Edge detection

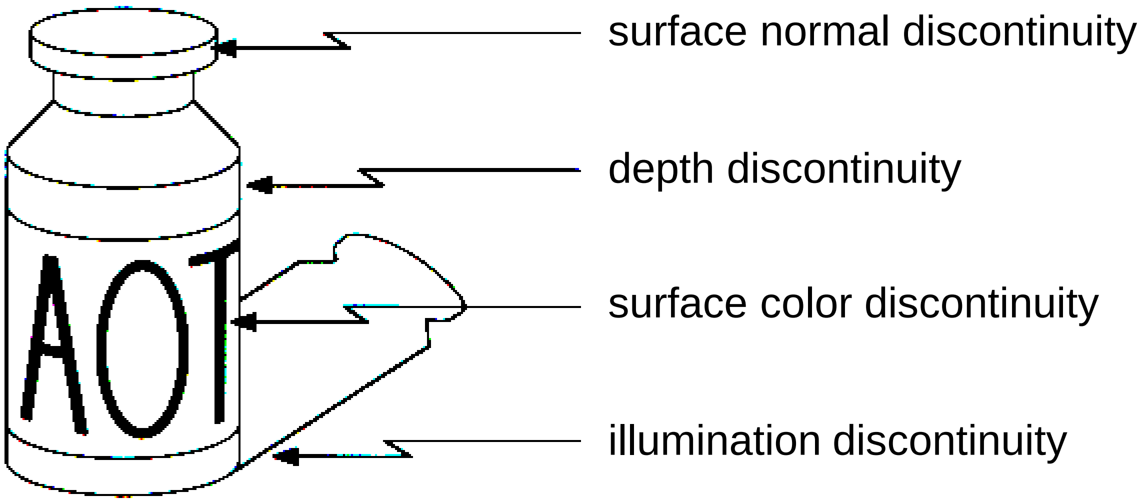
- **Goal:** Identify sudden changes (discontinuities) in an image
  - Intuitively, most semantic and shape information from the image can be encoded in the edges
  - More compact than pixels

- **Ideal:** artist's line drawing (but artist is also using object-level knowledge)

Source: D. Lowe

Origin of edges

Edges are caused by a variety of factors:



Source: Steve Seitz

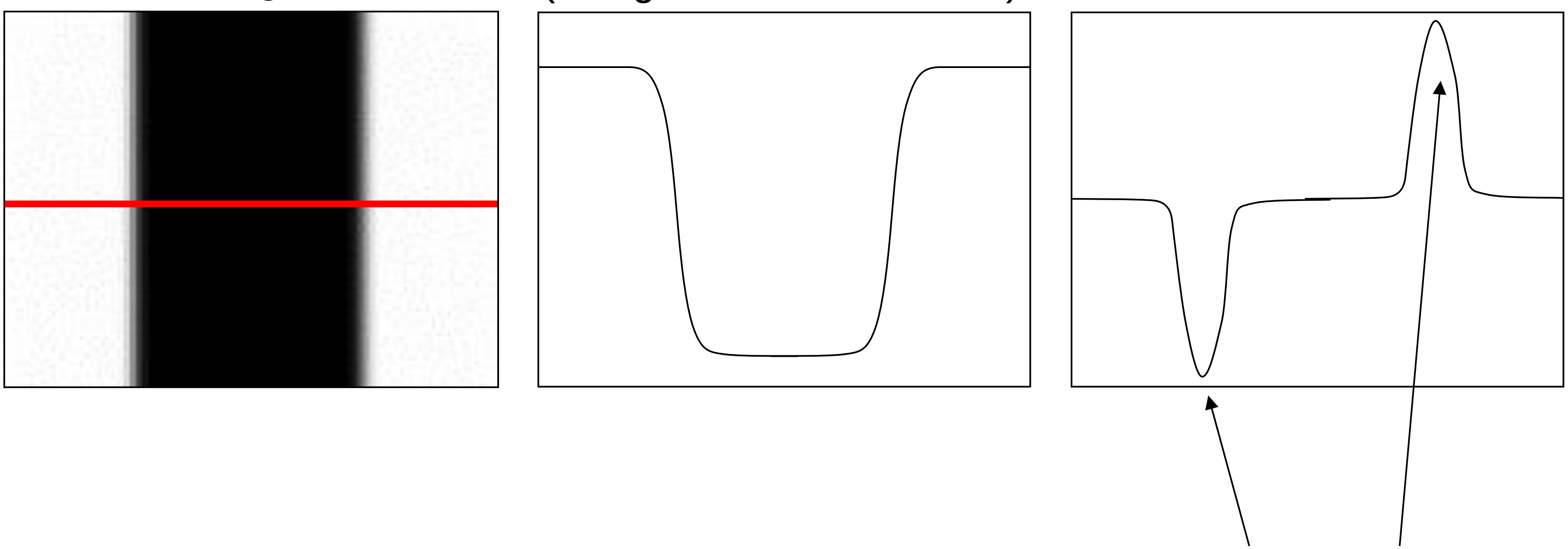
Characterizing edges

- An edge is a place of rapid change in the image intensity function

image

intensity function (along horizontal scanline)

first derivative



edges correspond to extrema of derivative

Derivatives with convolution

For 2D function f(x,y), the partial derivative is:

$$\frac{\partial f(x,y)}{\partial x} = \lim_{\varepsilon \to 0} \frac{f(x+\varepsilon, y) - f(x,y)}{\varepsilon}$$

For discrete data, we can approximate using finite differences:

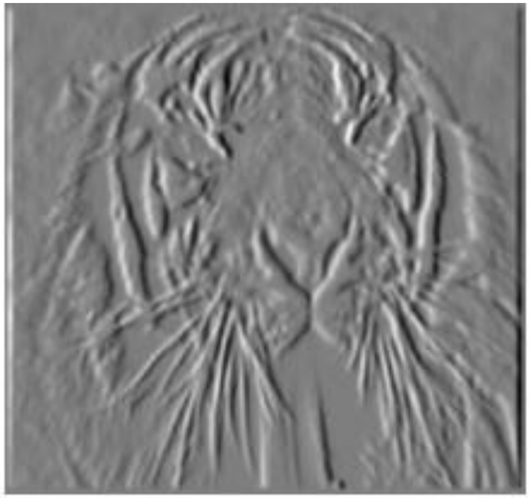
$$\frac{\partial f(x,y)}{\partial x} \approx \frac{f(x+1, y) - f(x,y)}{1}$$

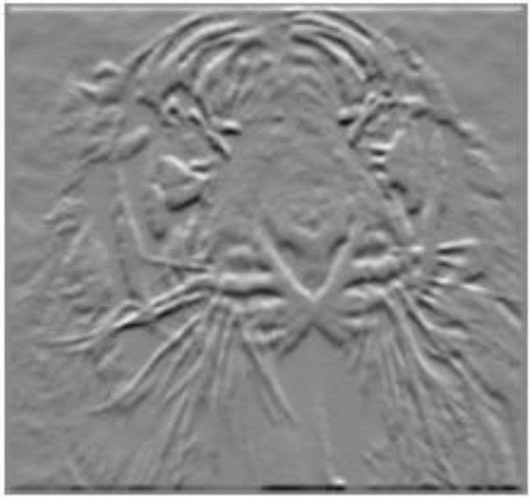
To implement above as convolution, what would be the associated filter?

Source: K. Grauman

Partial derivatives of an image

$$\frac{\partial f(x,y)}{\partial x}$$





$$\frac{\partial f(x,y)}{\partial y}$$

-1	**1**

$\begin{bmatrix} -1 \\ 1 \end{bmatrix}$ **or** $\begin{bmatrix} 1 \\ -1 \end{bmatrix}$

Which shows changes with respect to x?

Finite difference filters

Other approximations of derivative filters exist:

Prewitt: $M_x = \begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline \end{array}$; $M_y = \begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -1 & -1 \\ \hline \end{array}$

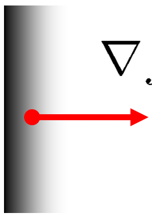
Sobel: $M_x = \begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline \end{array}$; $M_y = \begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -2 & -1 \\ \hline \end{array}$

Roberts: $M_x = \begin{array}{|c|c|}\hline 0 & 1 \\ \hline -1 & 0 \\ \hline \end{array}$; $M_y = \begin{array}{|c|c|}\hline 1 & 0 \\ \hline 0 & -1 \\ \hline \end{array}$

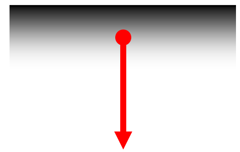
Source: K. Grauman

Image gradient

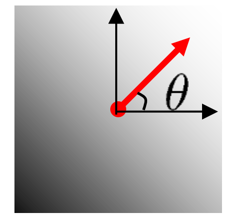
The gradient of an image: $$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$



$$\nabla f = \left[\frac{\partial f}{\partial x}, 0\right]$$



$$\nabla f = \left[0, \frac{\partial f}{\partial y}\right]$$



$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

The gradient points in the direction of most rapid increase in intensity

- How does this direction relate to the direction of the edge?

The gradient direction is given by $\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$

The *edge strength* is given by the gradient magnitude

$$||\nabla f|| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

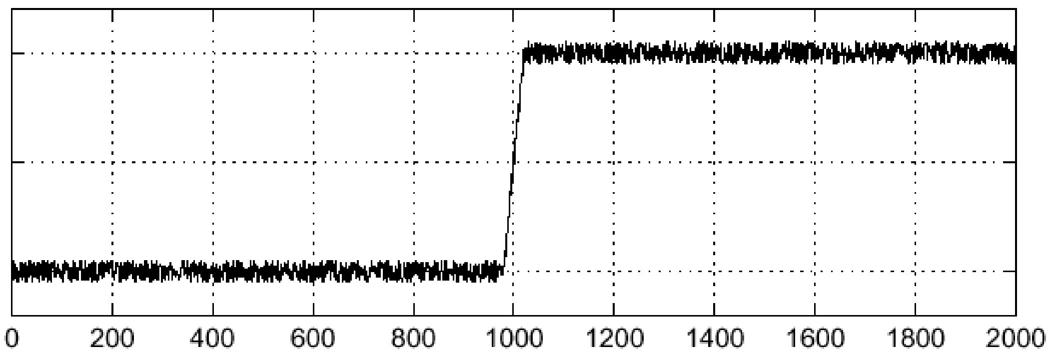
Source: Steve Seitz

Effects of noise

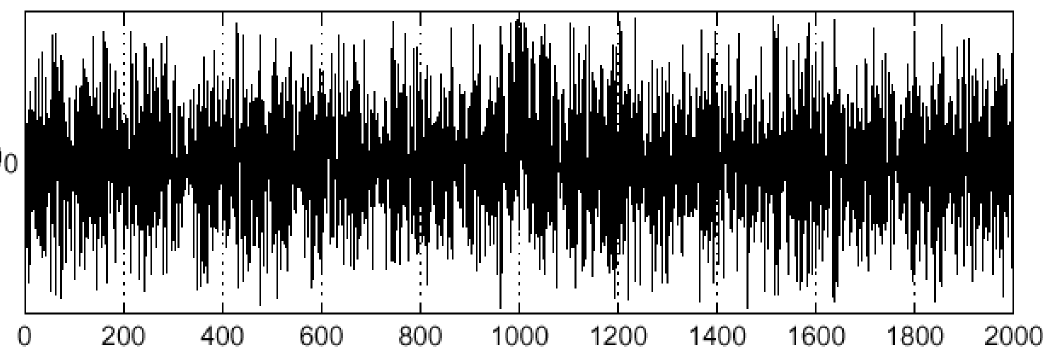
Consider a single row or column of the image

- Plotting intensity as a function of position gives a signal

$f(x)$



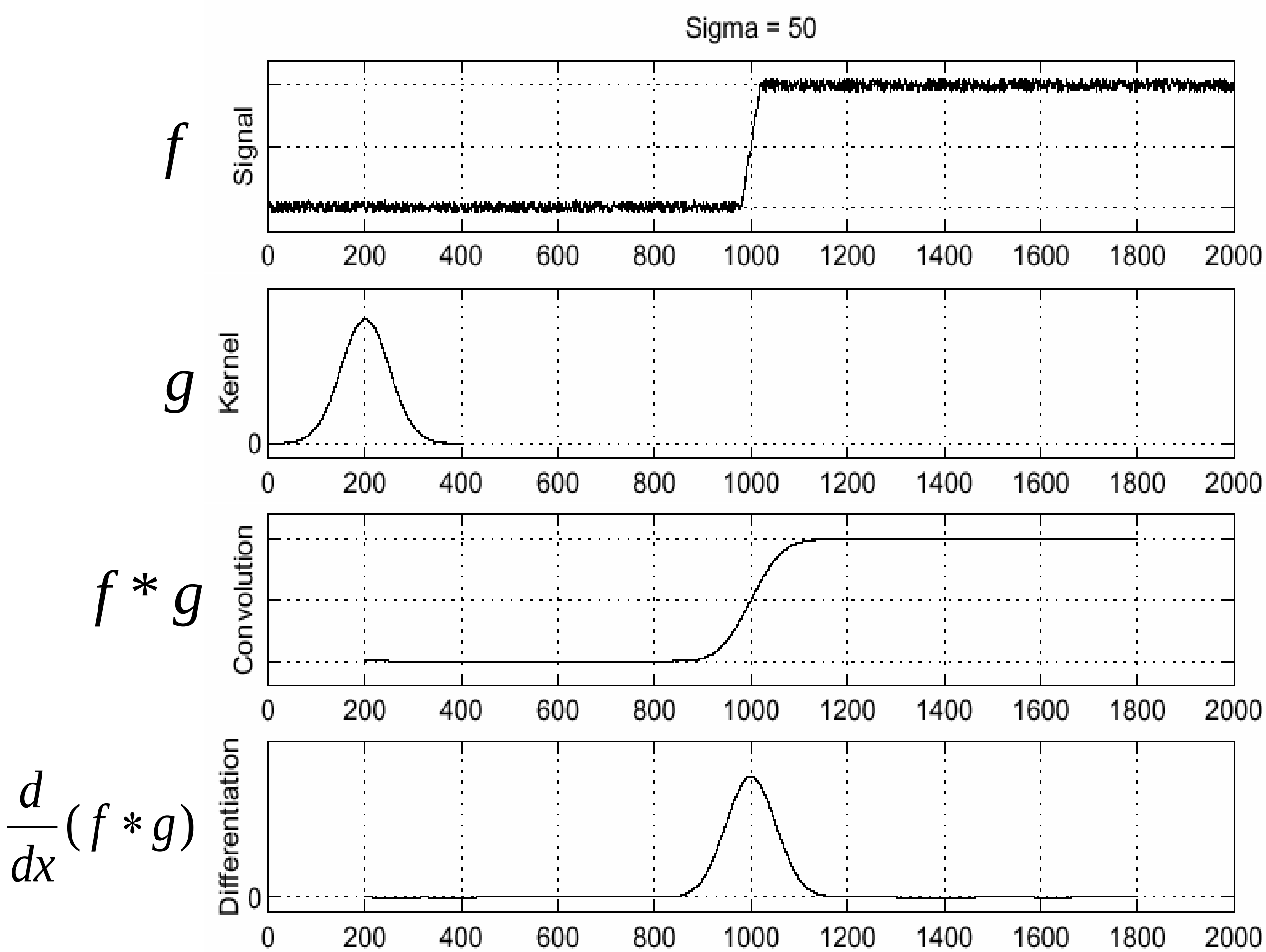
$\frac{d}{dx}f(x)$



Where is the edge?

Source: S. Seitz

Solution: smooth first



- To find edges, look for peaks in $\frac{d}{dx}(f * g)$

Source: S. Seitz

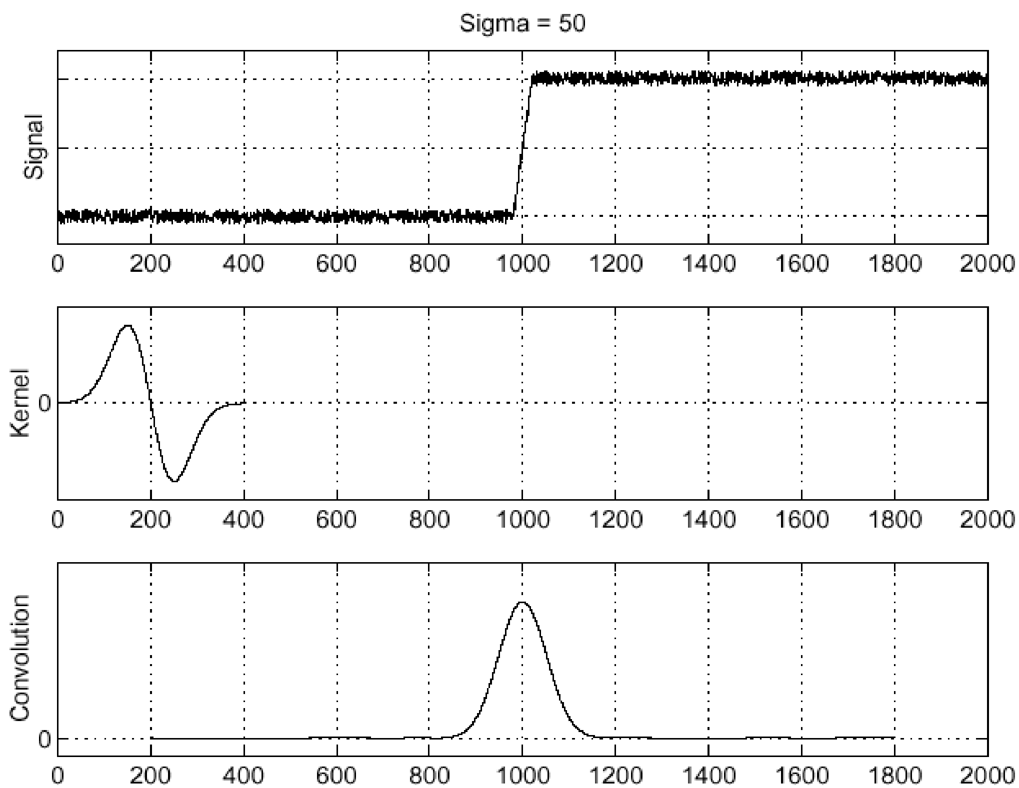
Derivative theorem of convolution

- Differentiation is convolution, and convolution is associative: $$\frac{d}{dx}(f * g) = f * \frac{d}{dx} g$$
- This saves us one operation:

f

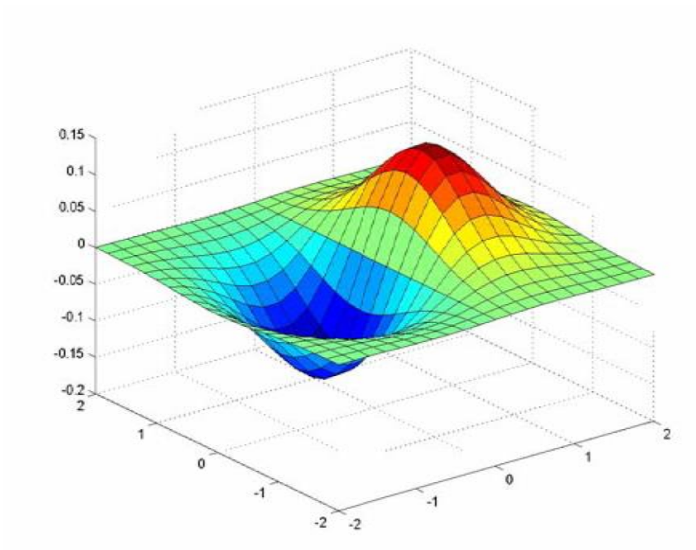
$\frac{d}{dx} g$

$f * \frac{d}{dx} g$

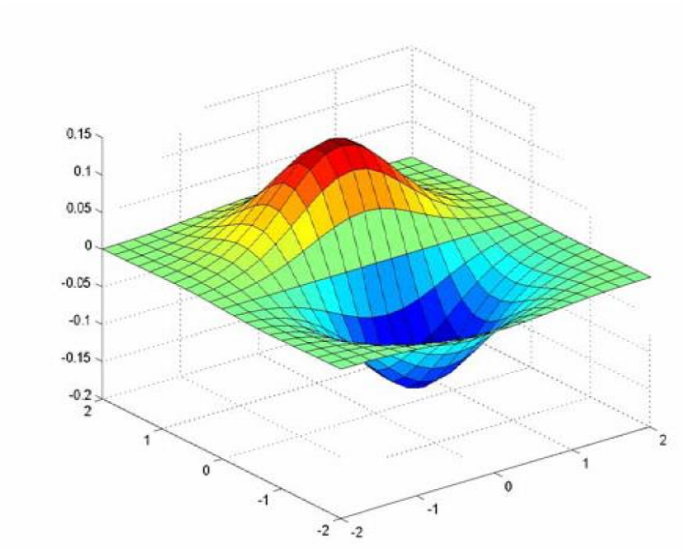


Source: S. Seitz

Derivative of Gaussian filter



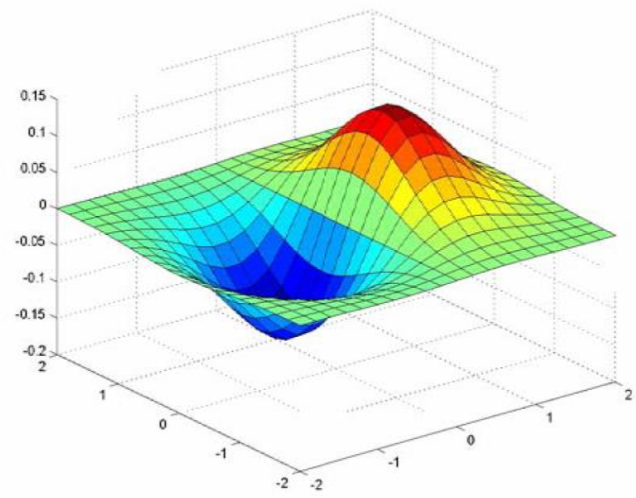
x-direction



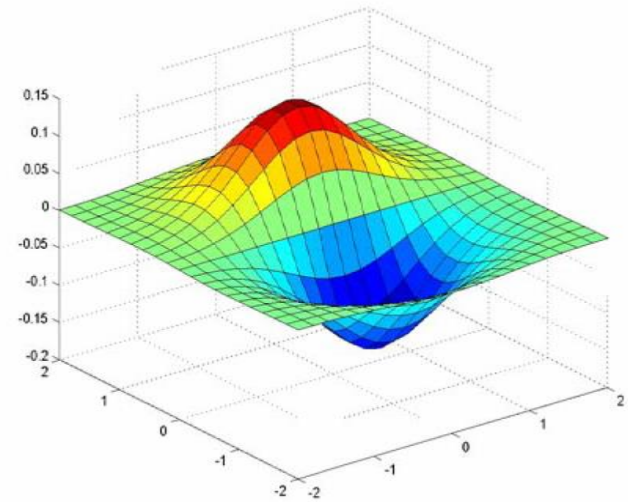
y-direction

Are these filters separable?

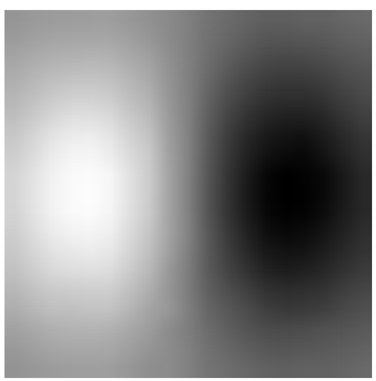
Derivative of Gaussian filter

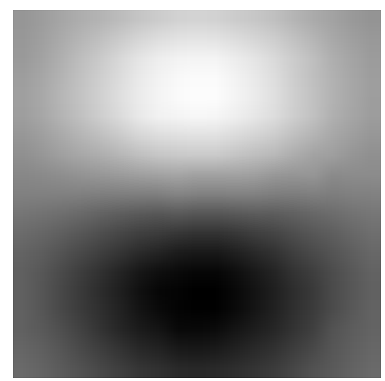


x-direction



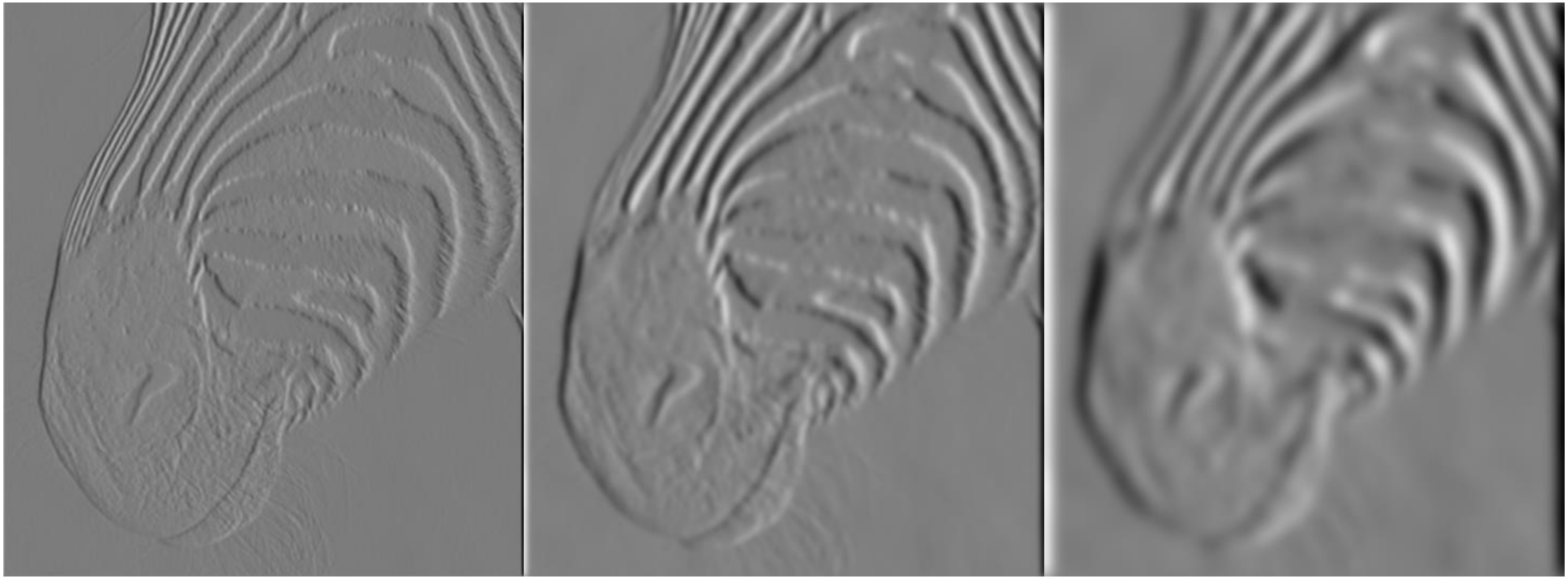
y-direction





Which one finds horizontal/vertical edges?

Scale of Gaussian derivative filter



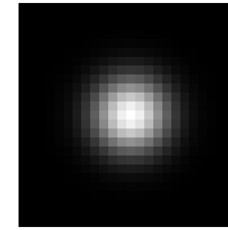
1 pixel 3 pixels 7 pixels

Smoothed derivative removes noise, but blurs edge. Also finds edges at different “scales”

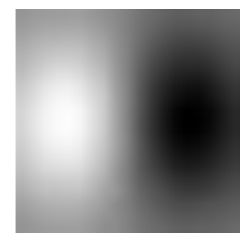
Source: D. Forsyth

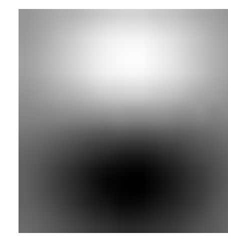
Review: Smoothing vs. derivative filters

Smoothing filters



- Gaussian: remove "high-frequency" components; "low-pass" filter
- Can the values of a smoothing filter be negative?
- What should the values sum to?
  - **One:** constant regions are not affected by the filter





Derivative filters

- Derivatives of Gaussian
- Can the values of a derivative filter be negative?
- What should the values sum to?
  - **Zero:** no response in constant regions
- High absolute value at points of high contrast

The Canny edge detector

original image

Slide credit: Steve Seitz

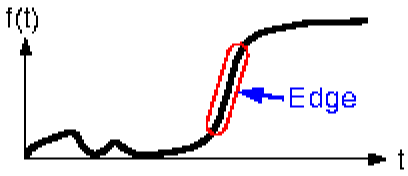
The Canny edge detector

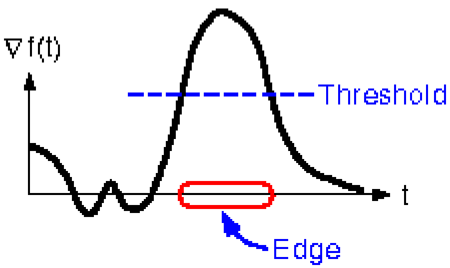
norm of the gradient

The Canny edge detector

thresholding

The Canny edge detector

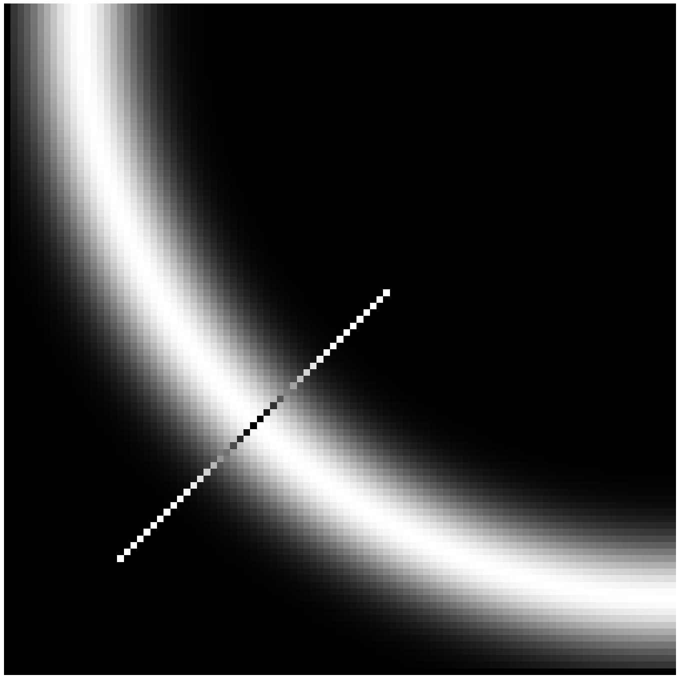


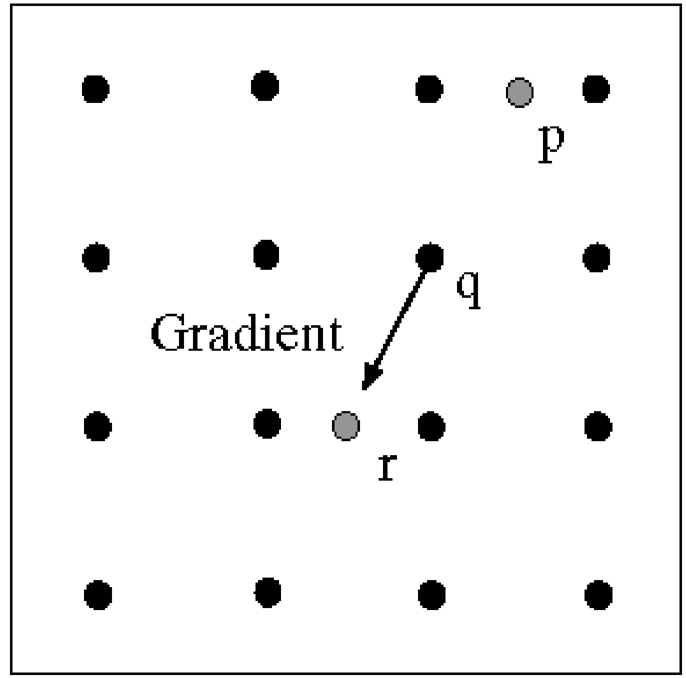


How to turn these thick regions of the gradient into curves?

thresholding

Non-maximum suppression





Check if pixel is local maximum along gradient direction, select single max across width of the edge

- requires checking interpolated pixels p and r

The Canny edge detector

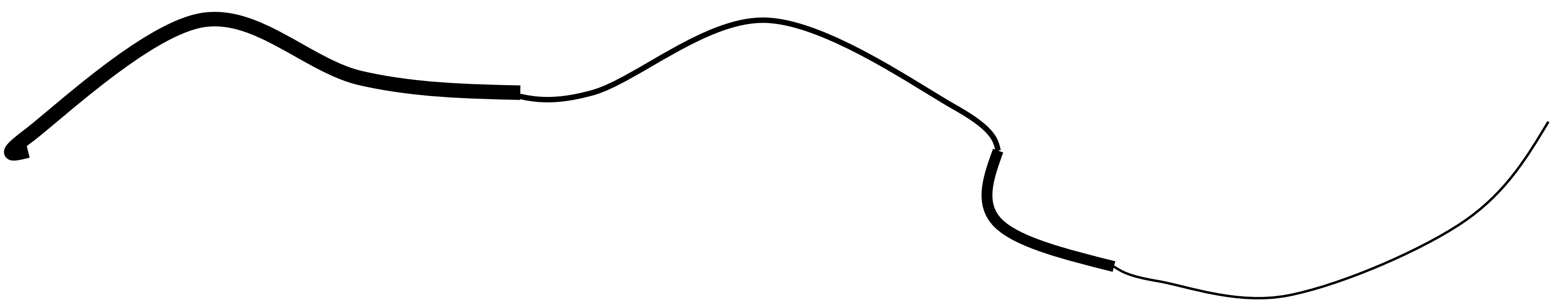
Problem: pixels along this edge didn't survive the thresholding

thinning

(non-maximum suppression)

Hysteresis thresholding

Use a high threshold to start edge curves, and a low threshold to continue them.



Source: Steve Seitz

Hysteresis thresholding

original image

high threshold (strong edges)

low threshold (weak edges)

hysteresis threshold

Source: L. Fei-Fei

Recap: Canny edge detector

1. Filter image with derivative of Gaussian
2. Find magnitude and orientation of gradient
3. **Non-maximum suppression**:
   - Thin wide “ridges” down to single pixel width
4. **Linking and thresholding** (**hysteresis**):
   - Define two thresholds: low and high
   - Use the high threshold to start edge curves and the low threshold to continue them

MATLAB: `edge(image, ‘canny’);`

J. Canny, ***A Computational Approach To Edge Detection***, IEEE Trans. Pattern Analysis and Machine Intelligence, 8:679-714, 1986.

Edge detection is just the beginning…

image | human segmentation | gradient magnitude

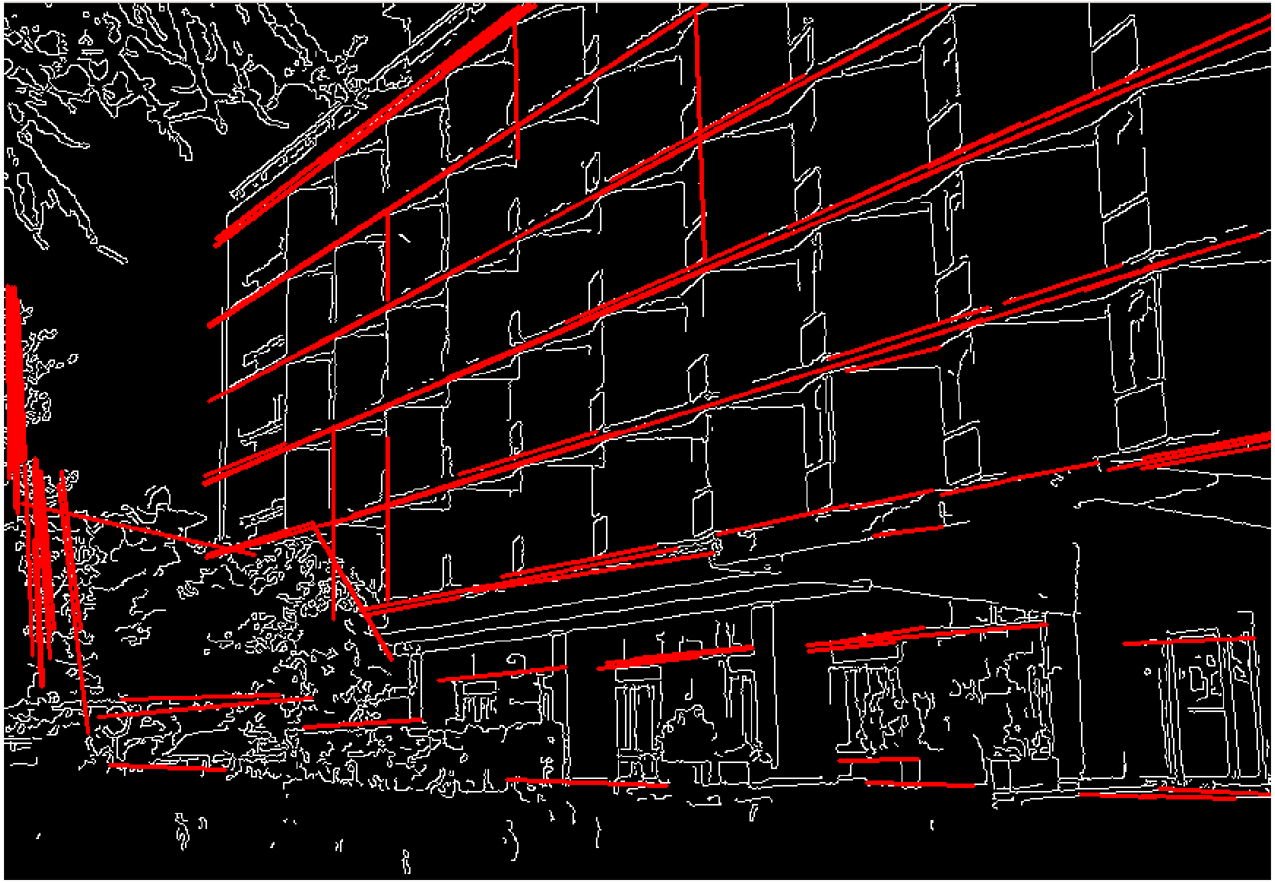
Berkeley segmentation database:

http://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/segbench/

Low-level edges vs. perceived contours

Background **Texture** **Shadows**

Fitting

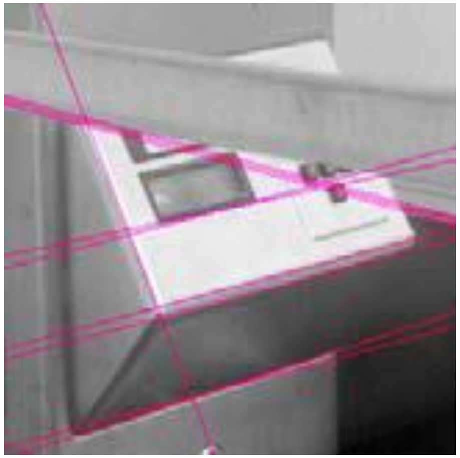


Fitting

- 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

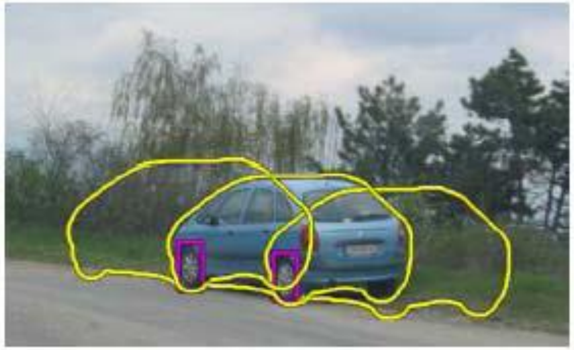
Fitting

- Choose a parametric model to represent a set of features



simple model: lines

simple model: circles



complicated model: car

Source: K. Grauman

Fitting: Issues

Case study: Line detection

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

Fitting: Overview

- If we know which points belong to the line, how do we find the “optimal” line parameters?
  - Least squares

- What if there are outliers?
  - Robust fitting, RANSAC

- What if there are many lines?
  - Voting methods: RANSAC, Hough transform

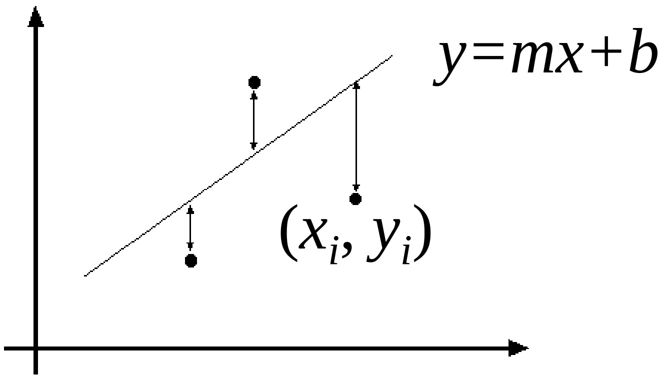
- What if we’re not even sure it’s a line?
  - Model selection

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



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

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

$$E = \|Y - XB\|^2 = (Y - XB)^T (Y - XB) = Y^T Y - 2(XB)^T Y + (XB)^T (XB)$$

$$\frac{dE}{dB} = 2X^T XB - 2X^T Y = 0$$

$$\boxed{X^T XB = X^T Y}$$

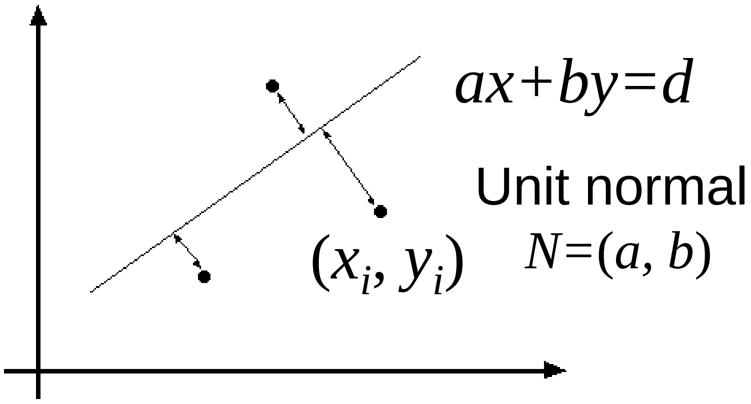
Normal equations: least squares solution to $XB=Y$

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 $ax+by=d$ $(a^2+b^2=1)$: $|ax_i + by_i - d|$

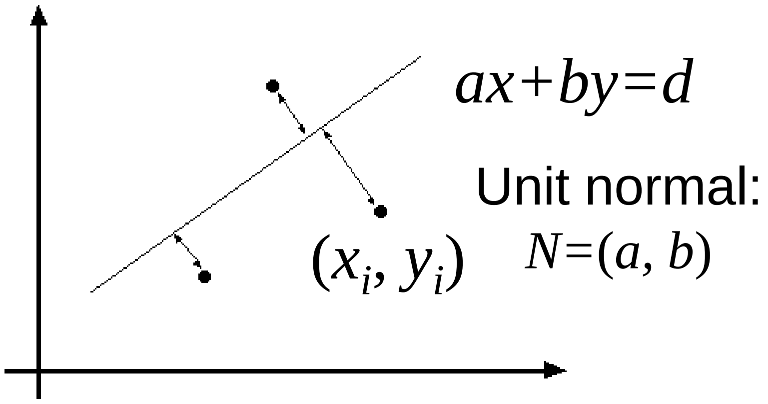


Total least squares

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

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

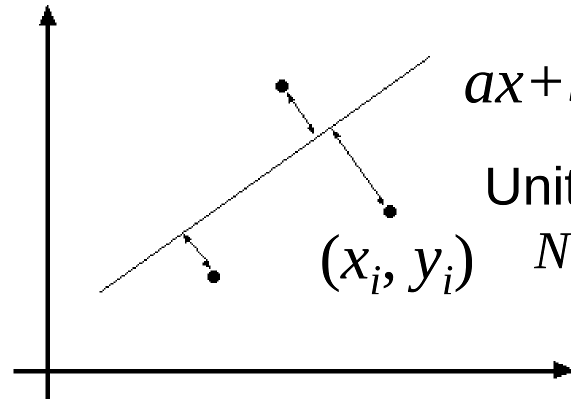
$$\boxed{E = \sum_{i=1}^{n}(ax_i + by_i - d)^2}$$



Total least squares

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

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



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

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

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

$$\frac{dE}{dN} = 2(U^T U)N = 0$$

Solution to $(U^T U)N = 0$, subject to $||N||^2 = 1$: eigenvector of $U^T U$ associated with the smallest eigenvalue (least squares solution to *homogeneous linear system* $UN = 0$)

Total least squares

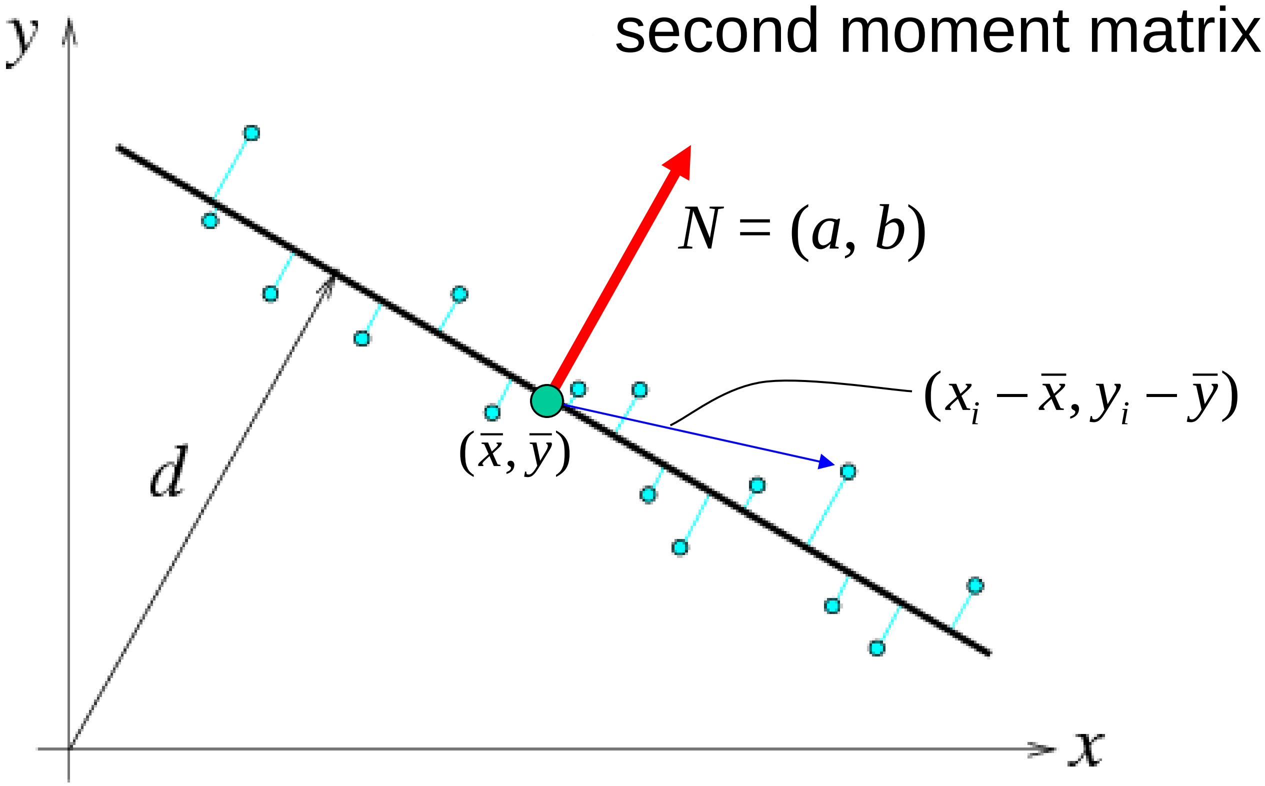
$$U = \begin{bmatrix} x_1 - \overline{x} & y_1 - \overline{y} \\ \vdots & \vdots \\ x_n - \overline{x} & y_n - \overline{y} \end{bmatrix} \qquad U^T U = \begin{bmatrix} \sum_{i=1}^{n} (x_i - \overline{x})^2 & \sum_{i=1}^{n} (x_i - \overline{x})(y_i - \overline{y}) \\ \sum_{i=1}^{n} (x_i - \overline{x})(y_i - \overline{y}) & \sum_{i=1}^{n} (y_i - \overline{y})^2 \end{bmatrix}$$

second moment matrix

Total least squares

$$U = \begin{bmatrix} x_1 - \bar{x} & y_1 - \bar{y} \\ \vdots & \vdots \\ x_n - \bar{x} & y_n - \bar{y} \end{bmatrix} \qquad U^T U = \begin{bmatrix} \sum_{i=1}^{n} (x_i - \bar{x})^2 & \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y}) \\ \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y}) & \sum_{i=1}^{n} (y_i - \bar{y})^2 \end{bmatrix}$$

second moment matrix



Least squares as likelihood maximization

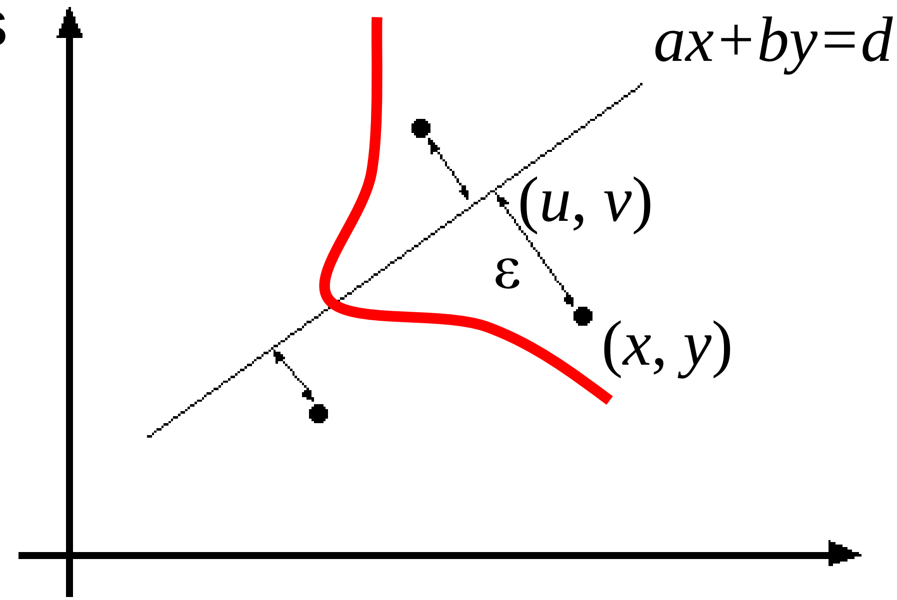
- **Generative model**: line points are sampled independently and corrupted by Gaussian noise in the direction perpendicular to the line

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

point on the line

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

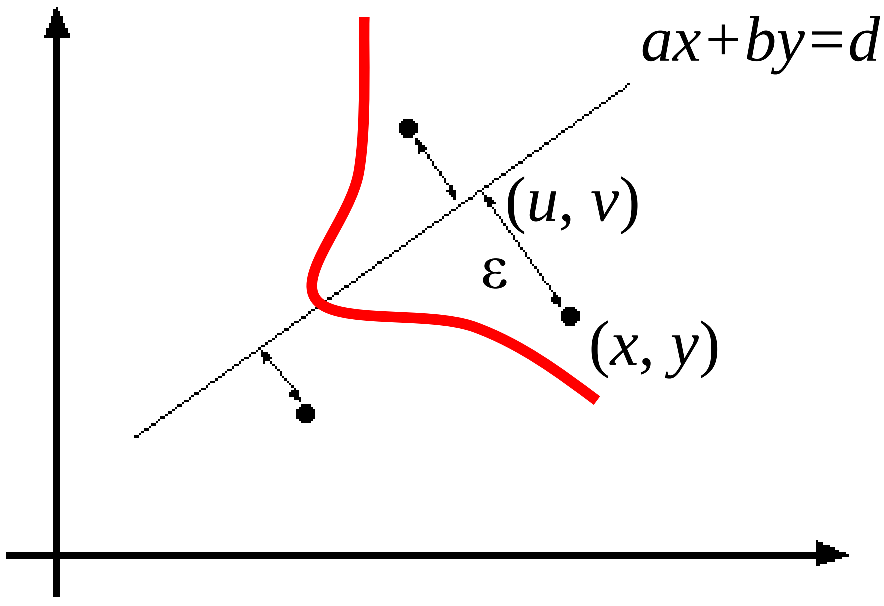
normal direction



Least squares as likelihood maximization

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

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



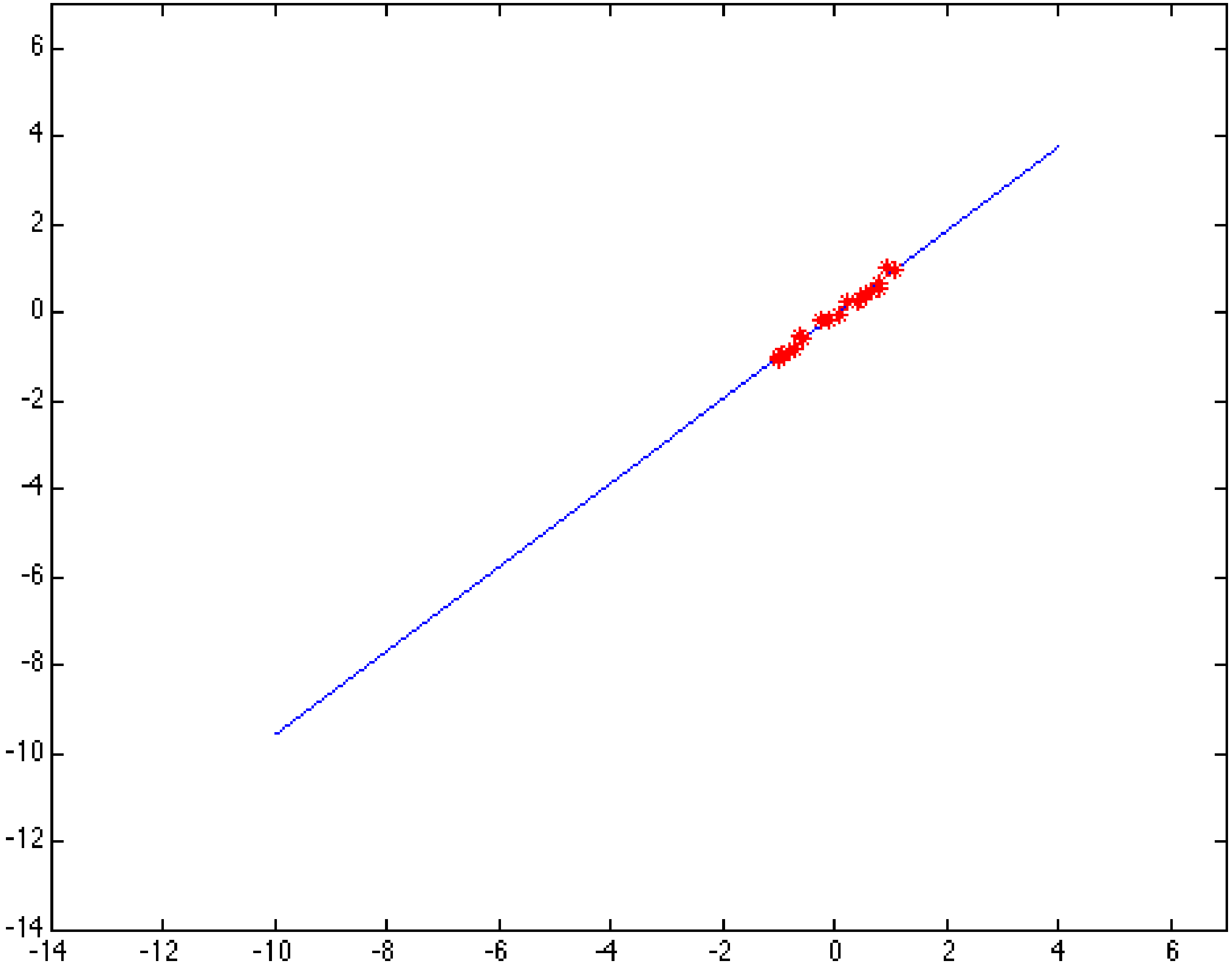
Likelihood of points given line parameters (*a*, *b*, *d*):

$$P(x_1, y_1, \ldots, x_n, y_n \mid a, b, d) = \prod_{i=1}^{n} P(x_i, y_i \mid a, b, d) \propto \prod_{i=1}^{n} \exp\left(-\frac{(ax_i + by_i - d)^2}{2\sigma^2} \right)$$

Log-likelihood: $L(x_1, y_1, \ldots, x_n, y_n \mid a, b, d) = -\frac{1}{2\sigma^2} \sum_{i=1}^{n} (ax_i + by_i - d)^2$

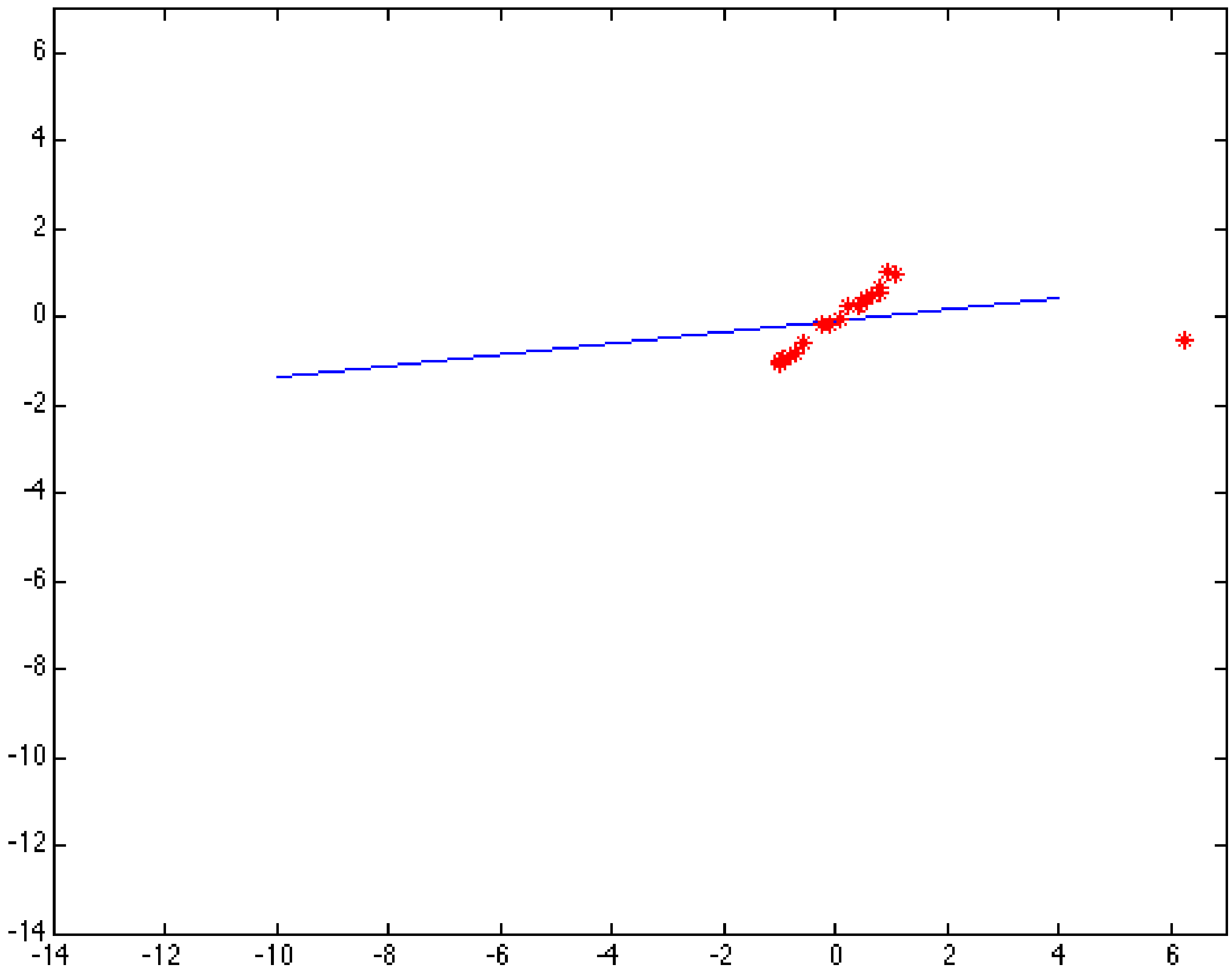
Least squares: Robustness to noise

Least squares fit to the red points:



Least squares: Robustness to noise

Least squares fit with an outlier:



Problem: squared error heavily penalizes outliers

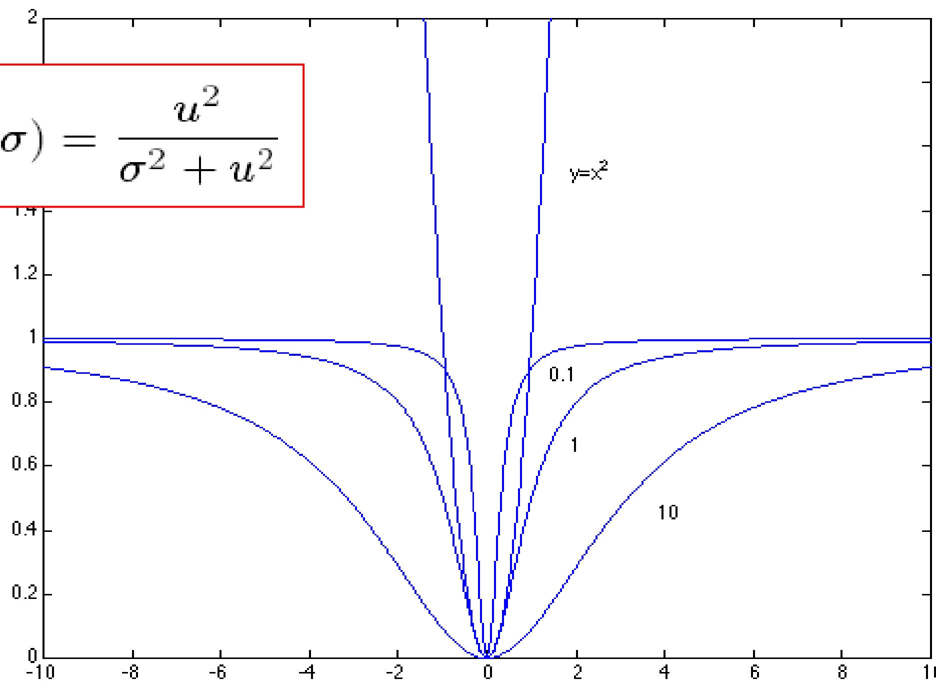
Robust estimators

- General approach: find model parameters θ that minimize

$$\sum_i \rho\left(r_i\left(x_i,\theta\right);\sigma\right)$$

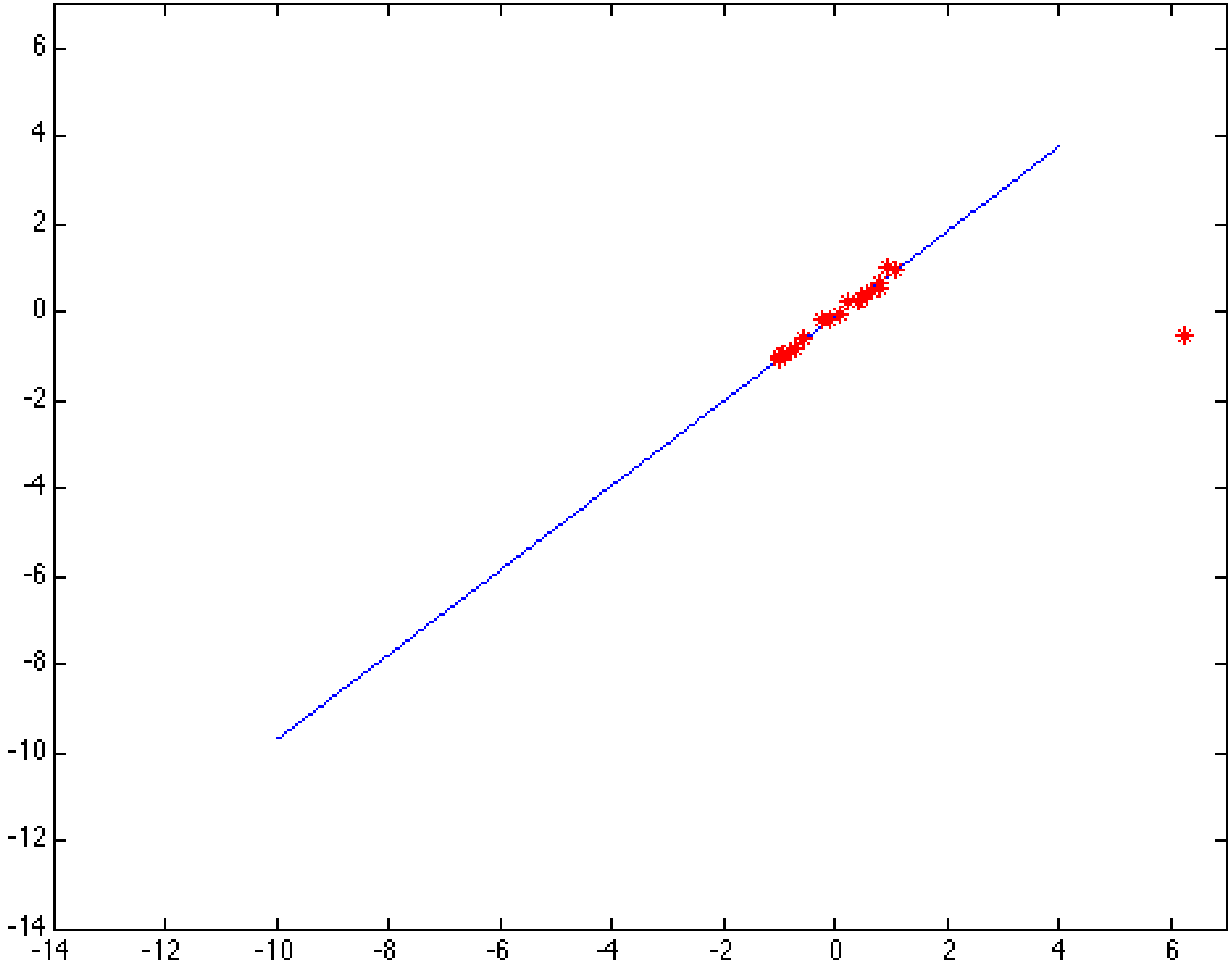
$r_i\,(x_i,\,\theta)$ – residual of ith point w.r.t. model parameters θ
ρ – robust function with scale parameter σ

$$\boxed{\rho(u;\sigma) = \frac{u^2}{\sigma^2 + u^2}}$$



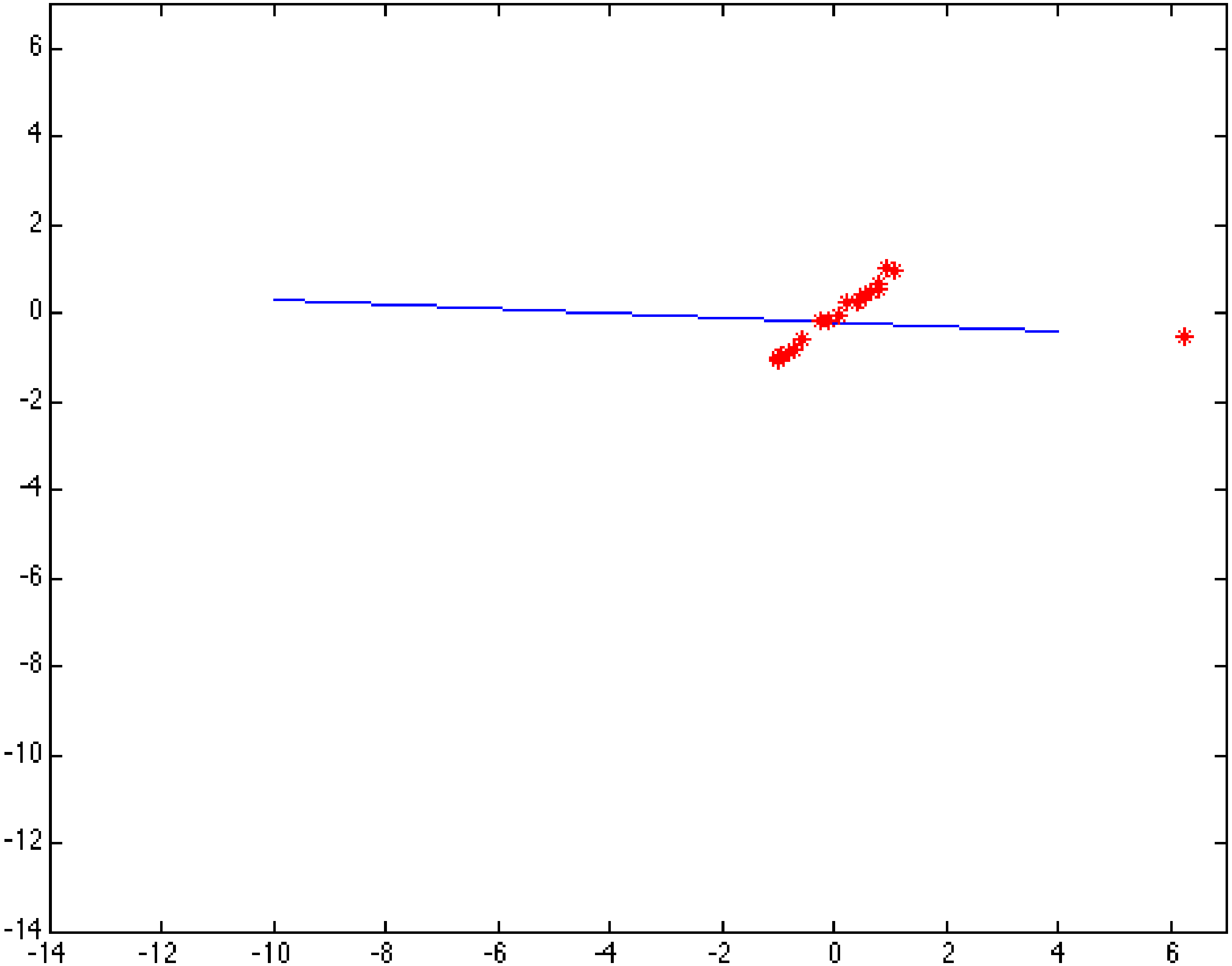
The robust function ρ behaves like squared distance for small values of the residual *u* but saturates for larger values of *u*

Choosing the scale: Just right



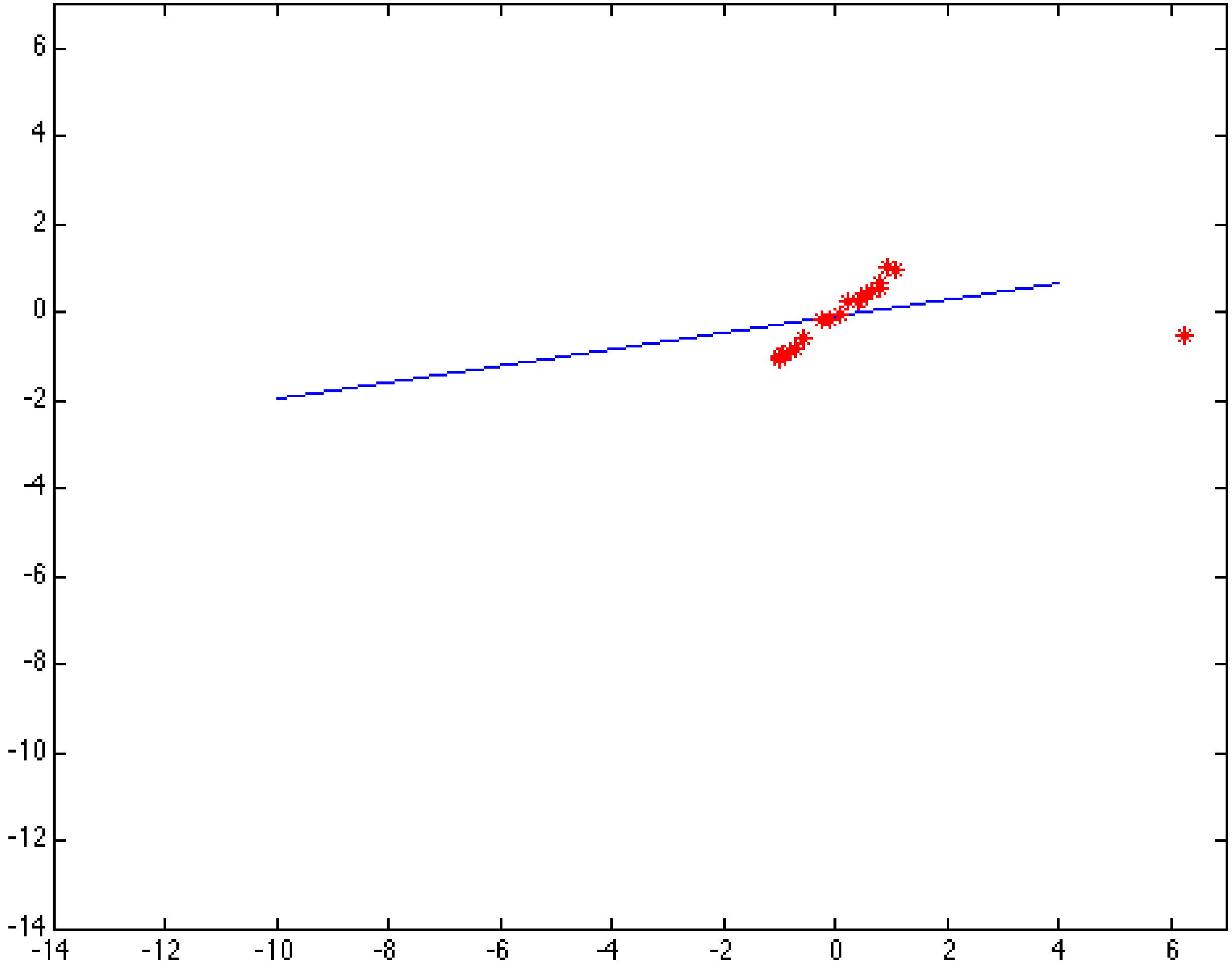
The effect of the outlier is minimized

Choosing the scale: Too small



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

Robust estimation: Details

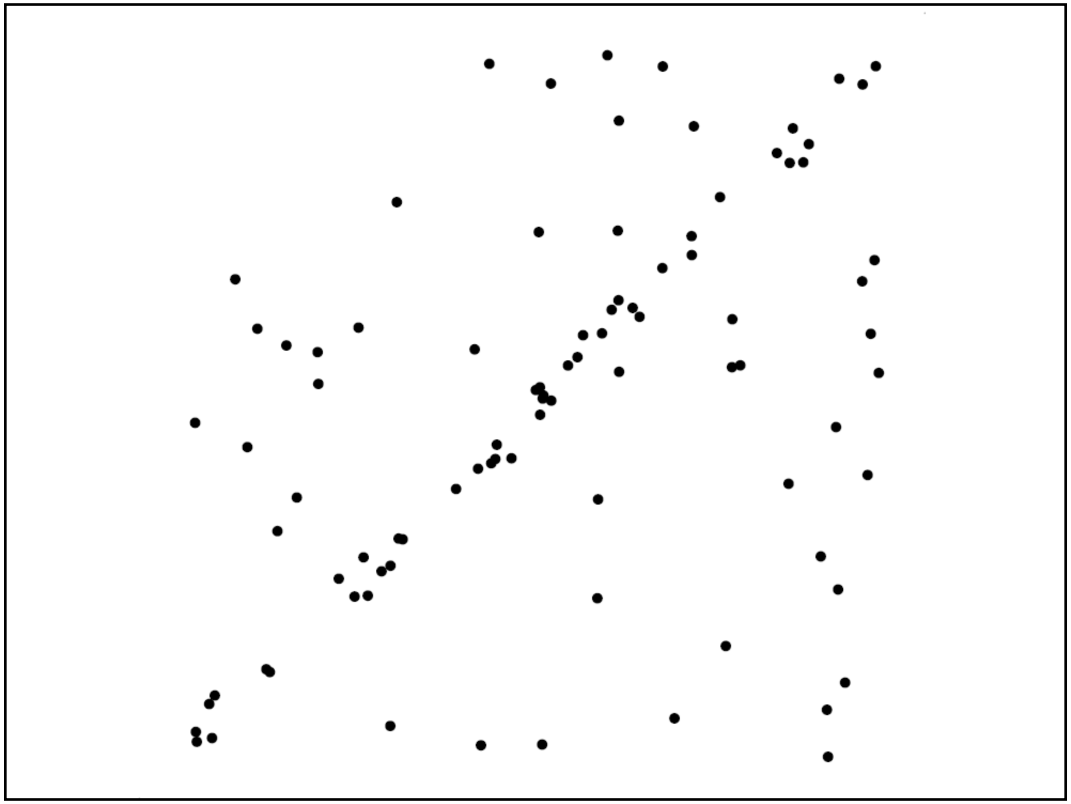
- Robust fitting is a nonlinear optimization problem that must be solved iteratively
- Least squares solution can be used for initialization
- Adaptive choice of scale: approx. 1.5 times median residual (F&P, Sec. 15.5.1)

RANSAC

- Robust fitting can deal with a few outliers – what if we have very many?
- Random sample consensus (RANSAC): Very general framework for model fitting in the presence of outliers
- Outline
  - Choose a small subset of points uniformly at random
  - Fit a model to that subset
  - Find all remaining points that are “close” to the model and reject the rest as outliers
  - Do this many times and choose the best model

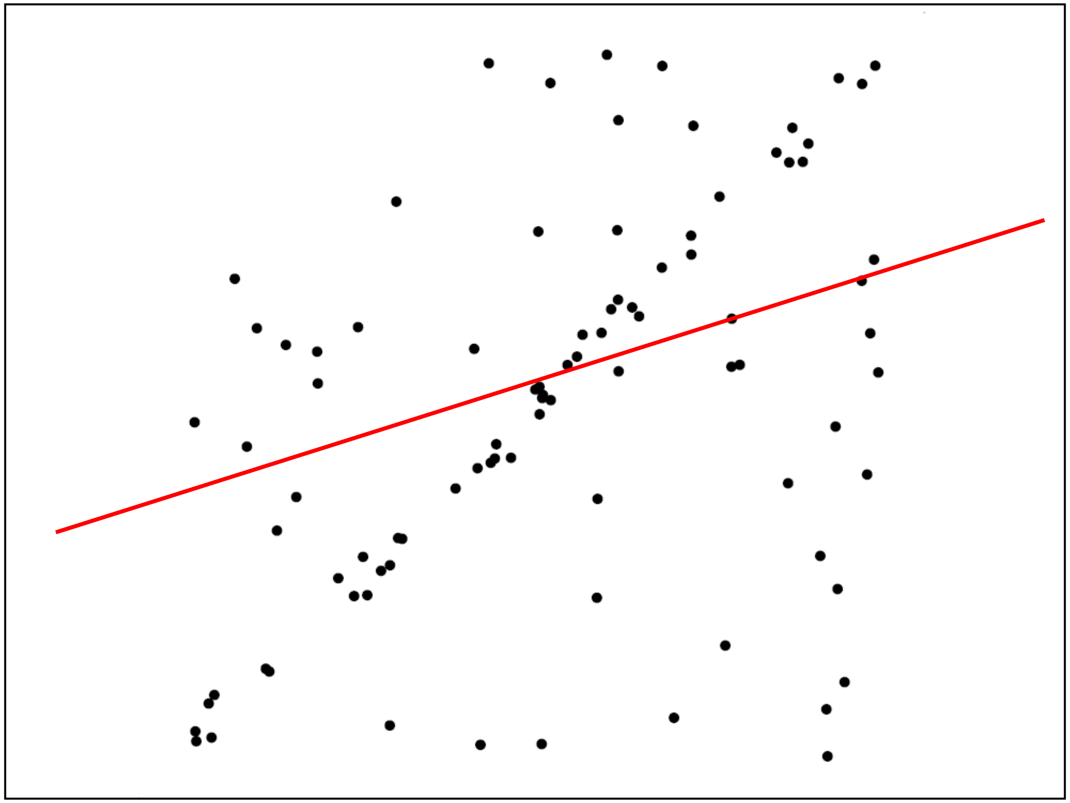
M. A. Fischler, R. C. Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. Comm. of the ACM, Vol 24, pp 381-395, 1981.

RANSAC for line fitting example



Source: R. Raguram

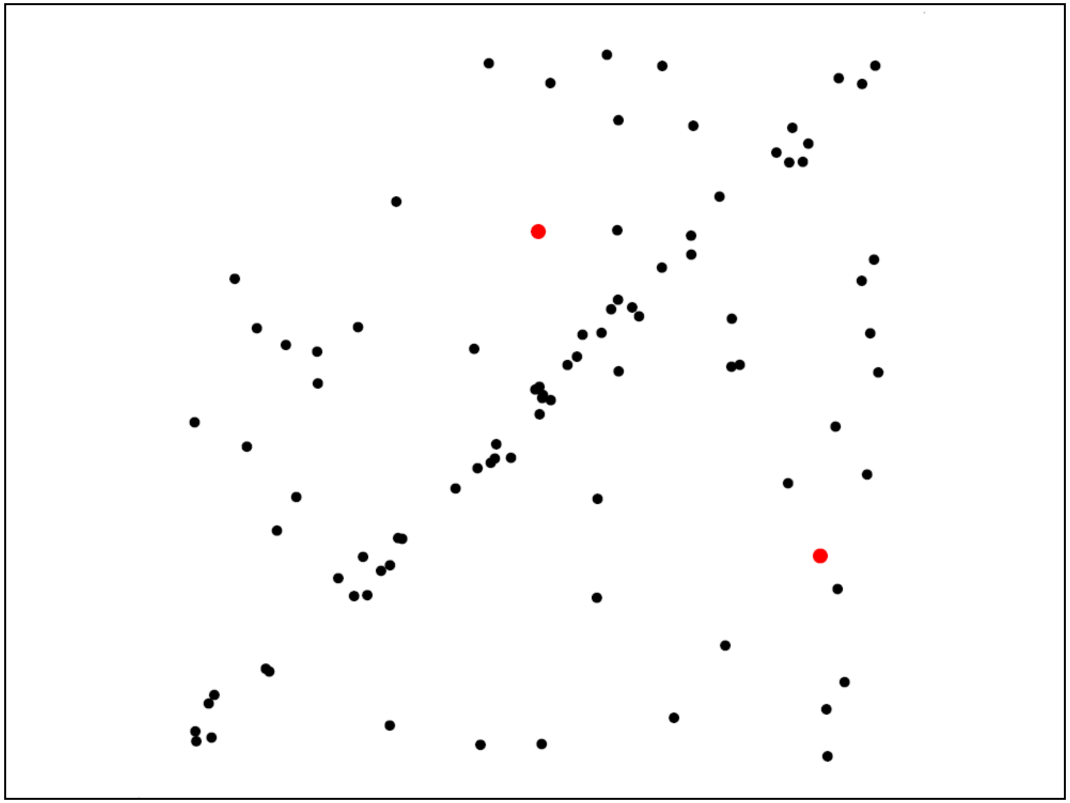
RANSAC for line fitting example



Least-squares fit

Source: R. Raguram

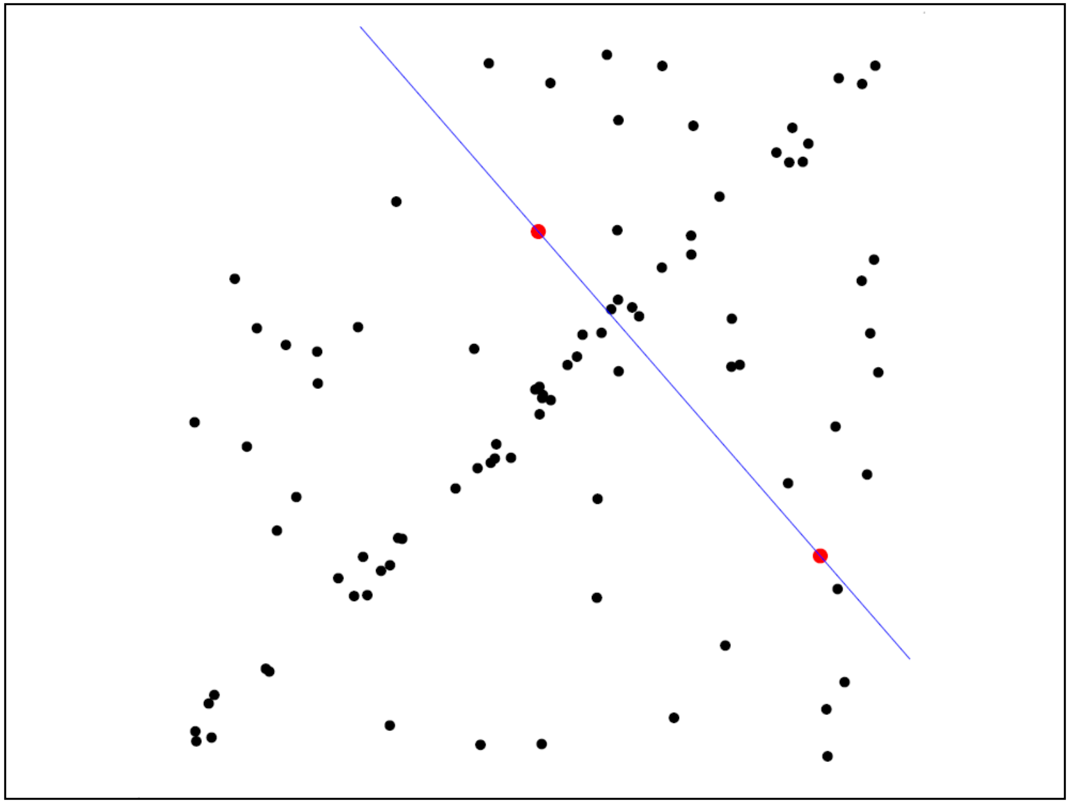
RANSAC for line fitting example



1. Randomly select minimal subset of points

Source: R. Raguram

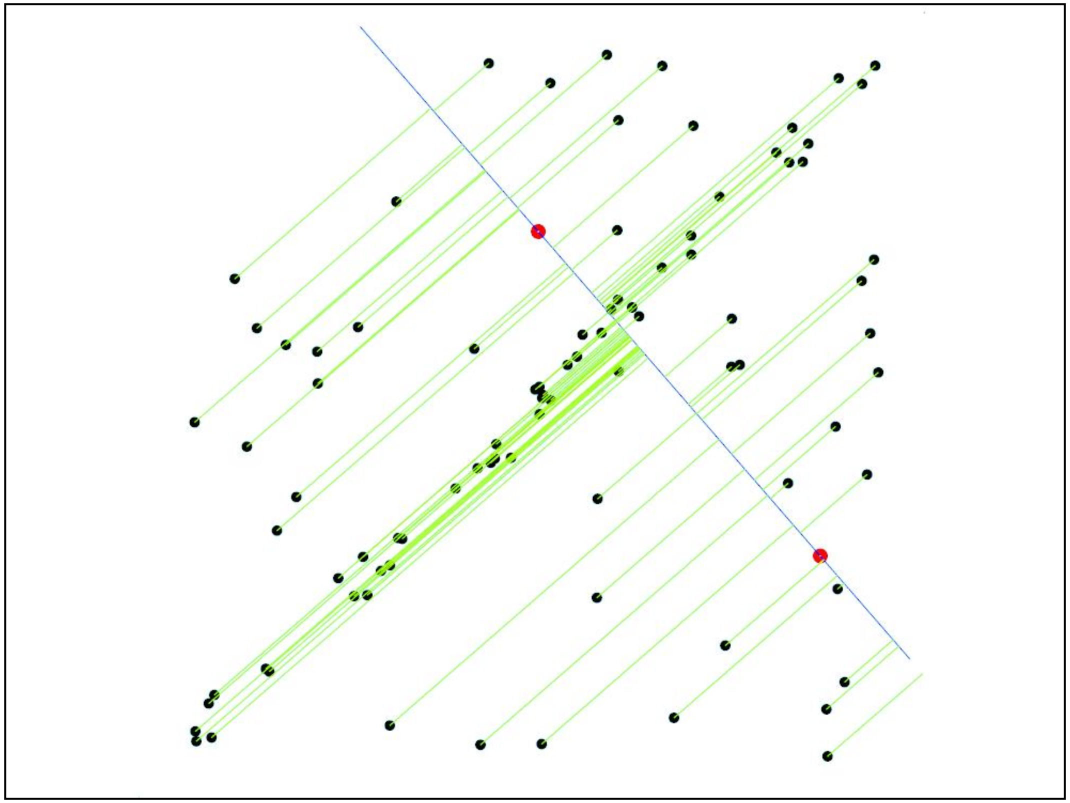
RANSAC for line fitting example



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

Source: R. Raguram

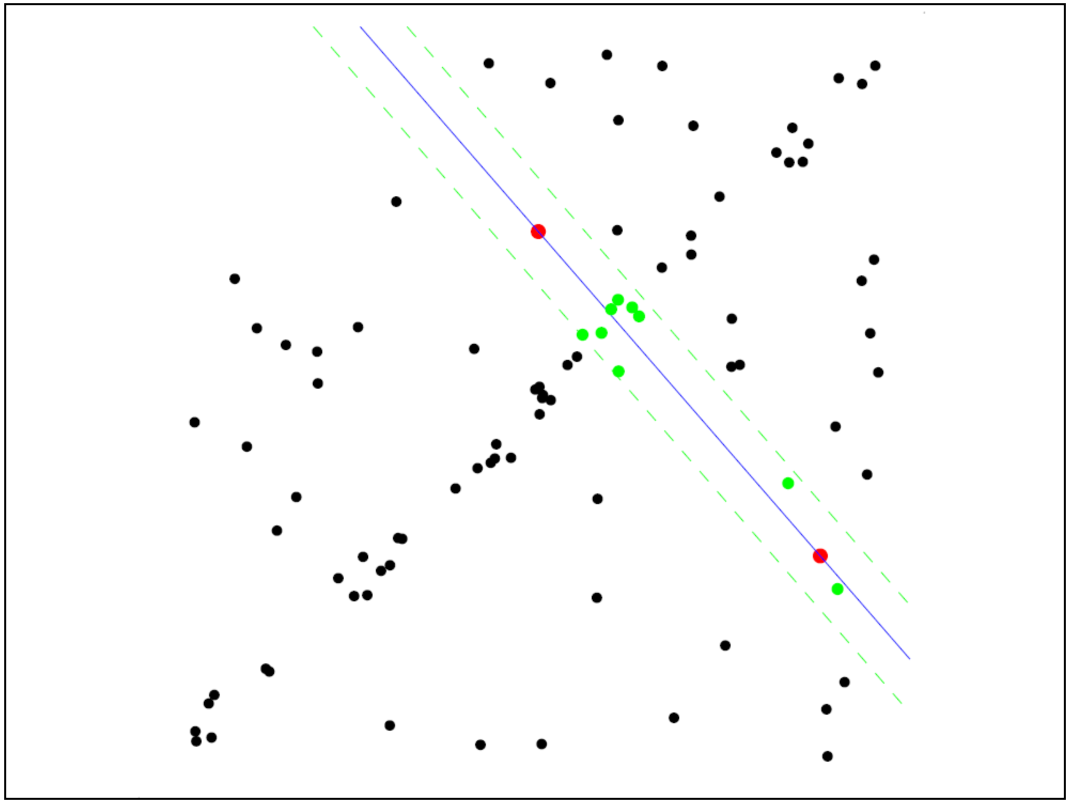
RANSAC for line fitting example



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

Source: R. Raguram

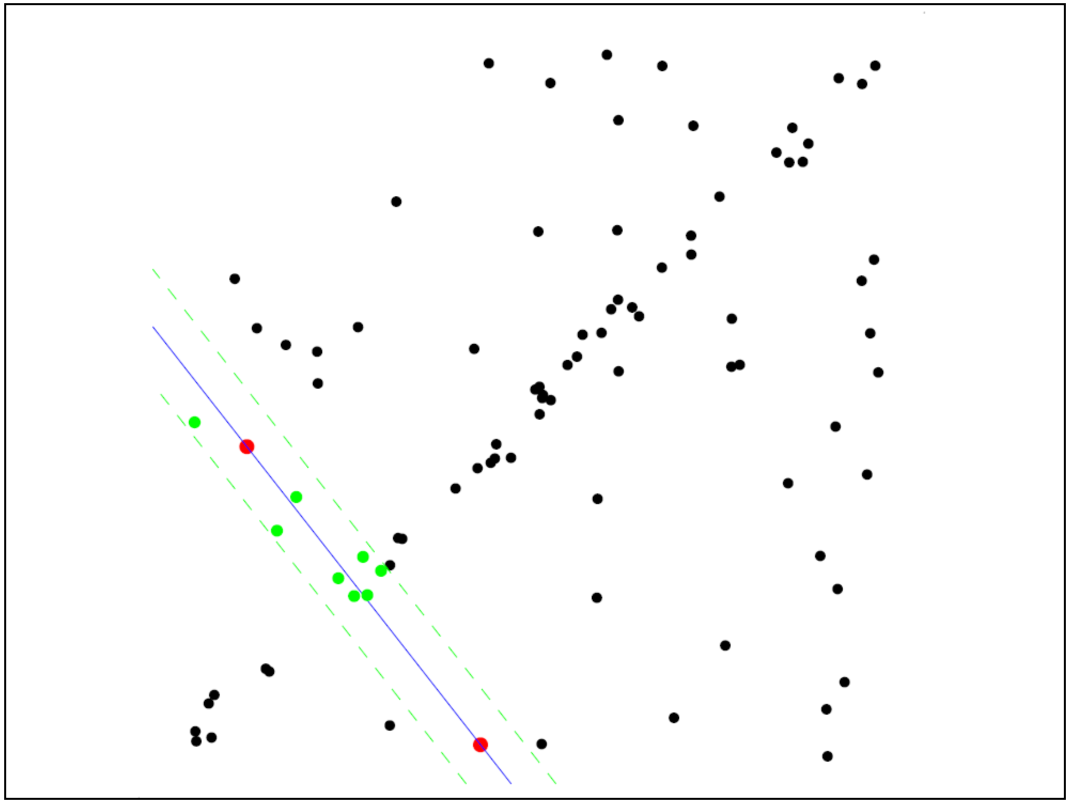
RANSAC for line fitting example



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

Source: R. Raguram

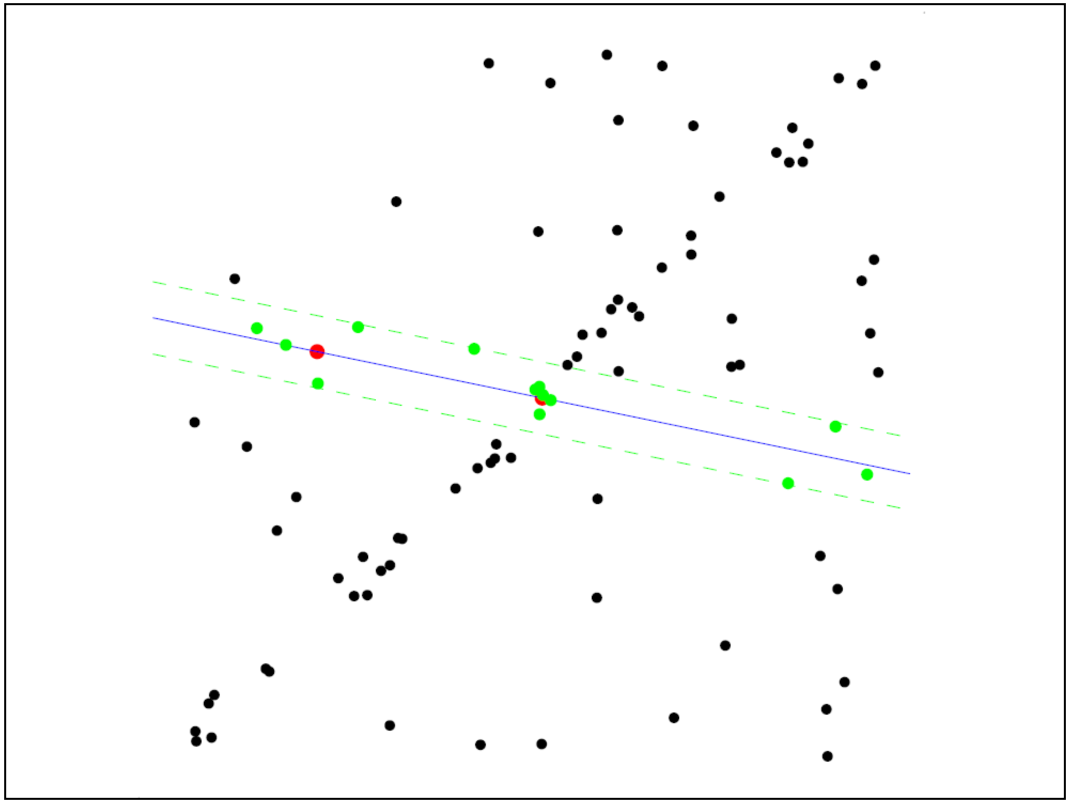
RANSAC for line fitting example



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

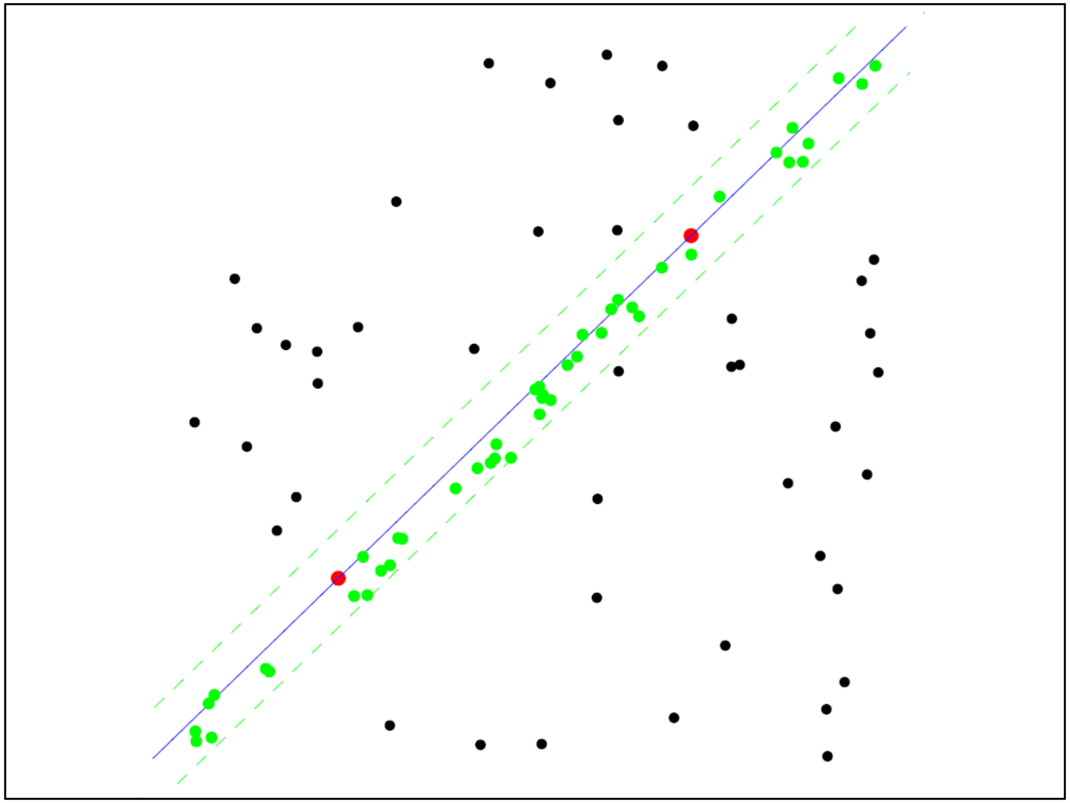


1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model
5. Repeat *hypothesize-and-verify* loop

Source: R. Raguram

RANSAC for line fitting example

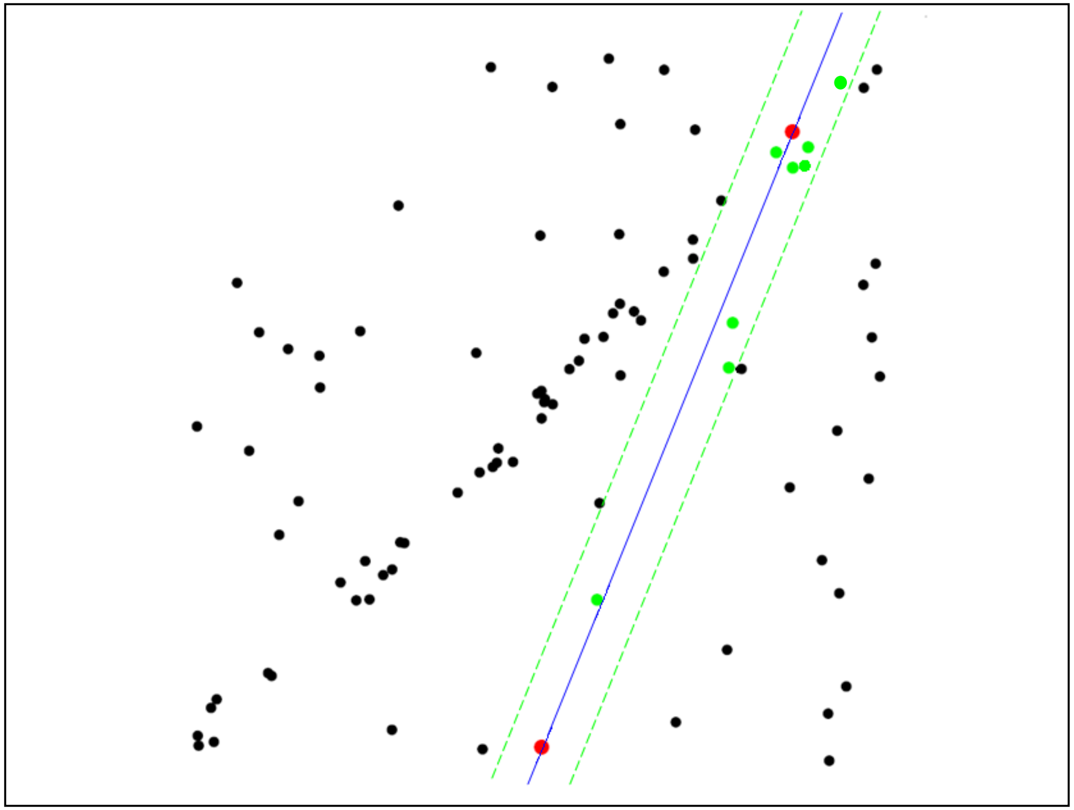
Uncontaminated sample



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 for line fitting

Repeat ***N*** times:

- Draw ***s*** points uniformly at random
- Fit line to these ***s*** points
- Find inliers to this line among the remaining points (i.e., points whose distance from the line is less than ***t***)
- If there are ***d*** or more inliers, accept the line and refit using all inliers

Choosing the parameters

- Initial number of points *s*
  - Typically minimum number needed to fit the model
- Distance threshold *t*
  - Choose *t* so probability for inlier is *p* (e.g. 0.95)
  - Zero-mean Gaussian noise with std. dev. σ: $t^2=3.84\sigma^2$
- Number of samples *N*
  - Choose *N* so that, with probability *p*, at least one random sample is free from outliers (e.g. *p*=0.99) (outlier ratio: *e*)

Source: M. Pollefeys

Choosing the parameters

- Initial number of points *s*
  - Typically minimum number needed to fit the model
- Distance threshold *t*
  - Choose *t* so probability for inlier is *p* (e.g. 0.95)
  - Zero-mean Gaussian noise with std. dev. σ: $t^2=3.84\sigma^2$
- Number of samples *N*
  - Choose *N* so that, with probability *p*, at least one random sample is free from outliers (e.g. *p*=0.99) (outlier ratio: *e*)

$$\boxed{\left(1-\left(1-e\right)^s\right)^N = 1-p}$$

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

	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

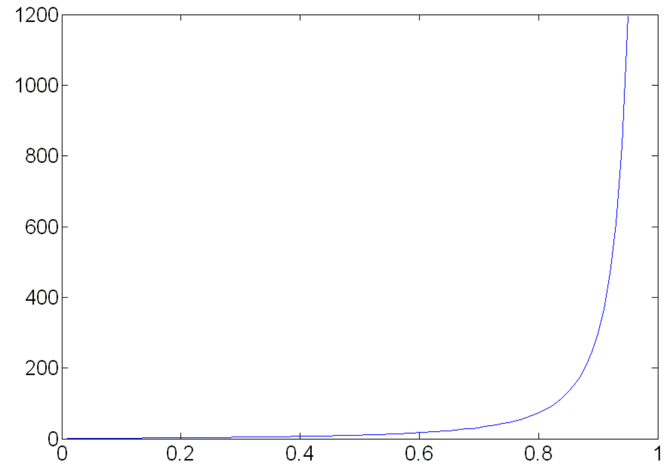
Source: M. Pollefeys

Choosing the parameters

- Initial number of points *s*
  - Typically minimum number needed to fit the model
- Distance threshold *t*
  - Choose *t* so probability for inlier is *p* (e.g. 0.95)
  - Zero-mean Gaussian noise with std. dev. σ: $t^2=3.84\sigma^2$
- Number of samples *N*
  - Choose *N* so that, with probability *p*, at least one random sample is free from outliers (e.g. *p*=0.99) (outlier ratio: *e*)

$$\boxed{\left(1-\left(1-e\right)^s\right)^N = 1-p}$$

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



Source: M. Pollefeys

Choosing the parameters

- Initial number of points *s*
  - Typically minimum number needed to fit the model
- Distance threshold *t*
  - Choose *t* so probability for inlier is *p* (e.g. 0.95)
  - Zero-mean Gaussian noise with std. dev. σ: $t^2=3.84\sigma^2$
- Number of samples *N*
  - Choose *N* so that, with probability *p*, at least one random sample is free from outliers (e.g. *p*=0.99) (outlier ratio: *e*)
- Consensus set size *d*
  - Should match expected inlier ratio

Source: M. Pollefeys

Adaptively determining the number of samples

- Inlier ratio *e* is often unknown a priori, so pick worst case, e.g. 50%, and adapt if more inliers are found, e.g. 80% would yield *e*=0.2
- Adaptive procedure:
  - $N=\infty$, *sample_count* =0
  - While *N* >*sample_count*
    - Choose a sample and count the number of inliers
    - Set e = 1 – (number of inliers)/(total number of points)
    - Recompute *N* from *e:*

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

    - Increment the *sample_count* by 1

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 tune
  - Doesn’t work well for low inlier ratios (too many iterations, or can fail completely)
  - Can’t always get a good initialization of the model based on the minimum number of samples