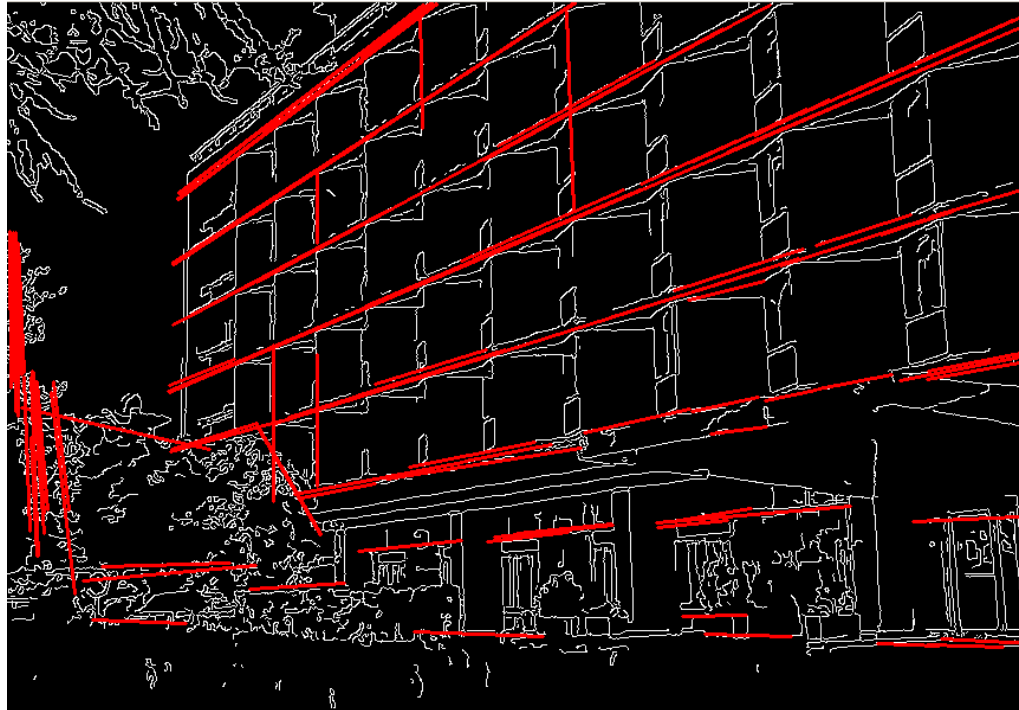


# Fitting

---



# Fitting

---

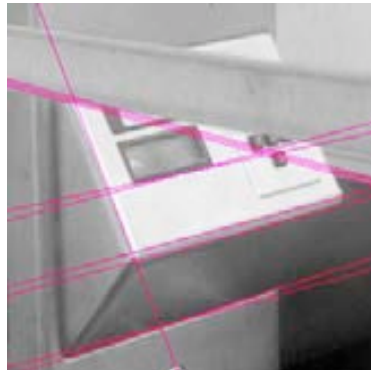
- We've learned how to detect edges, corners, blobs. Now what?
- We would like to form a higher-level, more compact representation of the features in the image by grouping multiple features according to a simple model



# Fitting

---

- Choose a *parametric model* to represent a set of features



simple model: lines



simple model: circles



complicated model: car

Source: K. Grauman

# Fitting: Challenges

---

## Case study: Line detection



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

## Fitting: Overview

---

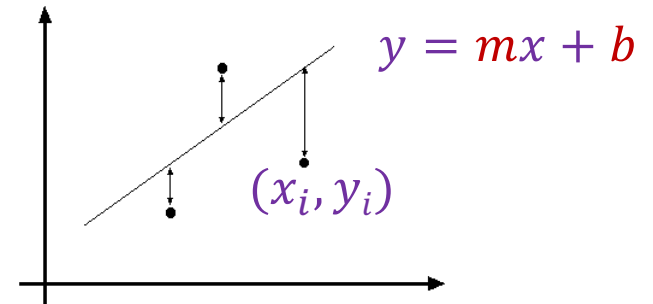
- Least squares line fitting
- Robust fitting
- RANSAC

## Least squares line fitting: First attempt

---

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

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



- Equivalent to finding least-squares solution to:

$$\begin{matrix} \begin{bmatrix} x_1 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{bmatrix} & \begin{pmatrix} m \\ b \end{pmatrix} & = & \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \\ X & B & & Y \end{matrix}$$

- Solution is given by  $X^T X B = X^T Y$

## Is this a good solution?

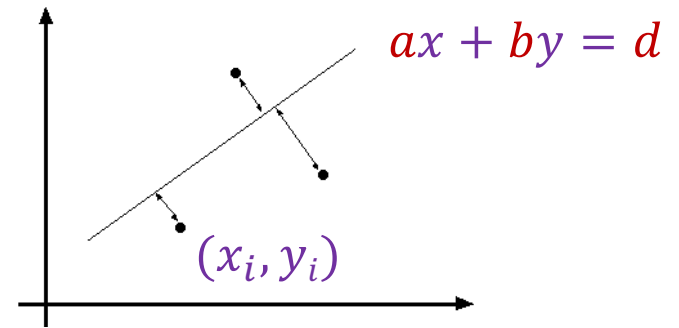
---

- Slope-intercept parametrization fails for vertical lines
- Solution is not equivariant w.r.t. rotation

# Total least squares

---

- Line parametrization:  $ax + by = d$ 
  - $(a, b)$  is the *unit normal* to the line (i.e.,  $a^2 + b^2 = 1$ )
  - $d$  is the distance between the line and the origin



- Perpendicular distance between point  $(x_i, y_i)$  and line  $ax + by = d$  (assuming  $a^2 + b^2 = 1$ ):  
 $|ax_i + by_i - d|$

- Objective function:

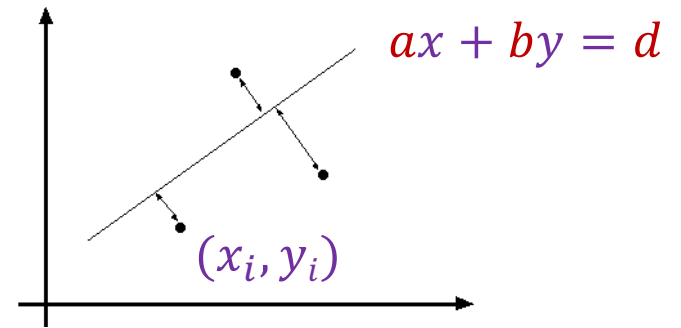
$$E = \sum_{i=1}^n (ax_i + by_i - d)^2$$



# Total least squares

---

$$E = \sum_{i=1}^n (ax_i + by_i - d)^2$$



- Solve for  $d$  first:

$$\frac{\partial E}{\partial d} = -2 \sum_{i=1}^n (ax_i + by_i - d) = 0$$

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

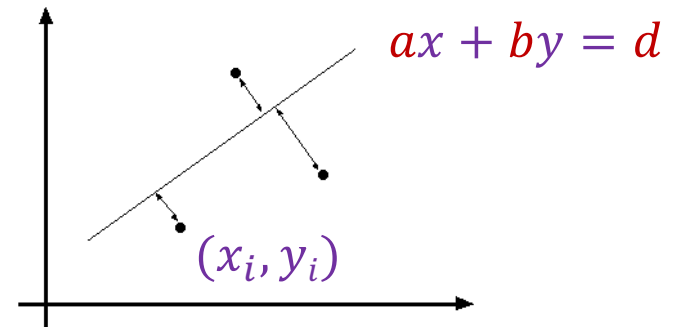
# Total least squares

---

$$E = \sum_{i=1}^n (ax_i + by_i - d)^2$$

- Solve for  $d$  first:  $d = a\bar{x} + b\bar{y}$
- Plugging back in:

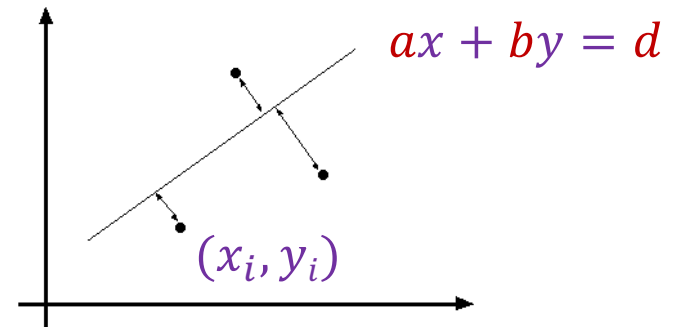
$$E = \sum_{i=1}^n (a(x_i - \bar{x}) + b(y_i - \bar{y}))^2$$



# Total least squares

---

$$E = \sum_{i=1}^n (ax_i + by_i - d)^2$$



- Solve for  $d$  first:  $d = a\bar{x} + b\bar{y}$
- Plugging back in:

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

$\underbrace{\hspace{10em}}_U \quad \underbrace{\hspace{2em}}_N$

- We want to find  $N$  that minimizes  $\|UN\|^2$  subject to  $\|N\|^2 = 1$ 
  - Solution is given by the eigenvector of  $U^T U$  associated with the smallest eigenvalue

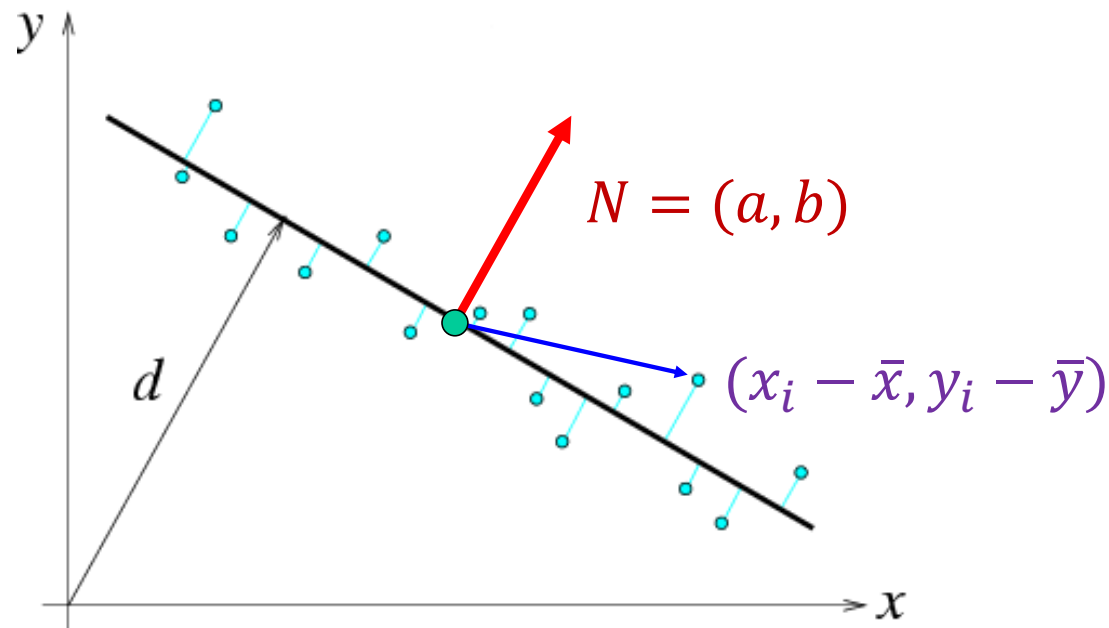
# Total least squares

---

$$U = \begin{bmatrix} x_1 - \bar{x} & y_1 - \bar{y} \\ \vdots & \vdots \\ x_n - \bar{x} & y_n - \bar{y} \end{bmatrix}$$

$$U^T U = \begin{bmatrix} \sum_{i=1}^n (x_i - \bar{x})^2 & \sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y}) \\ \sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y}) & \sum_{i=1}^n (y_i - \bar{y})^2 \end{bmatrix}$$

second moment matrix



## Application: Computing surface normals

---

Assume at every pixel you have distance to the 3D point (depth)

write:  $d(i, j)$

You would like to know the surface normal at each point.

Strategy:

- fit a plane to the a group of points in 3D

- use the normal of that plane

## Fitting a plane in 3D (Very much like fitting a line in 2D)

---

- Plane parametrization:  $ax + by + cz = d$ 
  - $(a, b, c)$  is the *unit normal* to the plane (i.e.,  $a^2 + b^2 + c^2 = 1$ )
  - $d$  is the distance between the line and the origin
- Perpendicular distance between point  $(x_i, y_i, z_i)$  and plane  $ax + by + cz = d$  (assuming  $a^2 + b^2 + c^2 = 1$ ):
$$|ax_i + by_i + cz_i - d|$$

- Objective function:

$$E = \sum_{i=1}^n (ax_i + by_i + cz_i - d)^2$$

## Total least squares

---

$$E = \sum_{i=1}^n (ax_i + by_i + cz_i - d)^2$$

- Solve for  $d$  first:  $d = a\bar{x} + b\bar{y} + c\bar{z}$
- etc
  - Solution is given by the eigenvector of  $U^T U$  associated with the smallest eigenvalue
  - But now  $U$  is 3x3

## Points close to line $\rightarrow$ TLS behaves well

---

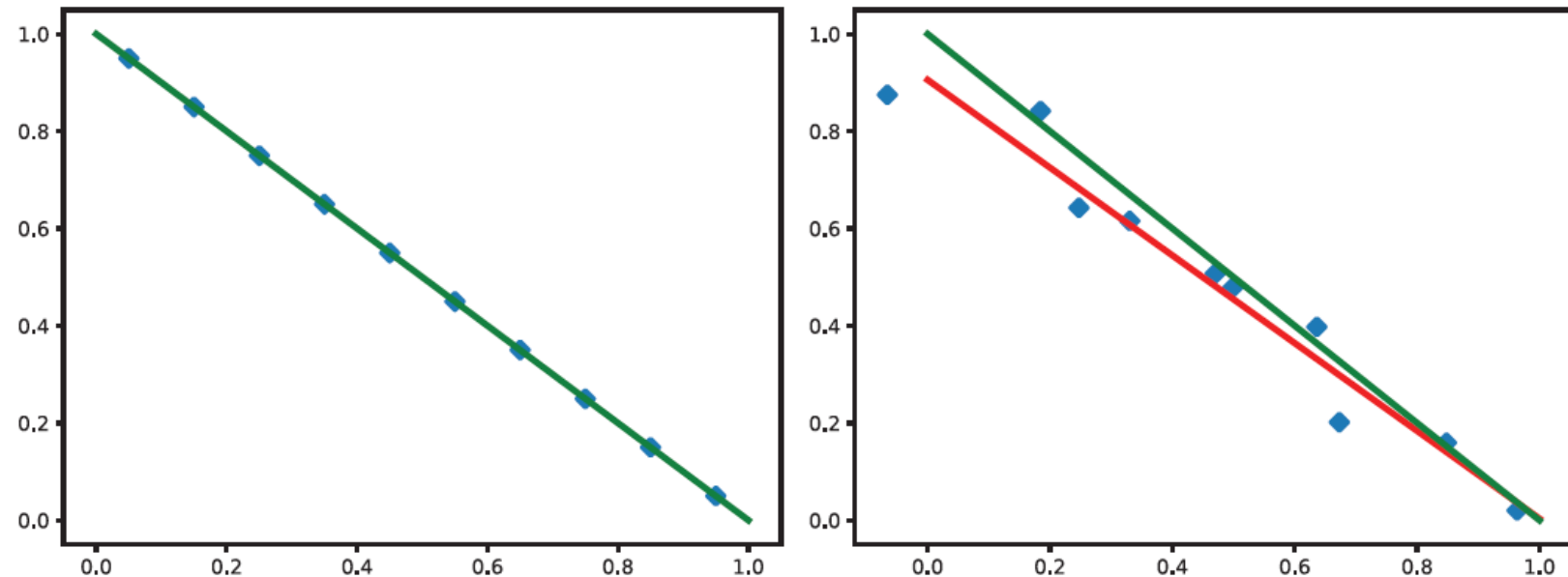


FIGURE 12.4: *Total least squares behaves well when all points lie fairly close to the line. **Left:** A line fitted with total least squares to a set of points that actually lie on the line. **Right:** The points have been perturbed by adding a small random term to both  $x$  and  $y$  coordinate, but the fitted line (red) is not that different from the original (green).*



## Large errors – TLS behaves badly

---

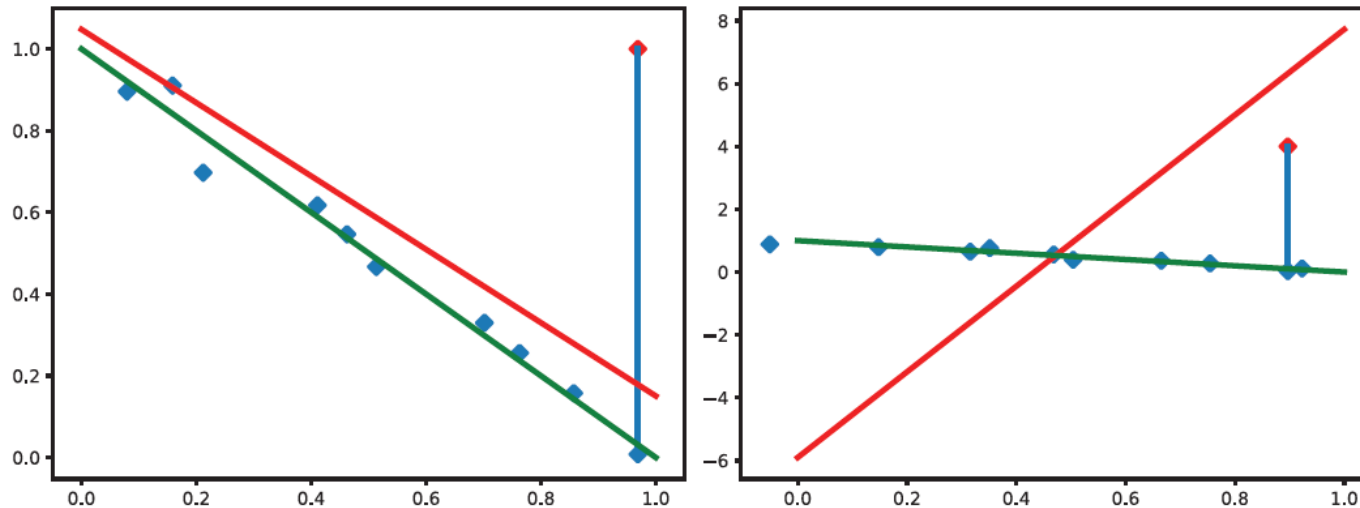
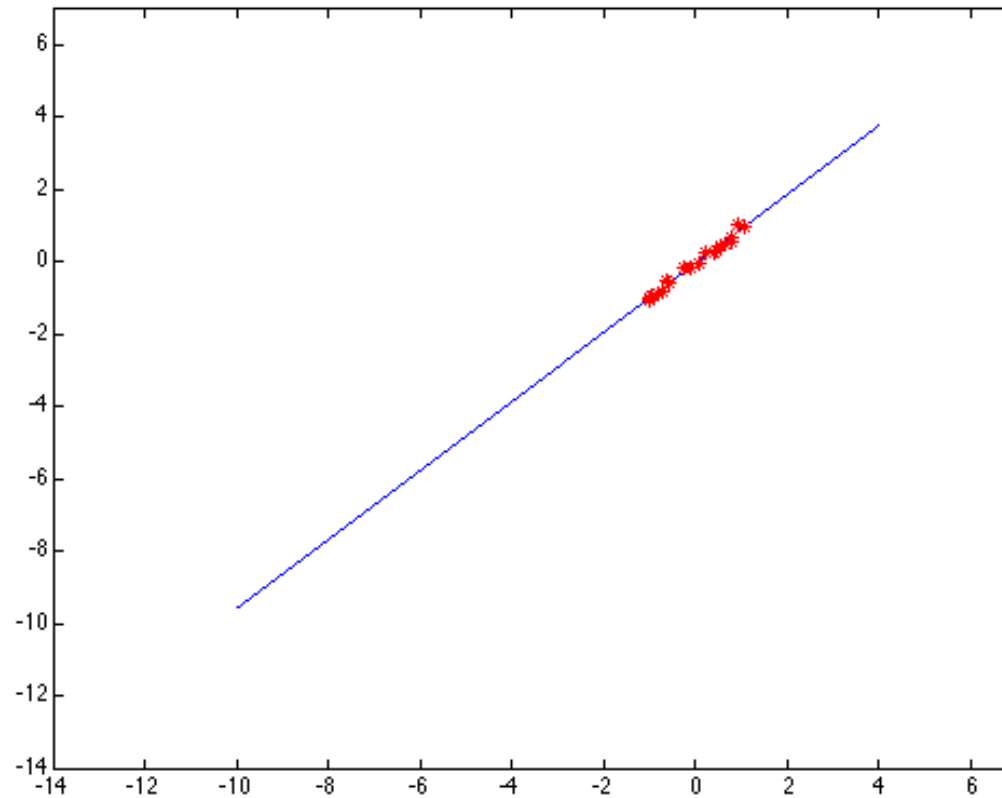


FIGURE 12.5: *Total least squares can misbehave when some points are significantly perturbed. Here the points are those of Figure 12.4, but one point has had its y coordinate replaced with a constant (new point in red, joined to its original position with a line). **Left:** The constant is 1, and the line produced by total least squares has shifted significantly. **Right:** The constant is 4, and now the fitted line is completely wrong. In each case, the original line is in green (and passes close to the points), and the new line is red.*

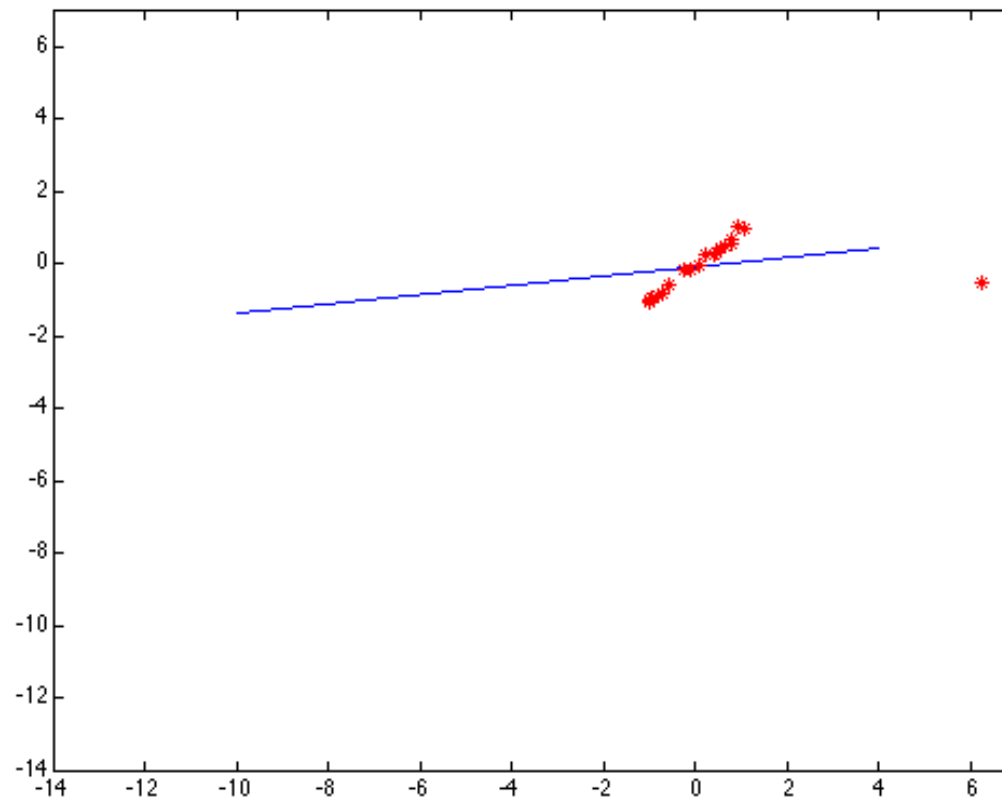
- Least squares fit to the red points:



# Least squares: Robustness to noise

---

- Least squares fit with an outlier:



Problem: squared error heavily penalizes outliers

# Robust estimators

---

- General approach: find model parameters  $\theta$  that minimize

$$\sum_i \rho_\sigma(r(x_i; \theta))$$

$r(x_i; \theta)$ : residual of  $x_i$  w.r.t. model parameters  $\theta$

eg for line,  $\theta = (a, b, d)$

residual  $r(x_i; \theta) = (ax_i + by_i - d)$

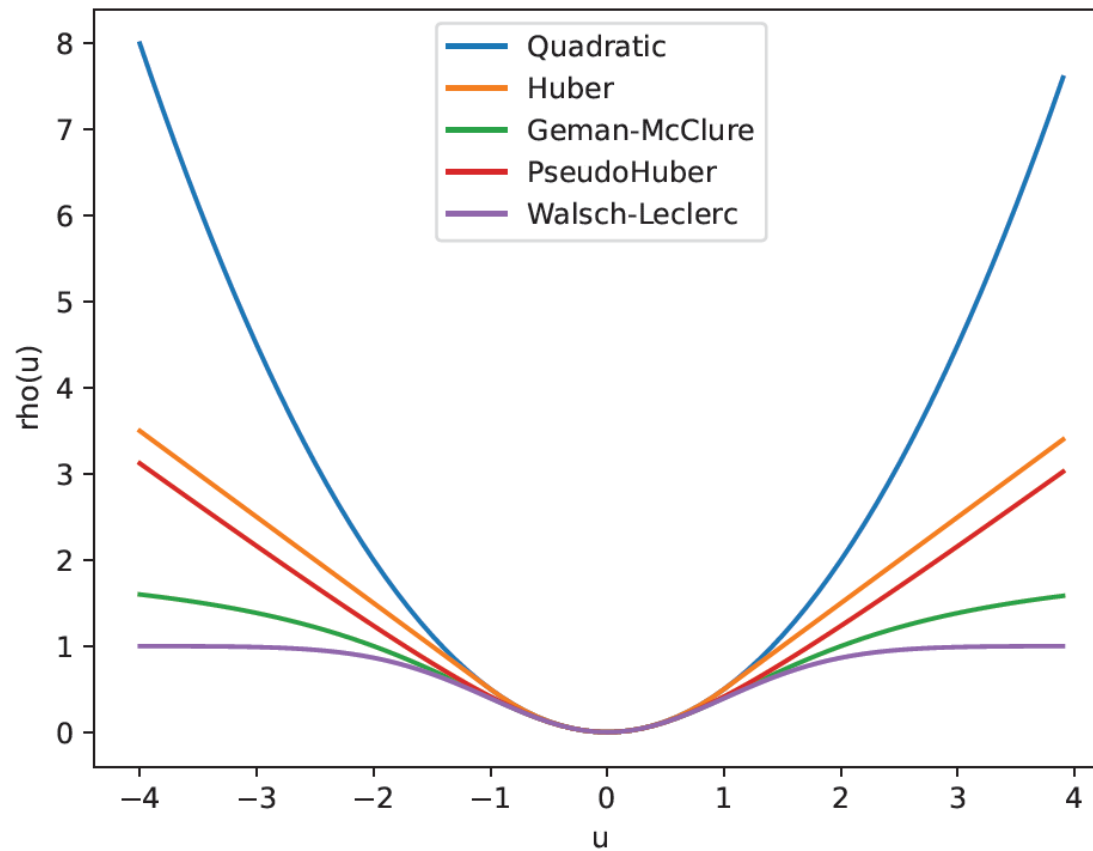
$\rho_\sigma$ : robust function with scale parameter  $\sigma$

Notice that  $\rho_\sigma(u) = u^2$

would give the original least squares loss

# Robust estimators

---




# The Huber loss

---

The *Huber loss* uses

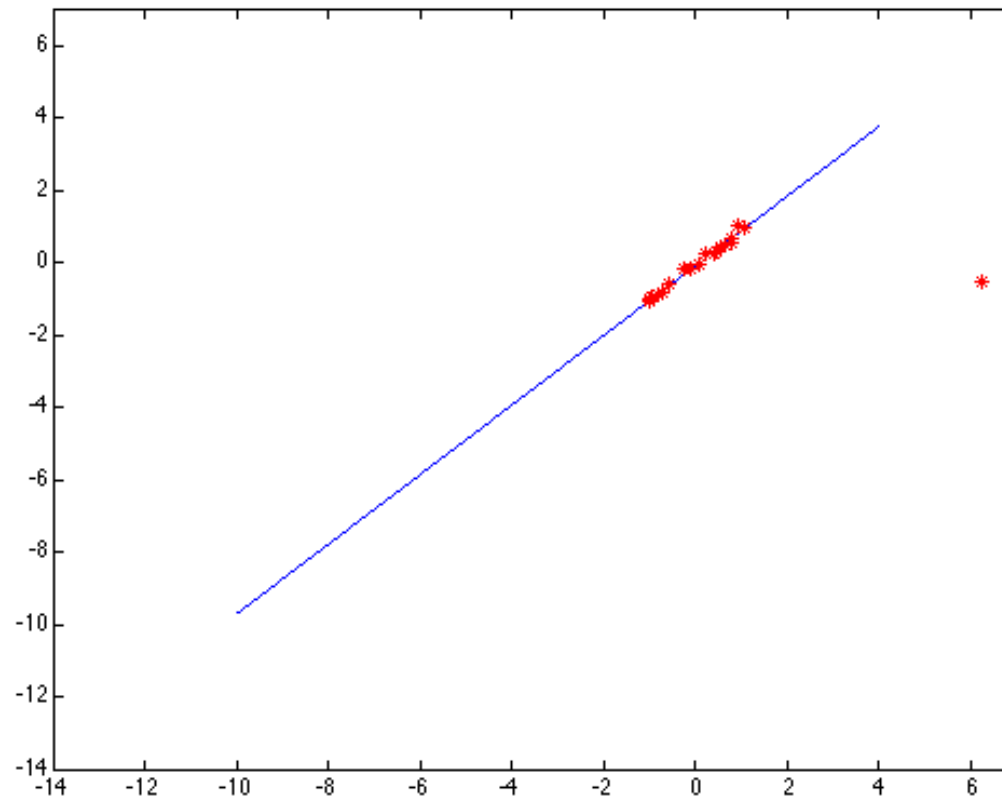
$$\rho(u; \sigma) = \begin{cases} \frac{u^2}{2} & |u| < \sigma \\ \sigma|u| - \frac{\sigma^2}{2} & |u| \geq \sigma \end{cases}$$

**Scale** 

which is the same as  $u^2/2$  for  $-\sigma \leq u \leq \sigma$ , switches to  $|u|$  for larger (or smaller)  $\sigma$ , and has continuous derivative at the switch. The Huber loss is convex (meaning that there will be a unique minimum for our models) and differentiable, but is not smooth. The choice of the parameter  $\sigma$  (which is known as *scale*) has an effect on the estimate. You should interpret this parameter as the distance that a point can

## Choosing the scale: Just right

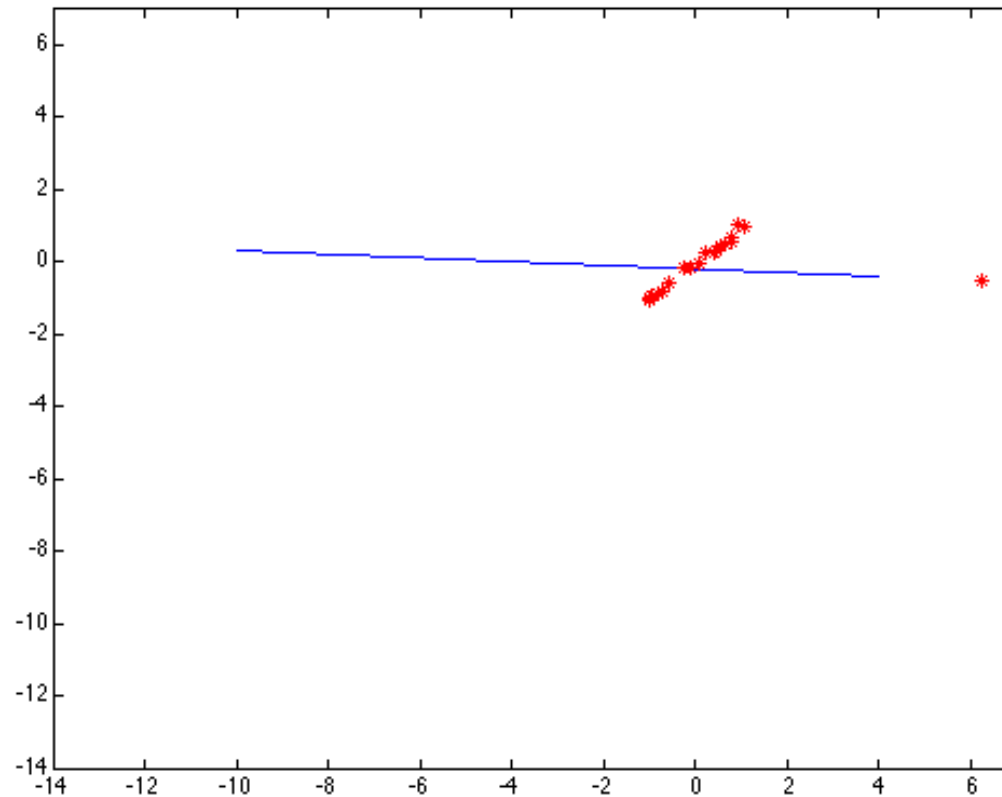
---



The effect of the outlier is minimized

## Choosing the scale: Too small

---

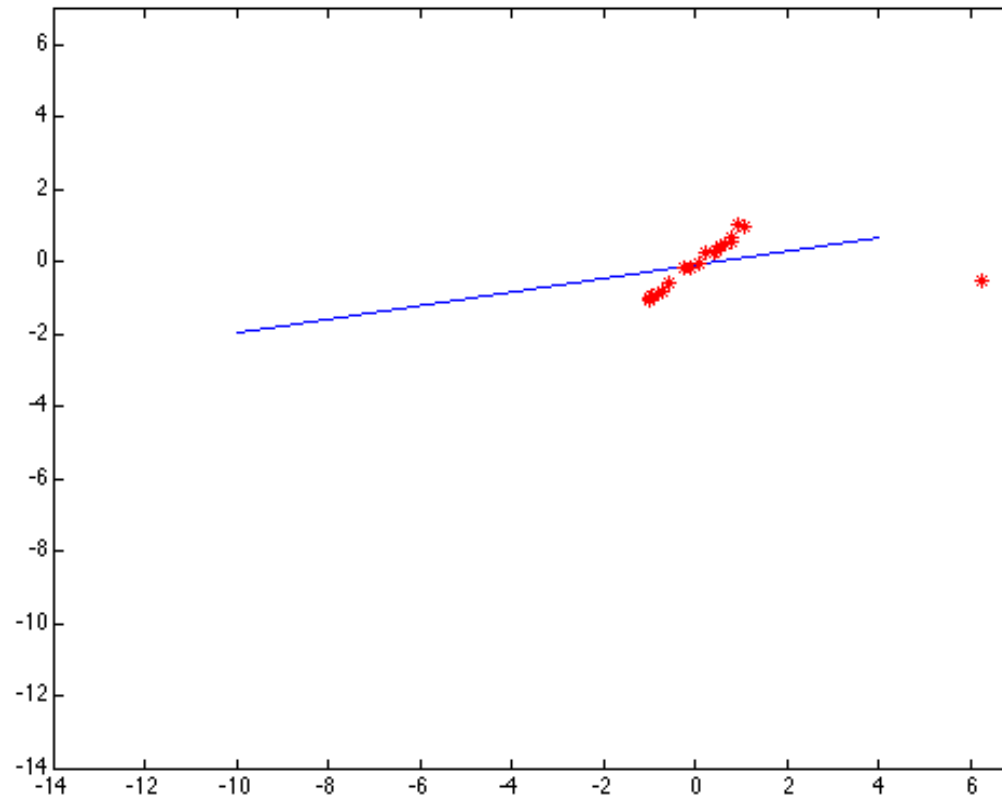


The error value is almost the same for every point and the fit is very poor



## Choosing the scale: Too large

---



Behaves much the same as least squares

## Finding the line

---

Now the line is chosen by minimizing

$$(1/2) \sum_i \rho(r(\mathbf{x}, \theta); \sigma)$$

with respect to  $\theta = (a_1, a_2, c)$ , subject to  $a_1^2 + a_2^2 = 1$ . The minimum occurs when

$$\begin{aligned} \nabla_{\theta} \left( \sum_i \rho(r(\mathbf{x}_i, \theta); \sigma) \right) &= \sum_i \left[ \frac{\partial \rho}{\partial u} \right] \nabla_{\theta} r(\mathbf{x}_i, \theta) \\ &= \lambda \begin{pmatrix} a_1 \\ a_2 \\ 0 \end{pmatrix}. \end{aligned}$$

## Finding the line - II

---

Here  $\lambda$  is a Lagrange multiplier and the derivative  $\frac{\partial \rho}{\partial u}$  is evaluated at  $r(\mathbf{x}_i, \theta)$ , so it is a function of  $\theta$ . Now notice that

$$\begin{aligned}\sum_i \left[ \frac{\partial \rho}{\partial u} \right] \nabla_{\theta} r(\mathbf{x}_i, \theta) &= \sum_i \left[ \left( \frac{\frac{\partial \rho}{\partial u}}{r(\mathbf{x}_i, \theta)} \right) \right] r(\mathbf{x}_i, \theta) \nabla_{\theta} r(\mathbf{x}_i, \theta) \\ &= \sum_i \left[ \left( \frac{\frac{\partial \rho}{\partial u}}{r(\mathbf{x}_i, \theta)} \right) \right] \nabla_{\theta} [(1/2)r(\mathbf{x}_i, \theta)]^2\end{aligned}$$

Now  $[r(\mathbf{x}_i, \theta)]^2$  is the squared error. At the true minimum  $\hat{\theta}$ , writing

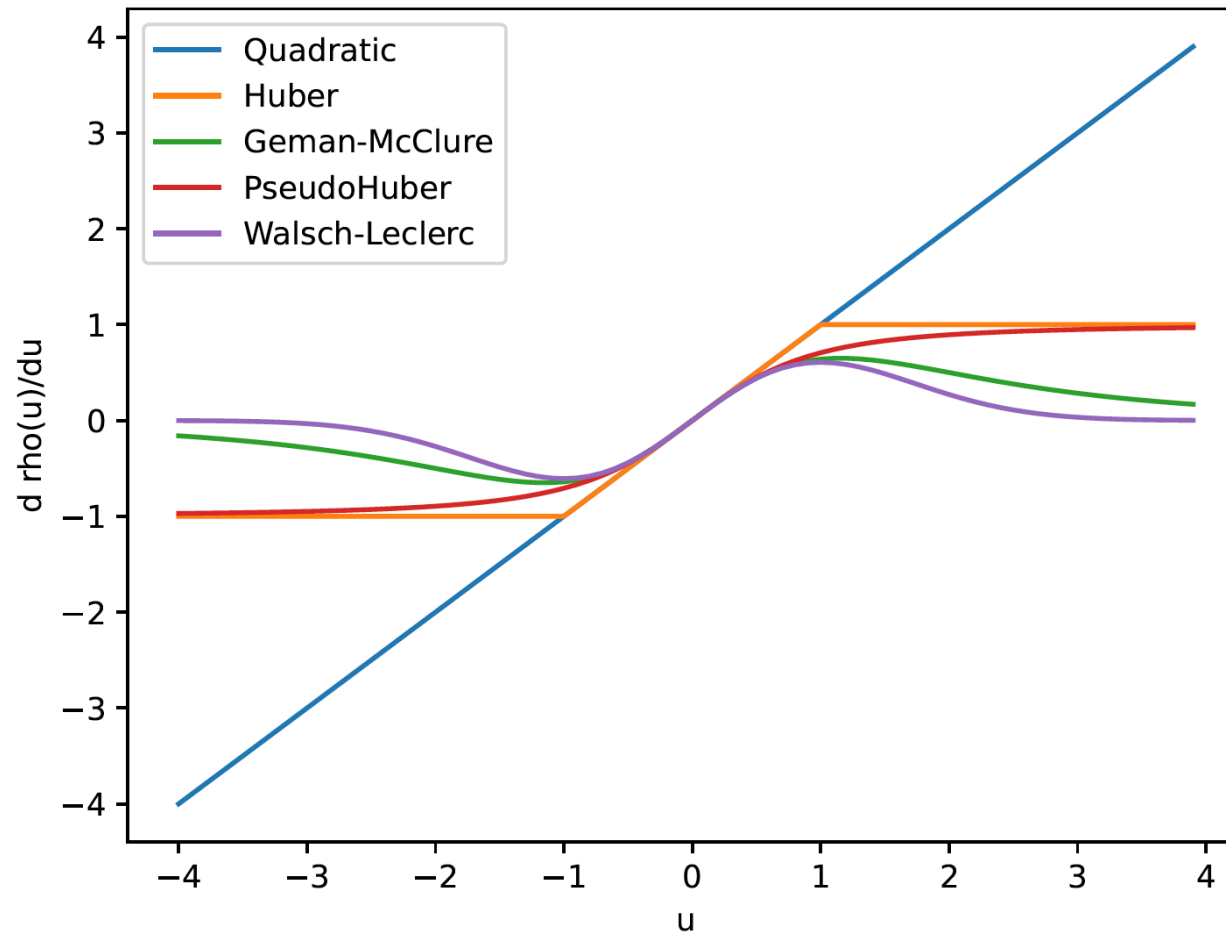
$$w_i = \left( \frac{\frac{\partial \rho}{\partial u}}{r(\mathbf{x}_i, \hat{\theta})} \right)$$

(where the derivatives are evaluated at that  $\hat{\theta}$ ), then

$$\sum_i w_i \nabla_{\theta} [r(\mathbf{x}_i, \theta)]^2 = \lambda \begin{pmatrix} 2a_1 \\ 2a_2 \\ 0 \end{pmatrix}.$$

# Robust estimators – influence functions

---



## Idea – iteratively reweighted least squares

---

Start with initial line

get weights, scale from line

Iterate:

estimate line using weights, scale

estimate scale using line

estimate weights using scale, line

We \*know\* that one stationary point is the true minimum

No other guarantees I'm aware of, but quite well behaved

# Starting iteratively reweighted least squares

---

Iterate:

- Initial line:

  - Draw two points at random from dataset

  - Pass line through them

- Fit line with IRLS

Use the best you encounter

# IRLS

---

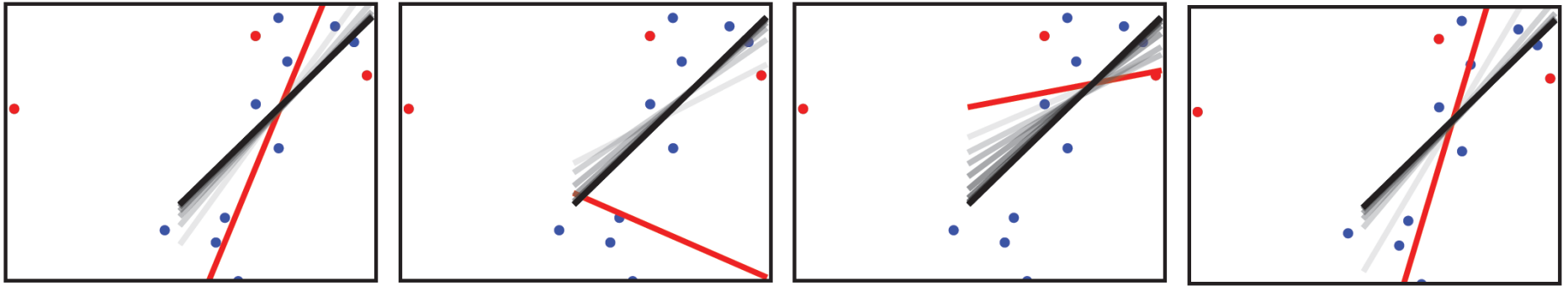


FIGURE 12.7: *Robust losses can control the influence of outliers. **Blue** points lie on a line, and have been perturbed by noise; **red** points are outliers. The **red** line shows a starting line, obtained by drawing a small random sample from the dataset, then fitting a line; the **gray** lines show iterates of IRLS applied to a Huber loss (later iterates are more opaque; scales are estimated as in the text). The procedure converges from a range of start points, some quite far from the “true” line. Notice how each start point results in the same line.*

# IRLS isn't perfect...

---

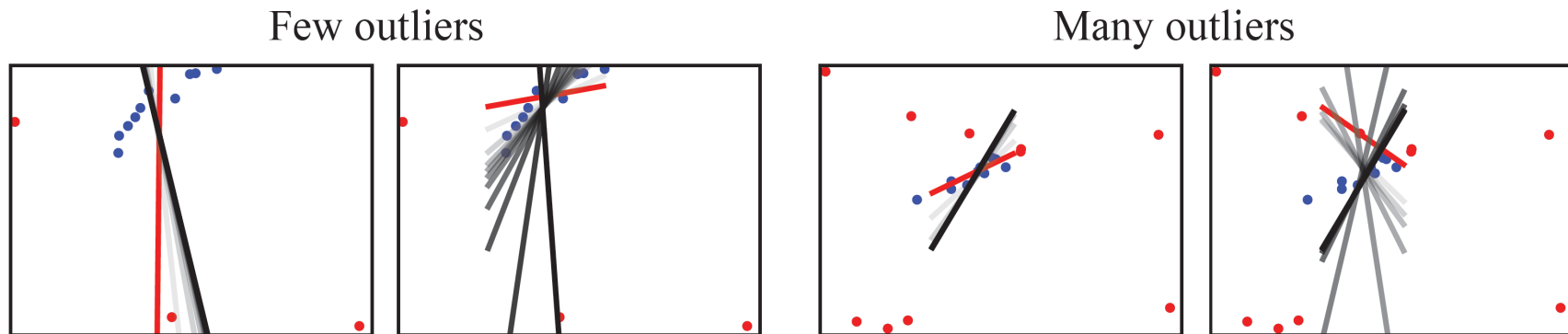


FIGURE 12.8: *Robust losses can fail, particularly when distant points still have some weight or if there are many outliers. **Left**: a bad start point leads to a bad line; **center left**: on the same data set, quite a good start point still converges to a bad line. Here there are few outliers, but they are far from the data and they contribute a significant weight to the loss. When there are many outliers, this effect worsens. Because each outlier still contributes a significant weight to the loss, even a good start fails (**center right**). A poor start (**right**) also fails, and produces the same line as the good start – in fact, most starts end up close to this line. Again, **blue** points lie on a line, and have been perturbed by noise; **red** points are outliers; the*



# Fitting: Overview

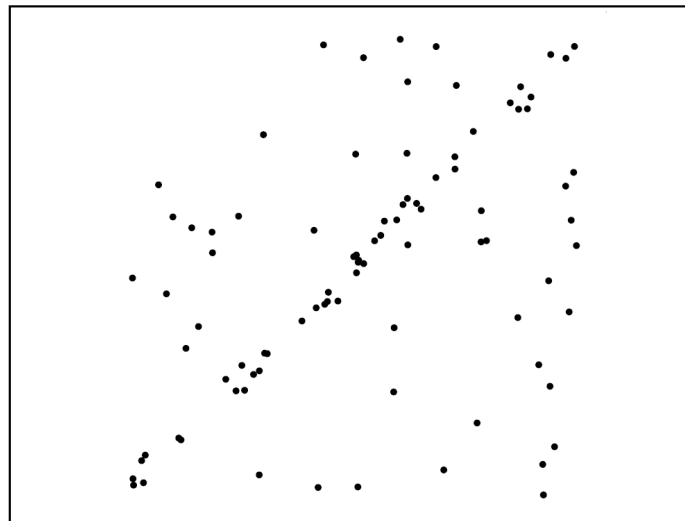
---

- Least squares line fitting
- Robust fitting
- RANSAC

# Voting schemes

---

- Robust fitting can deal with a few outliers – what if we have very many?
- Basic idea: let each point *vote* for all the models that are compatible with it
  - Hopefully the outliers will not vote consistently for any single model
  - The model that receives the most votes is the best fit to the image



# RANSAC

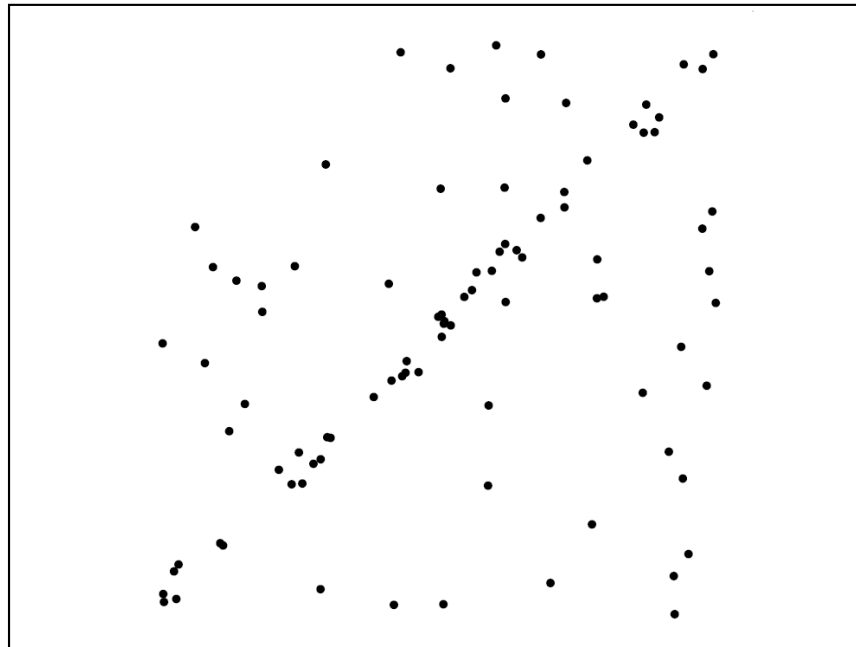
---

- Random sample consensus: very general framework for model fitting in the presence of outliers
- Outline:
  - Randomly choose a small initial subset of points
  - Fit a model to that subset
  - Find all inlier points that are “close” to the model and reject the rest as outliers
  - Do this many times and choose the model with the most inliers

M. Fischler and R. Bolles. [Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography](#). Comm. of the ACM, Vol 24, pp 381-395, 1981

# RANSAC for line fitting example

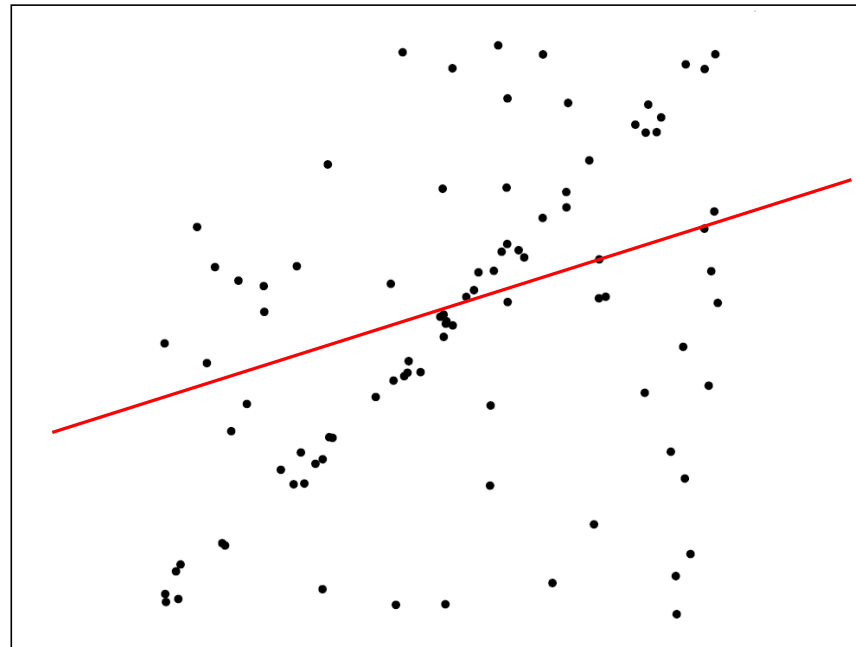
---



Source: R. Raguram

# RANSAC for line fitting example

---

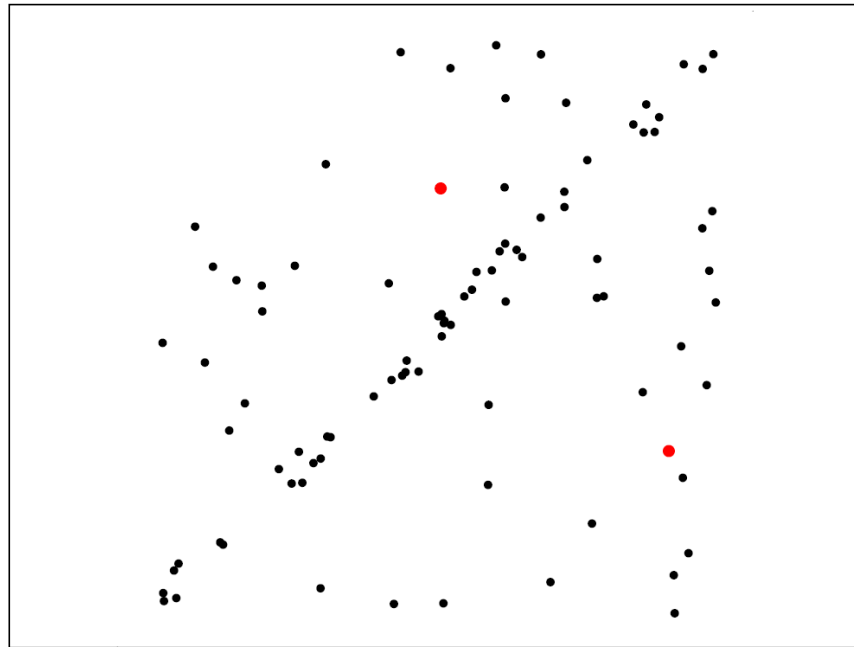


Least-squares fit

Source: R. Raguram

# RANSAC for line fitting example

---

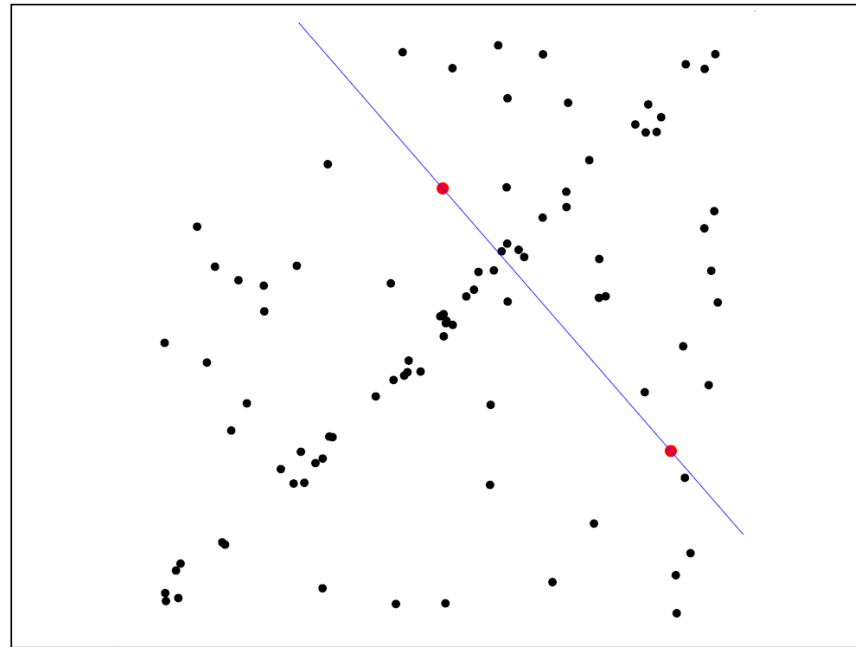


1. Randomly select minimal subset of points

Source: R. Raguram

# RANSAC for line fitting example

---

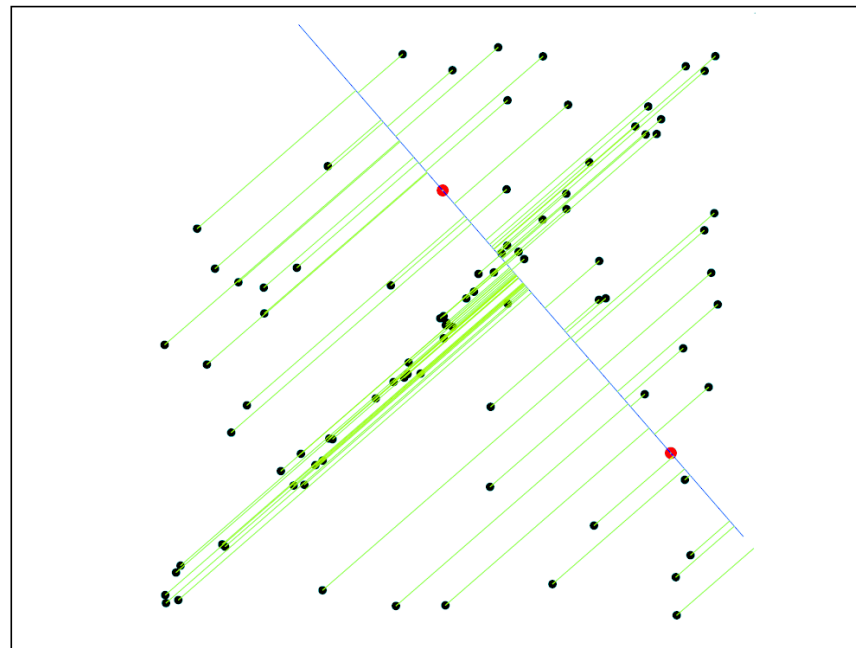


1. Randomly select minimal subset of points
2. Hypothesize a model

Source: R. Raguram

# RANSAC for line fitting example

---



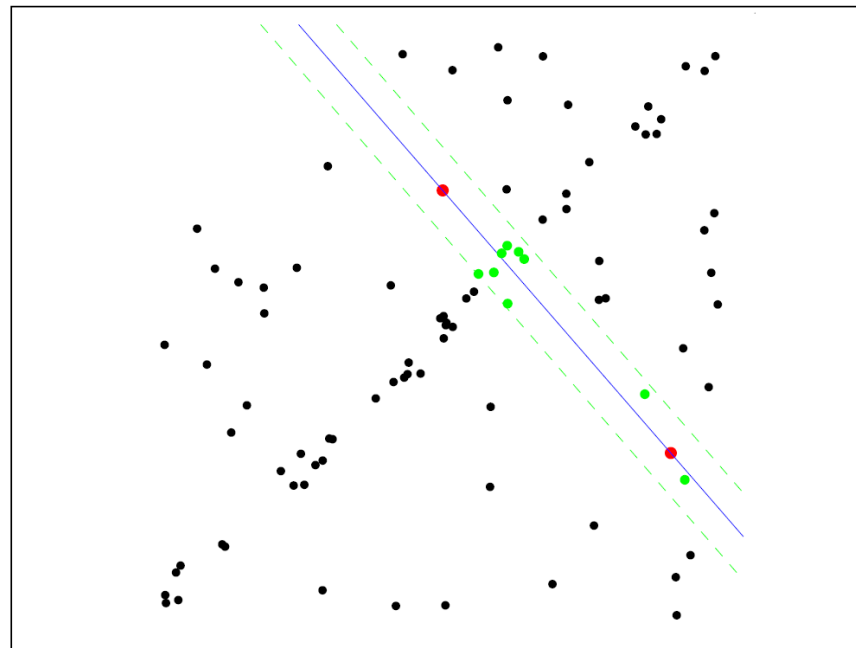
1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function

Source: R. Raguram



# RANSAC for line fitting example

---

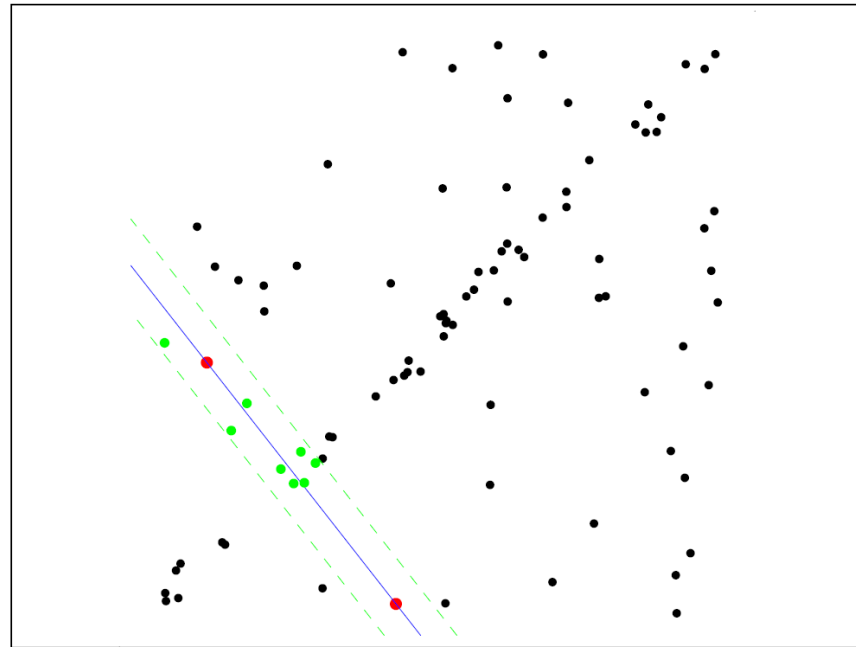


1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. **Select points consistent with model**

Source: R. Raguram

# RANSAC for line fitting example

---

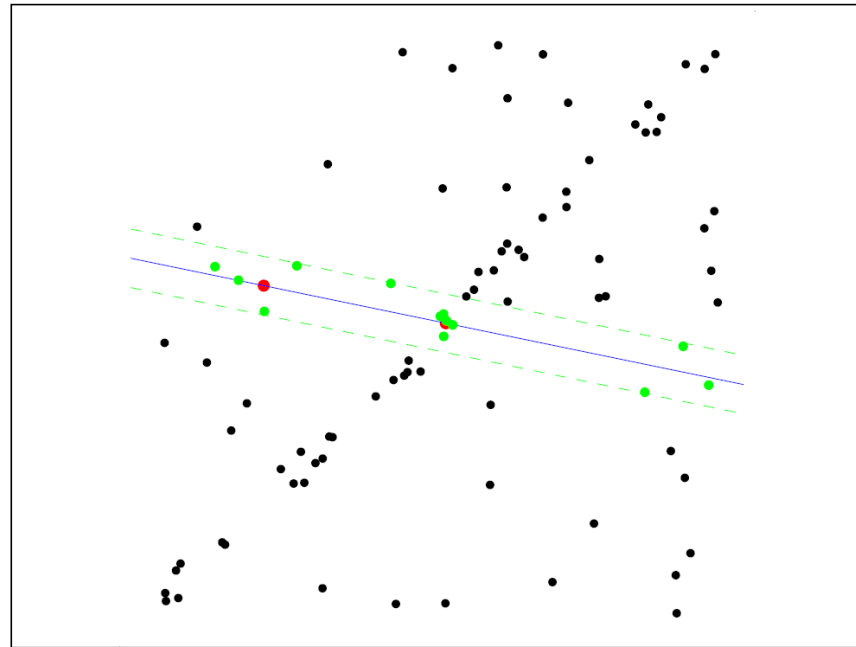


1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model
5. Repeat *hypothesize-and-verify* loop

Source: R. Raguram

# RANSAC for line fitting example

---



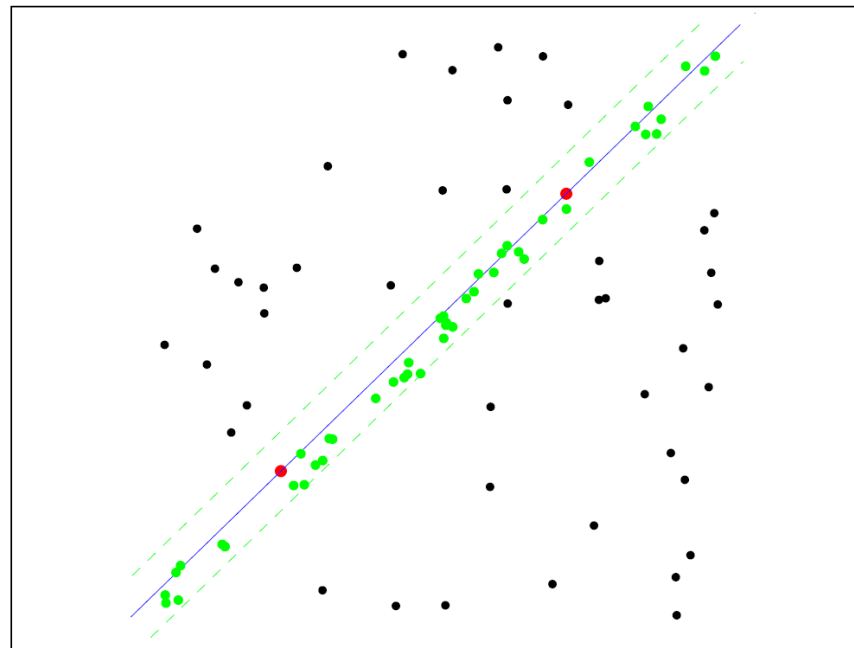
1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model
5. Repeat *hypothesize-and-verify* loop

Source: R. Raguram

# RANSAC for line fitting example

---

## Uncontaminated sample

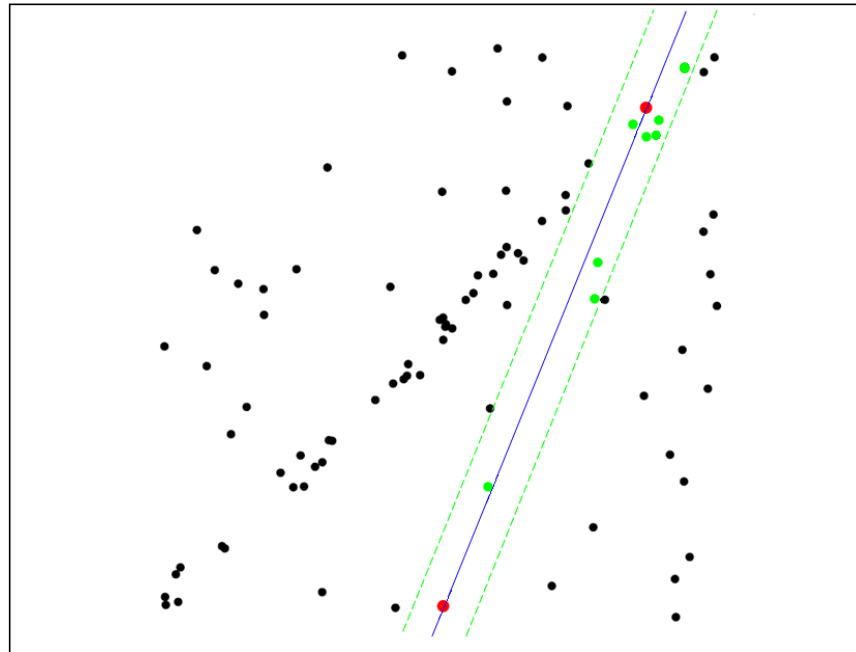


1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model
5. Repeat *hypothesize-and-verify* loop

Source: R. Raguram

# RANSAC for line fitting example

---



1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model
5. Repeat *hypothesize-and-verify* loop

Source: R. Raguram

## RANSAC loop

---

Repeat  $N$  times:

- Draw  $s$  points uniformly at random
- Fit model to these  $s$  points
- Find *inliers* to the model among the remaining points (points whose distance or residual w.r.t. model is less than  $t$ )
- If there are  $d$  or more inliers, accept the model and refit using all inliers

# RANSAC: Choosing the parameters

---

- Initial number of points  $s$ 
  - Typically minimum number needed to fit the model
- Distance threshold  $t$  for inliers
  - Need suitable assumptions, e.g., given zero-mean Gaussian noise with std. dev.  $\sigma$ ,  $t = 1.96\sigma$  will give ~95% probability of capturing all inliers
- Consensus set size  $d$ 
  - Should match expected inlier ratio

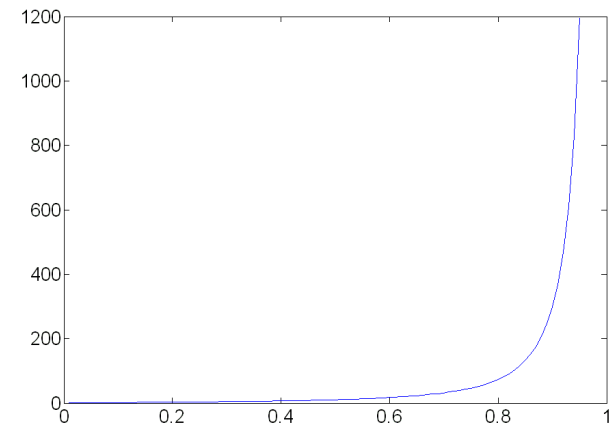
# RANSAC: Choosing the parameters

---

- Choosing the number of iterations (initial samples)  $N$ :
  - Choose  $N$  so that, with probability  $p$  (e.g. 99%), at least one initial sample is free from outliers
  - Assuming an outlier ratio of  $e$ :

$$(1 - (1 - e)^s)^N = 1 - p$$
$$N = \log(1 - p) / \log(1 - (1 - e)^s)$$

proportion of outliers $e$							
s	5%	10%	20%	25%	30%	40%	50%
2	2	3	5	6	7	11	17
3	3	4	7	9	11	19	35
4	3	5	9	13	17	34	72
5	4	6	12	17	26	57	146
6	4	7	16	24	37	97	293
7	4	8	20	33	54	163	588
8	5	9	26	44	78	272	1177



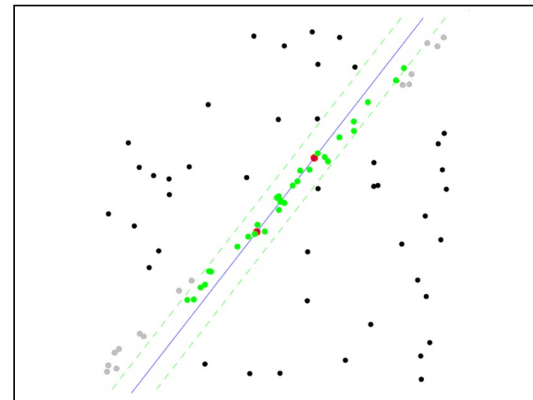
Source: M. Pollefeys



# RANSAC pros and cons

---

- Pros
  - Simple and general
  - Applicable to many different problems
  - Often works well in practice
- Cons
  - Lots of parameters to set
  - Number of iterations grows **exponentially** as outlier ratio increases
  - Can't always get a good initialization of the model based on the minimum number of samples



# Incremental RANSAC

---

To fit many lines to a dataset:

Iterate:

- Fit a line with RANSAC

- using IRLS (crucial!)

- Remove inliers from dataset

Q: when to stop?

A: when you have the right number of lines

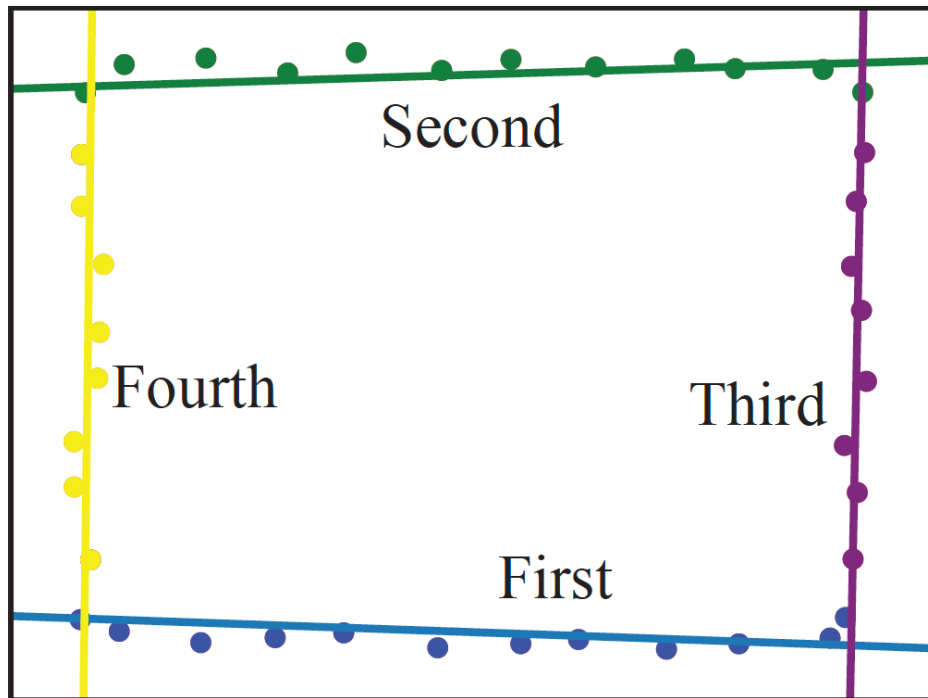
- when too little data is left

- when the last line has few inliers

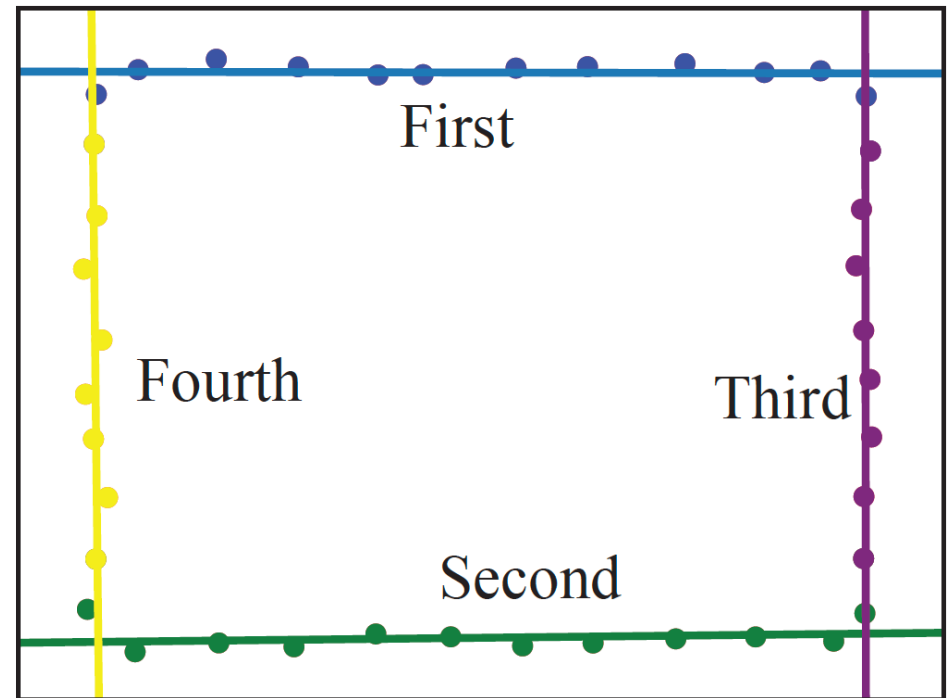
**Depends on  
application!**

## IRLS matters...

---



Least squares on inliers



IRLS squares on inliers

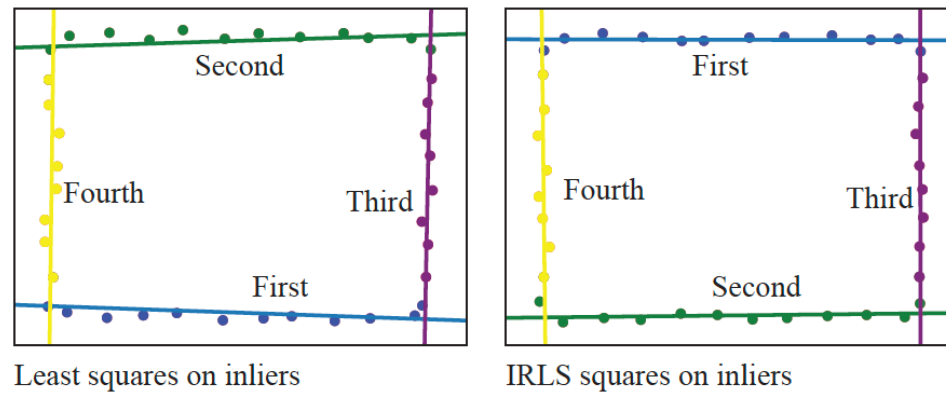


FIGURE 12.9: *Incremental RANSAC can successfully fit multiple lines to a noisy dataset. Points lie on the outline of a square, and have been perturbed by noise. I applied the strategy of Section 12.3.1 using a RANSAC line fitter. Colors show inliers and line for each round (in the order blue, green, purple, yellow). On the left, the final fit to the inlying points is by total least squares. Because some inliers can be quite far from the line at the corners, the lines tend not to run close to the data points. On the right, the final fit uses 10 iterations of IRLS, with the Huber loss. The lines are now very close to the data points.*