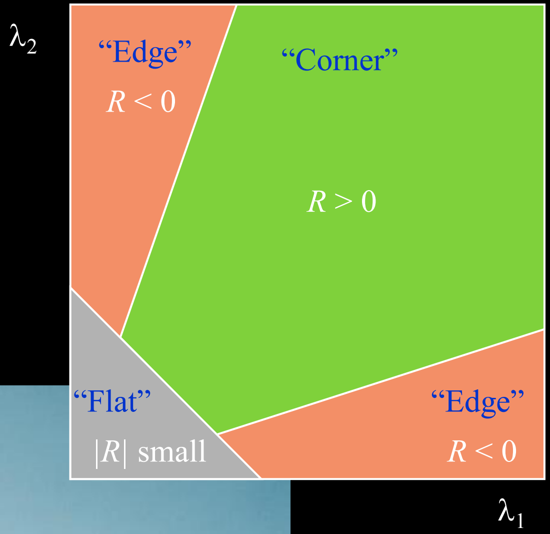
# CS4495/6495
# Introduction to Computer Vision

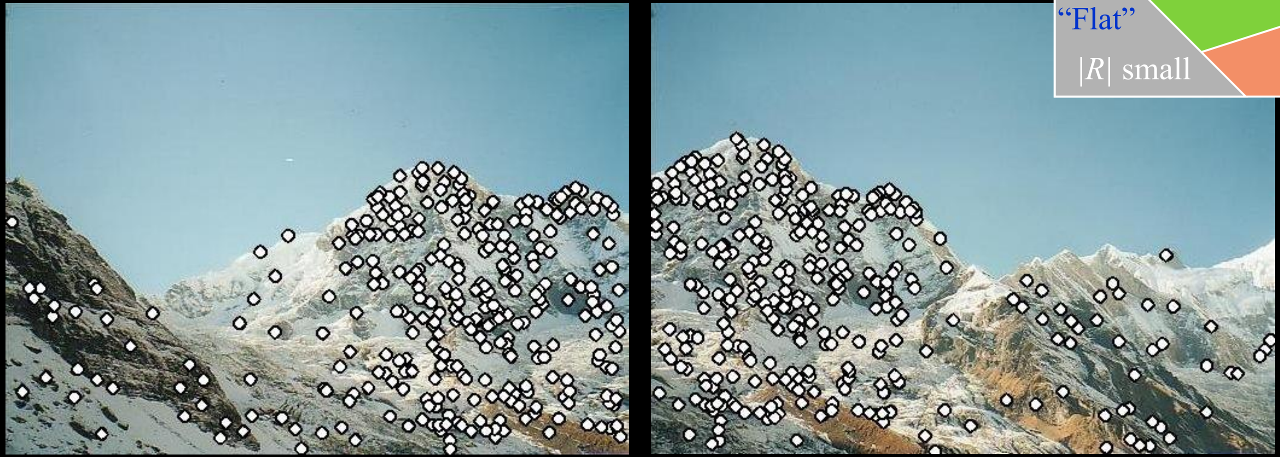
## 4C-L1 *Robust error functions*

# Feature-based alignment to find transforms

Overall strategy:

1. Compute features – detect and describe

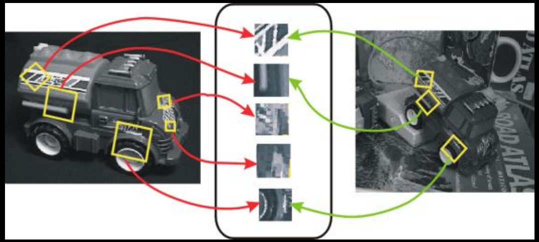


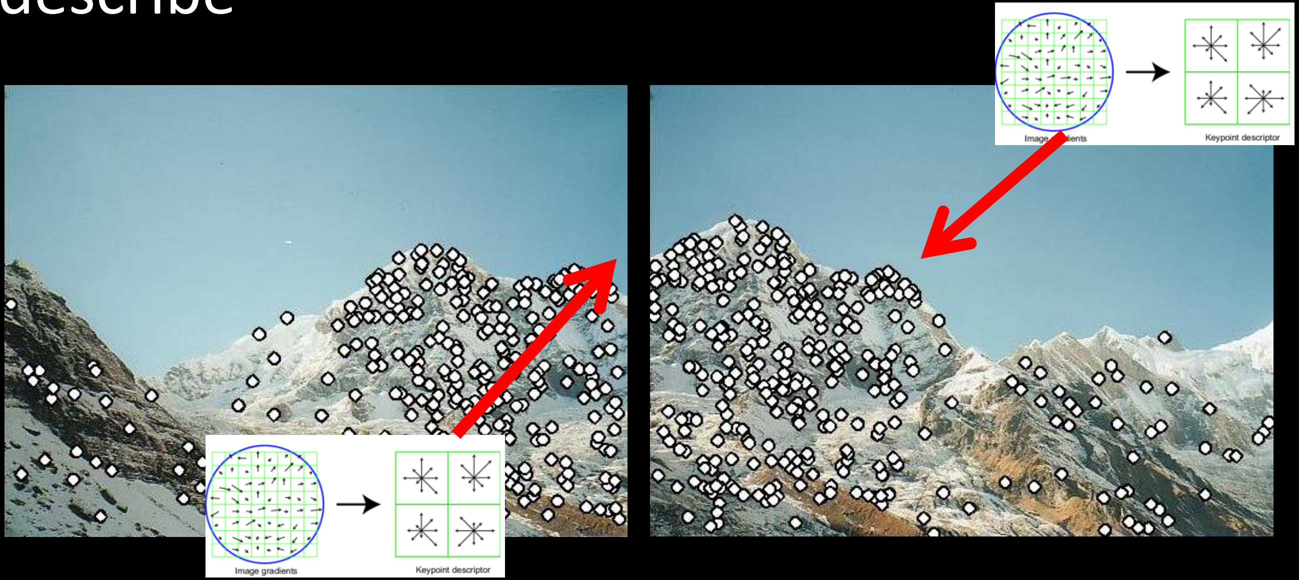



# Feature-based alignment to find transforms

Overall strategy:

1. Compute features – detect and describe



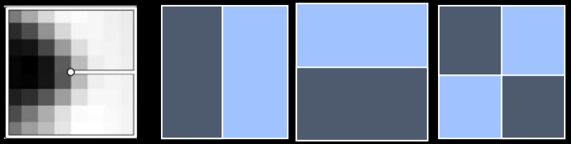


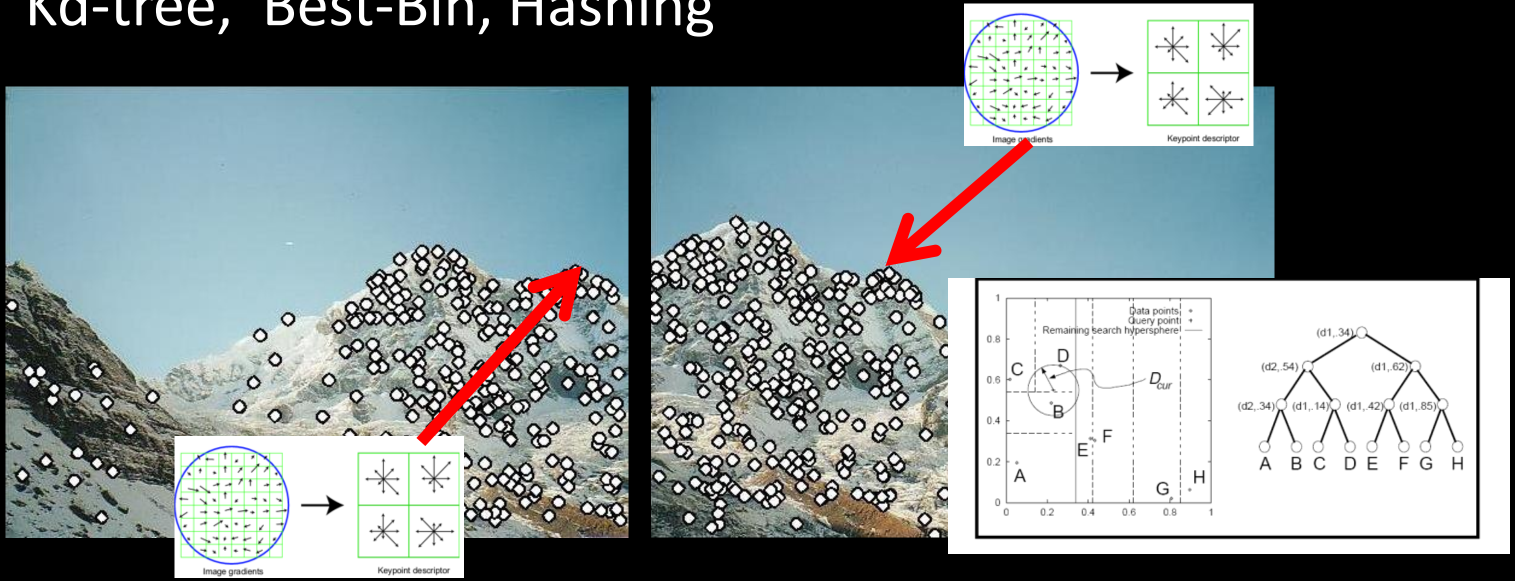
# Feature-based alignment to find transforms

## Overall strategy:

2. Find some useful matches:

   Kd-tree,  Best-Bin, Hashing



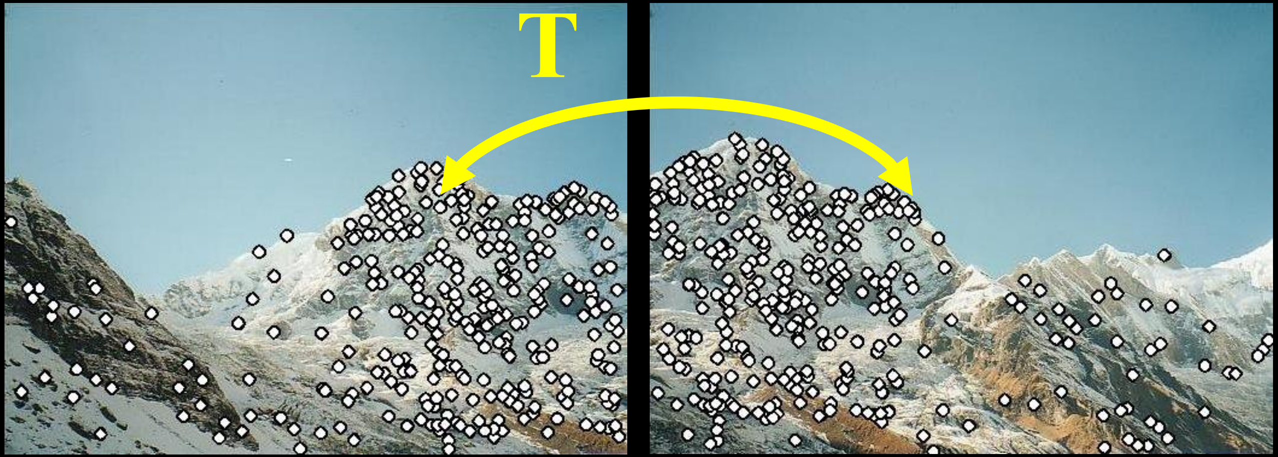


# Feature-based alignment to find transforms

Overall strategy:

3. Compute and apply the best transformation:

   *e.g. affine, translation, or homography*



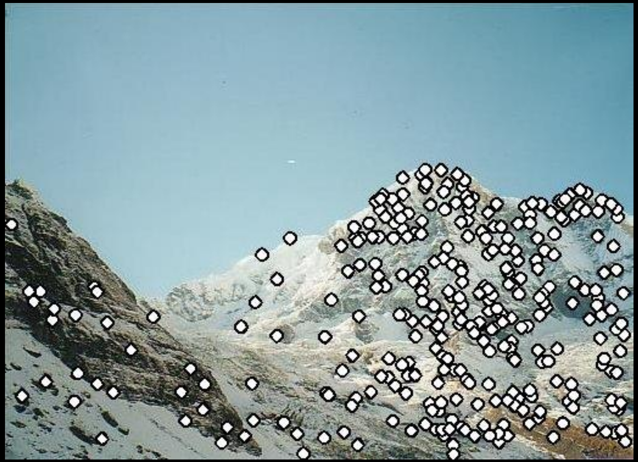
# Feature-based alignment to find transforms

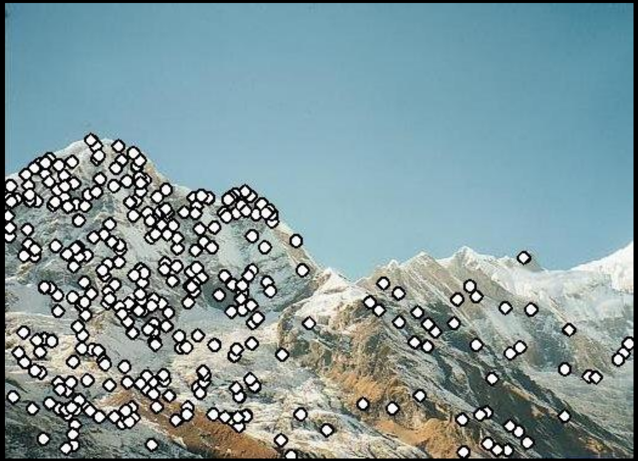
Overall strategy:

3. Compute and apply the best transformation:

   *e.g. affine, translation, or homography*

# Feature-based alignment algorithm





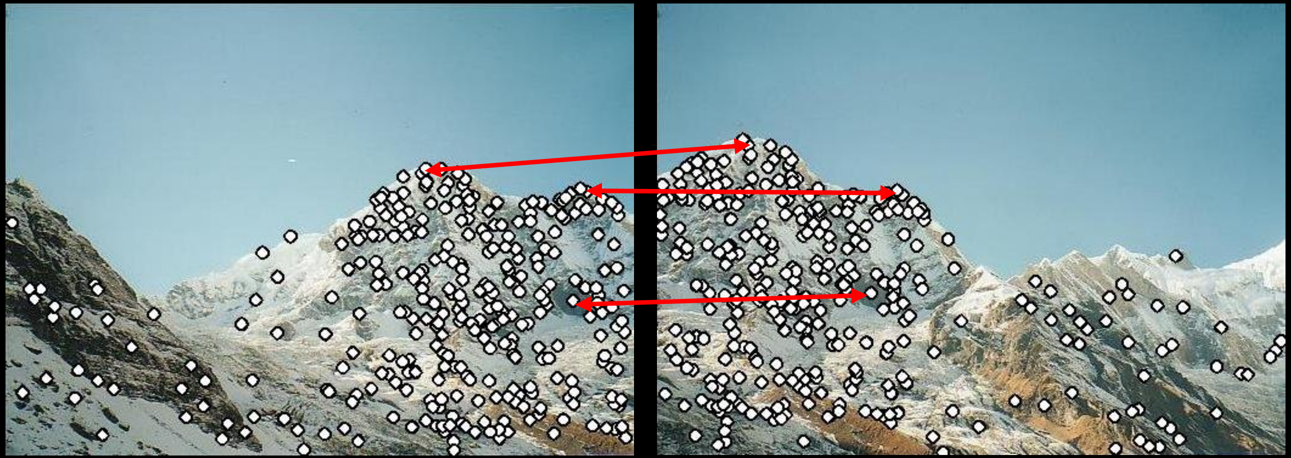
1. Extract features

# Feature-based alignment algorithm

2. Compute ***putative*** *matches* – e.g. “closest descriptor”

   *Kd-tree, best bin, etc…*

# Feature-based alignment algorithm



3. Loop until happy:
   - *Hypothesize* transformation *T* from some matches
   - *Verify* transformation (search for other matches consistent with *T*) – mark best

# Feature-based alignment algorithm

4. Apply best transformation.

# How to get “putative” matches?

# Feature matching

- Exhaustive search
- Hashing
- Nearest neighbor techniques

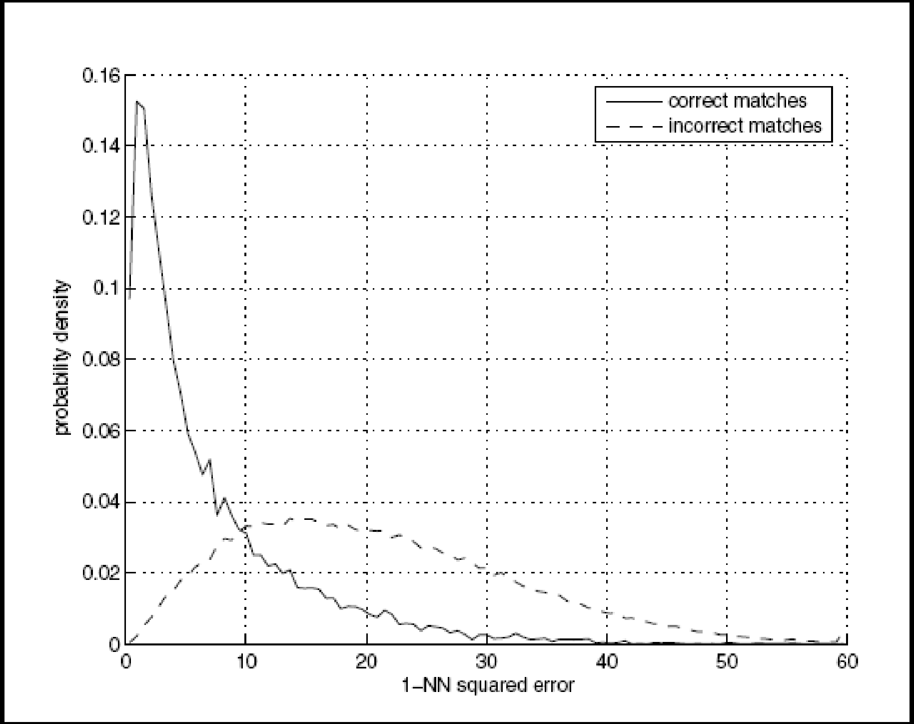
…. but these give the best match. How do we know it's a good one?

# Feature-space outlier rejection

- Let’s not match all features, but only these that have “similar enough” matches?
- How can we do it?
  - SSD(patch1,patch2) < threshold
  - How to set threshold?

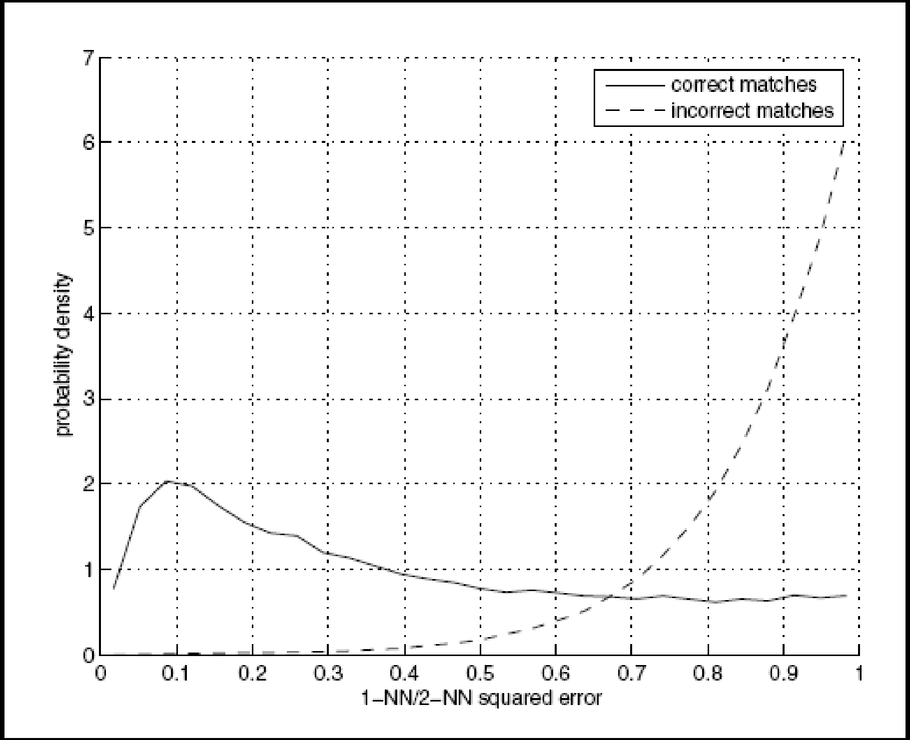
# Feature-space outlier rejection

- How to set threshold?



# A better way [Lowe, 1999]:

- 1-NN: SSD of the closest match
- 2-NN: SSD of the second-closest match
- Look at how much better 1-NN is than 2-NN, e.g. 1-NN/2-NN



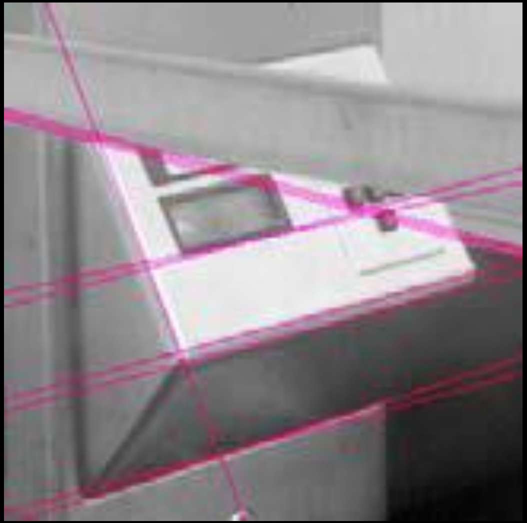
# Feature matching

- Exhaustive search
- Hashing
- Nearest neighbor techniques
- But…remember the distinctive vs invariant competition? Implies:

- *Problem: Even when pick best match, still lots (and lots) of wrong matches – "outliers". What should we do?*

# Model Fitting

- Choose a parametric model to represent a set of features – *remember this???*



simple model: lines

simple model: circles

# Fitting: Issues

Case study: Line detection

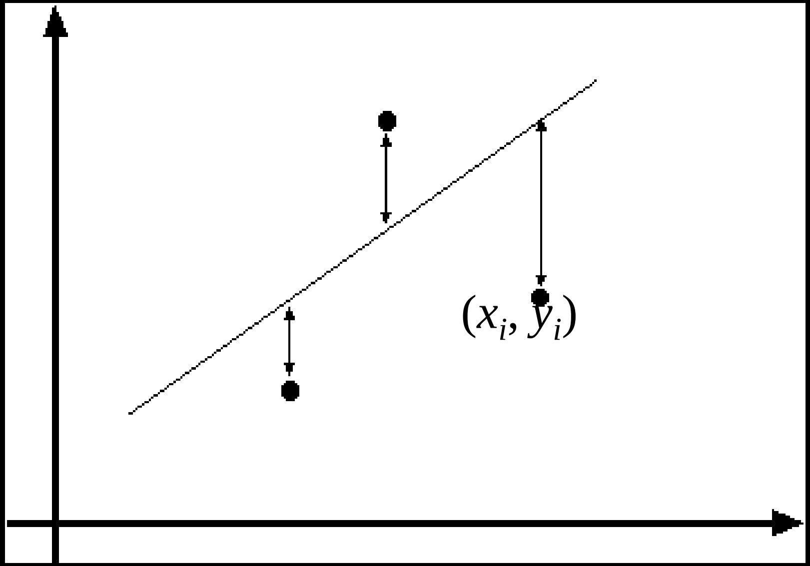
- **Noise** in the measured feature locations
- **Extraneous data**: clutter (outliers), multiple lines
- **Missing data**: occlusions

Slide: S. Lazebnik

# Typical least squares line fitting

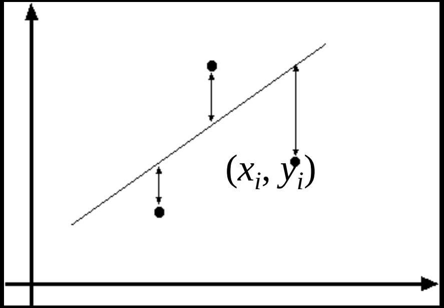
- Data: $(x_1, y_1), \ldots, (x_n, y_n)$
- Line equation: $y_i = m\,x_i + b$
- Find $(m, b)$ to minimize:

$$E = \sum_{i=1}^{n} (y_i - m\,x_i - b)^2$$



# Typical least squares line fitting

$$E=\sum_{i=1}^{n}(y_i-mx_i-b)^2$$




$$E=\sum_{i=1}^{n}\left(y_i-\begin{bmatrix}x_i & 1\end{bmatrix}\begin{bmatrix}m\\ b\end{bmatrix}\right)^2=\left\|\begin{bmatrix}y_1\\ \vdots\\ y_n\end{bmatrix}-\begin{bmatrix}x_1 & 1\\ \vdots & \vdots\\ x_n & 1\end{bmatrix}\underbrace{\begin{bmatrix}m\\ b\end{bmatrix}}_{\mathbf{h}}\right\|^2=\|\mathbf{y}-\mathbf{X}\mathbf{h}\|^2$$

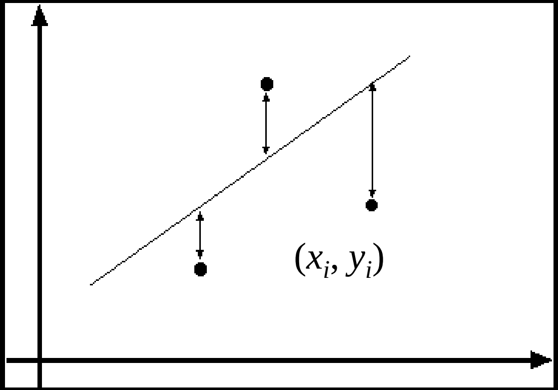
$$E=(\mathbf{y}-\mathbf{X}\mathbf{h})^T(\mathbf{y}-\mathbf{X}\mathbf{h})=\mathbf{y}^T\mathbf{y}-2(\mathbf{X}\mathbf{h})^T\mathbf{y}+(\mathbf{X}\mathbf{h})^T(\mathbf{X}\mathbf{h})$$

# Typical least squares line fitting

$$E = (\mathbf{y} - \mathbf{X}\mathbf{h})^T(\mathbf{y} - \mathbf{X}\mathbf{h}) = \mathbf{y}^T\mathbf{y} - 2(\mathbf{X}\mathbf{h})^T\mathbf{y} + (\mathbf{X}\mathbf{h})^T(\mathbf{X}\mathbf{h})$$

$$\Rightarrow \frac{dE}{d\mathbf{h}} = 2\mathbf{X}^T\mathbf{X}\mathbf{h} - 2\mathbf{X}^T\mathbf{y} = 0$$

$$\mathbf{X}^T\mathbf{X}\mathbf{h} = \mathbf{X}^T\mathbf{y} \Rightarrow \mathbf{h} = \underbrace{(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T}_{\textit{pseudoinverse}}\mathbf{y}$$



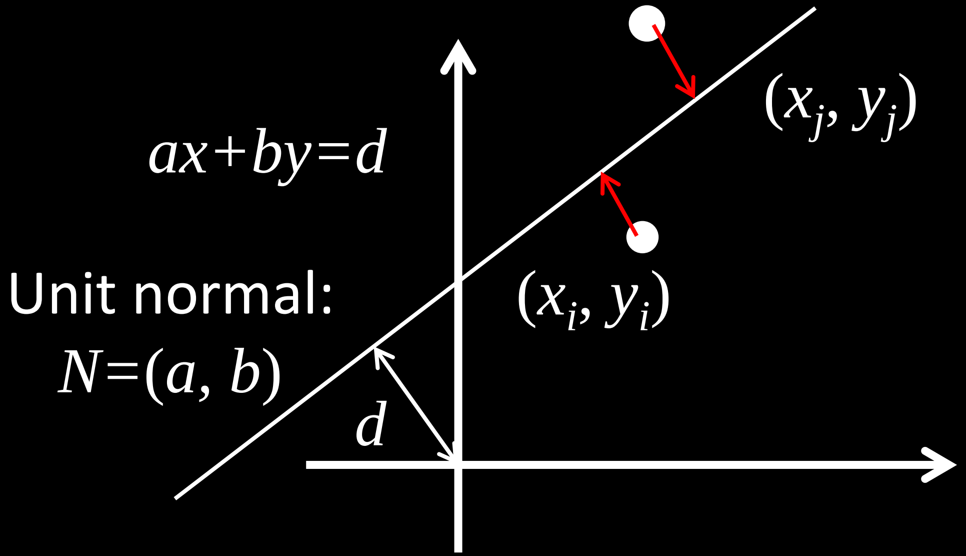

*Standard over-constrained least squares solution*

# Problem with “vertical” least squares

- Not rotation-invariant
- Fails completely for vertical lines

# Total least squares

- Distance between point $(x_i, y_i)$ and line $ax + by = d$

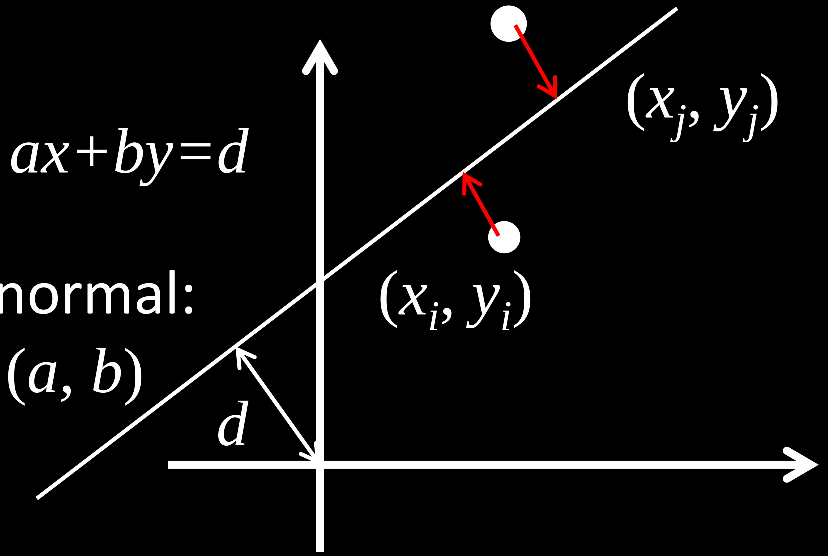



- Find (a, b, d) to minimize the sum of squared *perpendicular* distances

$$E = \sum_{i=1}^{n} (a x_i + b y_i - d)^2$$

# Total least squares

$$E = \sum_{i=1}^{n}(a x_i + b y_i - d)^2$$



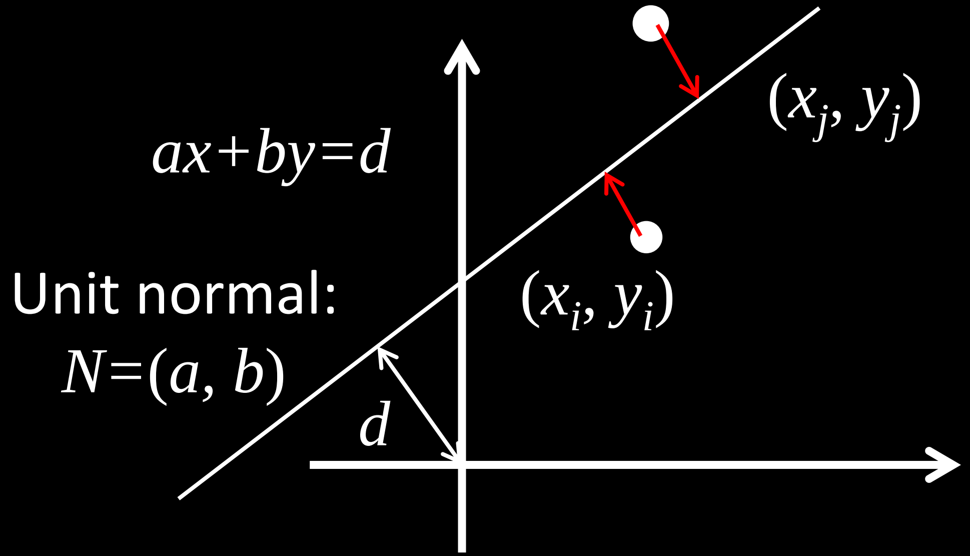

$$\frac{\partial E}{\partial d} = \sum_{i=1}^{n} -2(a x_i + b y_i - d) = 0$$

$$\Rightarrow \; d = \frac{a}{n}\sum_{i=1}^{n} x_i + \frac{b}{n}\sum_{i=1}^{n} x_i = a\bar{x} + b\bar{y}$$

# Total least squares

$$E = \sum_{i=1}^{n} (a x_i + b y_i - d)^2$$

$$d = a\bar{x} + b\bar{y}$$




$$E = \sum_{i=1}^{n} (a(x_i - \bar{x}) + b(y_i - \bar{y}))^2 = \left\| \begin{bmatrix} x_1 - \bar{x} & y_1 - \bar{y} \\ \vdots & \vdots \\ x_n - \bar{x} & y_n - \bar{y} \end{bmatrix} \underbrace{\begin{bmatrix} a \\ b \end{bmatrix}}_{\mathbf{h}} \right\|^2 = (U\mathbf{h})^T(U\mathbf{h})$$

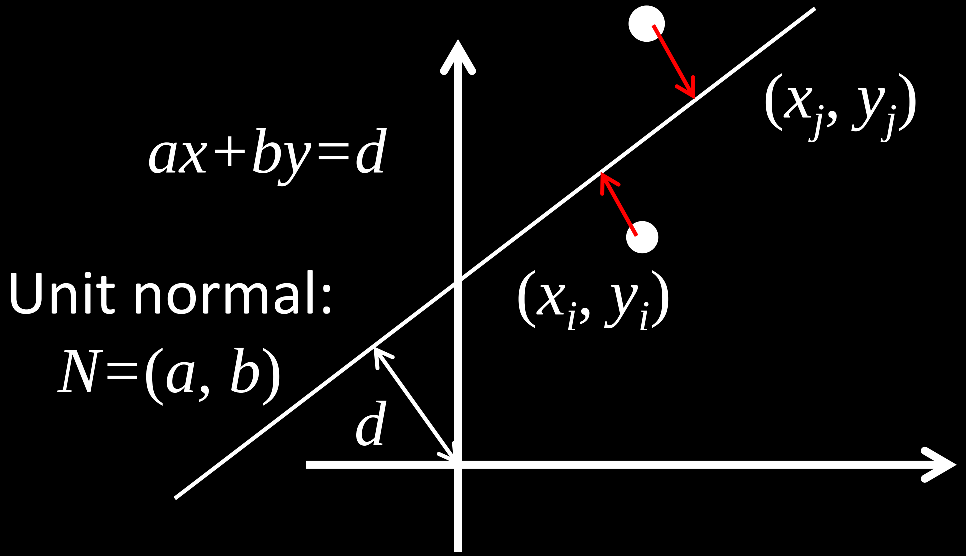
$$\frac{dE}{d\mathbf{h}} = 2(U^T U)\mathbf{h} = 0$$

# Total least squares

$$E = \sum_{i=1}^{n} (a x_i + b y_i - d)^2$$

$$d = a\bar{x} + b\bar{y}$$

$$\frac{dE}{d\mathbf{h}} = 2(U^T U)\mathbf{h} = 0$$



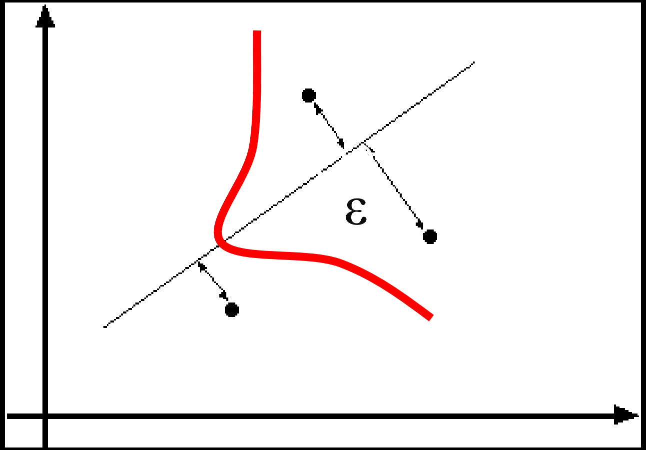

Solution to $(U^T U)\mathbf{h} = 0$, subject to $||\mathbf{h}||^2 = 1$:

eigenvector of $U^T U$ associated with the smallest eigenvalue

(Again SVD to least squares solution to *homogeneous linear system)*

# Least squares as likelihood maximization

- **Generative model**: Line points are corrupted by Gaussian noise in the direction perpendicular to the line



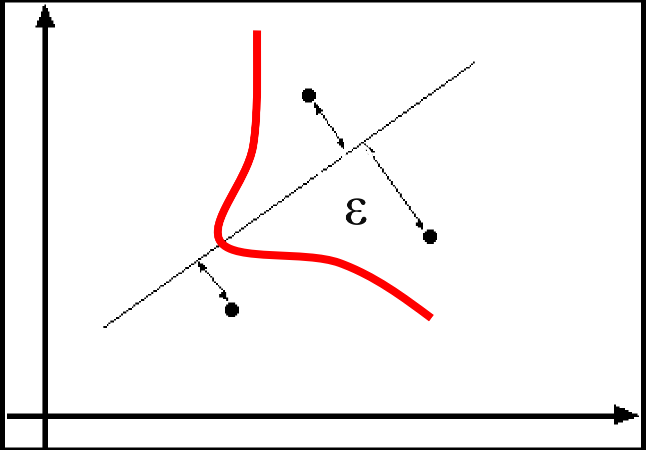
# Least squares as likelihood maximization

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} u \\ v \end{pmatrix} + \varepsilon \begin{pmatrix} a \\ b \end{pmatrix}$$

point on the line

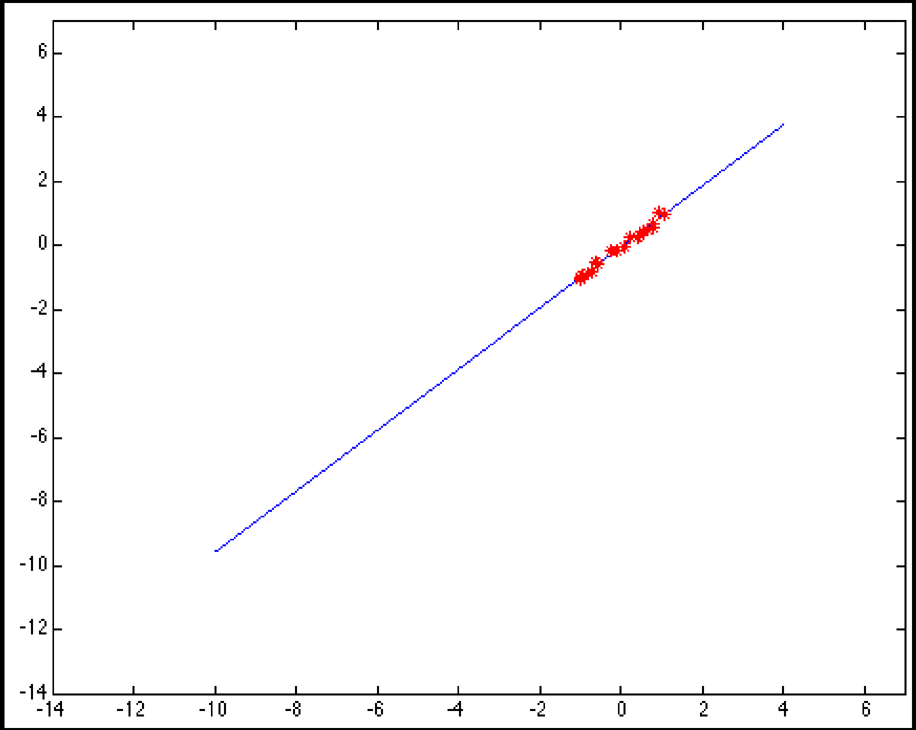
noise: sampled from zero-mean Gaussian with std. dev. σ

normal direction



# Least squares: Non-robustness to (very) non-Gaussian noise

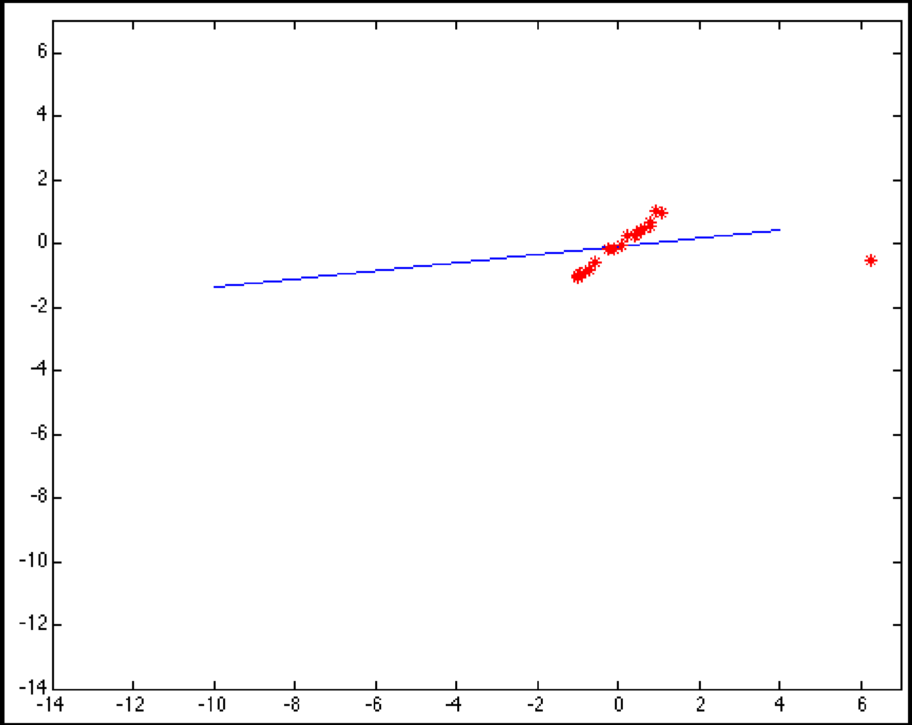
- Least squares fit to the red points:



# Least squares: Non-robustness to (very) non-Gaussian noise

- Least squares fit with an outlier…

Problem: *squared error heavily penalizes outliers*



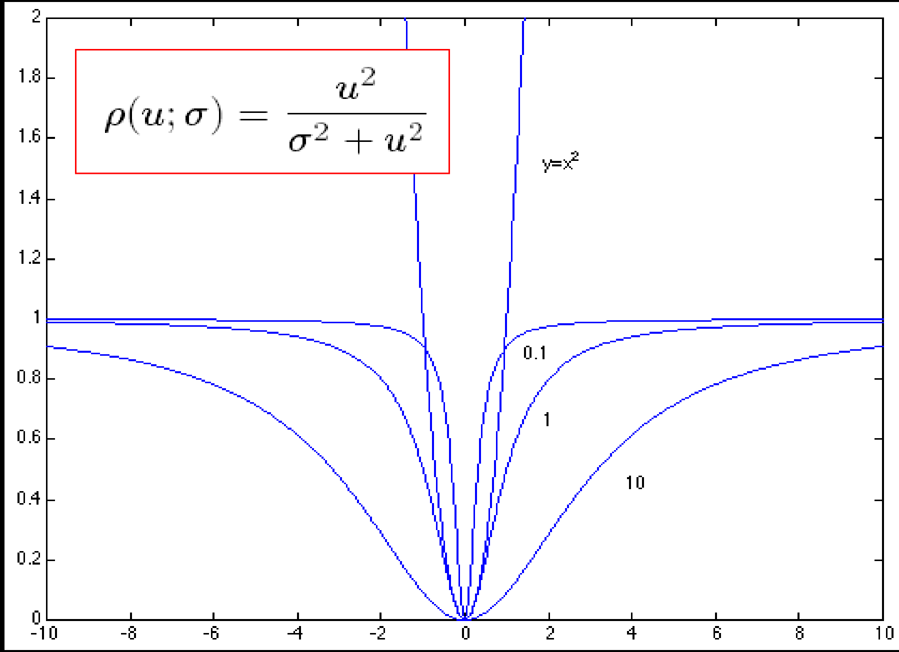
# Robust estimators

General approach: minimize $\sum_i \rho\left(r_i\left(x_i, \theta\right); \sigma\right)$

$r_i(x_i, \theta)$ – residual of $i^{th}$ point w.r.t. model parameters $\theta$

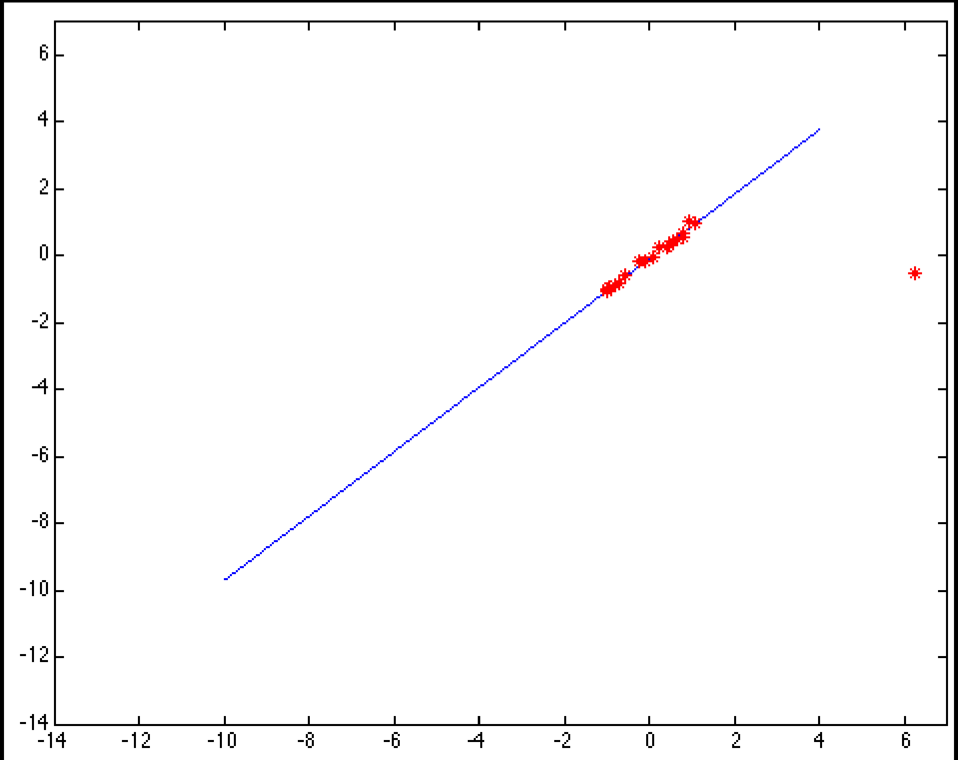
$\rho$ – robust function with scale parameter $\sigma$

# Robust estimators



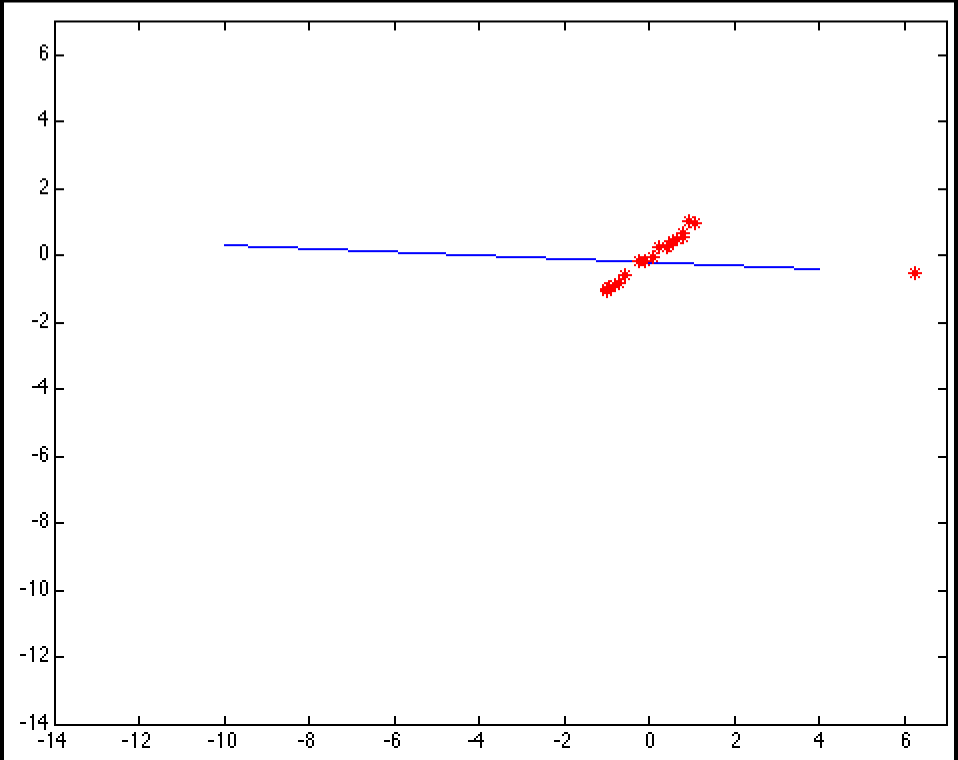

The robust function $\rho$ behaves like squared distance for small values of the residual $u$ but saturates for larger values of $u$

# Choosing the scale: Just right

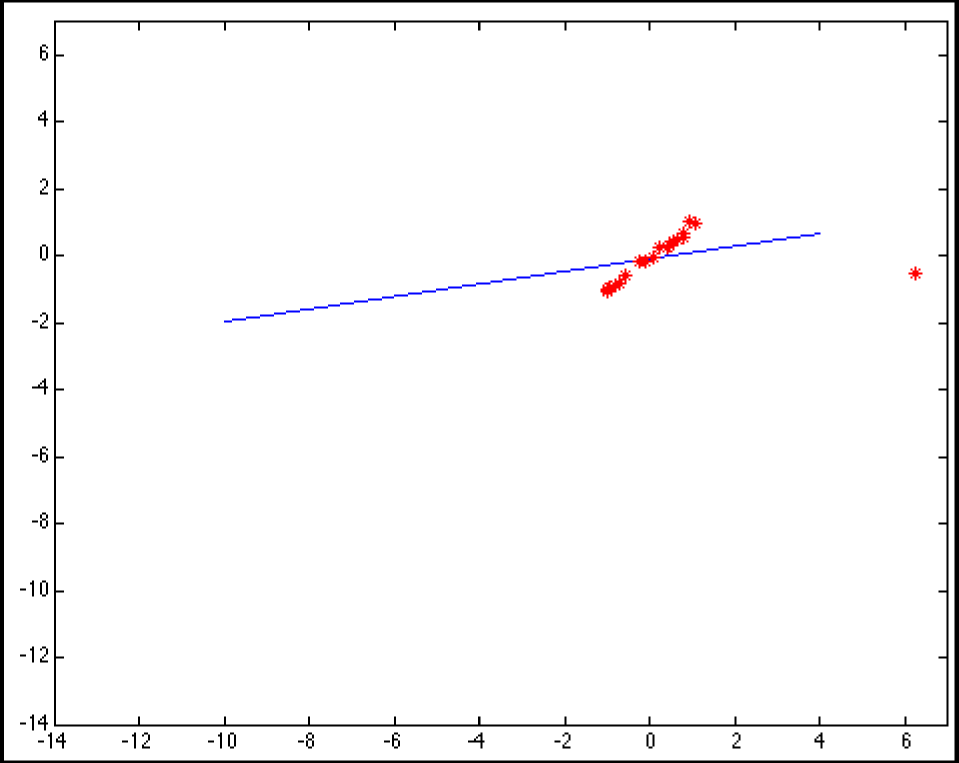



The effect of the outlier is minimized

# Choosing the scale: Too small



# Choosing the scale: Too large




*Behaves much the same as least squares*