V^T V Z \Sigma) = \\ &= \text{trace}(Z\Sigma \cancel{V^T V}) = \text{trace}(Z\Sigma) \\ &= \sum_{i=1}^{n} z_{ii}\sigma_i \le \sum_{i=1}^{n} \sigma_i, \end{aligned}$$

where the inequality follows from $z_{ii} \le 1$ for any orthogonal matrix Z. Furthermore, the bound is attained for $Z = I$. Notice that if $D^T C$ is rank deficient the solution is not unique (cf. [4]). So we have proved the following theorem:

Theorem

The Procrustes problem $\|C - DP\|_F^2 = \min$ is solved by the orthogonal polar factor of $D^T C$, i.e. $P = UV^T$ where $U\Sigma V^T$ is the singular value decomposition of $D^T C$.

The polar decomposition of a matrix is the generalization of the polar representation of complex numbers. The matrix is decomposed into the product of an orthogonal times a symmetric positive (semi-)definite matrix. The decomposition can be computed by the singular value decomposition or by other algorithms [3]. In our case we have

$$D^T C = U\Sigma V^T = \underbrace{UV^T}_{\text{orthogonal}} \; \underbrace{V\Sigma V^T}_{\substack{\text{positive}\\ \text{semidefinite}}} .$$

23.5 Algorithm

Given measured points $\boldsymbol{\xi}_i$ and corresponding nominal points $\mathbf{x}_i$ for $i = 1, \ldots, m$. We want to determine $\mathbf{t}$ and Q orthogonal such that $\boldsymbol{\xi}_i \approx Q\mathbf{x}_i + \mathbf{t}$.

1. Compute the centers of gravity:

$$\bar{\boldsymbol{\xi}} = \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{\xi}_i \quad \text{and} \quad \bar{\mathbf{x}} = \frac{1}{m}\sum_{i=1}^{m} \mathbf{x}_i .$$

2. Compute the *relative coordinates*:

$$\begin{array}{rclrcl} A &:=& (\mathbf{a}_1, \ldots, \mathbf{a}_m), & \mathbf{a}_i &:=& \mathbf{x}_i - \bar{\mathbf{x}} \\ B &:=& (\mathbf{b}_1, \ldots, \mathbf{b}_m), & \mathbf{b}_i &:=& \boldsymbol{\xi}_i - \bar{\boldsymbol{\xi}} \end{array}$$

3. Solve the Procrustes problem $\|C - DP\|_F^2 = \min$ with $C = B^T$, $D = A^T$ and $P = Q^T$. Compute first the singular value decomposition

$$AB^T = U\Sigma V^T.$$

4. $Q^T = UV^T$ or $Q = VU^T$.

5. $\mathbf{t} = \bar{\boldsymbol{\xi}} - Q\bar{\mathbf{x}}$

For technical reasons it may be important to decompose the orthogonal matrix Q into elementary rotations. The algorithm that we developed so far computes an orthogonal matrix but there is no guarantee that Q can be represented as a product of rotations and that no reflection occurs. Q can be represented as a product of rotations if $\det(Q) = 1$. However, if $\det(Q) = -1$ then a reflection is necessary and this may be of no practical use. Therefore one would like to find the best orthogonal matrix with $\det(Q) = 1$.

It is shown in [4] that the constrained Procrustes problem

$$\|C - DP\|_F^2 = \min, \quad \text{subject to} \quad \det(P) = 1$$

has the solution

$$P = U\,\text{diag}(1, \ldots, 1, \mu)V^T,$$

where $D^T C = U\Sigma V^T$ is the singular value decomposition and $\mu = \det(UV^T)$.

The proof is based on the fact, that for a real orthogonal $n \times n$ matrix Z with $\det(Z) < 1$ the trace is bounded by

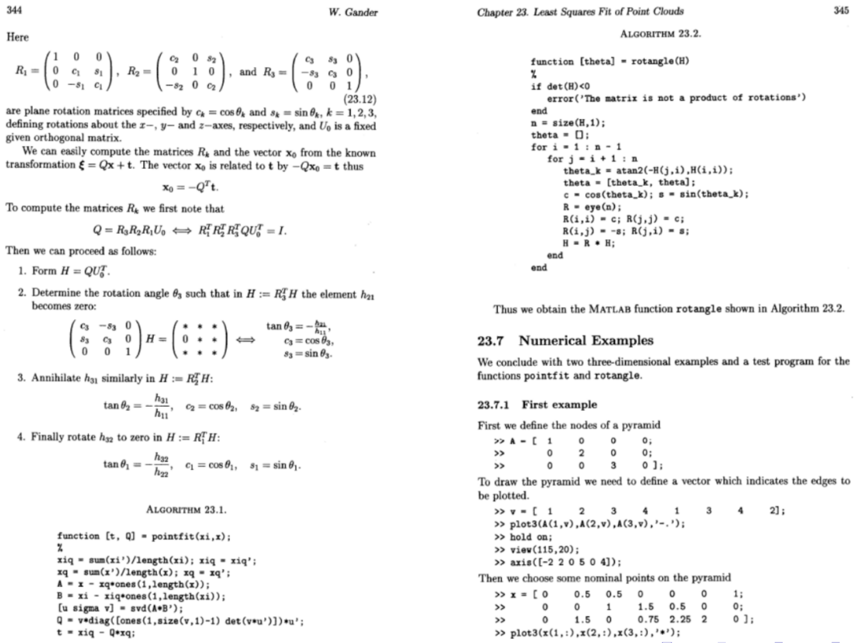
$$\text{trace}(Z) \le n - 2 \quad \text{and} \quad \text{trace}(Z) = n - 2 \iff \lambda_i(Z) = \{1, \ldots, 1, -1\}.$$

This can be seen by considering the real Schur form [5] of Z. The maximum of $\text{trace}(Z\Sigma)$ is therefore $\sum_{i=1}^{n-1} \sigma_i - \sigma_n$ and is achieved for $Z = \text{diag}(1, \ldots, 1, -1)$. Thus we obtain the MATLAB function `pointfit` shown in Algorithm 23.1:

23.6 Decomposing the Orthogonal Matrix

As mentioned before it may be useful to decompose the orthogonal matrix Q into elementary rotations. In [1] the affine transformation which maps the given nominal points onto the measured points is written as

$$\boldsymbol{\xi} = R_3 R_2 R_1 U_0 (\mathbf{x} - \mathbf{x}_0)$$



Here

$$R_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & c_1 & s_1 \\ 0 & -s_1 & c_1 \end{pmatrix}, \quad R_2 = \begin{pmatrix} c_2 & 0 & s_2 \\ 0 & 1 & 0 \\ -s_2 & 0 & c_2 \end{pmatrix}, \quad \text{and} \quad R_3 = \begin{pmatrix} c_3 & s_3 & 0 \\ -s_3 & c_3 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{23.12}$$

are plane rotation matrices specified by $c_k = \cos\theta_k$ and $s_k = \sin\theta_k$, $k = 1, 2, 3$, defining rotations about the $x-$, $y-$ and $z-$axes, respectively, and U_0 is a fixed given orthogonal matrix.

We can easily compute the matrices R_k and the vector $\mathbf{x}_0$ from the known transformation $\boldsymbol{\xi} = Q\mathbf{x} + \mathbf{t}$. The vector $\mathbf{x}_0$ is related to $\mathbf{t}$ by $-Q\mathbf{x}_0 = \mathbf{t}$ thus

$$\mathbf{x}_0 = -Q^T\mathbf{t}.$$

To compute the matrices R_k we first note that

$$Q = R_3R_2R_1U_0 \iff R_1^TR_2^TR_3^TQU_0^T = I.$$

Then we can proceed as follows:

1. Form $H = QU_0^T$.
2. Determine the rotation angle θ_3 such that in $H := R_3^TH$ the element h_{21} becomes zero:
$$\begin{pmatrix} c_3 & -s_3 & 0 \\ s_3 & c_3 & 0 \\ 0 & 0 & 1 \end{pmatrix} H = \begin{pmatrix} * & * & * \\ 0 & * & * \\ * & * & * \end{pmatrix} \iff \begin{aligned} \tan\theta_3 &= -\tfrac{h_{21}}{h_{11}}, \\ c_3 &= \cos\theta_3, \\ s_3 &= \sin\theta_3. \end{aligned}$$
3. Annihilate h_{31} similarly in $H := R_2^TH$:
$$\tan\theta_2 = -\frac{h_{31}}{h_{11}}, \quad c_2 = \cos\theta_2, \quad s_2 = \sin\theta_2.$$
4. Finally rotate h_{32} to zero in $H := R_1^TH$:
$$\tan\theta_1 = -\frac{h_{32}}{h_{22}}, \quad c_1 = \cos\theta_1, \quad s_1 = \sin\theta_1.$$

ALGORITHM 23.1.

```
function [t, Q] = pointfit(xi,x);
%
xiq = sum(xi')/length(xi); xiq = xiq';
xq = sum(x')/length(x); xq = xq';
A = x - xq*ones(1,length(x));
B = xi - xiq*ones(1,length(xi));
[u sigma v] = svd(A*B');
Q = v*diag([ones(1,size(v,1)-1) det(v*u')])*u';
t = xiq - Q*xq;
```

ALGORITHM 23.2.

```
function [theta] = rotangle(H)
%
if det(H)<0
   error('The matrix is not a product of rotations')
end
n = size(H,1);
theta = [];
for i = 1 : n - 1
   for j = i + 1 : n
      theta_k = atan2(-H(j,i),H(i,i));
      theta = [theta_k, theta];
      c = cos(theta_k); s = sin(theta_k);
      R = eye(n);
      R(i,i) = c; R(j,j) = c;
      R(i,j) = -s; R(j,i) = s;
      H = R * H;
   end
end
```

Thus we obtain the MATLAB function `rotangle` shown in Algorithm 23.2.

23.7 Numerical Examples

We conclude with two three-dimensional examples and a test program for the functions `pointfit` and `rotangle`.

23.7.1 First example

First we define the nodes of a pyramid

```
>> A = [ 1    0    0    0;
>>       0    2    0    0;
>>       0    0    3    0 ];
```

To draw the pyramid we need to define a vector which indicates the edges to be plotted.

```
>> v = [ 1    2    3    4    1    3    4    2];
>> plot3(A(1,v),A(2,v),A(3,v),'-.');
>> hold on;
>> view(115,20);
>> axis([-2 2 0 5 0 4]);
```

Then we choose some nominal points on the pyramid

```
>> x = [ 0    0.5  0.5  0    0    0    1;
>>       0    0    1    1.5  0.5  0    0;
>>       0    1.5  0    0.75 2.25 2    0 ];
>> plot3(x(1,:),x(2,:),x(3,:),'*');
```

ALGORITHM 23.3.

```
function [Qr] = ang2orth(theta)
% ANG2ORTH generate orthogonal matrix from given angles
Qr = eye(3);
n = 1;
for i = 2 : -1 : 1
   for j = 3 : -1 : i + 1
      t = theta(n);
      s = sin(t); c = cos(t);
      U = eye(3);
      U(i,i) =  c; U(i,j) = s;
      U(j,i) = -s; U(j,j) = c;
      Qr = U * Qr; n = n + 1;
   end
end
```

Now we will simulate the measured points. In order to give the reader reproducible results we commented out the randomly generated data and replaced them by fixed values. We construct an orthogonal matrix Qr by generating three angles und using the MATLAB function **ang2orth** shown in Algorithm 23.3:

```
>> %thetar = rand(1,1:3)
>> thetar = [pi/4 pi/15 -pi/6];
>> Qr = ang2orth(thetar);
```

and also a random translation vector:

```
>> %tr = rand(3,1)*3;
>> tr = [1;3;2];
```

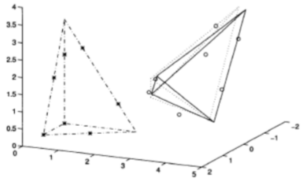
Now we can compute and plot the exactly transformed pyramid:

```
>> B = Qr * A + tr * ones(1, size(A,2));
>> plot3(B(1,v),B(2,v),B(3,v),':');
>> pause
```

We transform the nominal points and add some noise to simulate the measured points (again for reasons of reproducibility of results we give the points explicitly):

```
>> % (randomly distorted) measured points:
>> % xi = (Qr*x+tr*ones(1,length(x))+randn(size(x))/10);
>>
>> xi = [ 0.8314 0.9821 1.0211 0.1425 0.2572 0.5229 1.7713
>>        3.0358 4.5232 3.8075 4.4826 5.0120 4.5364 3.3987
>>        1.9328 2.8703 1.0573 1.5803 3.1471 3.5394 1.9054];
>> plot3(xi(1,:),xi(2,:),xi(3,:),'o');
>> pause
```

FIGURE 23.1. *Point Fit.*



Using the function **pointfit** we now estimate Q and t from the measured points and we plot the fitted pyramid (see Figure 23.1):

```
>> [t,Q]= pointfit(xi,x);
>> % the fitted pyramid:
>> C = Q * A + t * ones(1, size(A,2));
>> plot3(C(1,v),C(2,v),C(3,v),'-');
>> hold off;
```

Finally we compute the rotation angles of the fitted pyramid and compare them with the original ones.

```
>> theta = rotangle(Q)

theta =

    0.8282    0.1772   -0.3964

>> thetar

thetar =

    0.7854    0.2094   -0.5236
```

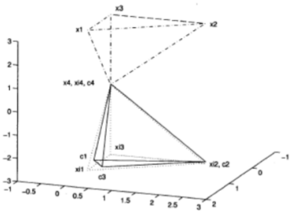
As a final check we generate the orthogonal matrix S from the computed angles θ and compare it with the result Q of **pointfit**:

```
>> S = ang2orth(theta)';
>> norm(Q - S)

ans =

  2.3497e-015
```

FIGURE 23.2.
Best Rotation: x_i nominal points, xi_i "measured points", c_i best fit



23.7.2 Second example

In this example we construct two sets of points which can be transformed exactly into one another by a reflection. We will compute the best solution using only rotations which cannot match the measured points since the orientation of the body would have to change. Consider again the vertices of a pyramid as the nominal points:

```
>> % Define nominal points.
>> A = [1 0 0 0;0 2 0 0;3 3 3 0];
>> v = [1 2 3 4 1 3 2 4];
>> plot3(A(1,v),A(2,v),A(3,v),'-.');
>> hold on;
>> view(110,20)
>> axis([-1 2 -1 3 -3 3])
```

As measured points we use the vertices of the pyramid which is reflected at the xy-plane:

```
>> B = [1 0 0 0;0 2 0 0;-3 -3 -3 0];
>> plot3(B(1,v),B(2,v),B(3,v),':')
```

We compute the best fit and plot the result in Figure 23.2:

```
>> [t,Q] = pointfit(B,A);
>> C = Q * A + t * ones(1, size(A,2))
```

```
>> plot3(C(1,v),C(2,v),C(3,v),'-')
>> text(1,-.25,3,'x1'); text(0,2.1,3,'x2')
>> text(0,.1,3.3,'x3'); text(0,-1,-0.1,'x4, xi4, c4')
>> text(1,-.25,-3,'xi1'); text(0,2.1,-3,'xi2, c2')
>> text(0,.1,-2.65,'xi3'); text(0.7,-0.3,-2.65,'c1')
>> text(1.75,.6,-2.65,'c3')
>> theta = 180/pi*rotangle(Q)
>> hold off;
```

Acknowledgments

I wish to thank Max Oertli for his help in finishing the last section.

References

[1] BUTLER, B. P., FORBES A. B. AND HARRIS, P. M., *Algorithms for Geometric Tolerance Assessment.* Technical Report DITC 228/94, National Physical Laboratory, Teddington, 1994.

[2] FORBES A. B., *Geometric Tolerance Assessment.* Technical Report DITC 210/92, National Physical Laboratory, Teddington, 1992.

[3] GANDER, W., *Algorithms for the Polar Decomposition*, SIAM J. on Sci. and Stat. Comp., Vol. 11, No. 6, 1990.

[4] HANSON, R. AND NORRIS, M. *Analysis of Measurements Based on the Singular Value Decomposition*, SIAM J. on Sci. and Stat. Comp., Vol. 2, No. 3, 1981.

[5] GOLUB, GENE H. AND VAN LOAN, CHARLES F., *Matrix Computations.* 2nd ed. Baltimore Johns Hopkins University Press, c1989.