Principle Component Analysis (PCA)

Felix Brockherde 2015
TU Berlin and MPI Halle

This Lecture

1. Lagrange Multipliers
2. Principle Component Analysis
    1. What are Principle Components?
    2. How to find/calculate them
    3. What can we do with them? / Applications

Lagrange Multipliers

- Imagine you have a constrained optimization problem:

Objective function

Read: subject to

$$\max_x f(x)$$
$$\text{s.t. } g(x) = 0$$

Constraint

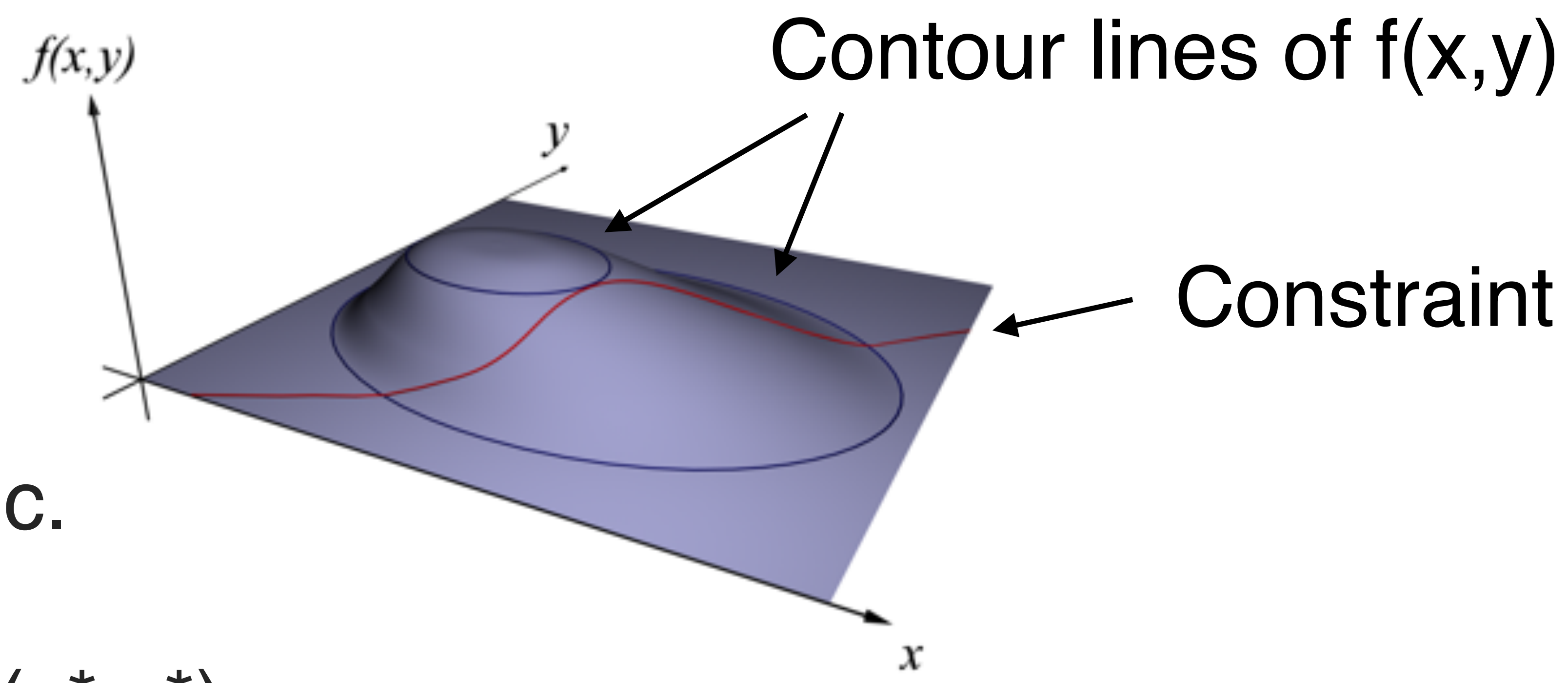
- Example:

$$\max_{\mathbf{x}} 1 - x_1^2 - x_2^2$$
$$\text{s.t. } x_1 + x_2 - 1 = 0$$

Reference: Bishop, Pattern Recognition and Machine Learning, Appendix E

Geometric intuition

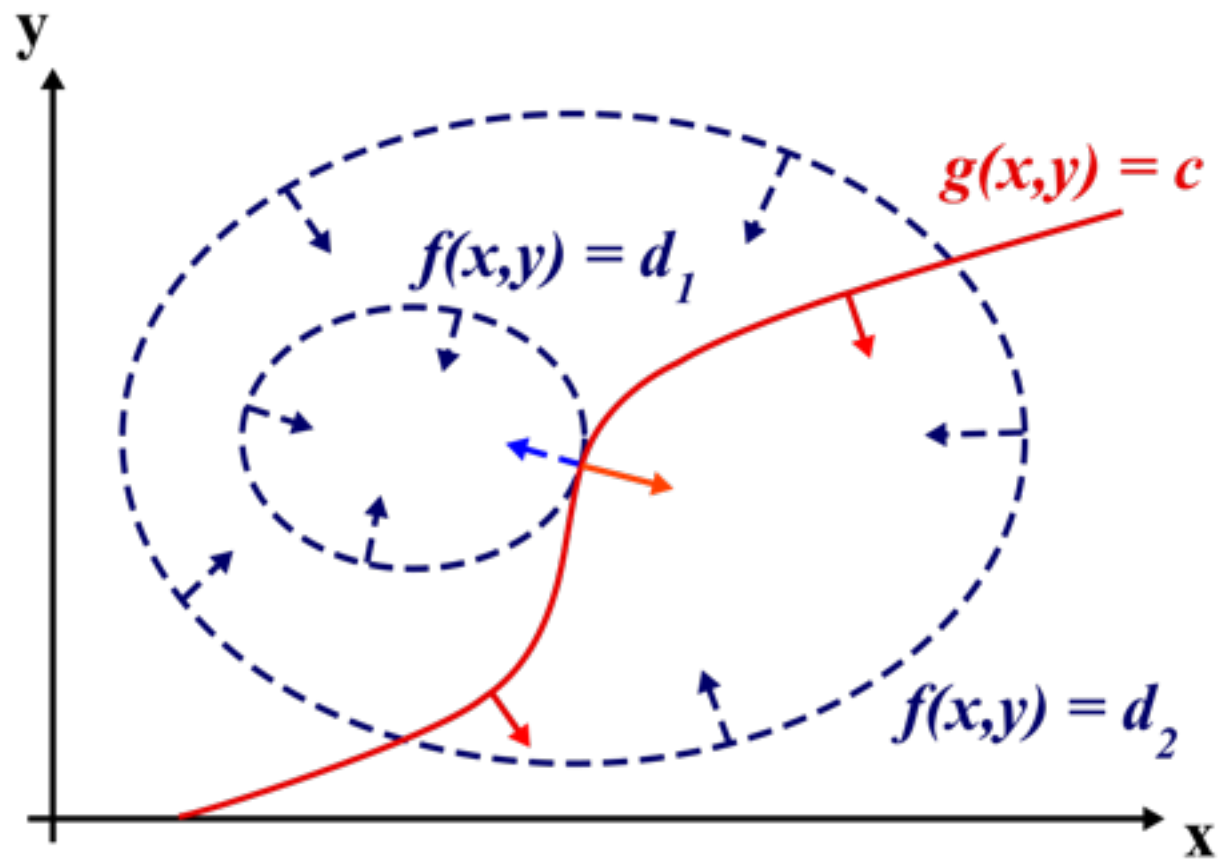
$$\max_x f(x)$$
$$\text{s.t. } g(x) = 0$$



Intuition 1: ∇g must be normal to the line g(x, y) = c.

Intuition 2: At a maximum f(x*,y*) can not increase in the direction of a neighboring point where g(x, y) = c. Thus, at (x*, y*) ∇f must be perpendicular to the line g(x, y) = c.

It follows: <u>In (x*, y*) the gradients ∇g and ∇f are parallel!</u>

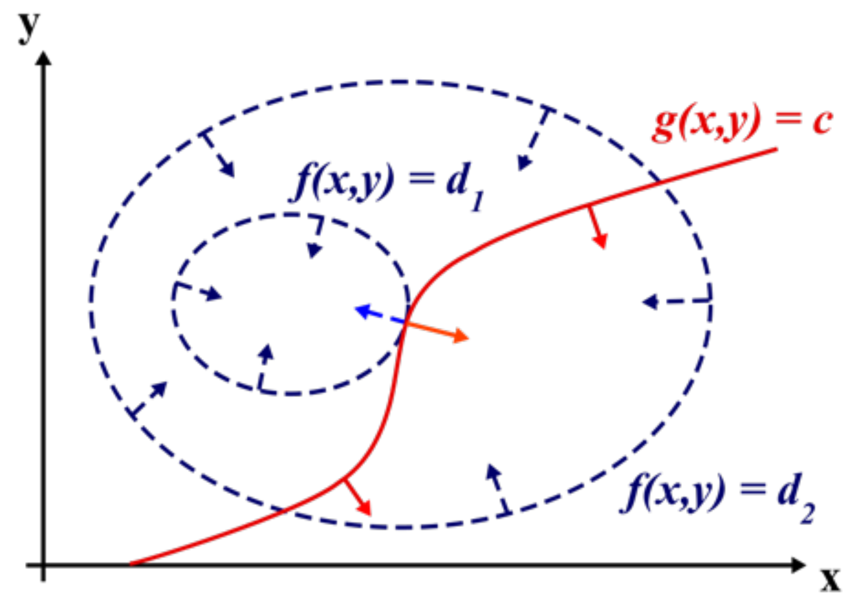


The Lagrange Multiplier

In (x*, y*) the gradients ∇g and ∇f are parallel. Thus for a maximum:

$$\nabla f = \lambda \nabla g$$

Lagrange Multiplier



Lagrangian function:

$$\mathcal{L}(x,\lambda) = f(x) + \lambda g(x)$$

∇*L* = 0 is a necessary (but not sufficient) condition for the optimization solution.

Thus, to solve a constrained optimization problem, we can define the Lagrangian and look for the points where its gradient vanishes.

Example

$$\max_{\mathbf{x}} 1 - x_1^2 - x_2^2$$
$$\text{s.t. } x_1 + x_2 - 1 = 0$$

1. Define Lagrangian:

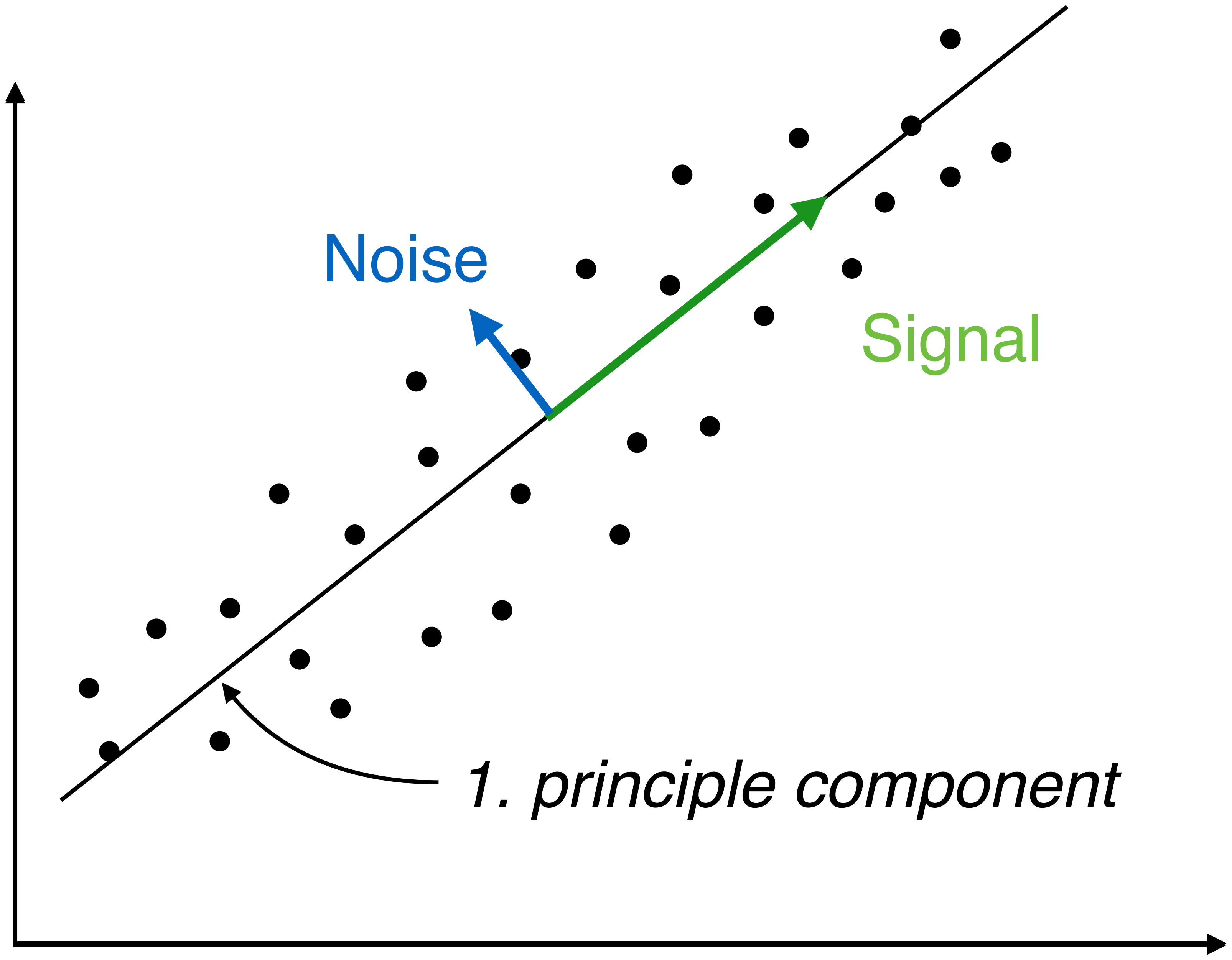
$$\mathcal{L}(x, \lambda) = 1 - x_1^2 - x_2^2 + \lambda(x_1 + x_2 - 1)$$

2. Set gradient of Lagrangian to zero:

$$\begin{aligned} -2x_1 + \lambda &= 0 \\ -2x_2 + \lambda &= 0 \\ x_1 + x_2 - 1 &= 0 \end{aligned}$$

3. Solve equations: $(x_1^*, x_2^*) = (\frac{1}{2}, \frac{1}{2}) \qquad \lambda = 1$

Principle Component Analysis (PCA)



Which line fits data best?

The line *w* that minimizes the noise and maximizes the signal [Pearson 1901].

Reference: Bishop, Pattern Recognition and Machine Learning, Chapter 12.1

Problem Definition

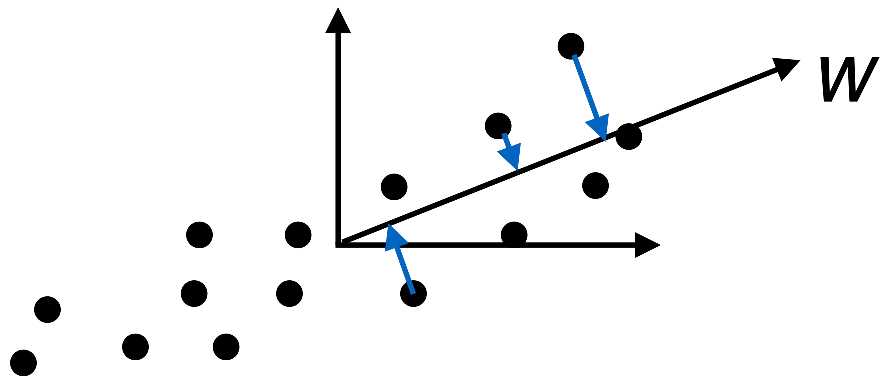
Given $x_1, \ldots, x_n \in \mathrm{R}^d$ find a *k*-dimensional subspace, so that the data projected on that subspace

1) is as close to the original data as possible (minimum noise)
2) has maximum variance (maximum signal).

2 Motivations: same result?

Assume ***<u>the data is centered</u>***: $\mu = \frac{1}{n}\sum_{i=1}^{n} x_i = 0$

Minimum distance



We are only interested in the direction of w, not its length, i.e. set $||w|| = w^{\mathrm{T}}w = 1$

$$\min_{w} \sum_{i=1}^{n} ||(w^{\mathrm{T}} x_i) w - x_i||^2$$

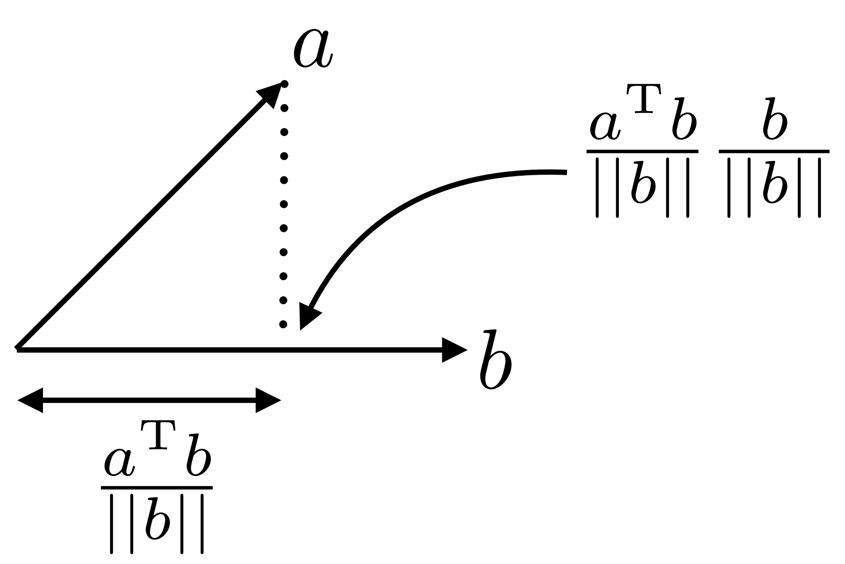
$$= \min_{w} \sum_{i=1}^{n} w^{\mathrm{T}} x_i x_i^{\mathrm{T}} w w^{\mathrm{T}} w - 2 w^{\mathrm{T}} x_i x_i^{\mathrm{T}} w + x_i^{\mathrm{T}} x_i$$

$$= \min_{w} - \sum_{i=1}^{n} w^{\mathrm{T}} x_i x_i^{\mathrm{T}} w$$

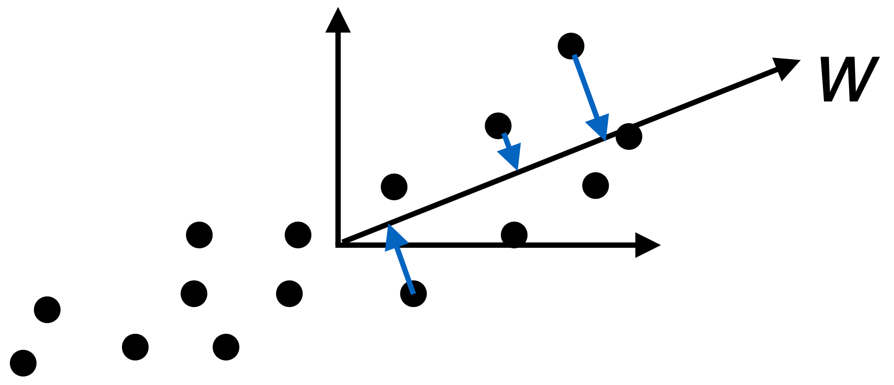
$$= \max_{w} \sum_{i=1}^{n} w^{\mathrm{T}} x_i x_i^{\mathrm{T}} w$$

$$= \max_{w} w^{\mathrm{T}} X X^{\mathrm{T}} w$$

<u>Reminder</u>


a
$\frac{a^{\mathrm{T}} b}{||b||} \frac{b}{||b||}$
b
$\frac{a^{\mathrm{T}} b}{||b||}$

Maximum Variance



X is centered

$$\max_w \mathrm{Var}(w^\mathrm{T} X) = \mathrm{E}[(w^\mathrm{T} X)^2]$$

$$= \max_w \frac{1}{n} \sum_{i=1}^{n} w^\mathrm{T} x_i x_i^\mathrm{T} w$$

$$= \max_w w^\mathrm{T} X X^\mathrm{T} w$$

✓ (same as minimum distance)

Introduce the *scatter matrix* $S = XX^\mathrm{T}$. This leads to the constrained optimization problem:

$$\max_w \; w^\mathrm{T} S w$$
$$\text{s.t. } ||w|| = 1$$

PCA optimization problem

$$\max_{w} \; w^{\mathrm{T}} S w$$
$$\text{s.t. } \; ||w|| = 1$$

1. Define Lagrangian: $\mathcal{L}(w, \lambda) = w^{\mathrm{T}} S w + \lambda(1 - w^{\mathrm{T}} w)$

2. Compute gradient: $\dfrac{\partial \mathcal{L}(w, \lambda)}{\partial w} = 2Sw - 2\lambda w$

3. Set to zero: $Sw = \lambda w$

This is an Eigenvalue problem!

PCA in multiple dimensions

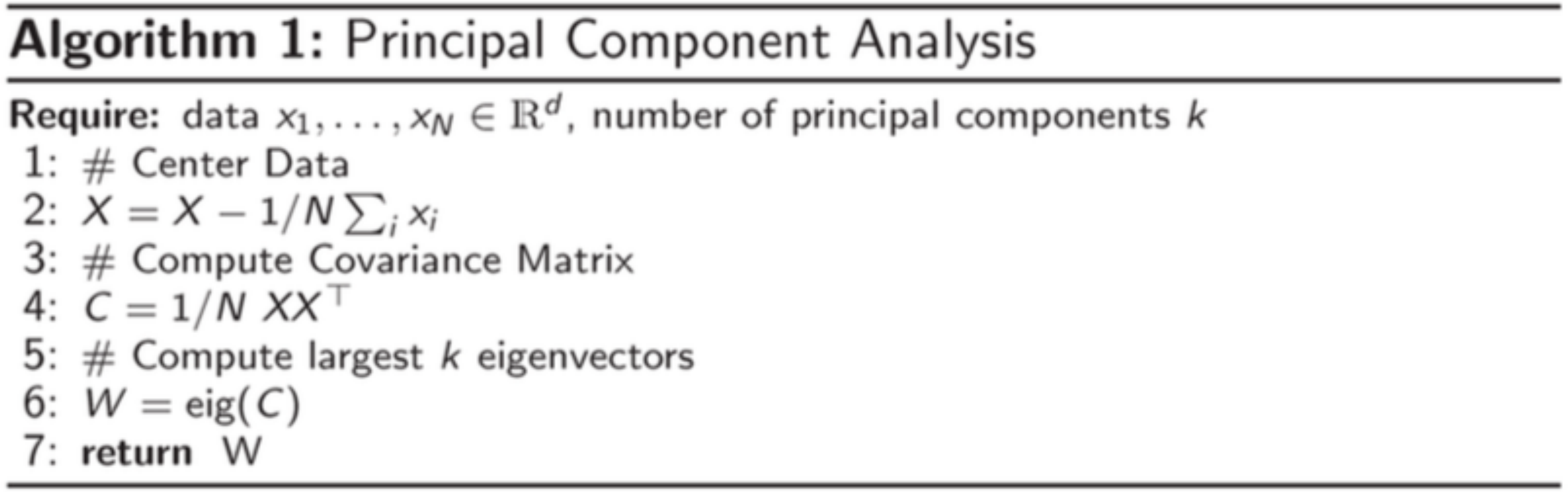
The i-th eigenvalue is the variance in the direction of the i-th eigenvector:

$$\mathrm{Var}(w_i^{\mathrm{T}} x) = \frac{1}{n} w_i^{\mathrm{T}} S w_i = \frac{1}{n} \lambda_i w_i^{\mathrm{T}} w_i = \frac{1}{n} \lambda_i$$

The direction of largest variance corresponds to the largest eigenvector (= the eigenvector with largest eigenvalue).

When we redo PCA in the orthogonal complement of the subspace spanned by the first EV ($\mathrm{span}\{w_1\}$), we get the direction of second principle component (w_2). This is the second largest EV of the original scatter matrix (w/o proof).

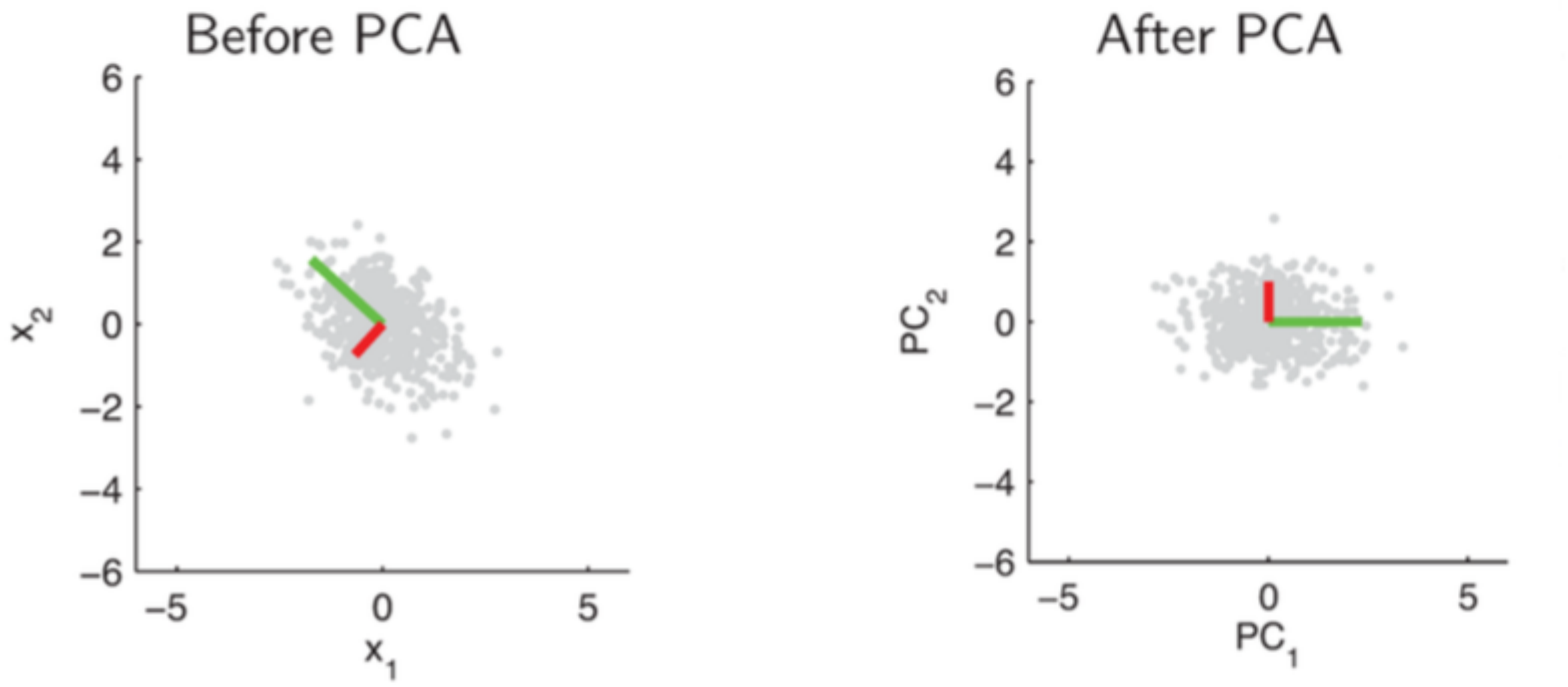
PCA Algorithm



Algorithm 1: Principal Component Analysis

```
Require: data $x_1, \dots, x_N \in \mathbb{R}^d$, number of principal components $k$
1: # Center Data
2: $X = X - 1/N \sum_i x_i$
3: # Compute Covariance Matrix
4: $C = 1/N \; XX^\top$
5: # Compute largest $k$ eigenvectors
6: $W = \text{eig}(C)$
7: return W
```

Example



PCA rotates data into new coordinate system with the directions of largest variances being the new coordinate axes.

Questions: What happens when the data is not centered?

Solving PCA with SVD

A singular value decomposition factorizes a matrix as:

$$U\Lambda V = M$$

where

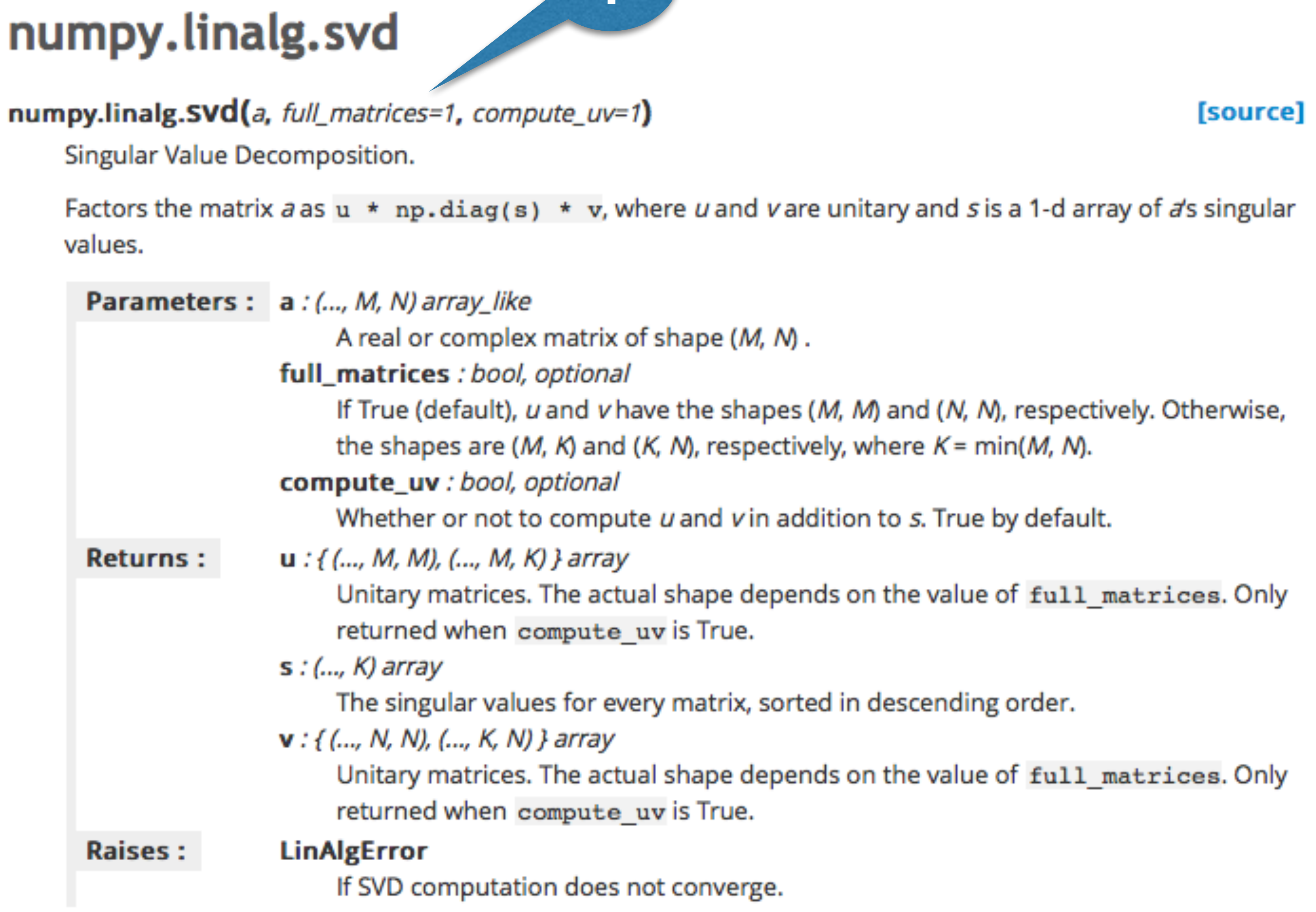
U are the Eigenvectors of MM^*.
V are the Eigenvectors of M^*M.
The square roots of the Eigenvalues of
MM^* are on the diagonal of Λ.

Instead of calculating the EV-decomposition of S, we can compute the SVD-decomposition of X. <u>This is computationally much more stable.</u>

E.g. Läuchli Matrix $\begin{pmatrix} 1 & 1 & 1 \\ \epsilon & 0 & 0 \\ 0 & \epsilon & 0 \\ 0 & 0 & \epsilon \end{pmatrix}^{\mathsf{T}}$

SVD in numpy

!



numpy.linalg.svd

numpy.linalg.svd(*a*, *full_matrices=1*, *compute_uv=1*) [source]

Singular Value Decomposition.

Factors the matrix *a* as `u * np.diag(s) * v`, where *u* and *v* are unitary and *s* is a 1-d array of *a*'s singular values.

Parameters :

a : *(..., M, N) array_like*
A real or complex matrix of shape (*M*, *N*) .

full_matrices : *bool, optional*
If True (default), *u* and *v* have the shapes (*M*, *M*) and (*N*, *N*), respectively. Otherwise, the shapes are (*M*, *K*) and (*K*, *N*), respectively, where *K* = min(*M*, *N*).

compute_uv : *bool, optional*
Whether or not to compute *u* and *v* in addition to *s*. True by default.

Returns :

u : *{ (..., M, M), (..., M, K) } array*
Unitary matrices. The actual shape depends on the value of `full_matrices`. Only returned when `compute_uv` is True.

s : *(..., K) array*
The singular values for every matrix, sorted in descending order.

v : *{ (..., N, N), (..., K, N) } array*
Unitary matrices. The actual shape depends on the value of `full_matrices`. Only returned when `compute_uv` is True.

Raises :

LinAlgError
If SVD computation does not converge.

Power iteration

The SVD has computational complexity $O(\min(n^2 d, d^2 n))$.

<u>But:</u> We often only need a few largest principle components and not all of them.

Algorithm 6.1 Simple vector iteration or power iteration

```
1: Choose a starting vector $\mathbf{x}^{(0)} \in \mathbb{F}^n$ with $\|\mathbf{x}^{(0)}\| = 1$.
2: $k = 0$.
3: repeat
4:     $k := k + 1$;
5:     $\mathbf{y}^{(k)} := A\mathbf{x}^{(k-1)}$;
6:     $\mu_k := \|\mathbf{y}^{(k)}\|$;
7:     $\mathbf{x}^{(k)} := \mathbf{y}^{(k)}/\mu_k$;
8: until a convergence criterion is satisfied
```

To find the second EV, remember $S = W \Lambda W^{\mathrm{T}} = \sum_{i=1}^{d} \lambda_i w_i w_i^{\mathrm{T}}$.

Thus, do power iteration on $\hat{S} = S - \lambda_1 w_1 w_1^{\mathrm{T}}$.

Power iteration intuition

Assume a diagonalizable matrix A.

The power iteration computes after step k: $A^k x$

where x is a random vector. Write the SVD of A as $A = U\Lambda U^{\mathrm{T}}$. then $A^k = U\Lambda^k U^{\mathrm{T}}$ (because $U^{\mathrm{T}}U = I$).

Transform x into a new coordinate system: $\tilde{x} = U^{\mathrm{T}}x$

This yields:

$$A^k x = U\Lambda U^{\mathrm{T}} x = U\Lambda^k \tilde{x} = \sum_{i=1}^{d}(\lambda_i^k \tilde{x}_i)u_i = \lambda_1^k \sum_{i=1}^{d}(\frac{\lambda_i^k}{\lambda_1^k}\tilde{x}_i)u_i$$

$$\Rightarrow \frac{A^k x}{||A^k x||} \xrightarrow{k\to\infty} \alpha u_1 \text{ where } \alpha \in \mathbb{R}$$

$$\text{because } \frac{\lambda_i}{\lambda_1} \leq 1 \text{ for } \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d \geq 0$$

Under the right conditions (x is linear independent from the first Ev) we thus get a multiple of the first EV.

Application: Eigenfaces

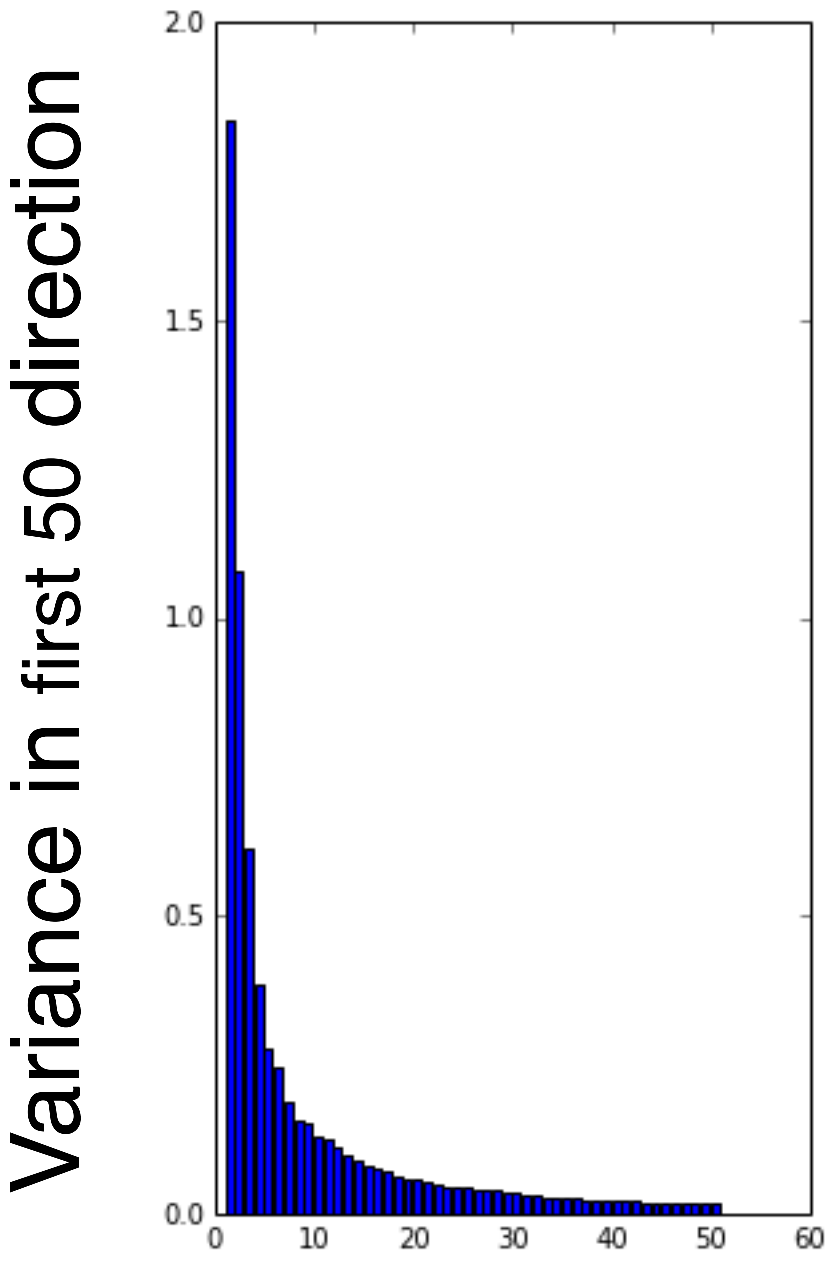
Idea: Faces look very similar in comparison to other random images. How many principle components would we need to accurately describe all faces?

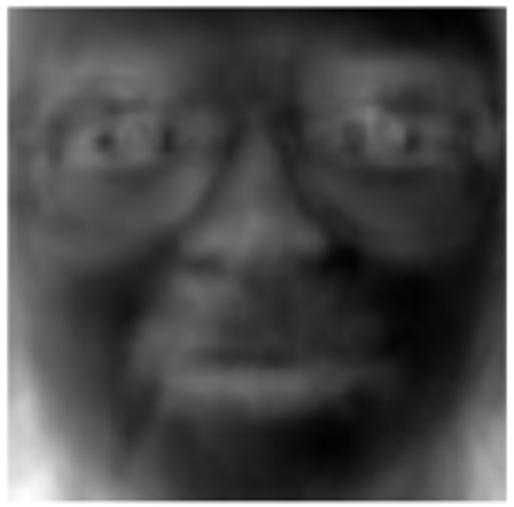
An 64x64 pixel image of a face can be represented as a 4096 dimensional vector where each entry has the pixel's grayscale value.

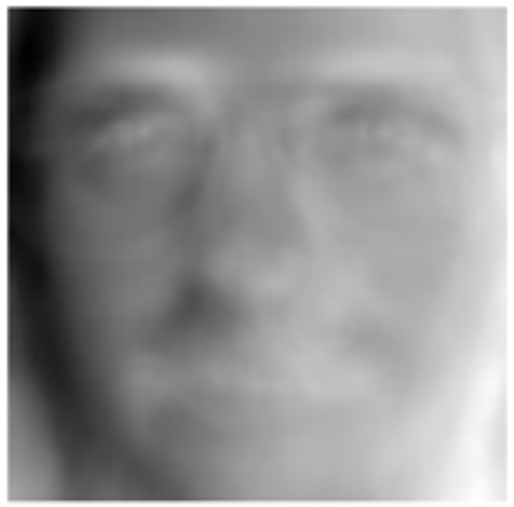
Reference: Turk, Matthew A and Pentland, Alex P. Face recognition using eigenfaces. Computer Vision and Pattern Recognition, 1991.

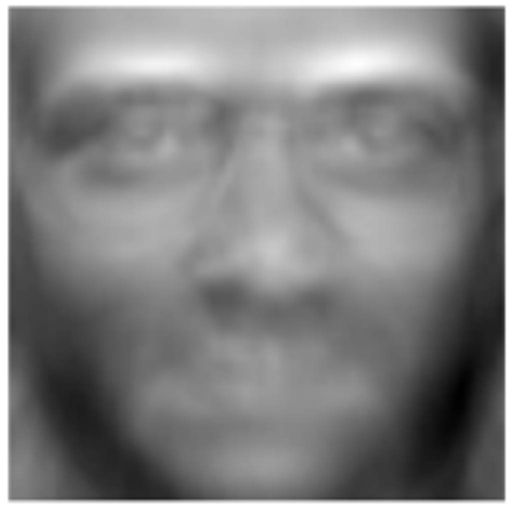
Eigenfaces

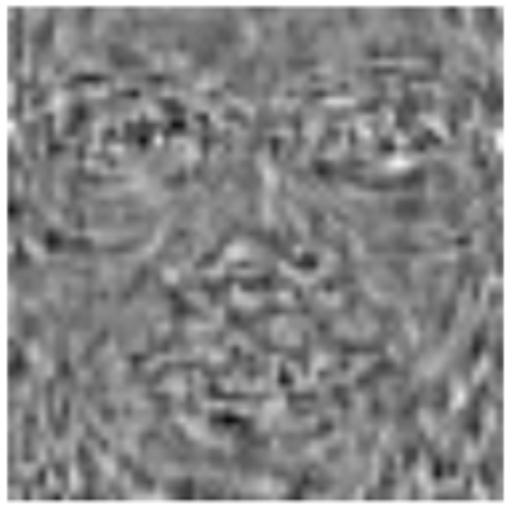
The principle components are directions in our faces space.
Thus, each principle component is a face representation, too.



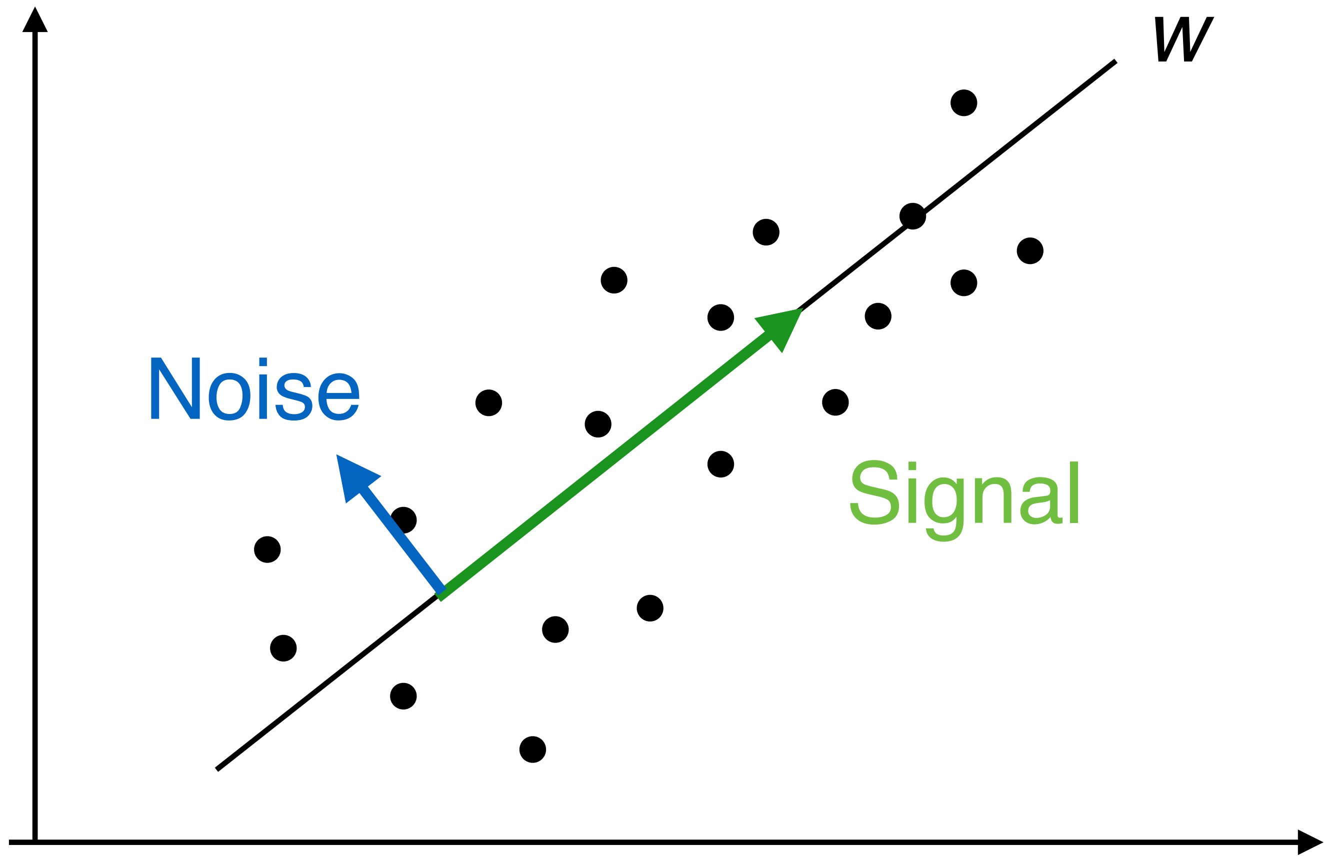

Principle component 1


Principle component 2


Principle component 3


Principle component 350

Application: Dimensionality Reduction



We can reduce the dimensionality of our dataset by projecting on the first k principle components.

How much signal are we going to loose?

$$\underbrace{\sum_{i=k+1}^{d}}_{} \mathrm{Var}(w^{\mathrm{T}}x) = \frac{1}{n} \underbrace{\sum_{i=k+1}^{d}}_{} \lambda_i$$

number of components

Projection on k principle components:

$$\begin{pmatrix} w_1^{\mathrm{T}} x \\ \vdots \\ w_k^{\mathrm{T}} x \end{pmatrix} = \begin{bmatrix} | & & | \\ w_1 & \cdots & w_k \\ | & & | \end{bmatrix}^{\mathrm{T}} x$$

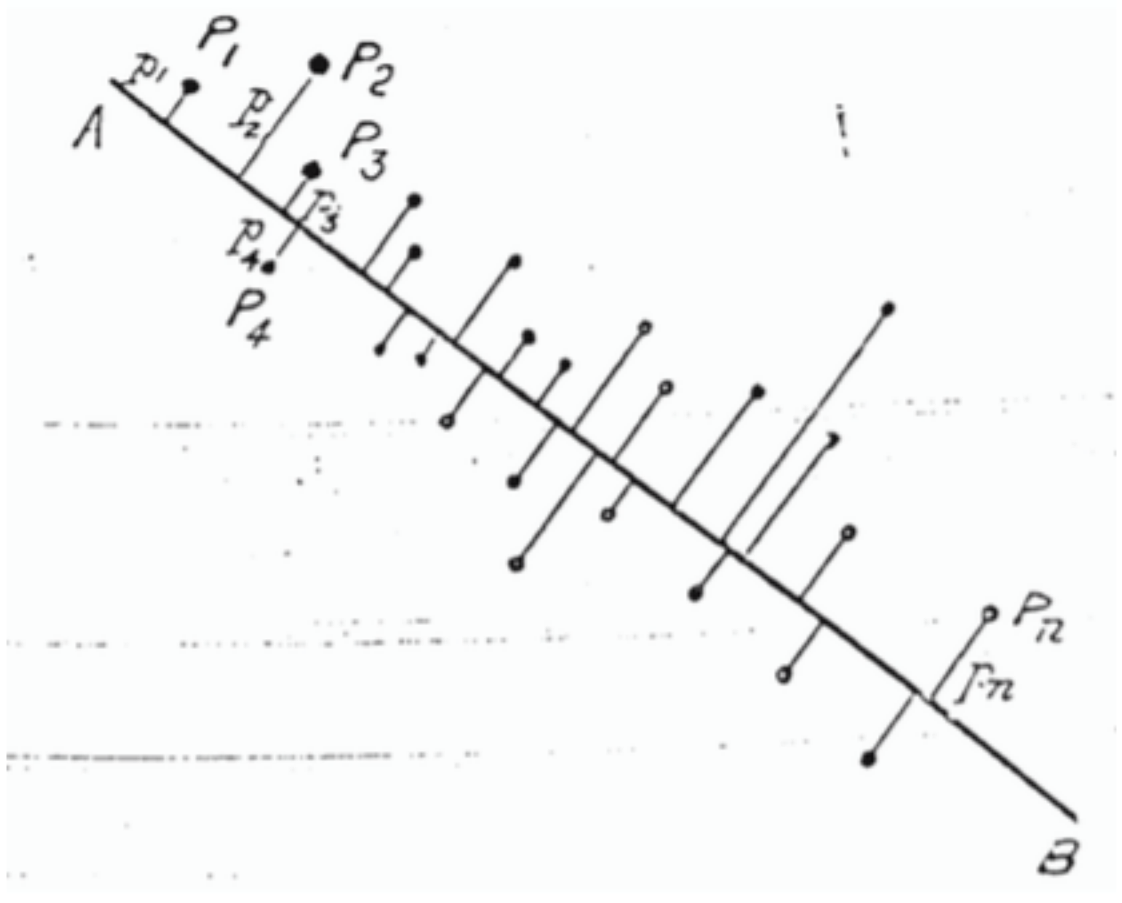
Application: Denoising

We can use PCA to denoise data:

Step 1: Reduce dimensionality to filter out "noise" directions: $(w_1^{\mathrm{T}} x, \ldots, w_k^{\mathrm{T}} x)^{\mathrm{T}}$

Step 2: Project back into original space:

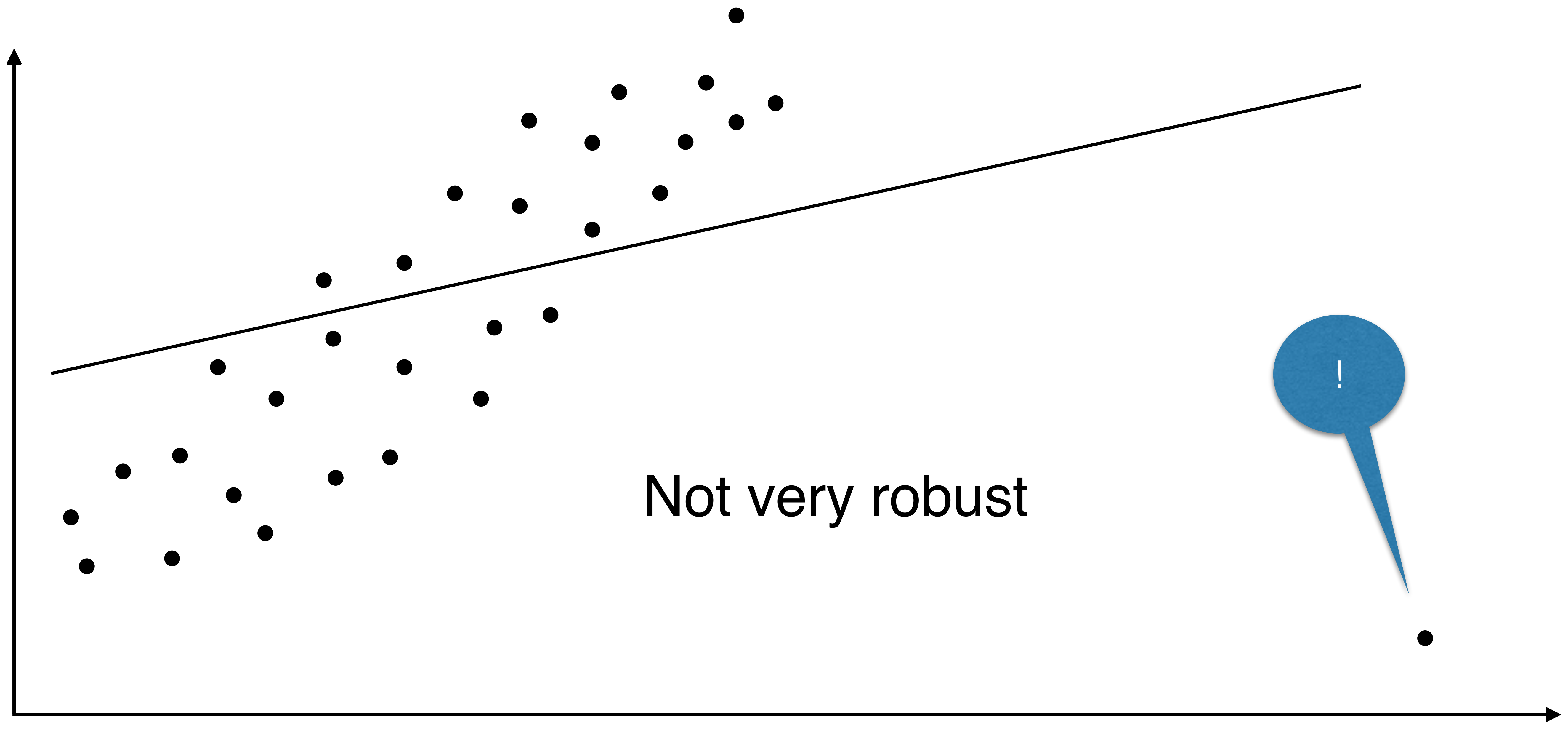
$$\sum_{i=1}^{k} (w_i^{\mathrm{T}} x) w_i$$



Step 3: Undo centering:

$$\sum_{i=1}^{k} (w_i^{\mathrm{T}} x) w_i + \sum_{i=1}^{n} x_i$$

How robust is PCA to outliers?



Not very robust