Model fitting

Hough transform
RANSAC
EM

Schedule (tentative)

#	date	topic
1	Sep.19	Introduction and geometry
2	Sep.26	Camera models and calibration
3	Oct.3	Invariant features
4	Oct.10	Optical flow & Particle Filters
5	Oct.17	Multiple-view geometry
6	Oct.24	**Model fitting (RANSAC, EM, …)**
7	Oct.31	Image segmentation
8	Nov.7	Stereo Matching & MVS
9	Nov.14	Structure-from-Motion & SLAM
10	Nov.21	Specific object recognition
11	Nov.28	Shape from X
12	Dec.5	Object category recognition
13	Dec.12	Tracking
14	Dec.19	Research Overview & Lab tours

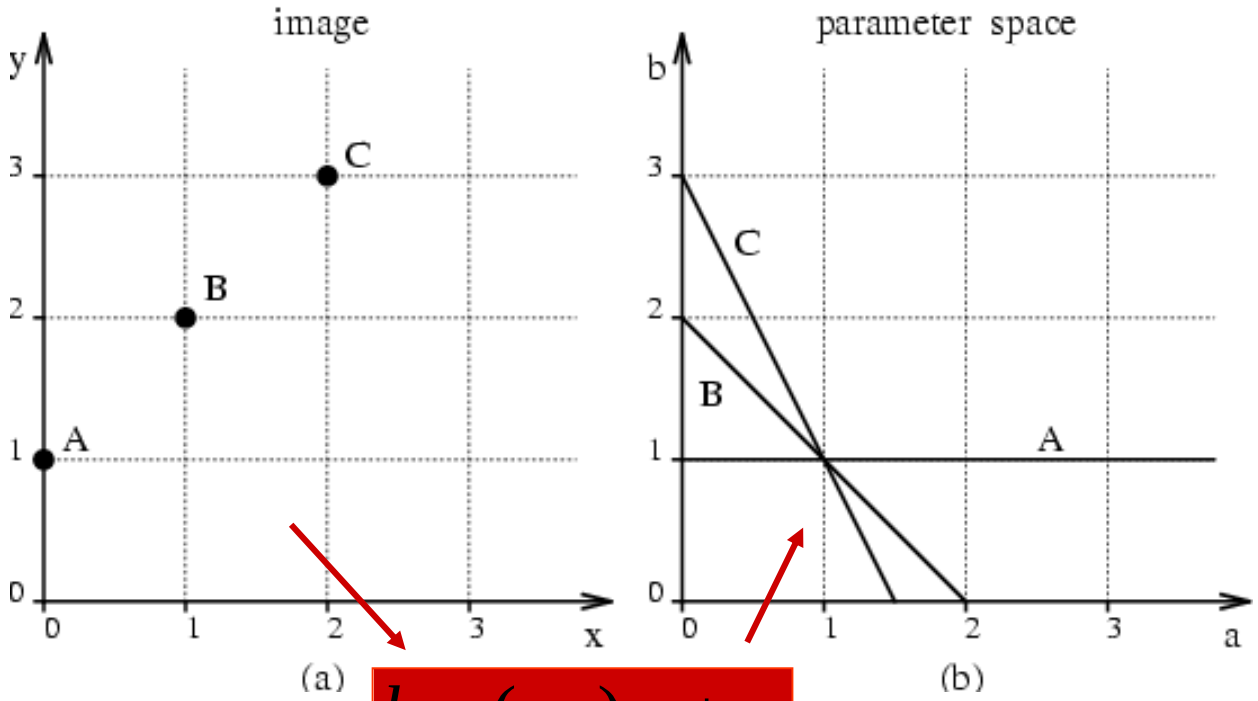
Fitting

- Choose a parametric object/some objects to represent a set of tokens
- Most interesting case is when criterion is not local
  - can't tell whether a set of points lies on a line by looking only at each point and the next.

- Three main questions:
  - what object represents this set of tokens best?
  - which of several objects gets which token?
  - how many objects are there?

(you could read line for object here, or circle, or ellipse or...)

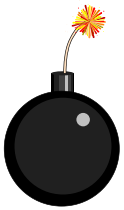
Hough transform : straight lines



implementation :

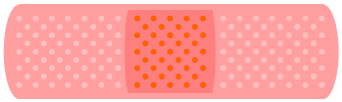
1. the parameter space is discretised
2. a counter is incremented at each cell where the lines pass
3. peaks are detected

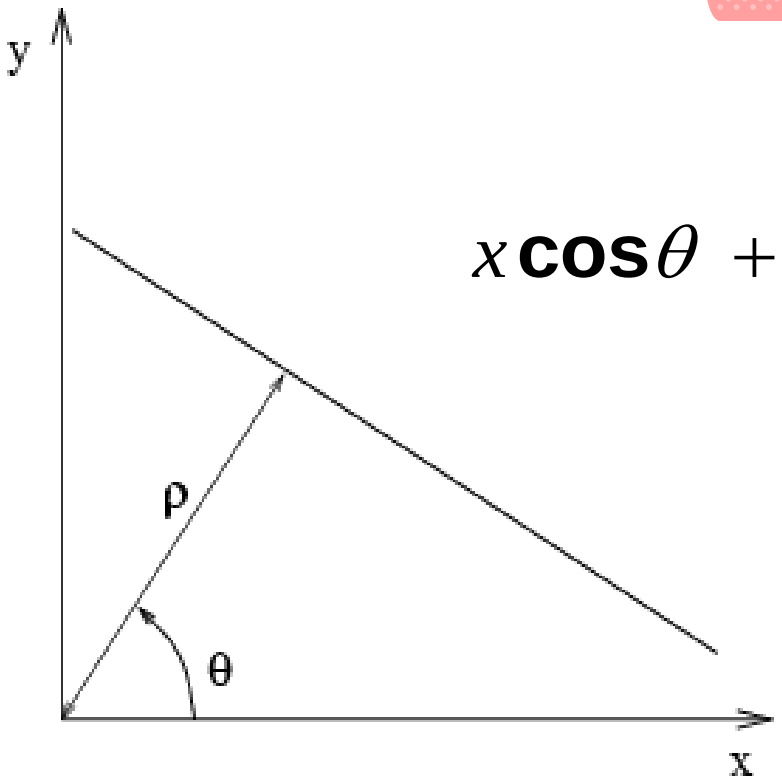
Hough transform : straight lines



problem : unbounded parameter domain
vertical lines require infinite *a*

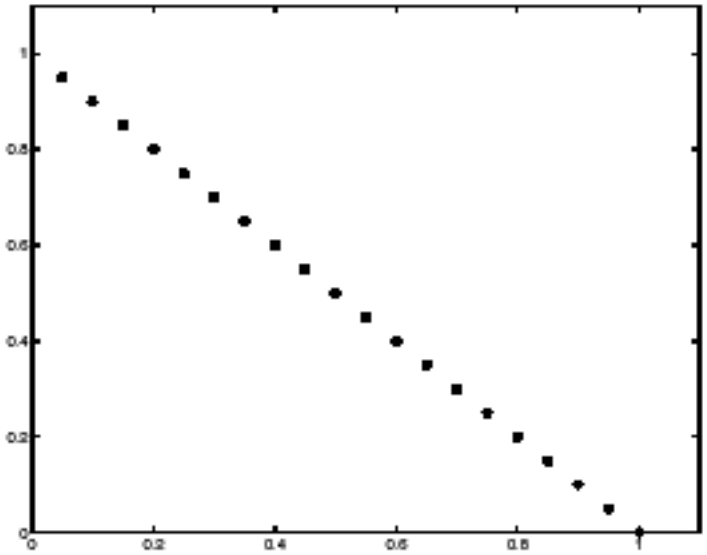
alternative representation:



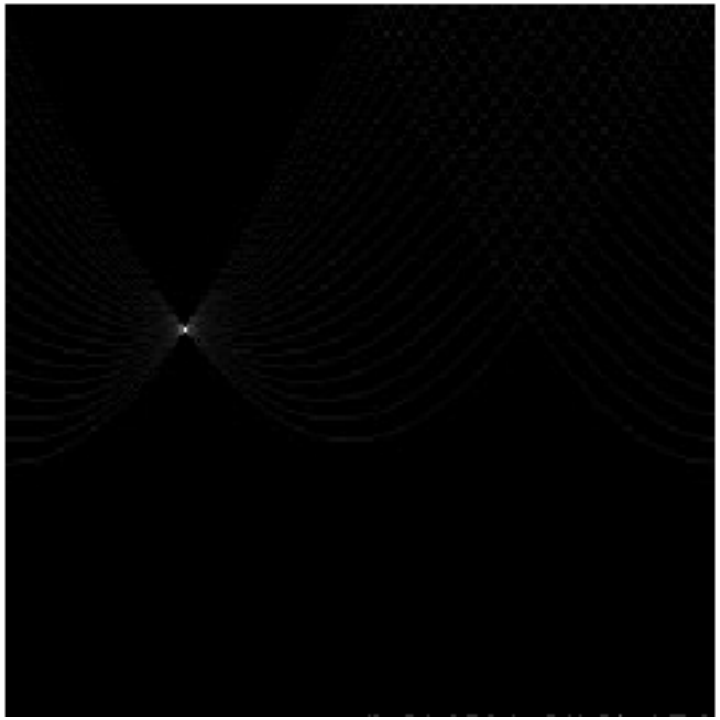


$$x\cos\theta + y\sin\theta = \rho$$

Each point will add a cosine function in the (θ,ρ) parameter space



tokens

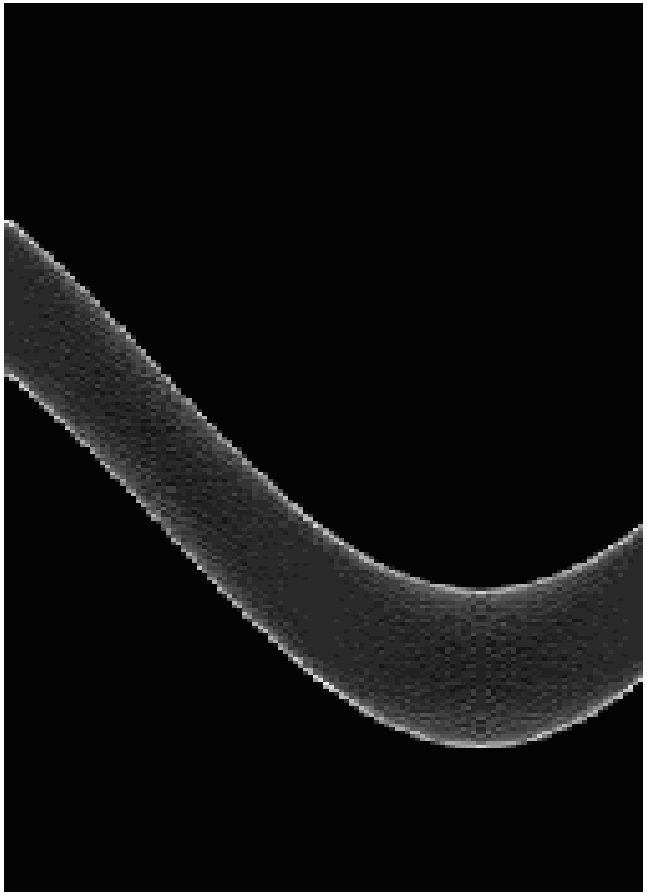


votes

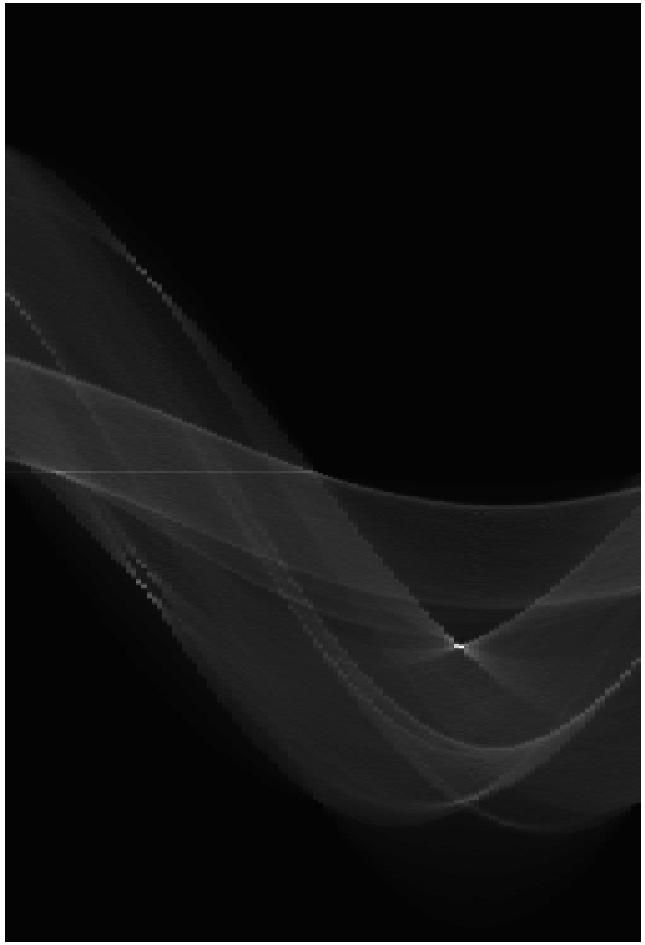
Hough transform : straight lines

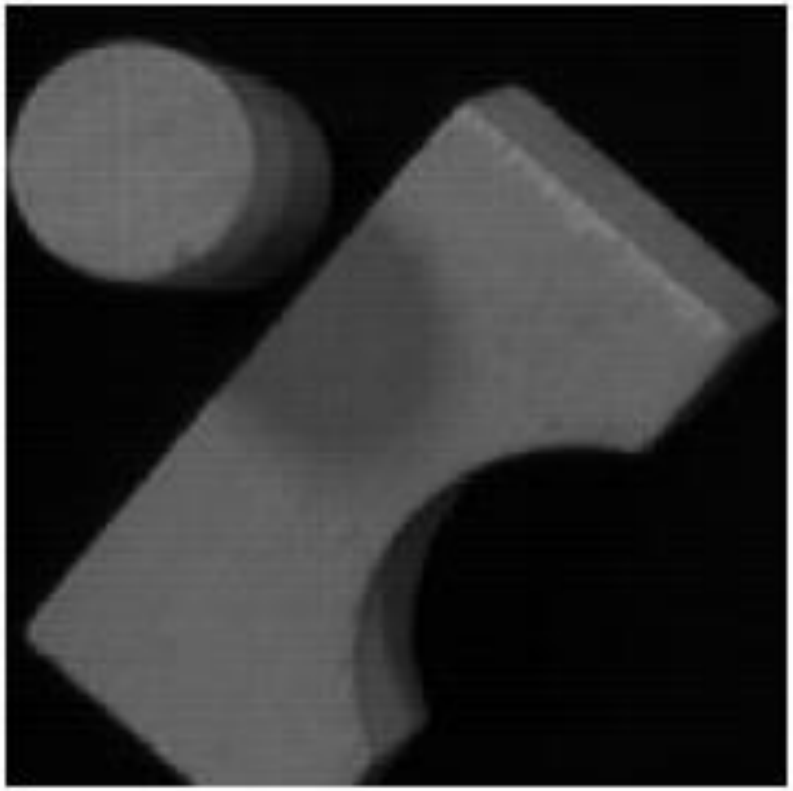
Square :

Circle :



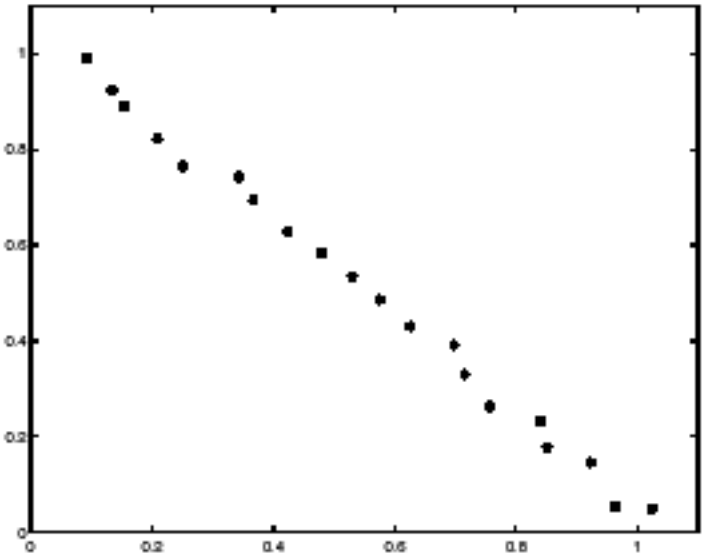
Hough transform : straight lines

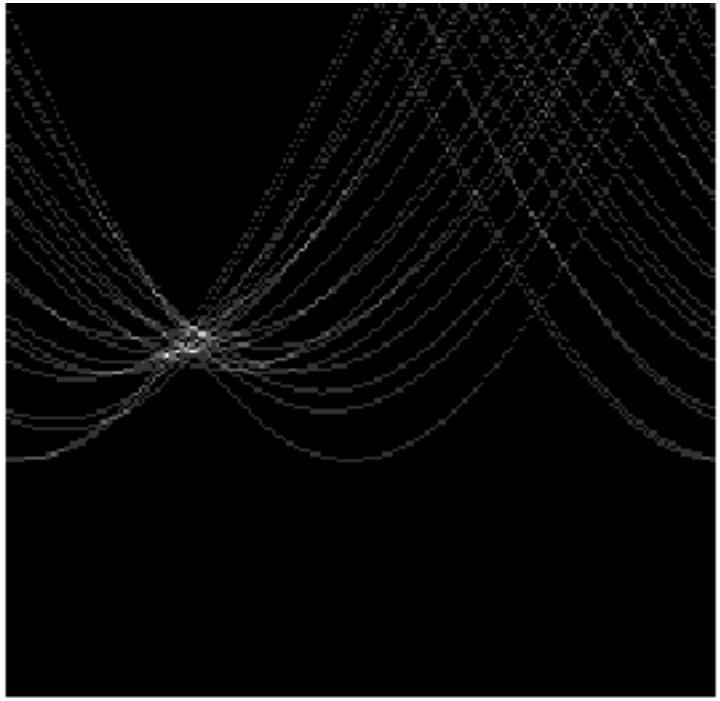




Mechanics of the Hough transform

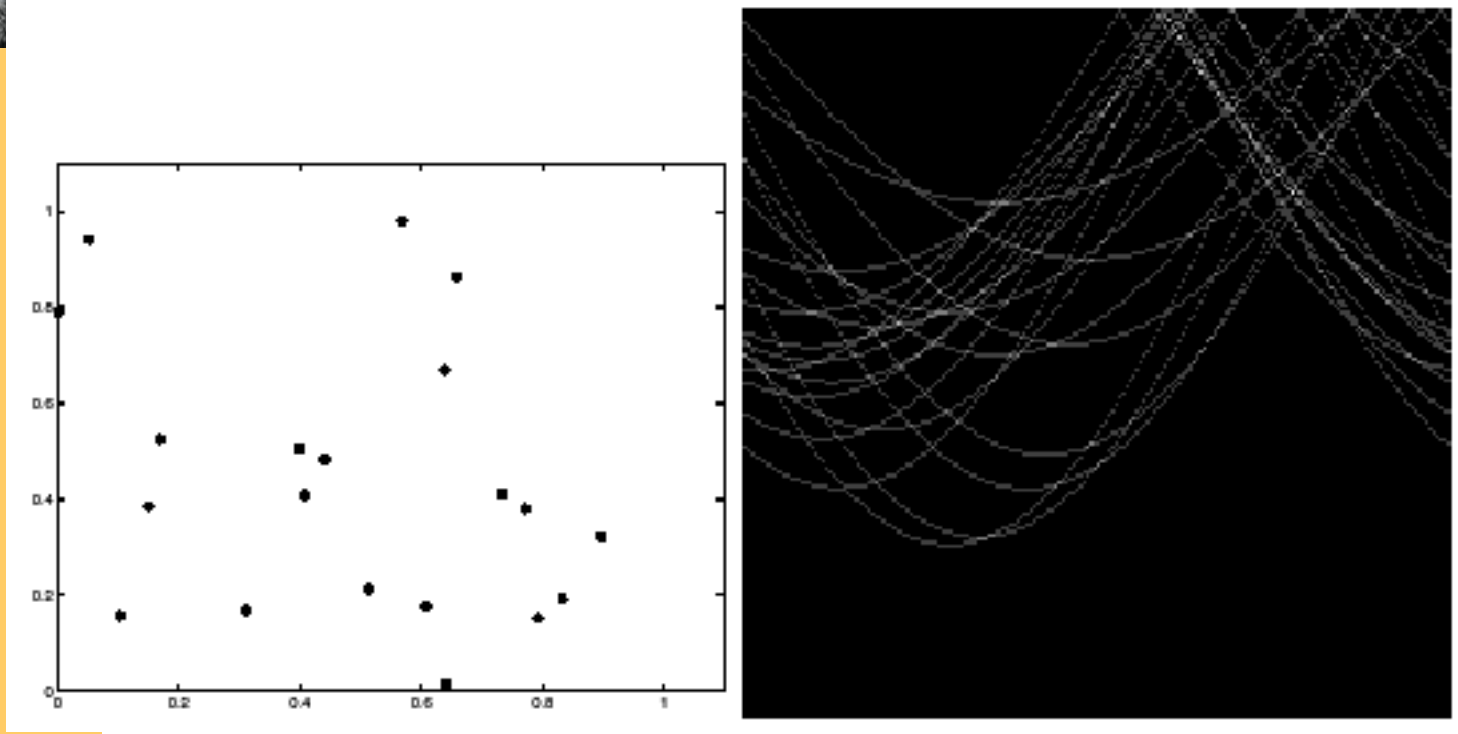
- Construct an array representing θ, d
- For each point, render the curve (θ, d) into this array, adding one at each cell
- Difficulties
  - how big should the cells be? (too big, and we cannot distinguish between quite different lines; too small, and noise causes lines to be missed)
- How many lines?
  - count the peaks in the Hough array
- Who belongs to which line?
  - tag the votes
- Hardly ever satisfactory in practice, because problems with noise and cell size defeat it

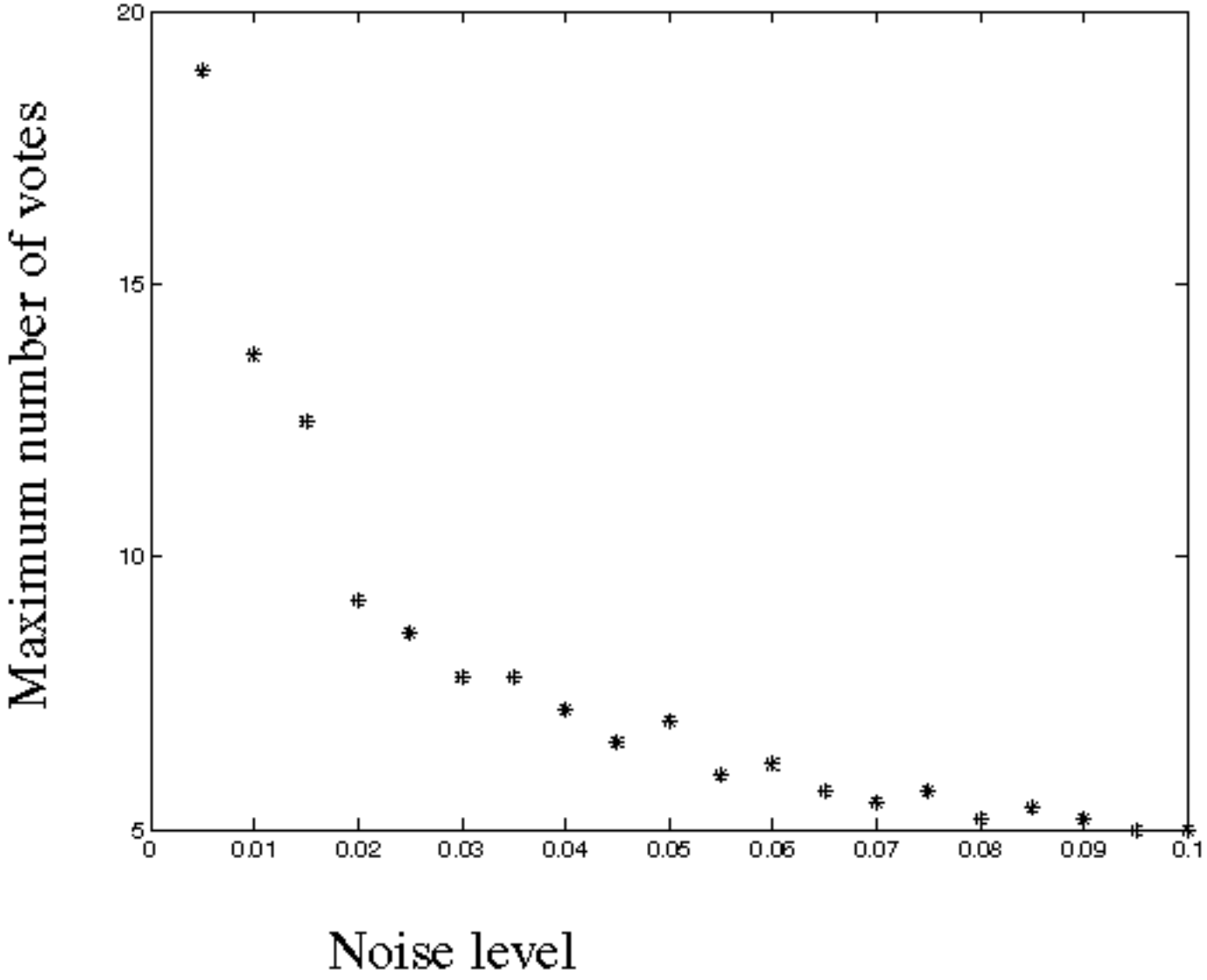


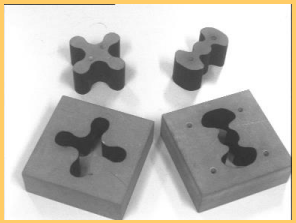


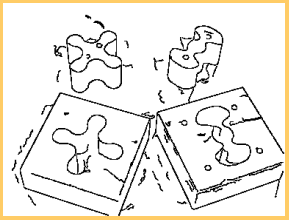
tokens

votes


0
0.2
0.4
0.6
0.8
1
0
0.2
0.4
0.6
0.8
1

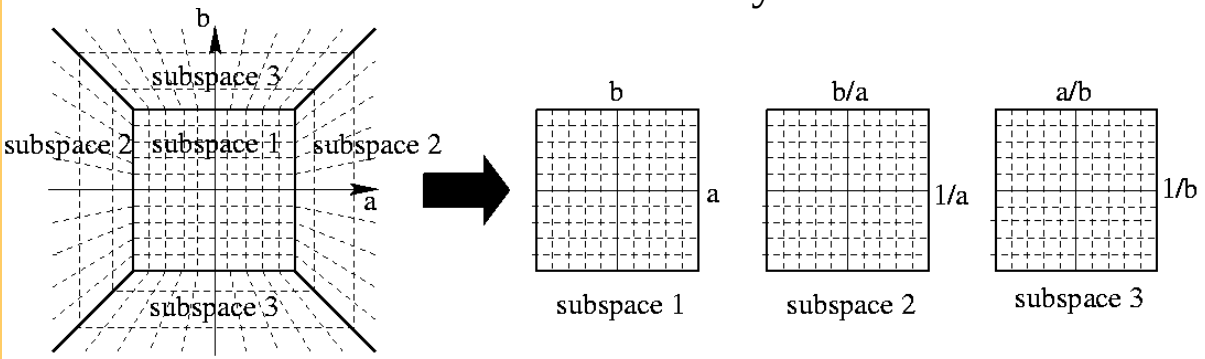

Maximum number of votes
20
15
10
5
0
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
0.09
0.1
Noise level

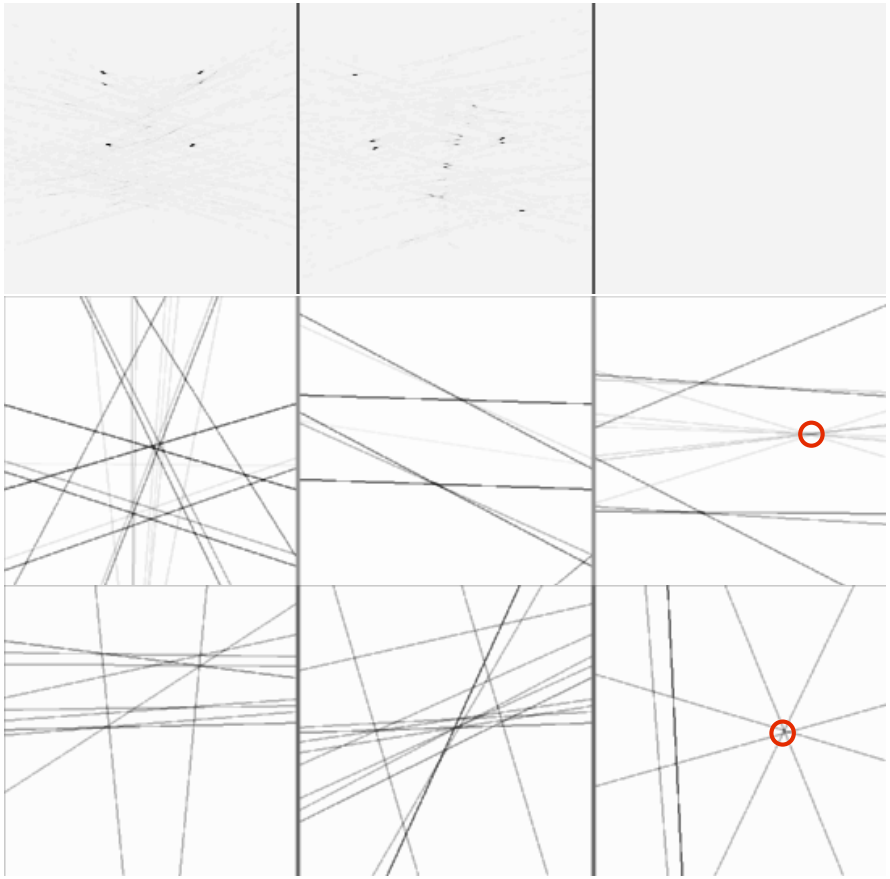


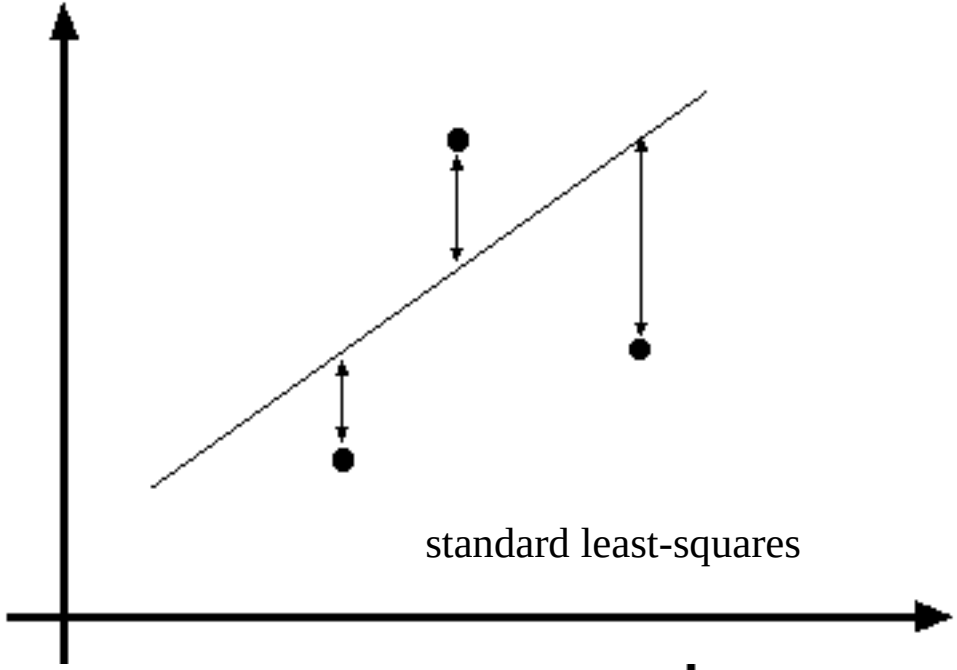


Cascaded hough transform

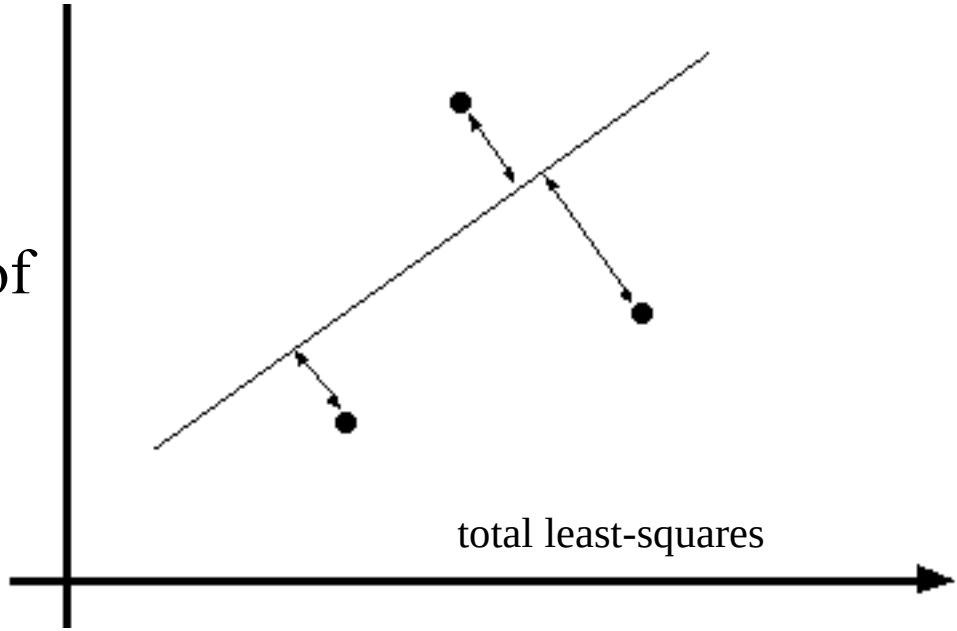
Tuytelaars and Van Gool ICCV ‘98







Line fitting can be max. likelihood - but choice of model is important



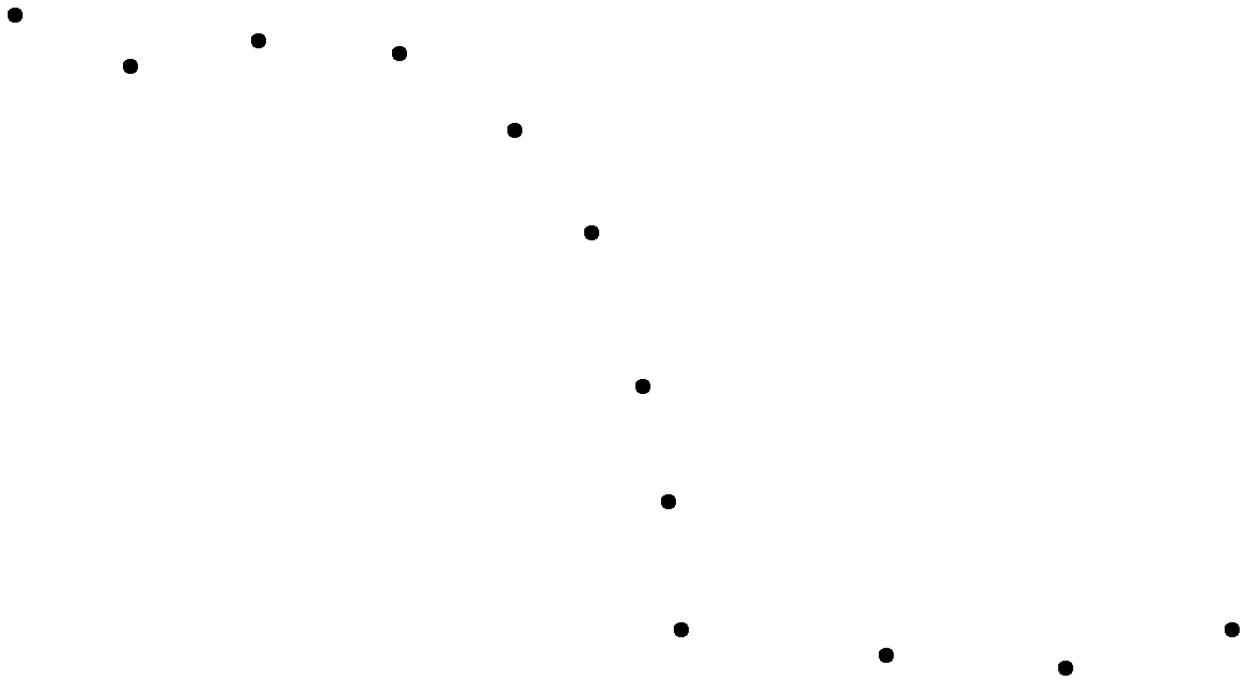
Who came from which line?

- Assume we know how many lines there are - but which lines are they?
  - easy, if we know who came from which line
- Three strategies
  - Incremental line fitting
  - K-means
  - Probabilistic (later!)

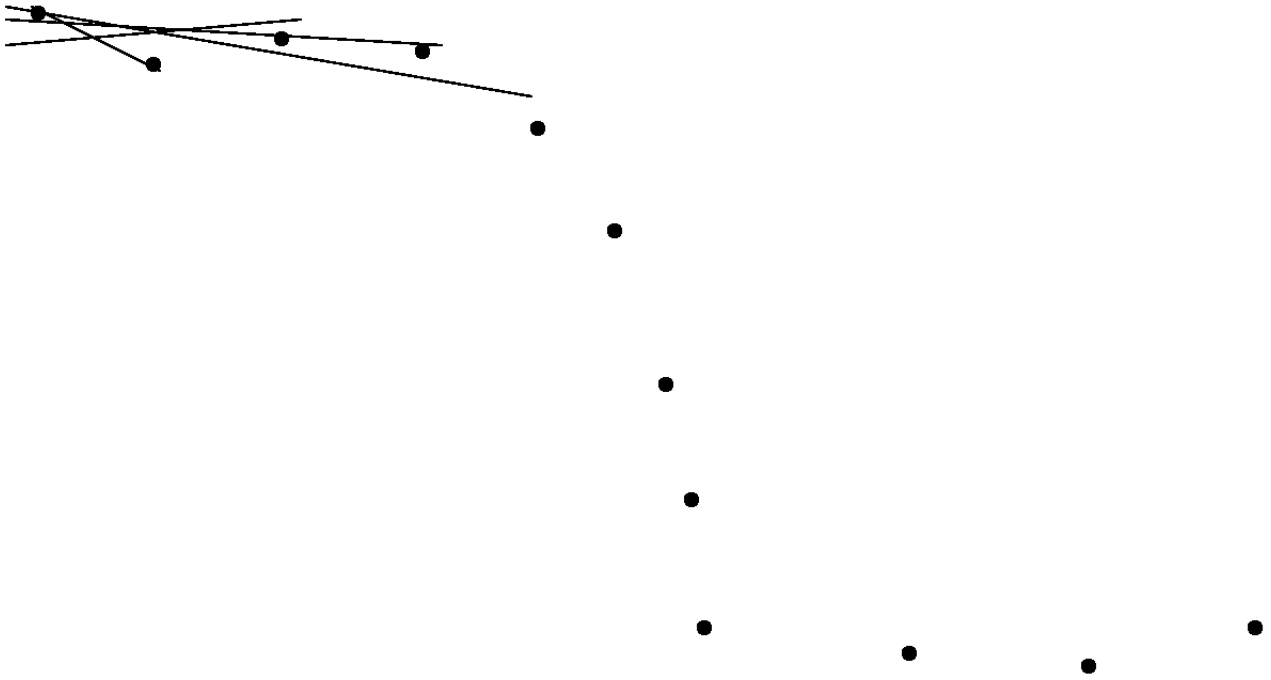
Algorithm 15.1: Incremental line fitting by walking along a curve, fitting a line to runs of pixels along the curve, and breaking the curve when the residual is too large

```
Put all points on curve list, in order along the curve
Empty the line point list
Empty the line list
Until there are too few points on the curve
   Transfer first few points on the curve to the line point list
   Fit line to line point list
   While fitted line is good enough
      Transfer the next point on the curve
         to the line point list and refit the line
   end
   Transfer last point(s) back to curve
   Refit line
   Attach line to line list
end
```

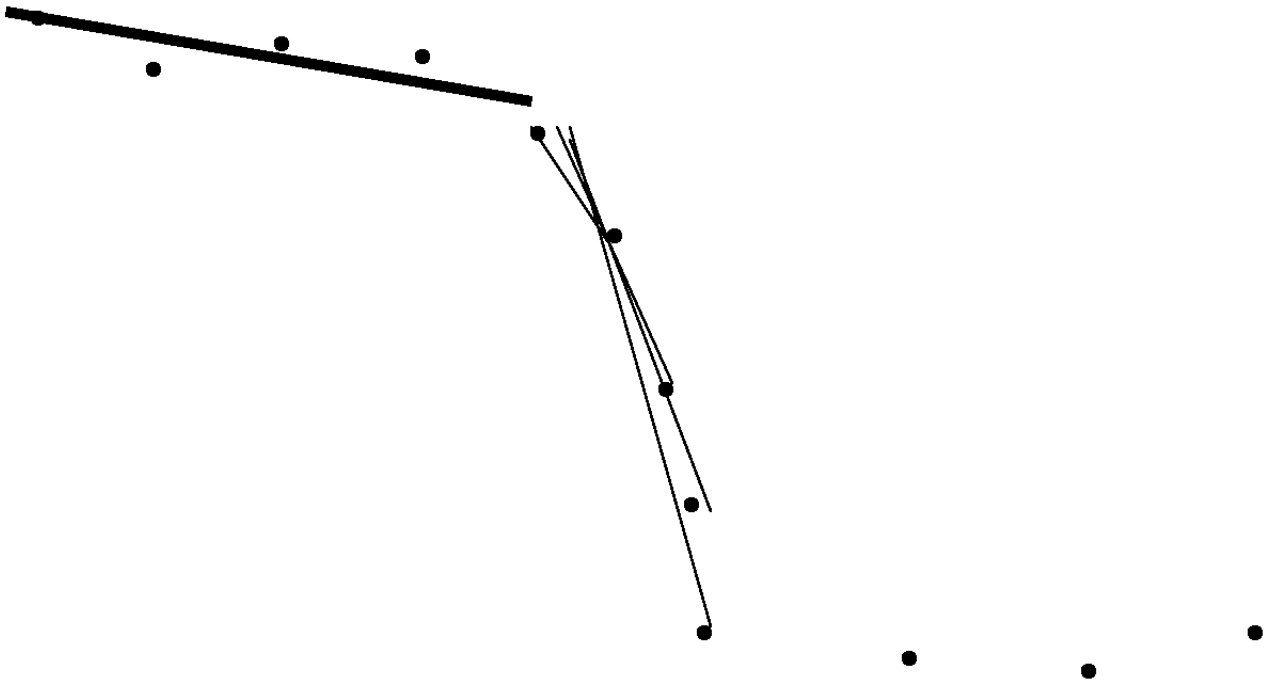
Incremental line fitting



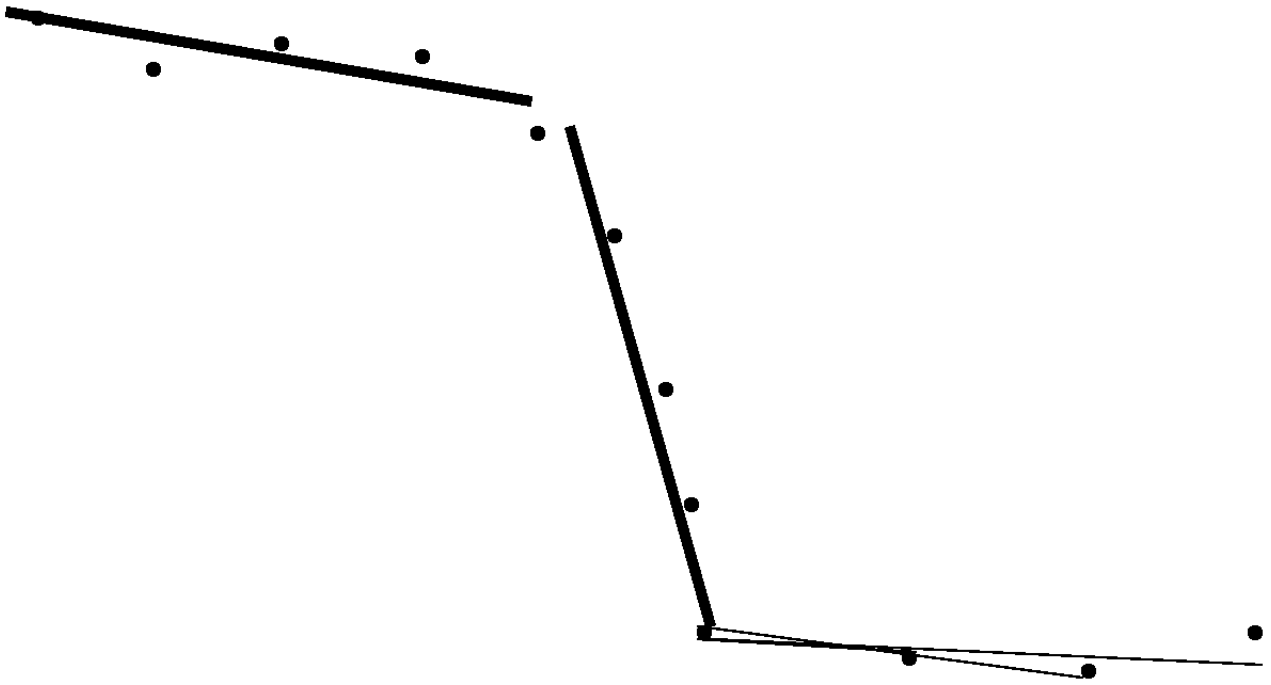
Incremental line fitting



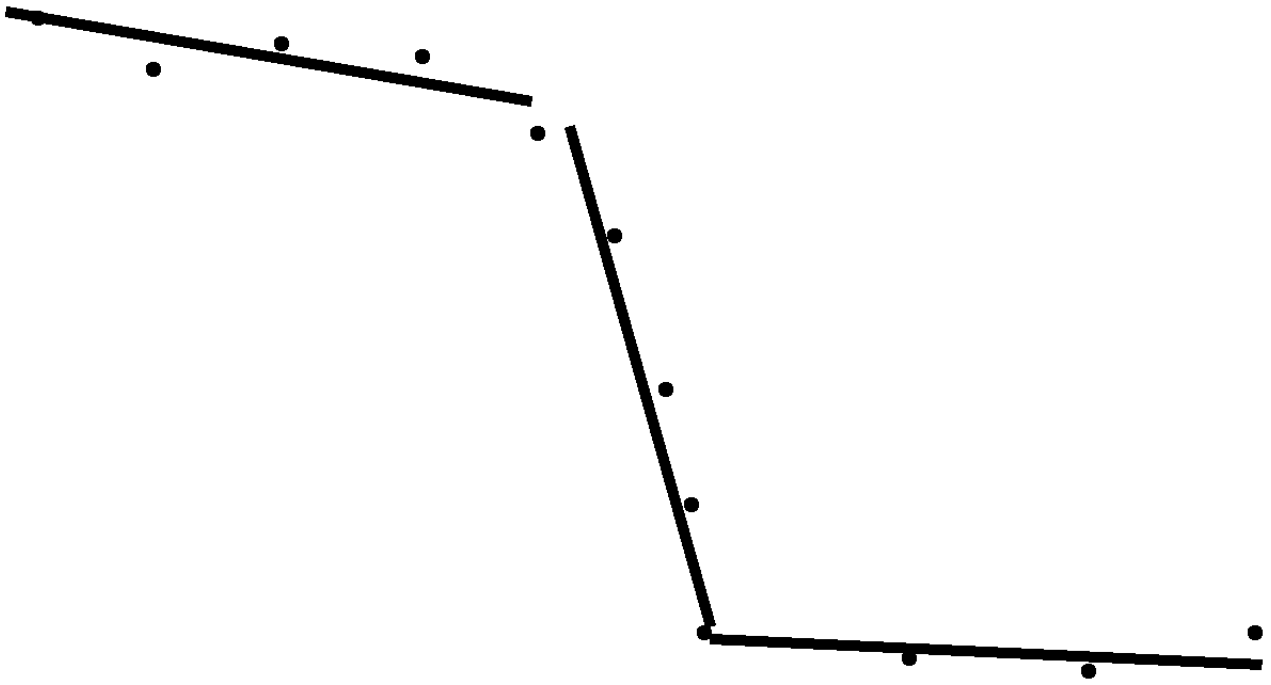
Incremental line fitting



Incremental line fitting



Incremental line fitting

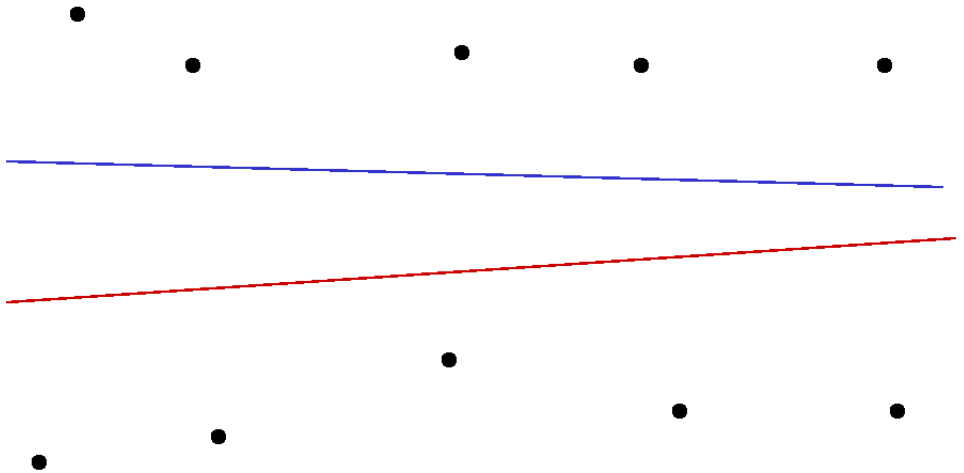


Algorithm 15.2: K-means line fitting by allocating points to the closest line and then refitting.

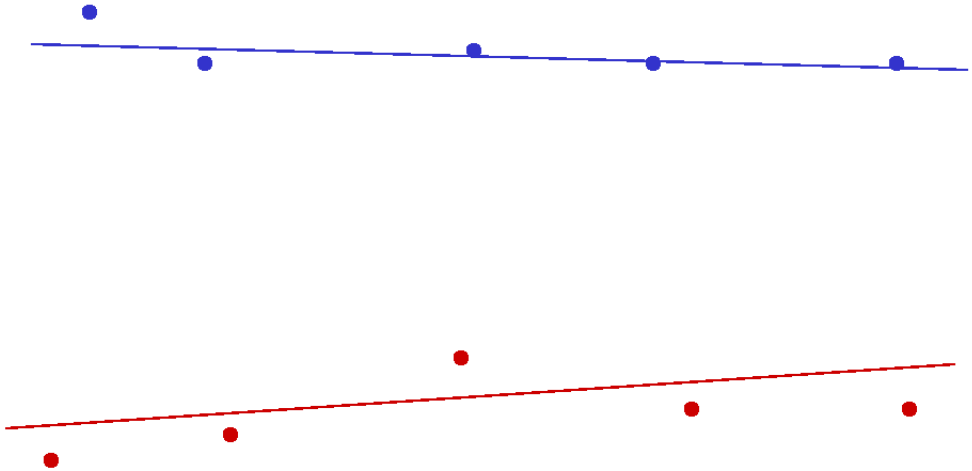
Hypothesize k lines (perhaps uniformly at random)
or
Hypothesize an assignment of lines to points
  and then fit lines using this assignment

Until convergence
  Allocate each point to the closest line
  Refit lines
end

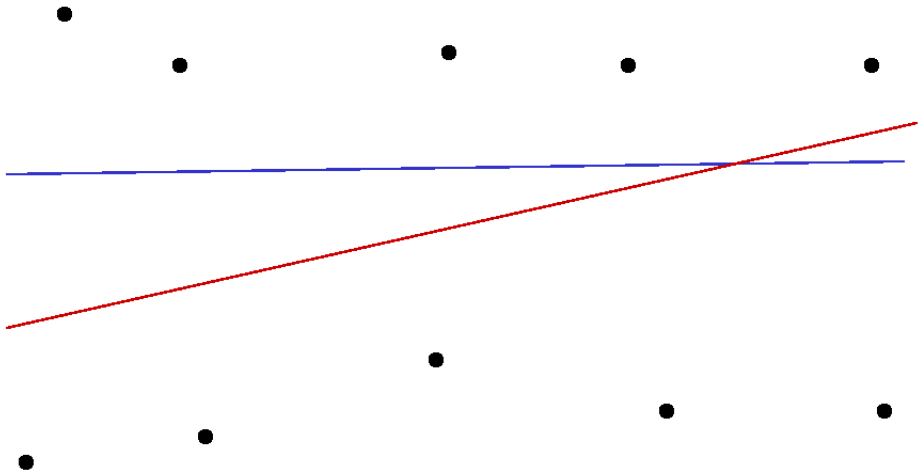
K-means line fitting



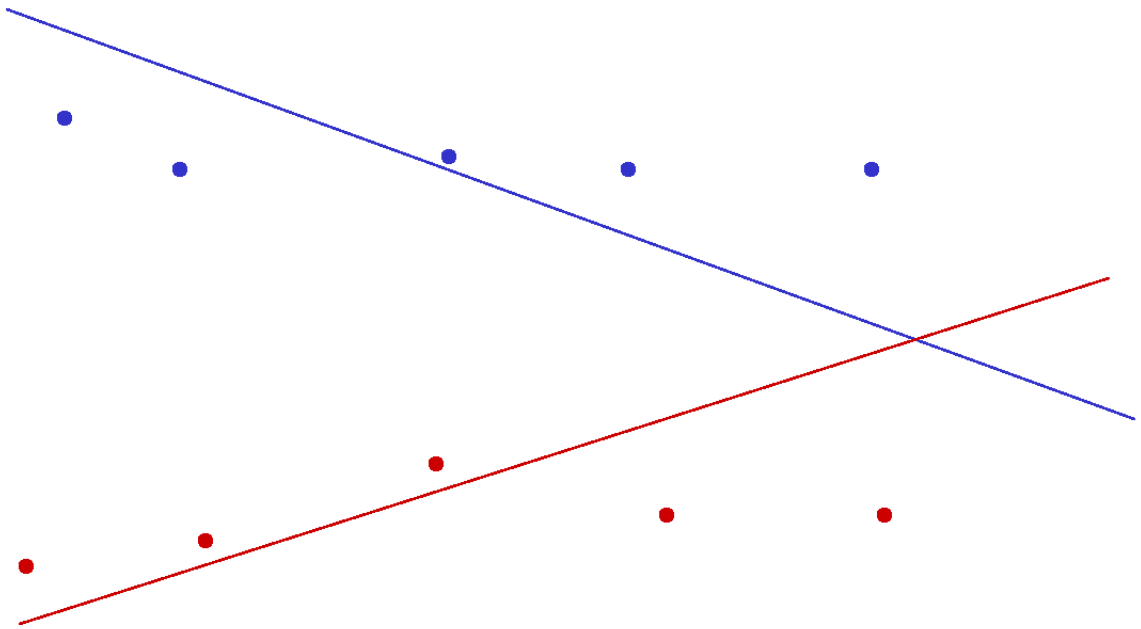
K-means line fitting



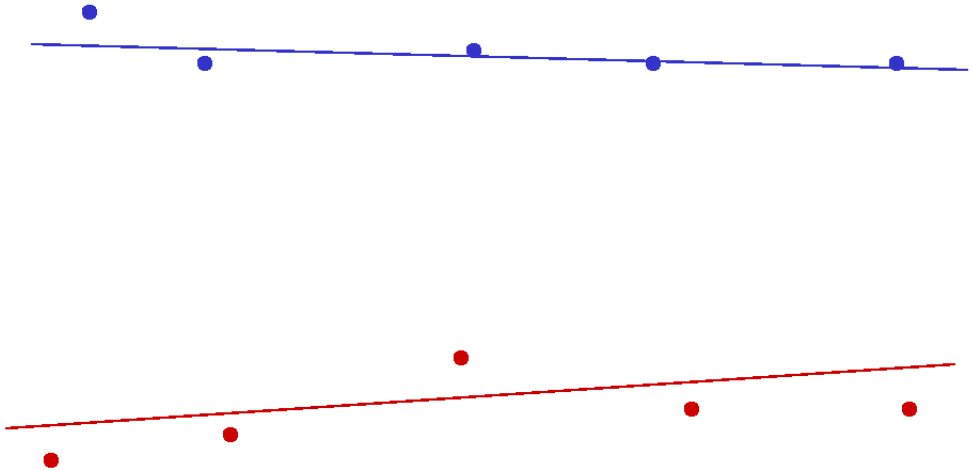
K-means line fitting



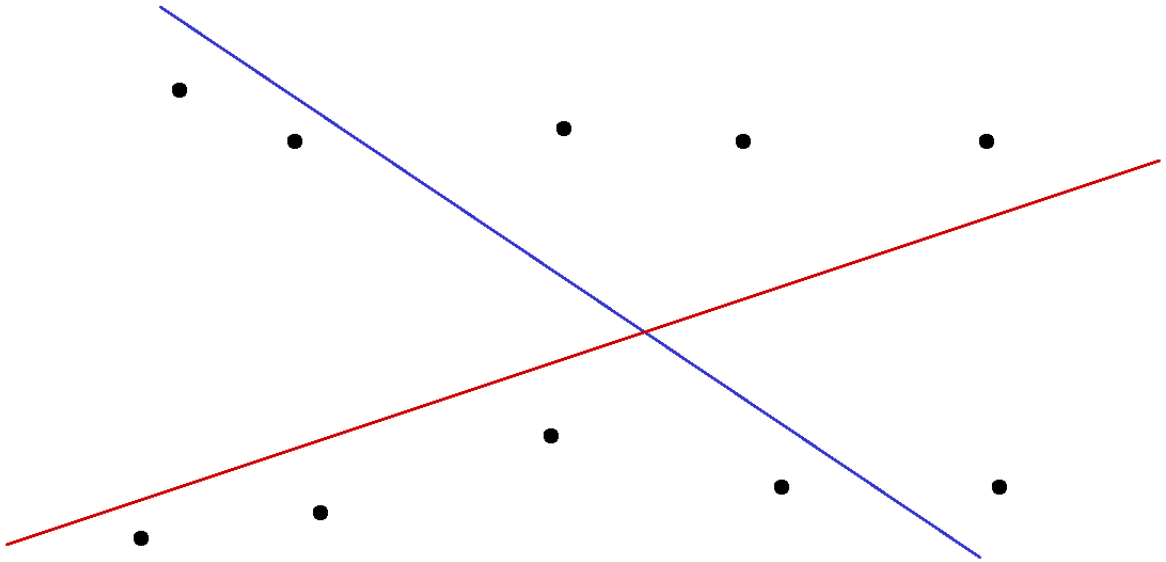
K-means line fitting



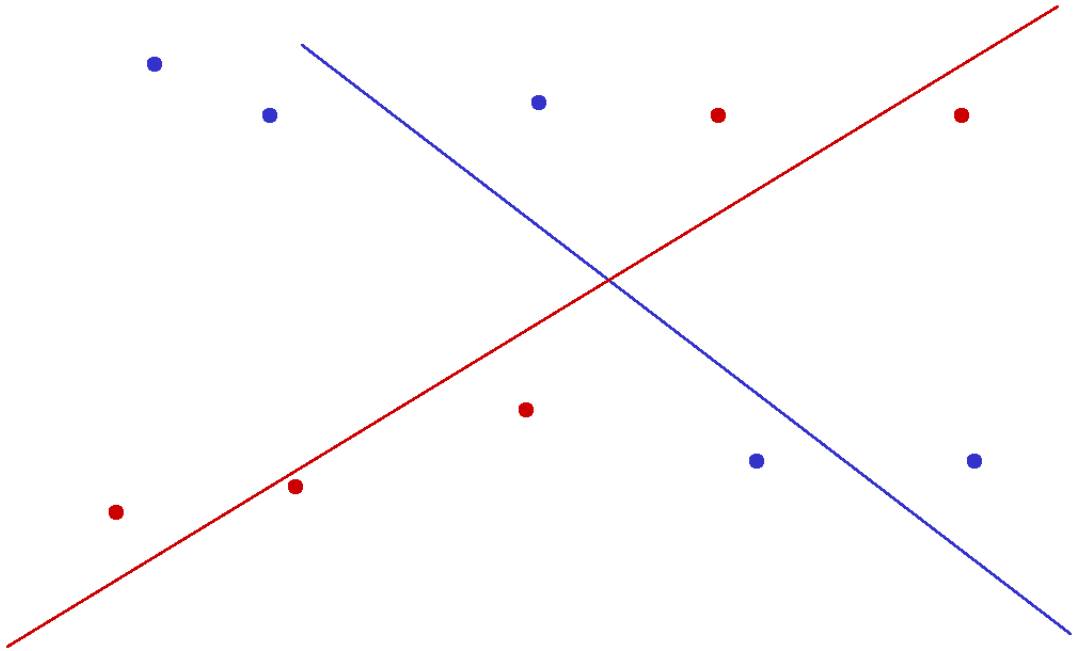
K-means line fitting



K-means line fitting

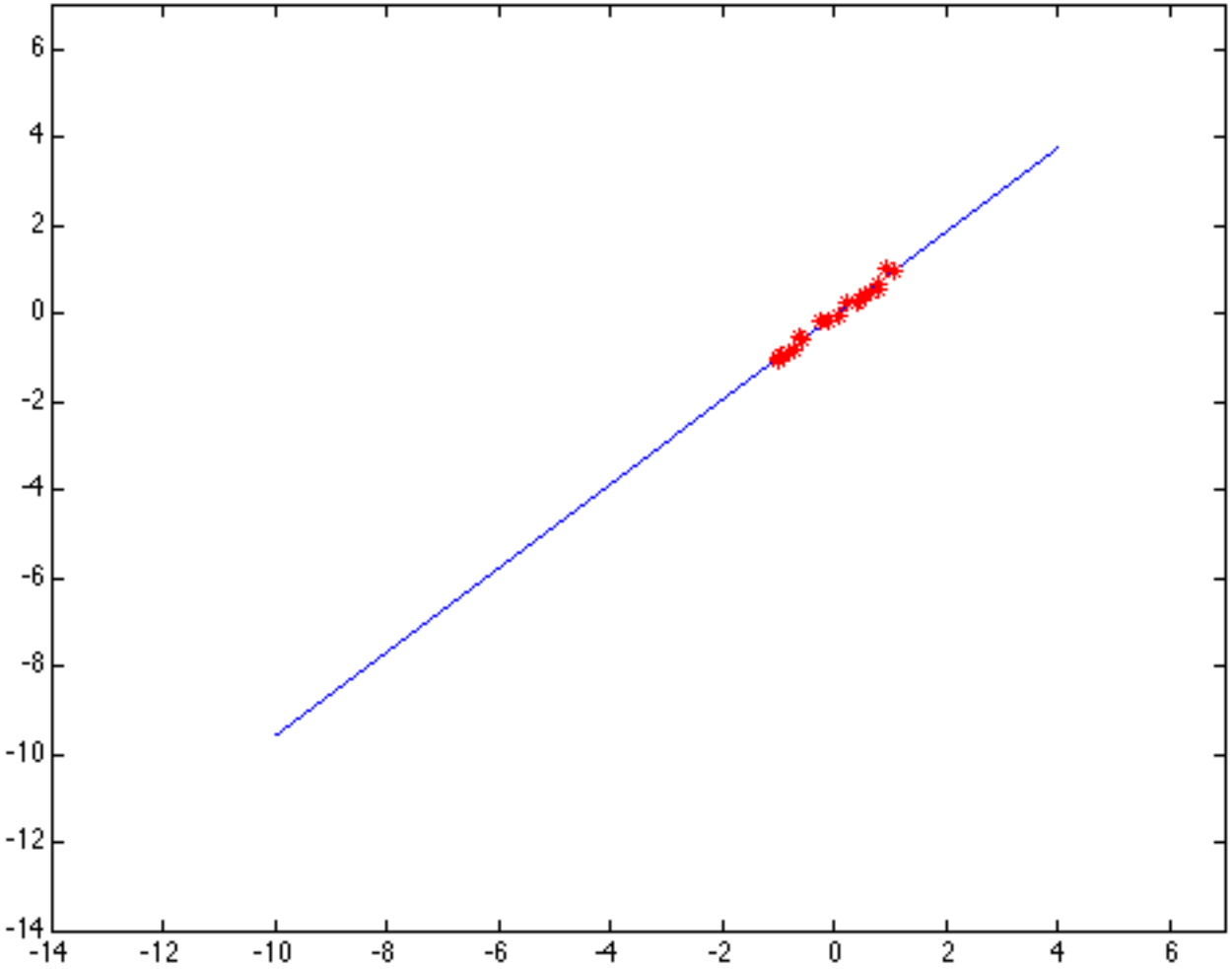


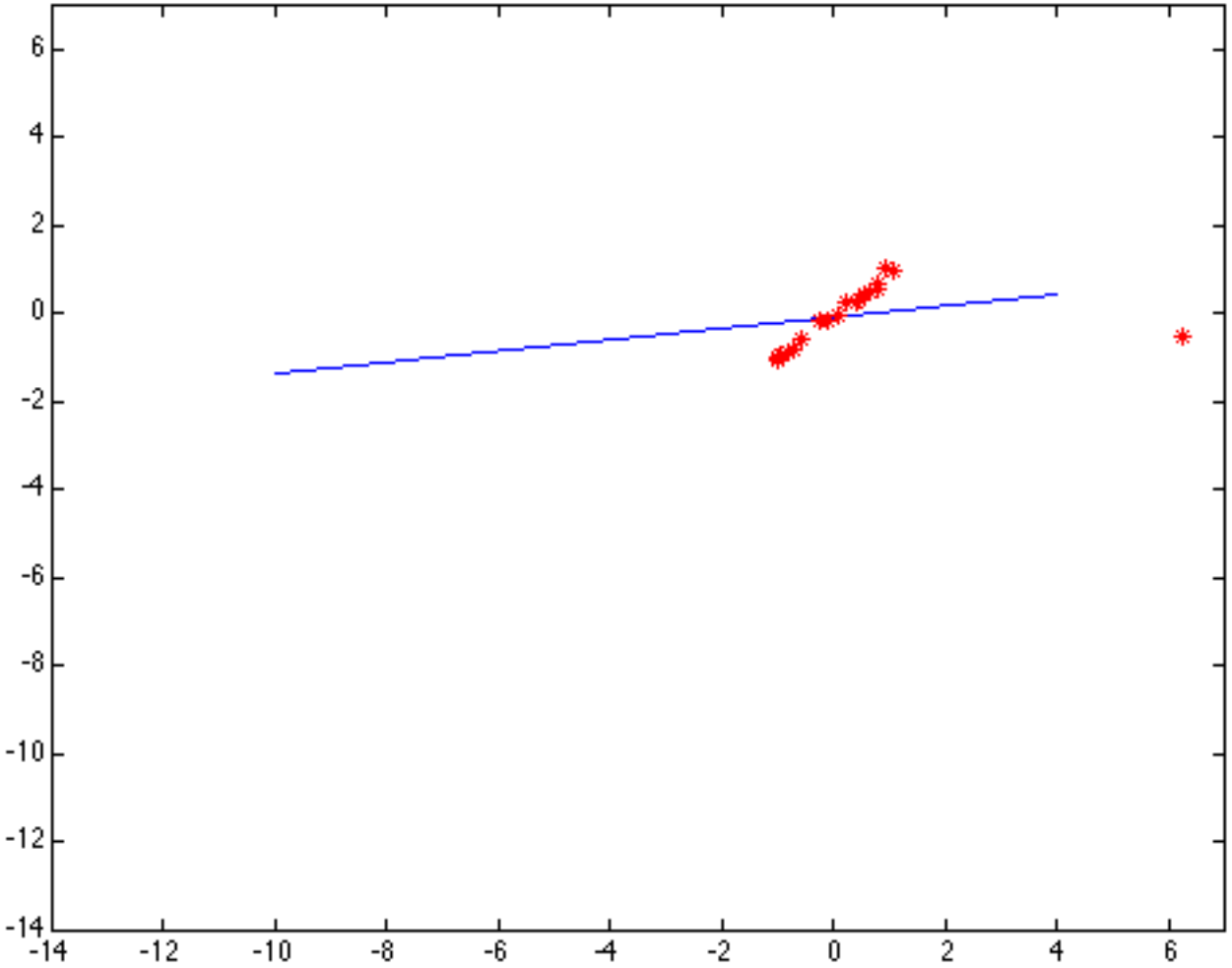
K-means line fitting

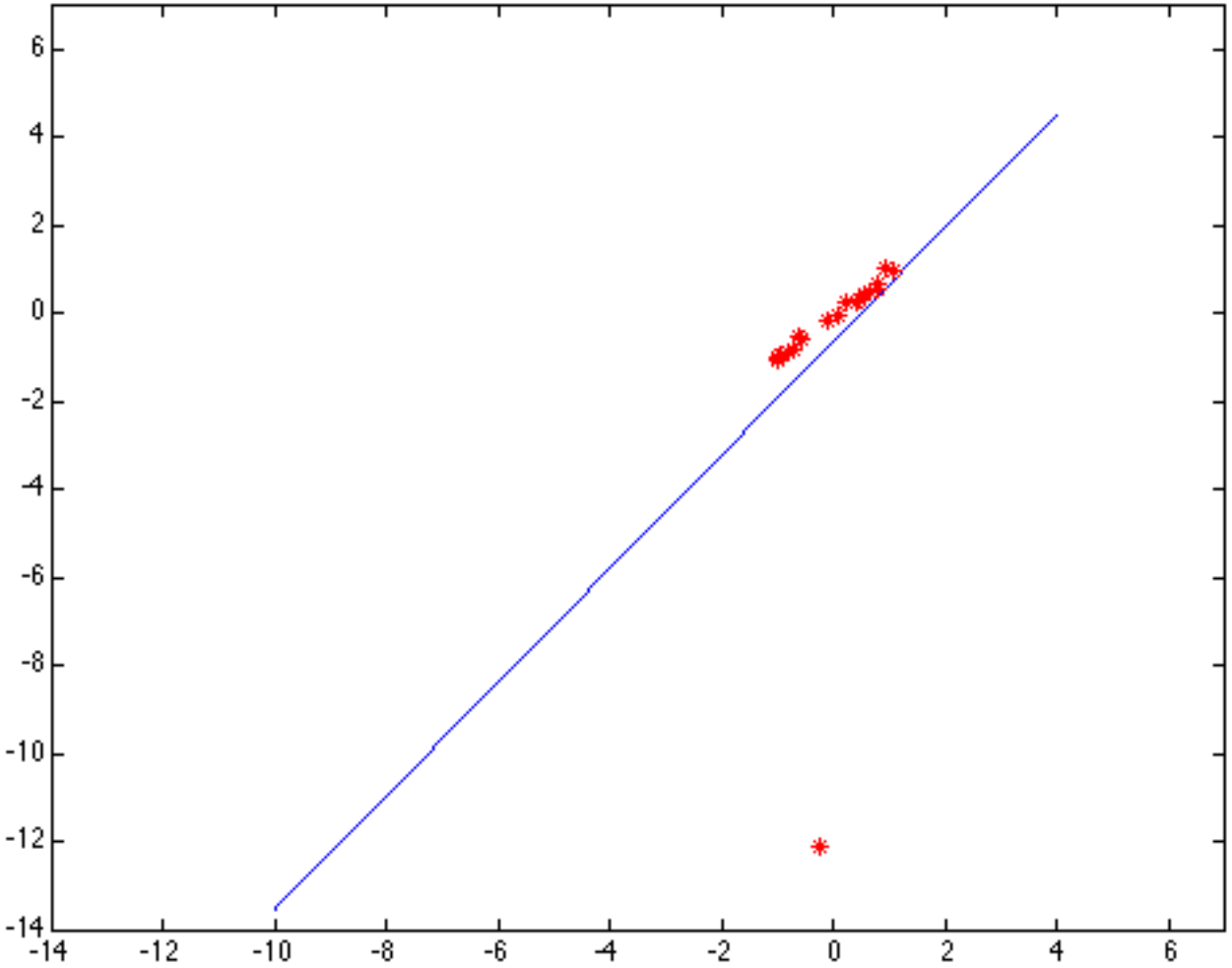


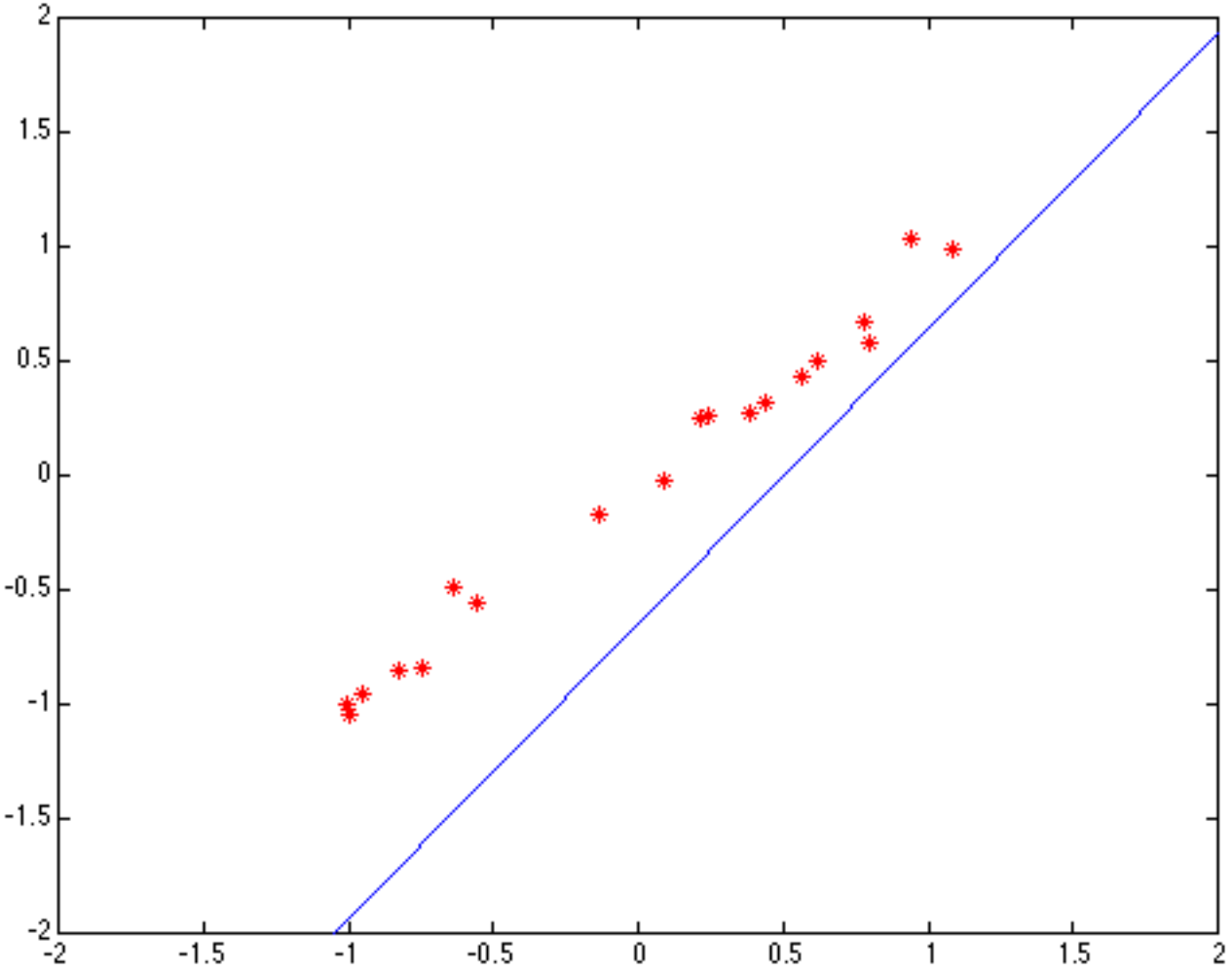
Robustness

- As we have seen, squared error can be a source of bias in the presence of noise points
  - One fix is EM - we'll do this shortly
  - Another is an M-estimator
    - Square nearby, threshold far away
  - A third is RANSAC
    - Search for good points


6
4
2
0
-2
-4
-6
-8
-10
-12
-14
-14
-12
-10
-8
-6
-4
-2
0
2
4
6


6
4
2
0
-2
-4
-6
-8
-10
-12
-14
-14
-12
-10
-8
-6
-4
-2
0
2
4
6


6
4
2
0
-2
-4
-6
-8
-10
-12
-14
-14
-12
-10
-8
-6
-4
-2
0
2
4
6

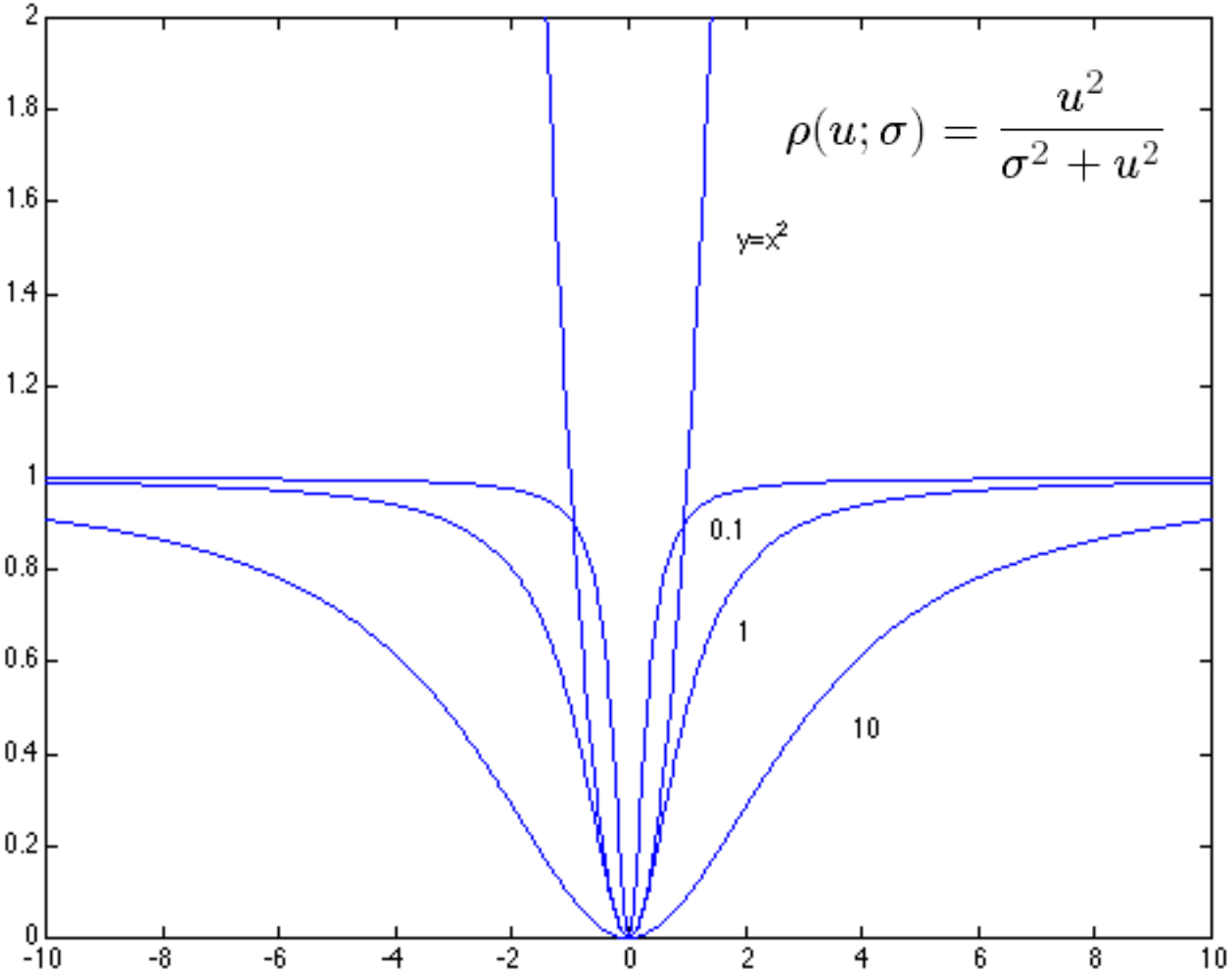

2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2

M-estimators

- Generally, minimize

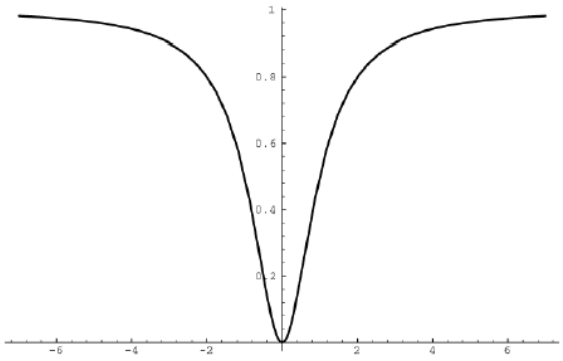
$$\sum_i \rho\left(r_i\left(x_i, \theta\right); \sigma\right)$$

where $r_i\left(x_i, \theta\right)$ is the residual


$\rho(u;\sigma) = \frac{u^2}{\sigma^2 + u^2}$
y=x²
0.1
1
10
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
-10
-8
-6
-4
-2
0
2
4
6
8
10

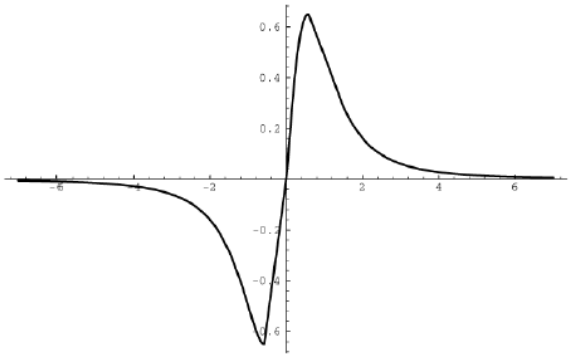
Robust Estimation

A quadratic ρ function gives too much weight to outliers
Instead, use robust norm:

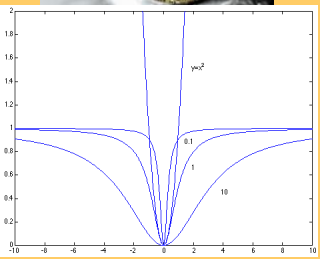


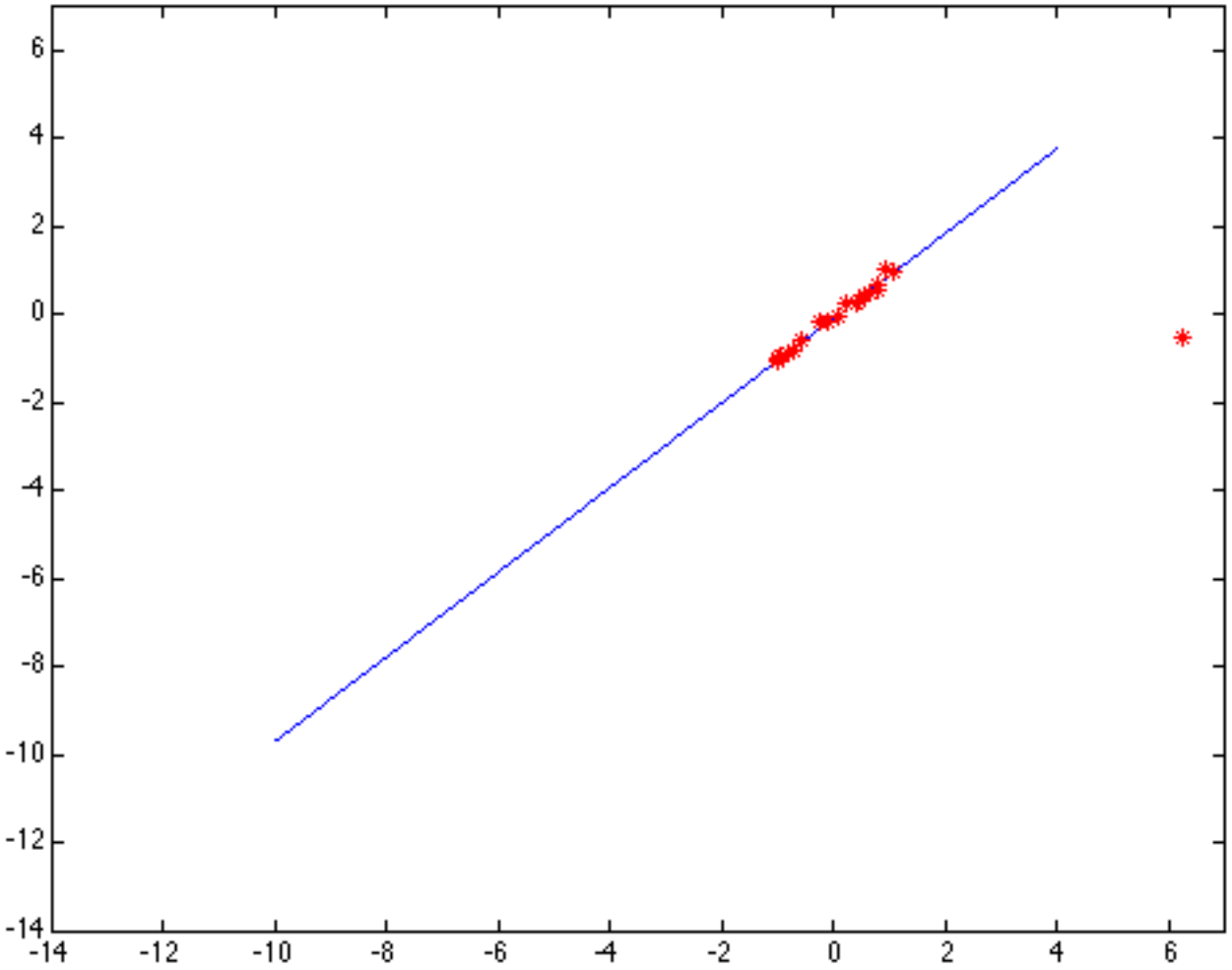
$$\rho(r,\sigma) = \frac{r^2}{\sigma^2 + r^2}$$

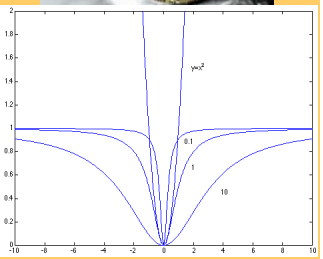
Influence function
(d/dr of norm):



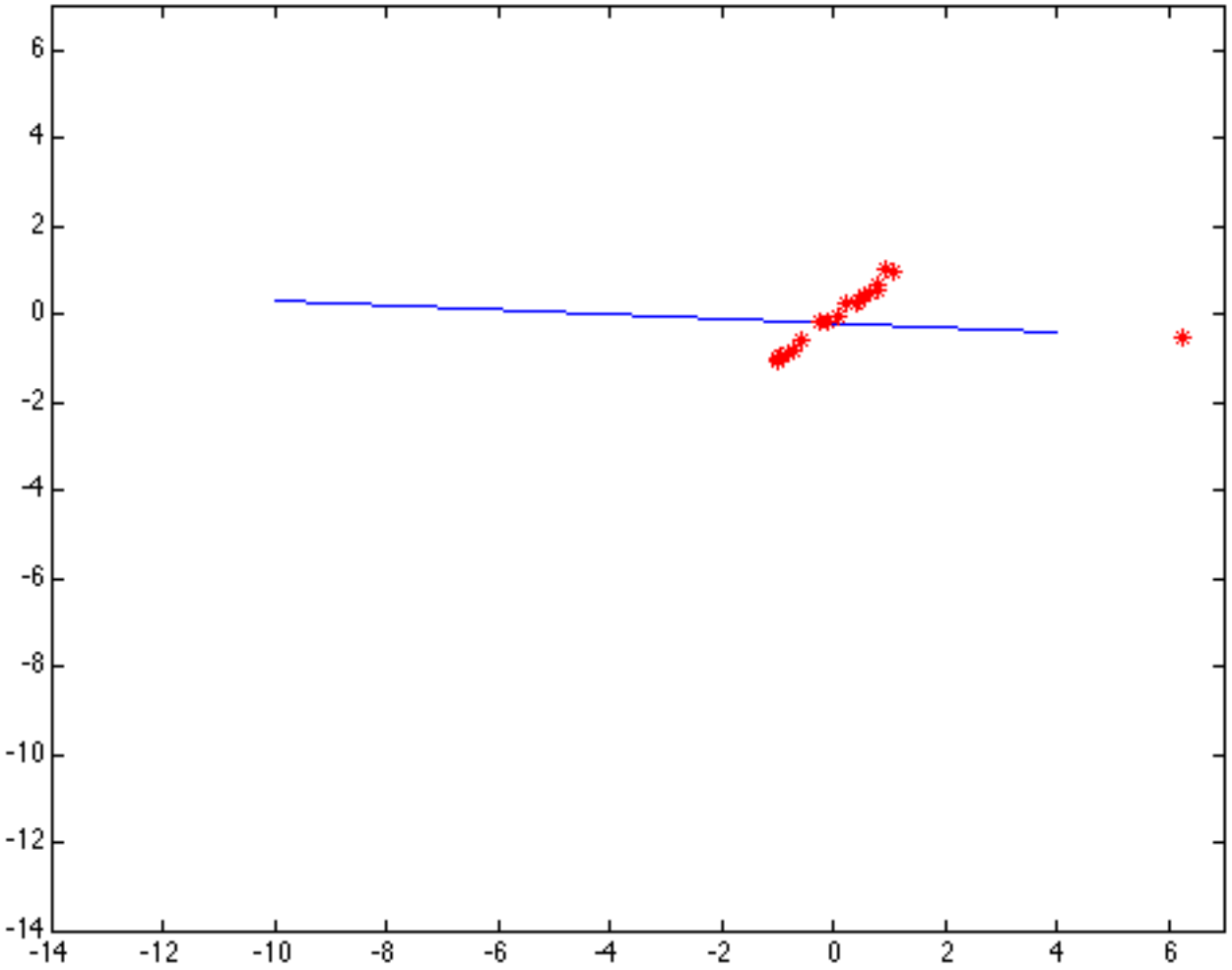
$$\psi(r,\sigma) = \frac{2r\sigma^2}{(\sigma^2 + r^2)^2}$$

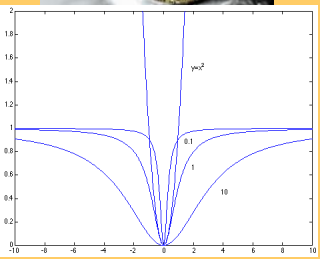

y=x²
0.1
1
10


6
4
2
0
-2
-4
-6
-8
-10
-12
-14
-14
-12
-10
-8
-6
-4
-2
0
2
4
6

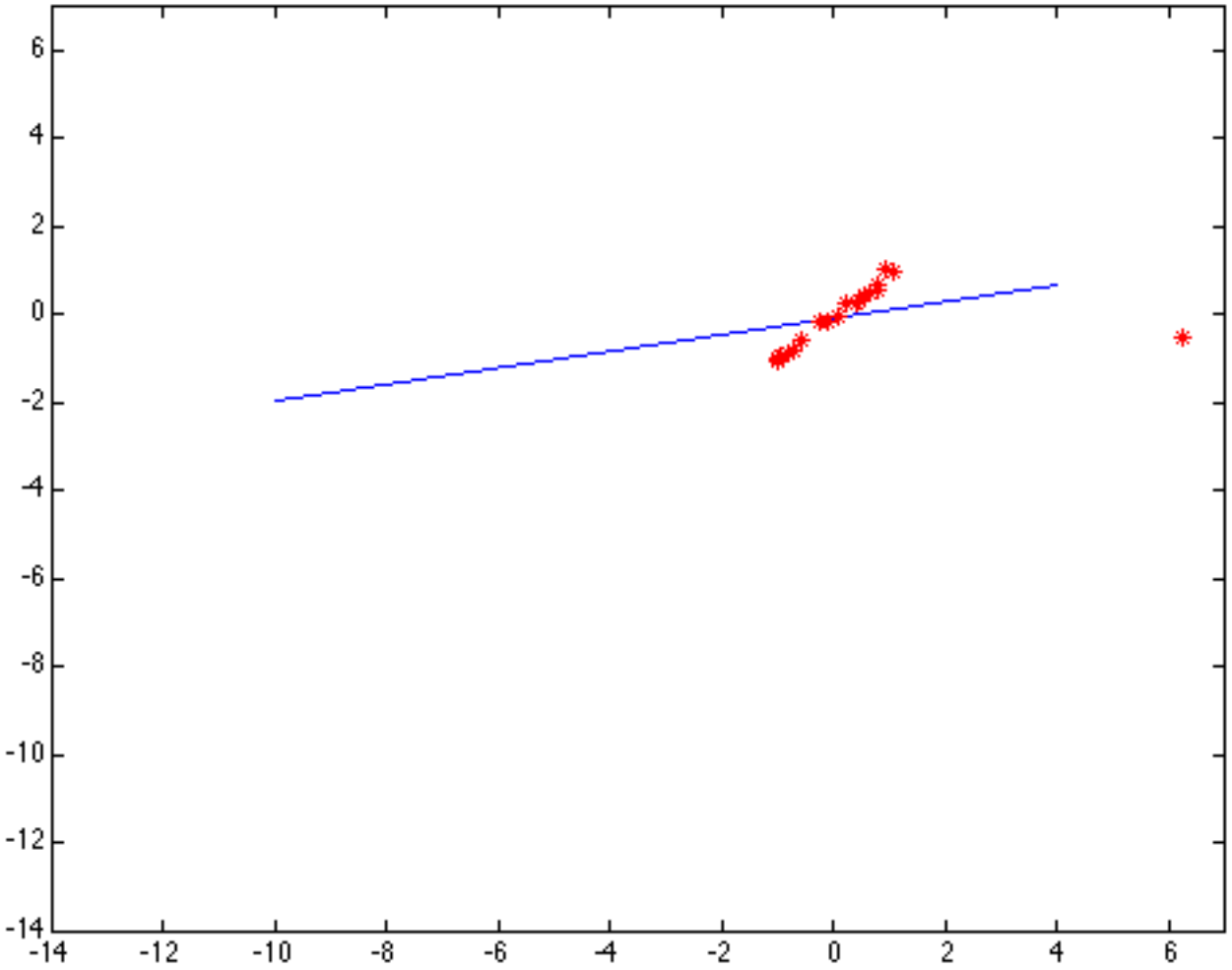


Too small





Too large



Robust scale

Scale is critical!

Popular choice:

$$\sigma^{(n)} = 1.4826 \, \text{median}_i \, |r_i^{(n)}(x_i; \theta^{(n-1)})|$$

RANSAC

- Choose a small subset uniformly at random
- Fit to that
- Anything that is close to result is signal; all others are noise
- Refit
- Do this many times and choose the best

- Issues
  - How many times?
    - Often enough that we are likely to have a good line
  - How big a subset?
    - Smallest possible
  - What does close mean?
    - Depends on the problem
  - What is a good line?
    - One where the number of nearby points is so big it is unlikely to be all outliers

Algorithm 15.4: RANSAC: fitting lines using random sample consensus

Determine:
    n — the smallest number of points required
    k — the number of iterations required
    t — the threshold used to identify a point that fits well
    d — the number of nearby points required
      to assert a model fits well
Until k iterations have occurred
    Draw a sample of n points from the data
      uniformly and at random
    Fit to that set of n points
    For each data point outside the sample
      Test the distance from the point to the line
        against t; if the distance from the point to the line
        is less than t, the point is close
    end
    If there are d or more points close to the line
      then there is a good fit. Refit the line using all
      these points.
end
Use the best fit from this collection, using the
  fitting error as a criterion

Distance threshold

Choose *t* so probability for inlier is α (e.g. 0.95)

- Often empirically
- Zero-mean Gaussian noise σ then $d_{\perp}^{2}$ follows χ_{m}^{2} distribution with m=codimension of model

(dimension+codimension=dimension space)

Codimension	Model	t^2
1	line,F	$3.84\sigma^2$
2	H,P	$5.99\sigma^2$
3	T	$7.81\sigma^2$

How many samples?

Choose N so that, with probability p, at least one random sample is free from outliers. e.g. p=0.99

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

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

alternatively, use Poisson:

$$f(k;\lambda)=\frac{\lambda^{k}e^{-\lambda}}{k!}$$, with k=0 (no occurrence)

λ=3: f≈0.05

λ=4.5: f≈0.01

	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

Acceptable consensus set?

- Typically, terminate when inlier ratio reaches expected ratio of inliers

$$T = (1 - e)n$$

Adaptively determining the number of samples

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

- $N=\infty$, *sample_count* =0
- While $N >$ *sample_count* repeat
  - Choose a sample and count the number of inliers
  - Set e=1-(number of inliers)/(total number of points)
  - Recompute N from e
  - Increment the *sample_count* by 1
- Terminate

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

RANSAC for Fundamental Matrix

Step 1. Extract features

Step 2. Compute a set of potential matches

Step 3. do

- Step 3.1 select minimal sample (i.e. 7 matches)
- Step 3.2 compute solution(s) for F

} (generate hypothesis)

- Step 3.3 determine inliers (verify hypothesis)

until Γ(*#inliers*,*#samples*)>95%

Step 4. Compute F based on all inliers

Step 5. Look for additional matches

Step 6. Refine F based on all correct matches

$$\Gamma = 1 - (1 - \left(\frac{\#inliers}{\#matches}\right)^7)^{\#samples}$$

#inliers	90%	80%	70%	60%	50%
#samples	5	13	35	106	382

Randomized RANSAC for Fundamental Matrix

Step 1. Extract features

Step 2. Compute a set of potential matches

Step 3. do

- Step 3.1 select minimal sample (i.e. 7 matches)
- Step 3.2 compute solution(s) for F

} (generate hypothesis)

- Step 3.3 Randomize verification
  - 3.3.1 verify if inlier
  - while hypothesis is still promising

} (verify hypothesis)

while Γ(*#inliers*,*#samples*)<95%

Step 4. Compute F based on all inliers

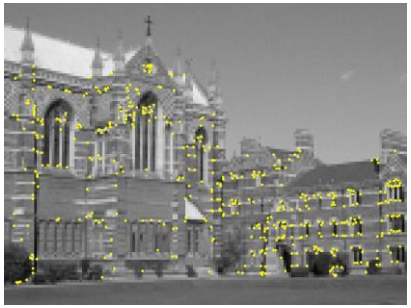
Step 5. Look for additional matches

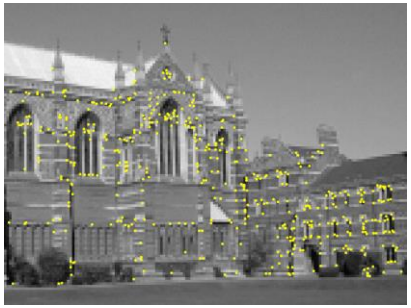
Step 6. Refine F based on all correct matches

#in	1-e	adapt. *N*
6	2%	20M
10	3%	2.5M
44	16%	6,922
58	21%	2,291
73	26%	911
151	56%	43

Example: robust computation

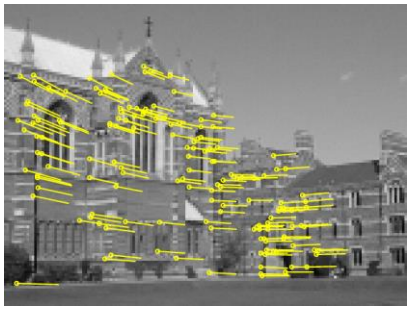
from H&Z

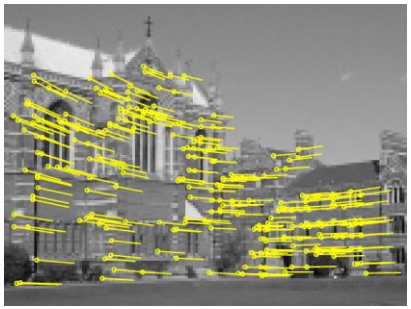




Interest points (500/image)
(640x480)

Putative correspondences (268)
(Best match,SSD<20,±320)
Outliers (117)
(*t*=1.25 pixel; 43 iterations)





Inliers (151)

Final inliers (262)
(2 MLE-inlier cycles;
$d_{\perp}$=0.23→$d_{\perp}$=0.19;
$Iter_{Lev\text{-}Mar}$=10)

More on robust estimation

- LMedS, an alternative to RANSAC (minimize Median residual in stead of maximizing inlier count)

- Enhancements to RANSAC
  - Randomized RANSAC
  - Sample ‘good’ matches more frequently
  - …

- RANSAC is also somewhat robust to bugs, sometimes it just takes a bit longer…

RANSAC for quasi-degenerate data

- Often data can be almost degenerate
  e.g. 337 matches on plane, 11 off plane

planar points only
provide 6 in stead of 8
linearly independent
equations for F

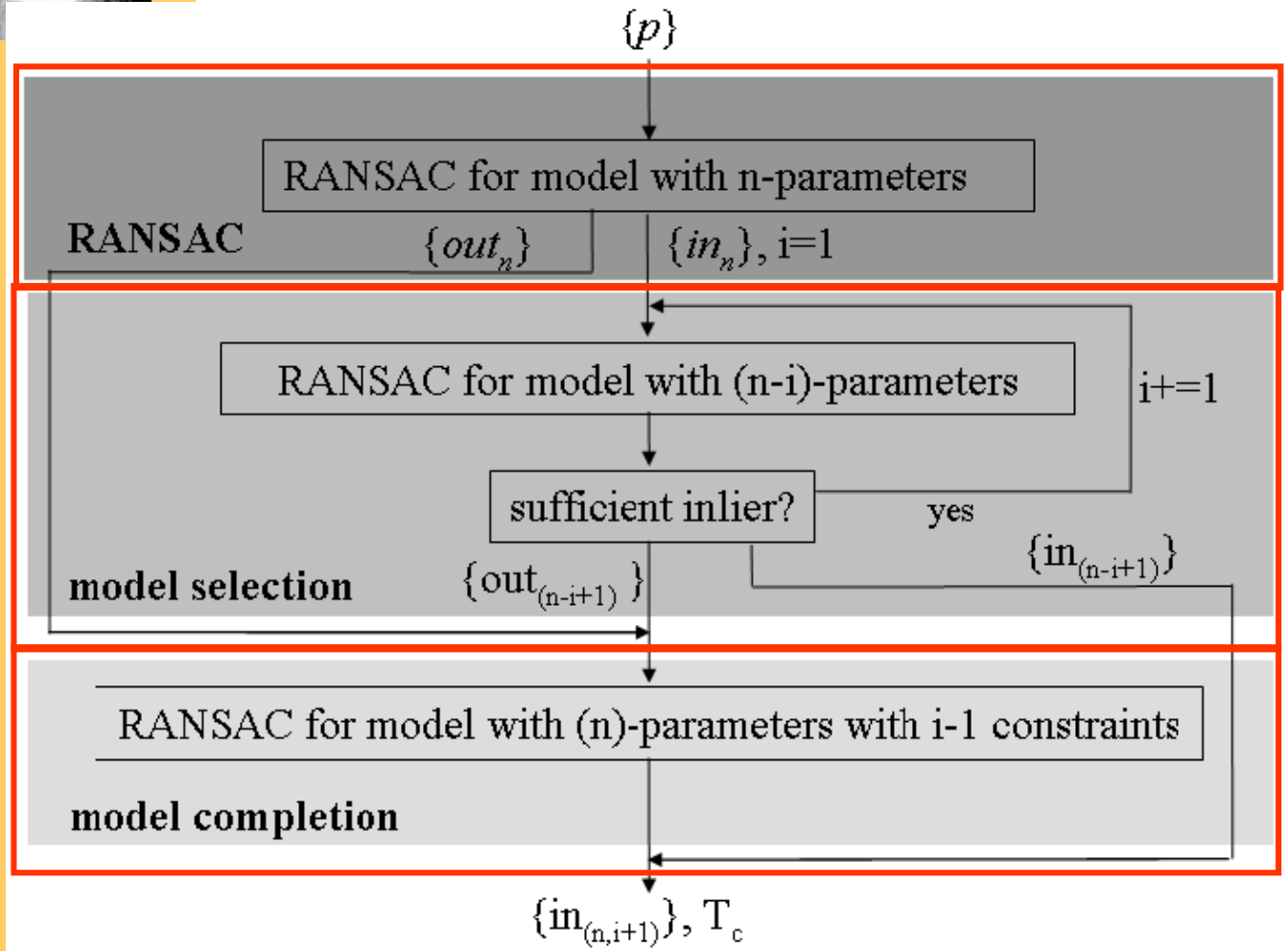
- RANSAC gets confused by quasi-degenerate data

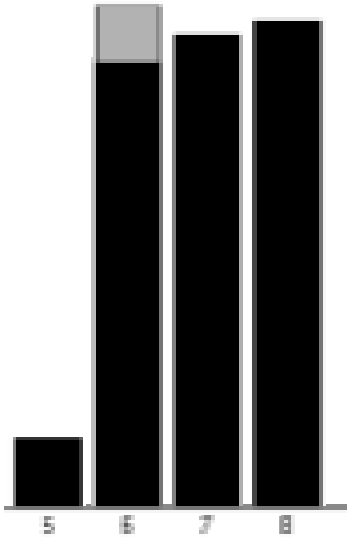
	$\epsilon = 0.9$				$\epsilon = 0.8$			
	$\epsilon_d = 0.85$		$\epsilon_d = 0.88$		$\epsilon_d = 0.75$		$\epsilon_d = 0.78$	
	P_{nd}	P_s	P_{nd}	P_s	P_{nd}	P_s	P_{nd}	P_s
F	3%	36%	0.5%	5%	1.4%	47%	0.3%	8%
P	2.1%	19%	0.4%	3%	1.3%	26%	0.2%	4%
H	1.6%	54%	0.7%	9%	1.4%	66%	0.3%	13%

P_{nd} Probability of valid non-degenerate sample
P_s Probability of success for RANSAC (aiming for 99%)

RANSAC for quasi-degenerate data (QDEGSAC)

(Frahm and Pollefeys, CVPR06)





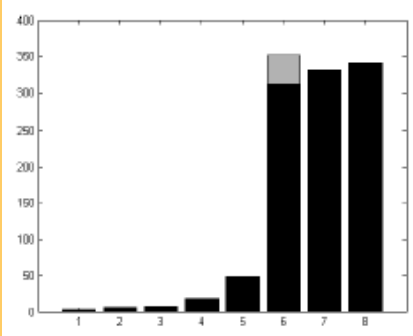
QDEGSAC estimates robust rank of datamatrix

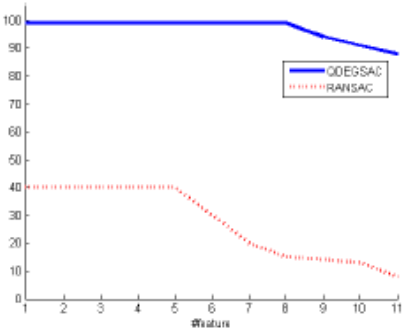
(i.e. large % of rows approx. fit in a lower dimensional subspace)

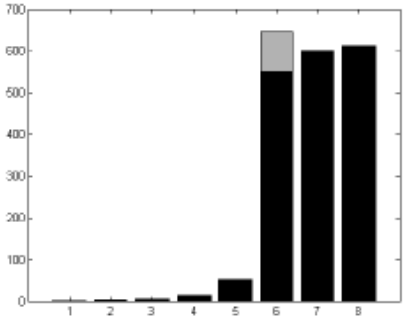
Experiments for computing F matrix

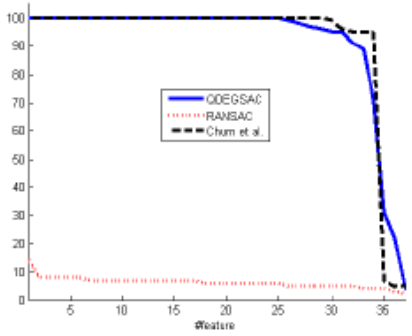
Comparison to Chum et al. (CVPR'05)

Data courtesy of Chum et al.





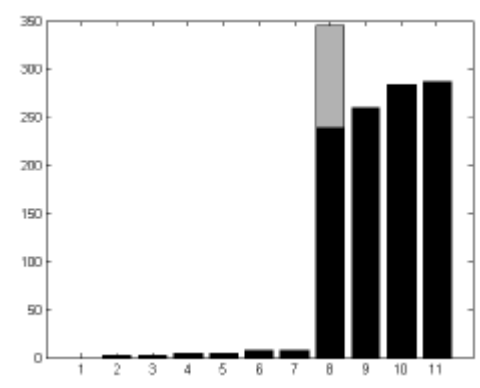


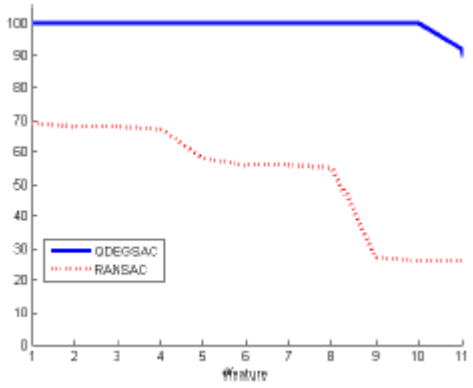


QDEGSAC (and DEGENSAC) are successful at dealing with quasi-degenerate data

Experiments for computing P matrix

Similar results for H, Q, …





QDEGSAC is general and can deal with other robust estimation problems

QDEGSAC does not require to know which degeneracies can occur

Fitting curves other than lines

- In principle, an easy generalisation
  - The probability of obtaining a point, given a curve, is given by a negative exponential of distance squared
- In practice, rather hard
  - It is generally difficult to compute the distance between a point and a curve

Missing variable problems

- In many vision problems, if some variables were known the maximum likelihood inference problem would be easy
  - fitting; if we knew which line each token came from, it would be easy to determine line parameters
  - segmentation; if we knew the segment each pixel came from, it would be easy to determine the segment parameters
  - fundamental matrix estimation; if we knew which feature corresponded to which, it would be easy to determine the fundamental matrix
  - etc.
- This sort of thing happens in statistics, too

Missing variable problems

- Strategy
  - estimate appropriate values for the missing variables
  - plug these in, now estimate parameters
  - re-estimate appropriate values for missing variables, continue
- e.g.
  - guess which line gets which point
  - now fit the lines
  - now reallocate points to lines, using our knowledge of the lines
  - now refit, etc.
- We've seen this line of thought before (k-means)

Missing variables - strategy

- We have a problem with parameters, missing variables
- This suggests:
- Iterate until convergence
  - replace missing variable with expected values, given **fixed** values of parameters
  - fix missing variables, choose parameters to maximise likelihood given fixed values of missing variables
- e.g., iterate till convergence
  - allocate each point to a line with a weight, which is the probability of the point given the line
  - refit lines to the weighted set of points
- Converges to local extremum
- Somewhat more general form is available

Lines and robustness

- We have one line, and n points
- Some come from the line, some from "noise"
- This is a mixture model:
- We wish to determine
  - line parameters
  - p(comes from line)

$$
\begin{aligned}
&P(\text{point} \mid \text{line and noise params}) \\
&\quad = P(\text{point} \mid \text{line})P(\text{comes from line}) + \\
&\quad P(\text{point} \mid \text{noise})P(\text{comes from noise}) \\
&\quad = P(\text{point} \mid \text{line})\lambda + P(\text{point} \mid \text{noise})(1-\lambda)
\end{aligned}
$$

Estimating the mixture model

- Introduce a set of hidden variables, δ, one for each point. They are one when the point is on the line, and zero when off.
- If these are known, the negative log-likelihood becomes (the line's parameters are ϕ, c):
- Here K is a normalising constant, k_n is the noise intensity (we'll choose this later).

$$Q_c(x;\theta) = \sum_i \left(\begin{array}{l} \delta_i \left(\frac{(x_i \cos\phi + y_i \sin\phi + c)^2}{2\sigma^2} \right) + \\ (1-\delta_i) k_n \end{array} \right) + K$$

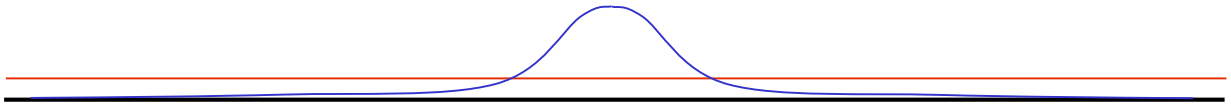
Substituting for delta

- We shall substitute the expected value of δ, for a given θ
- recall $\theta=(\phi, c, \lambda)$
- $E(\delta\_i)=1.$ $P(\delta\_i=1|\theta)+0....$

- Notice that if k_n is small and positive, then if distance is small, this value is close to 1 and if it is large, close to zero

$$P(\delta_i=1|\theta,\mathbf{x}_i)=\frac{P(\mathbf{x}_i|\delta_i=1,\theta)P(\delta_i=1)}{P(\mathbf{x}_i|\delta_i=1,\theta)P(\delta_i=1)+P(\mathbf{x}_i|\delta_i=0,\theta)P(\delta_i=0)}$$

$$=\frac{\exp\left(-1/2\sigma^2\left[x_i\cos\phi+y_i\sin\varphi+c\right]^2\right)\lambda}{\exp\left(-1/2\sigma^2\left[x_i\cos\phi+y_i\sin\varphi+c\right]^2\right)\lambda+\exp(-k_n)(1-\lambda)}$$



Algorithm for line fitting

- Obtain some start point

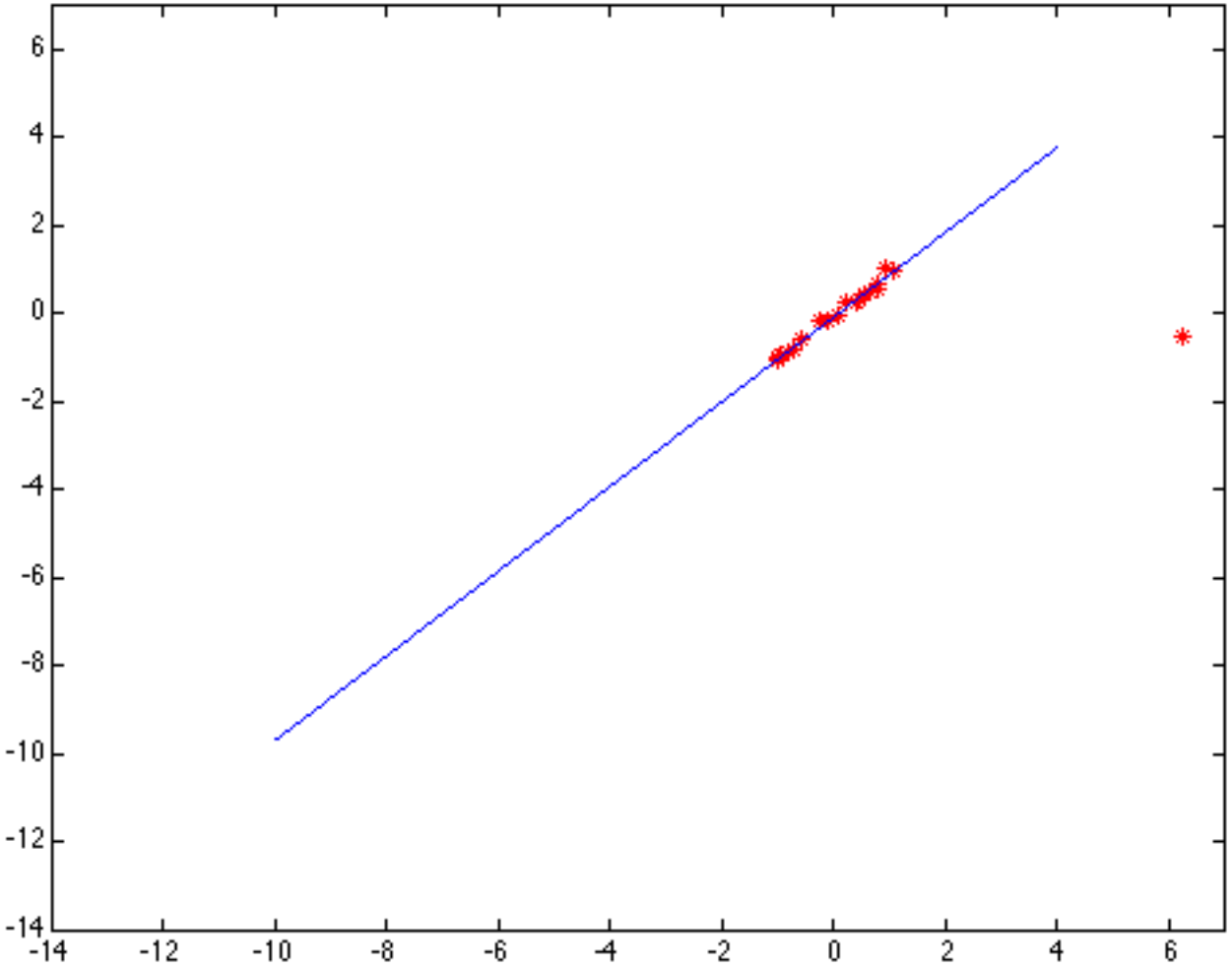
$$\theta^{(0)} = \left(\phi^{(0)}, c^{(0)}, \lambda^{(0)}\right)$$

- Now compute δ's using formula above

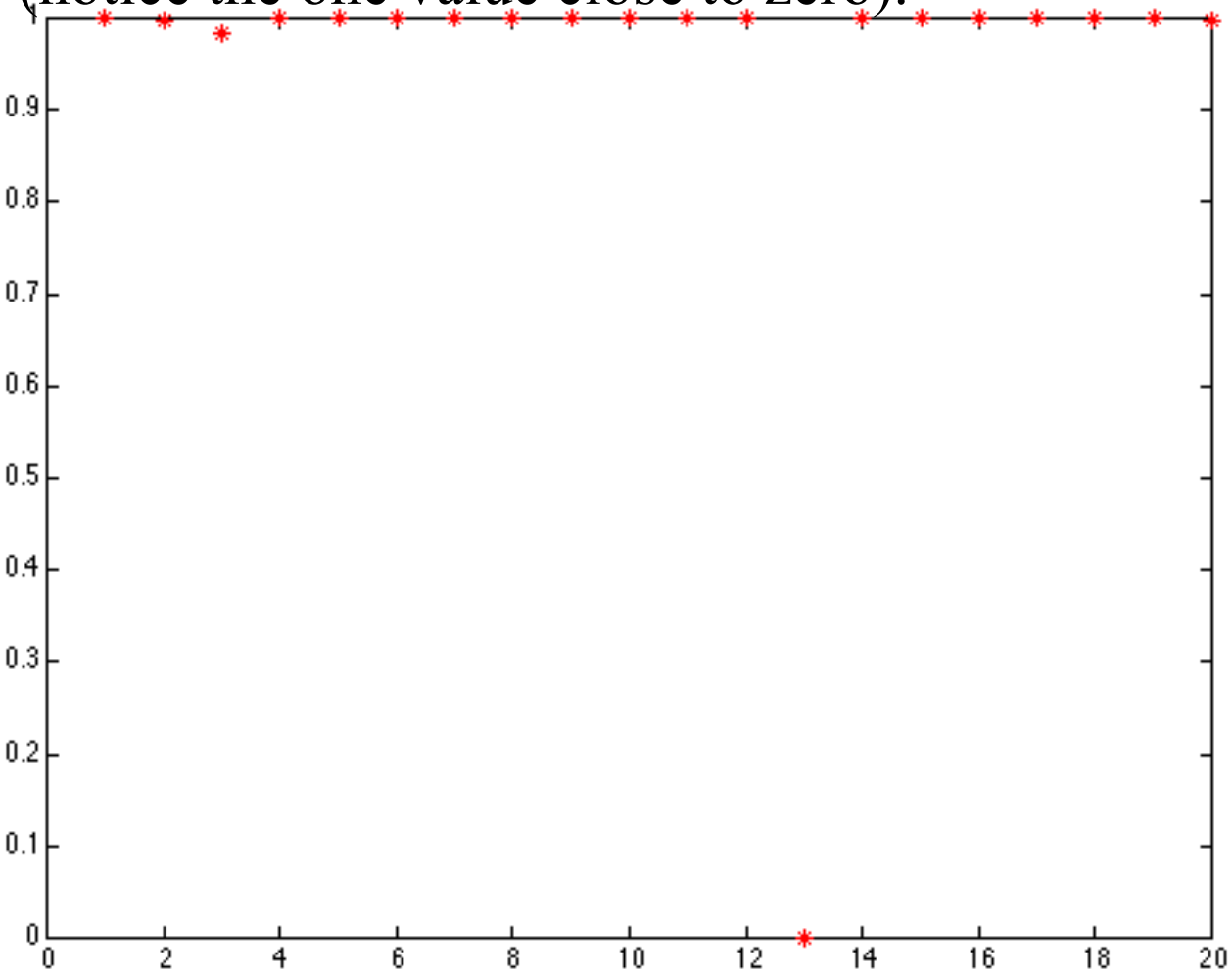
- Now compute maximum likelihood estimate of $\theta^{(1)}$

  - ϕ, c come from fitting to weighted points
  - λ comes by counting

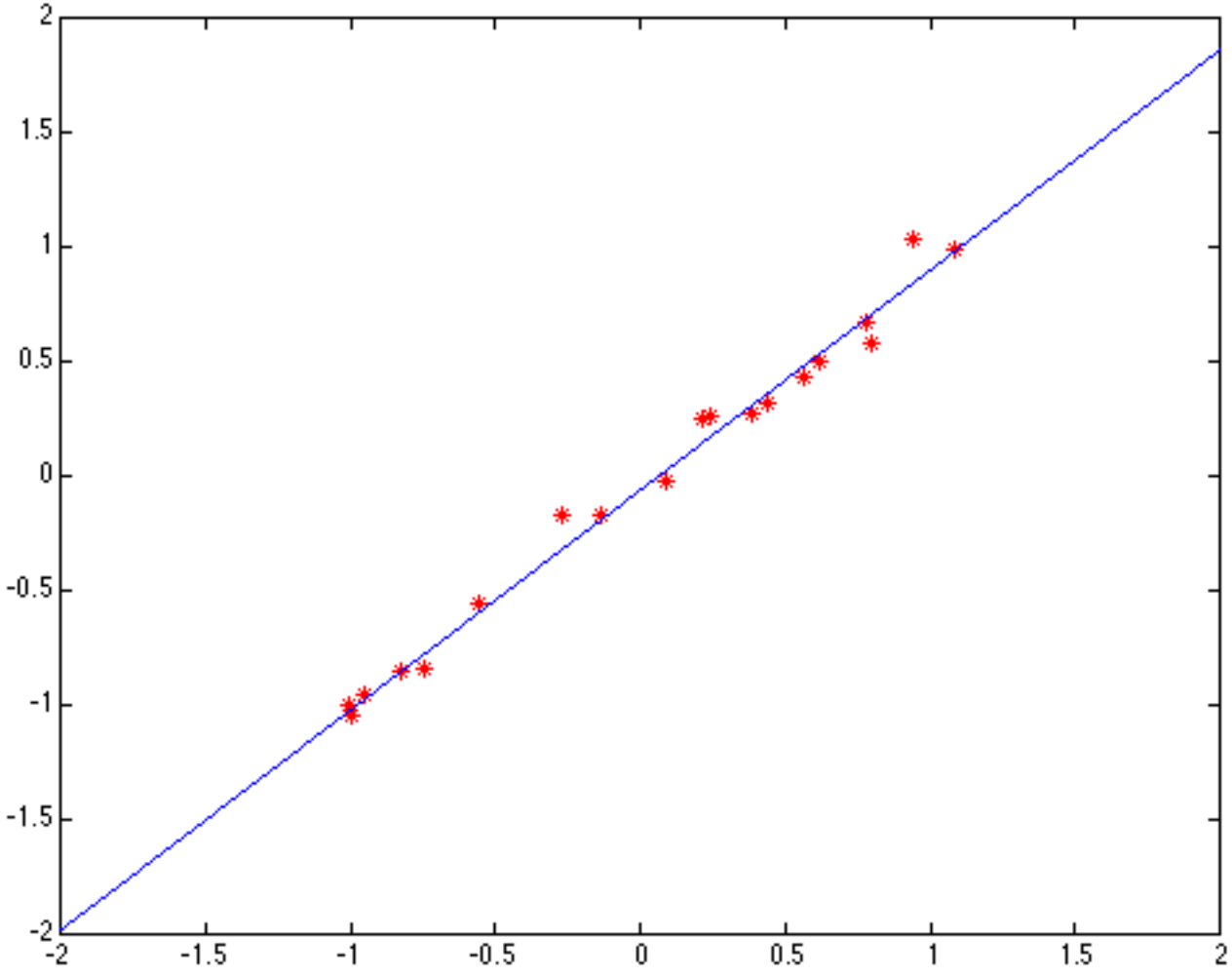
- Iterate to convergence


6
4
2
0
-2
-4
-6
-8
-10
-12
-14
-14
-12
-10
-8
-6
-4
-2
0
2
4
6

The expected values of the deltas at the maximum (notice the one value close to zero).



Closeup of the fit



Choosing parameters

- What about the noise parameter, and the sigma for the line?
  - several methods
    - from first principles knowledge of the problem (seldom really possible)
    - play around with a few examples and choose (usually quite effective, as precise choice doesn't matter much)
  - notice that if k_n is large, this says that points very seldom come from noise, however far from the line they lie
    - usually biases the fit, by pushing outliers into the line
    - rule of thumb; its better to fit to the better fitting points, within reason; if this is hard to do, then the model could be a problem

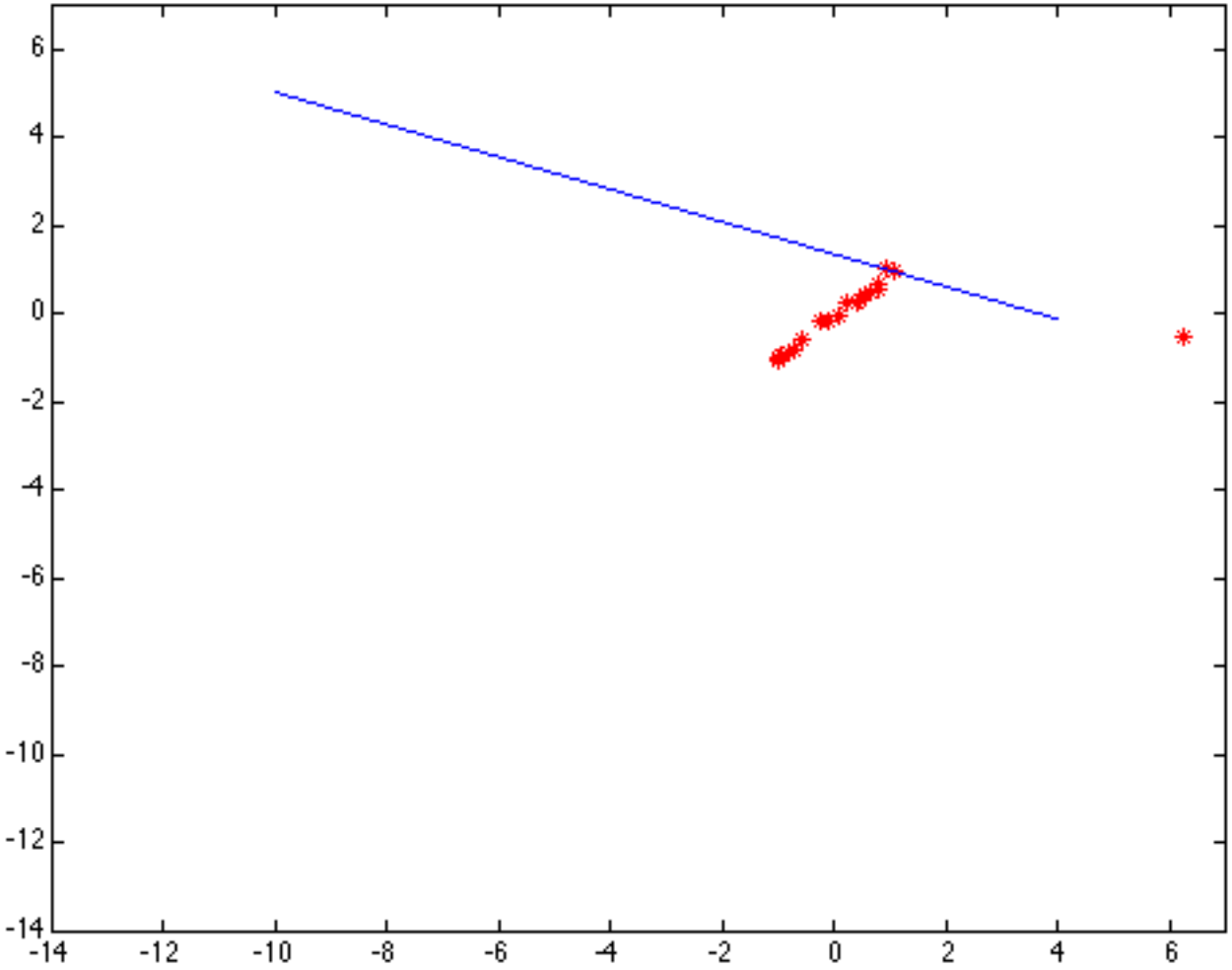
Other examples

- Segmentation
  - a segment is a gaussian that emits feature vectors (which could contain colour; or colour and position; or colour, texture and position).
  - segment parameters are mean and (perhaps) covariance
  - if we knew which segment each point belonged to, estimating these parameters would be easy
  - rest is on same lines as fitting line
- Fitting multiple lines
  - rather like fitting one line, except there are more hidden variables
  - easiest is to encode as an array of hidden variables, which represent a table with a one where the i'th point comes from the j'th line, zeros otherwise
  - rest is on same lines as above

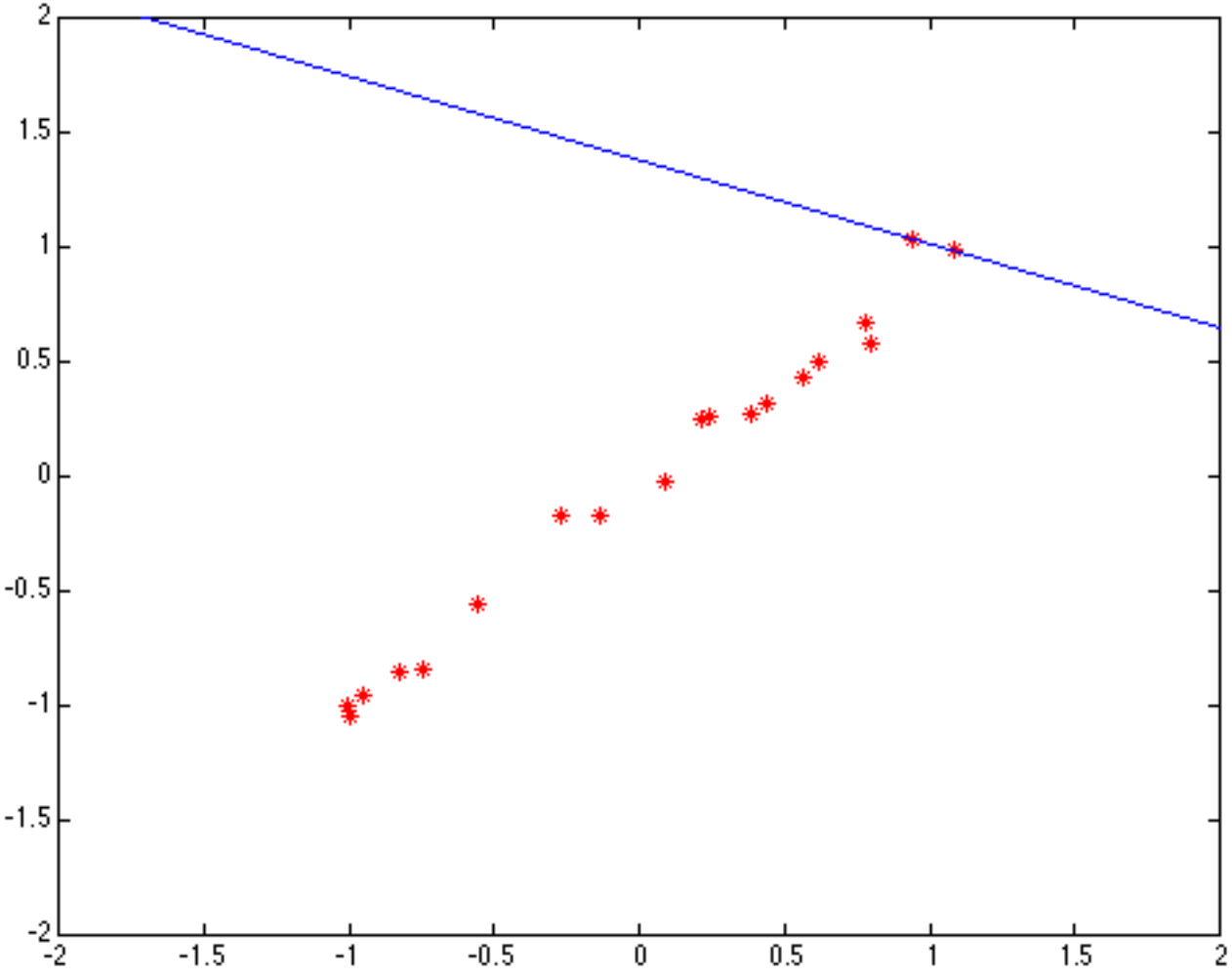
Issues with EM

- Local maxima
  - can be a serious nuisance in some problems
  - no guarantee that we have reached the “right” maximum
- Starting
  - k means to cluster the points is often a good idea

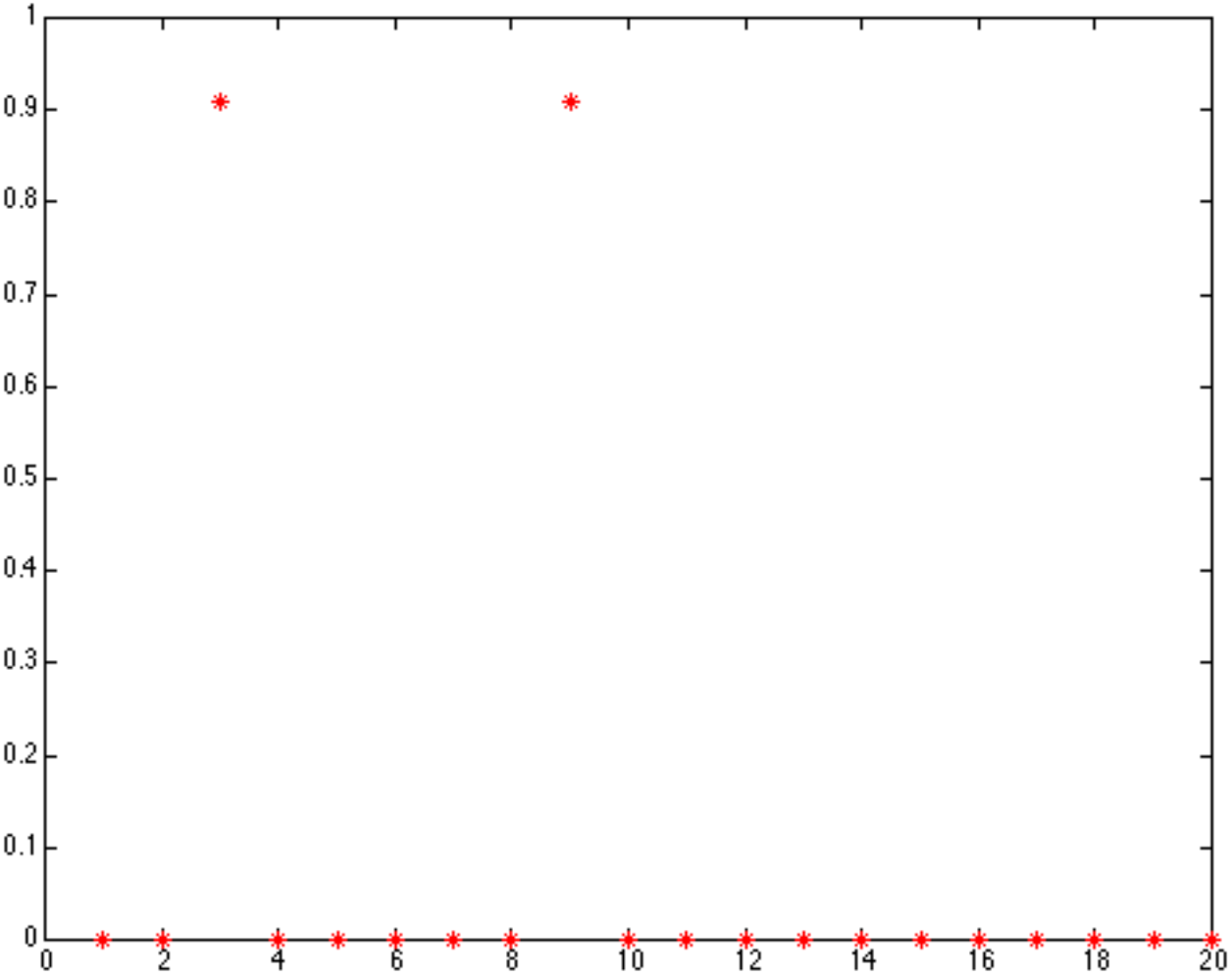
Local maximum



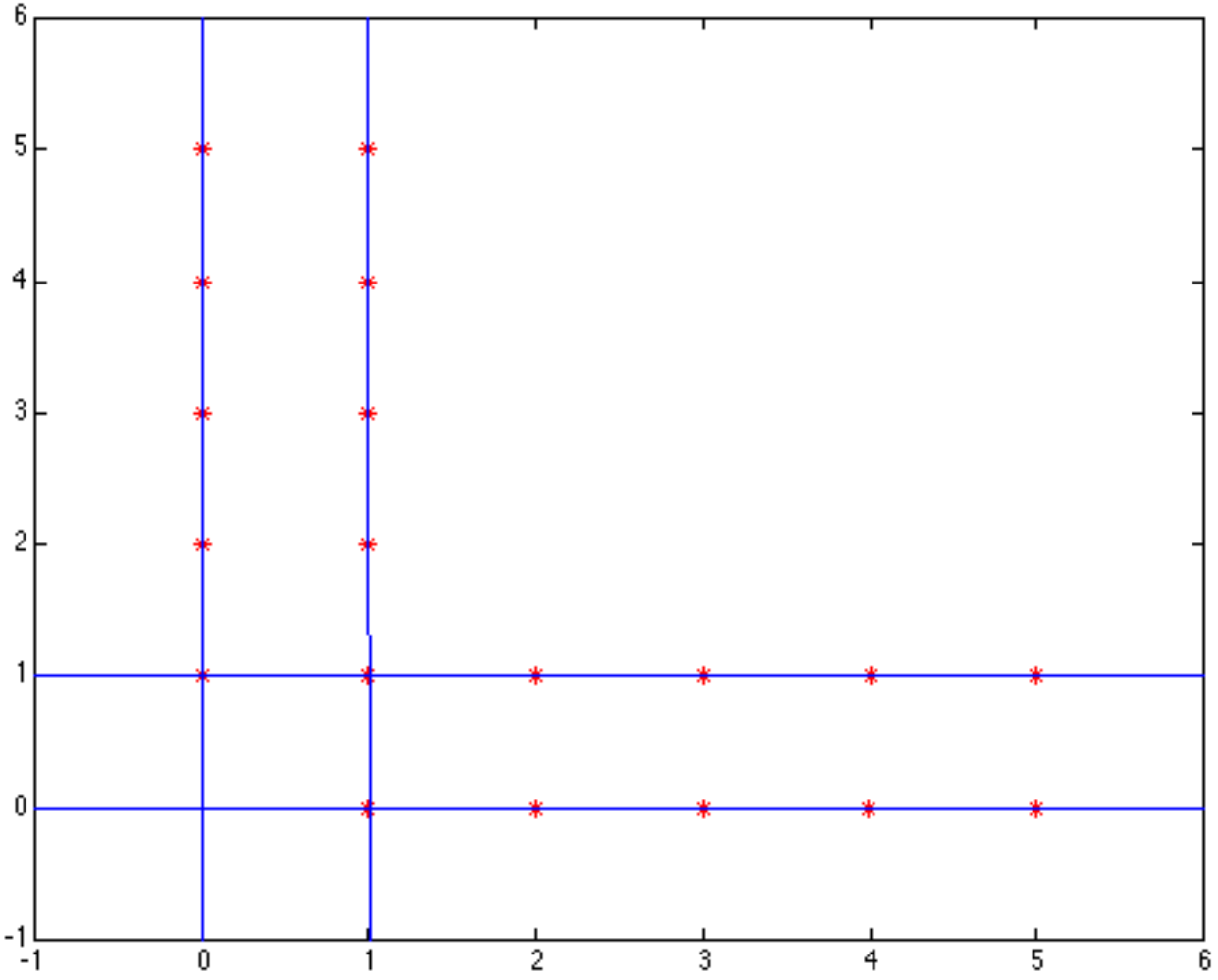
which is an excellent fit to some points



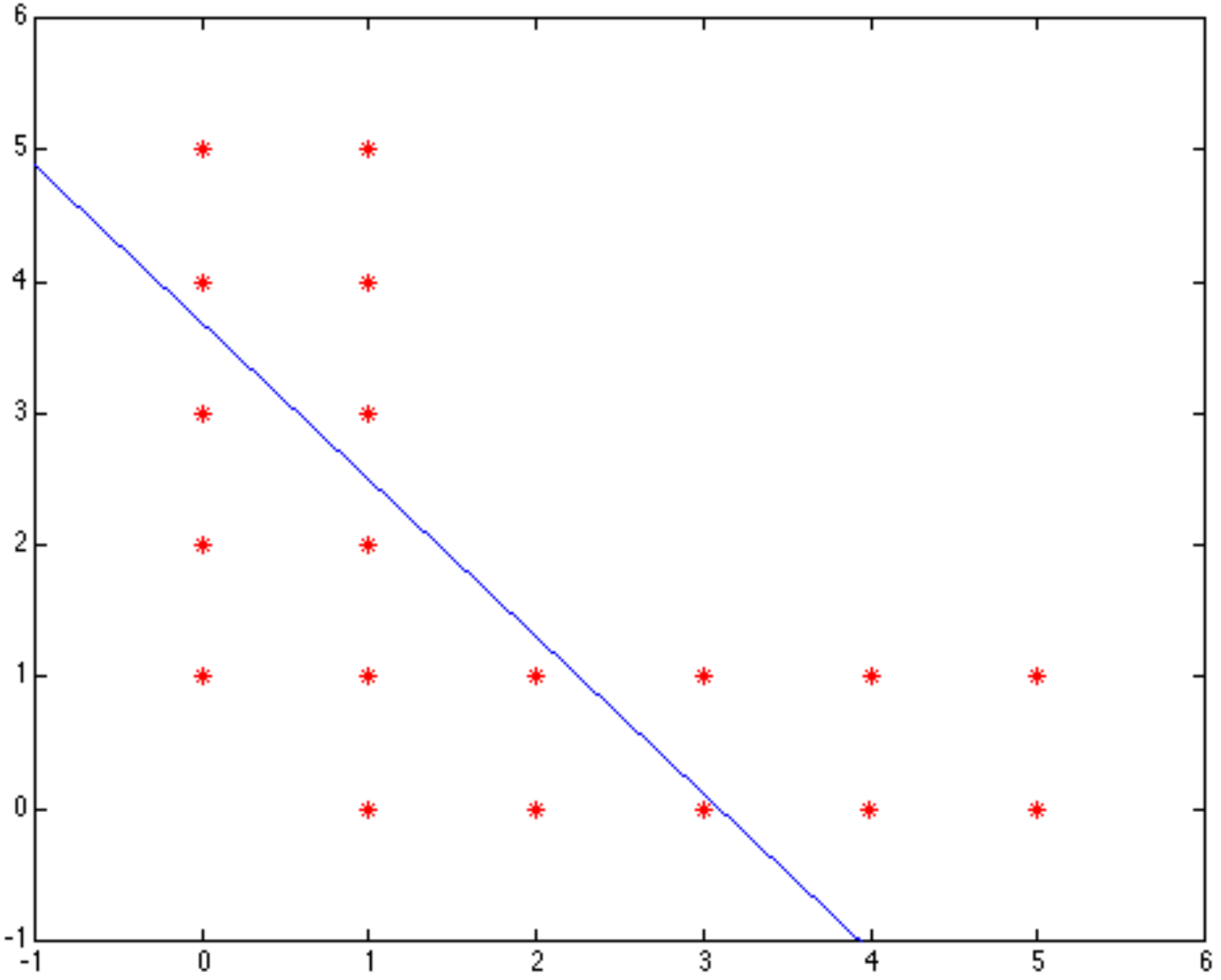
and the deltas for this maximum



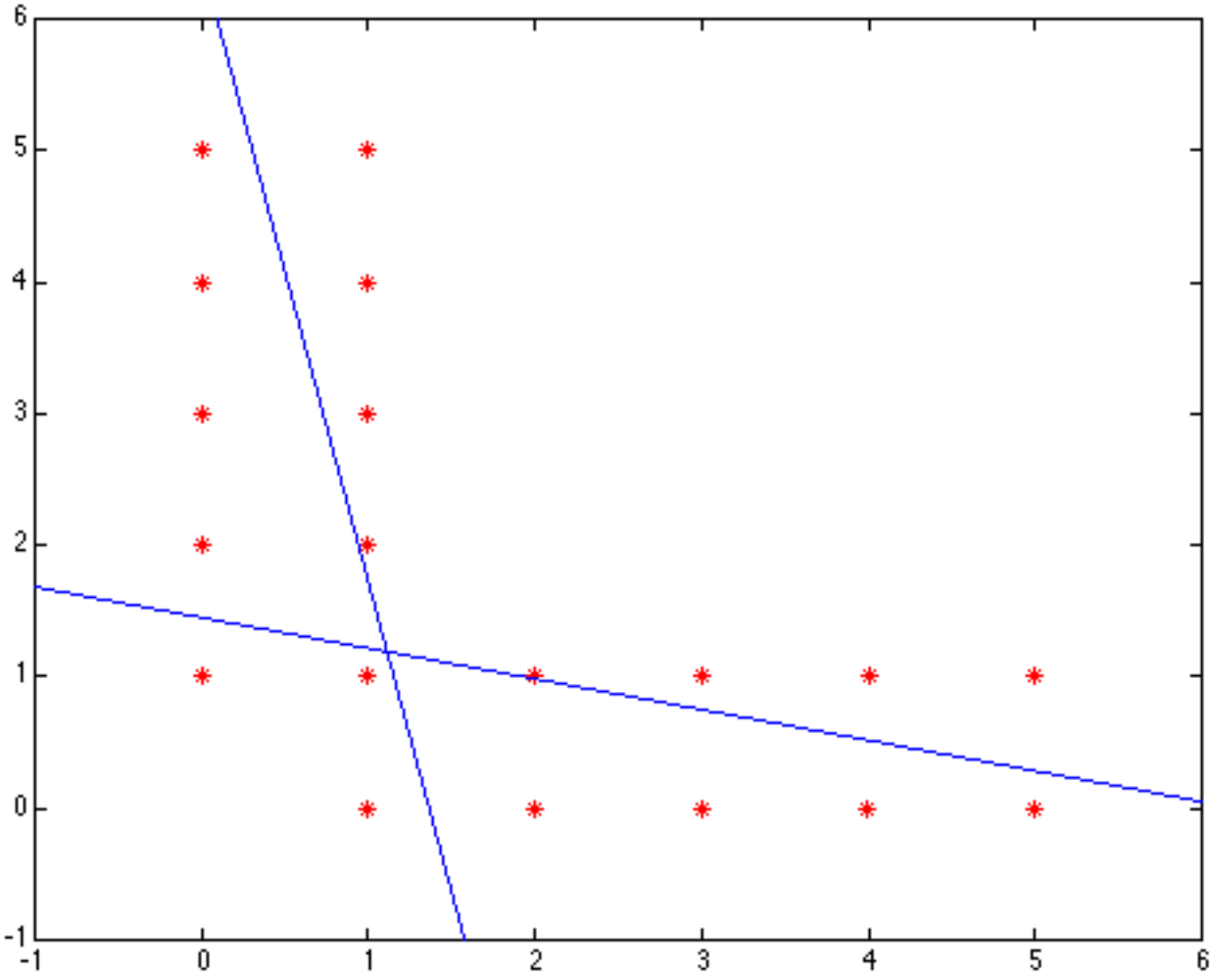
A dataset that is well fitted by four lines



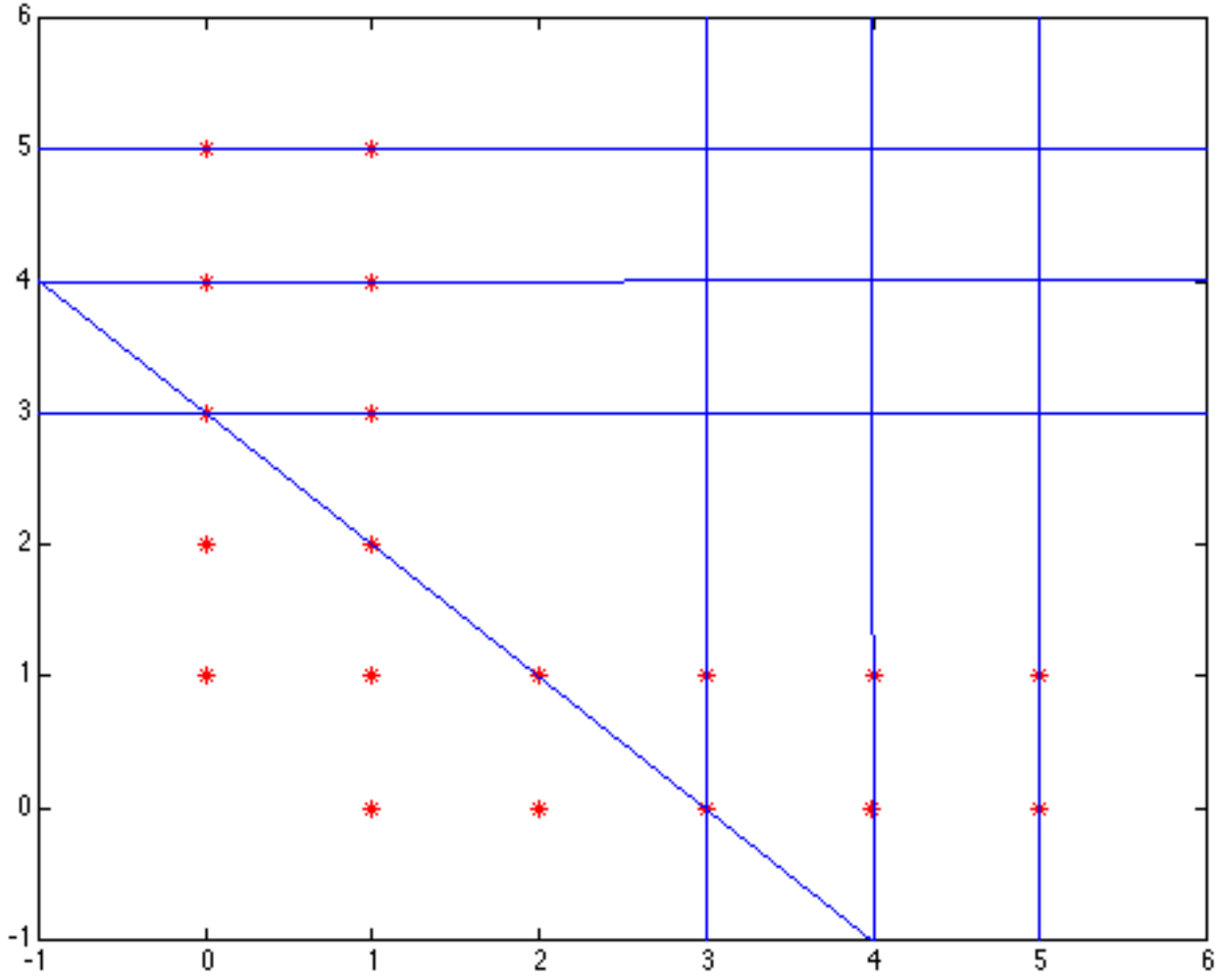
Result of EM fitting, with one line (or at least, one available local maximum).



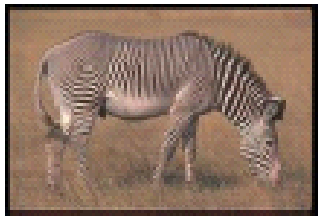
Result of EM fitting, with two lines (or at least, one available local maximum).



Seven lines can produce a rather logical answer



Segmentation with EM



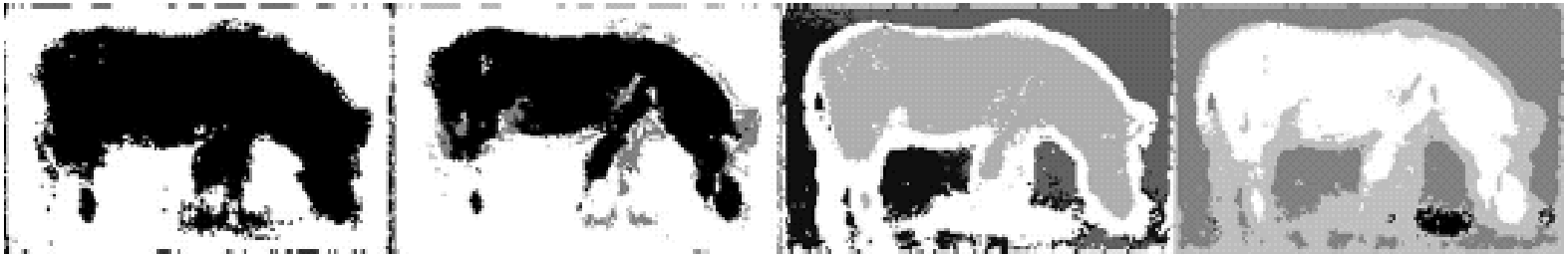


Figure from “Color and Texture Based Image Segmentation Using EM and Its Application to Content Based Image Retrieval”,S.J. Belongie et al., Proc. Int. Conf. Computer Vision, 1998, c1998, IEEE

Motion segmentation with EM

- Model image pair (or video sequence) as consisting of regions of parametric motion
  - affine motion is popular

$$\begin{pmatrix} v_x \\ v_y \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}$$

- Now we need to
  - determine which pixels belong to which region
  - estimate parameters

- Likelihood
  - assume

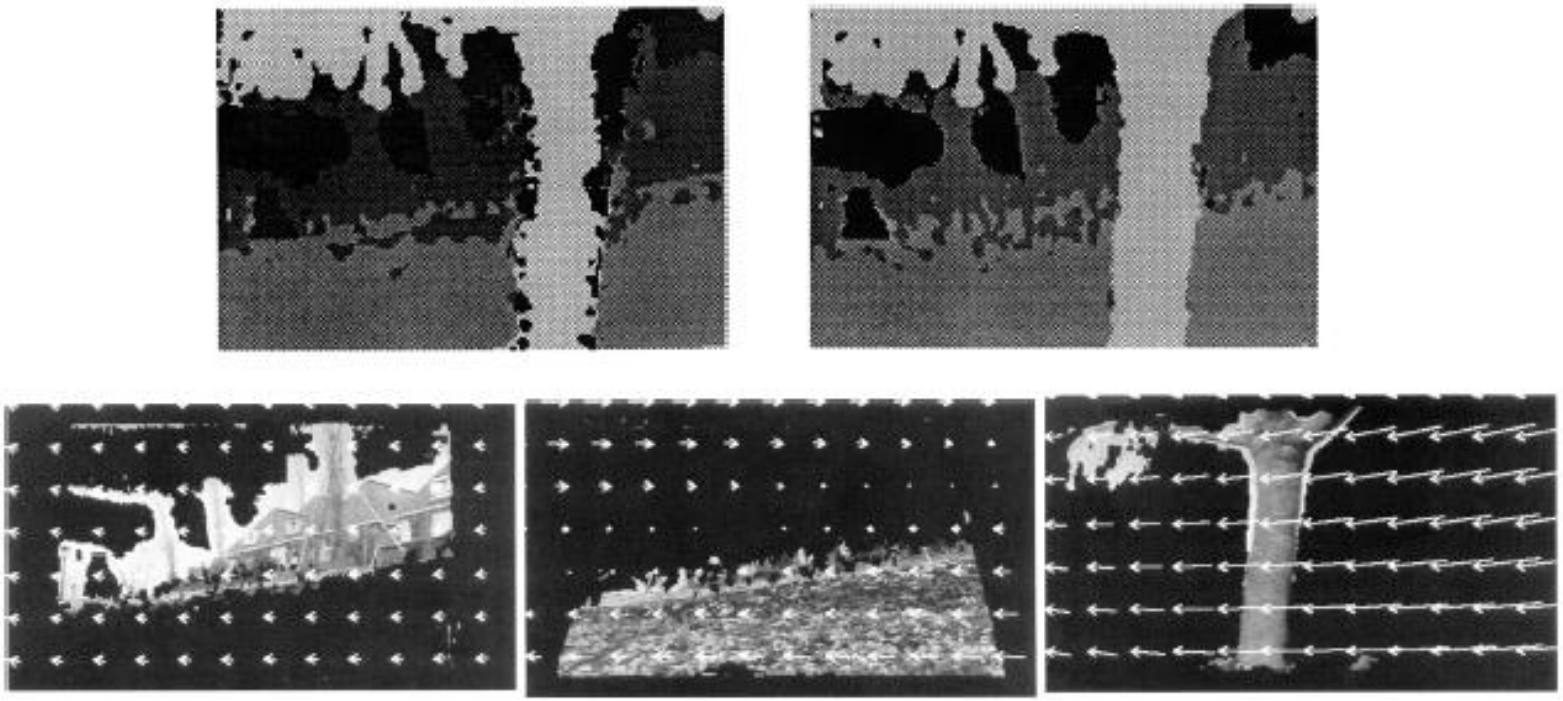
$$I(x,y,t) = I(x+v_x, y+v_y, t+1) + noise$$

- Straightforward missing variable problem, rest is calculation

Three frames from the MPEG “flower garden” sequence

Figure from “Representing Images with layers,”, by J. Wang and E.H. Adelson, IEEE Transactions on Image Processing, 1994, c 1994, IEEE

Grey level shows region no. with highest probability



Segments and motion fields associated with them

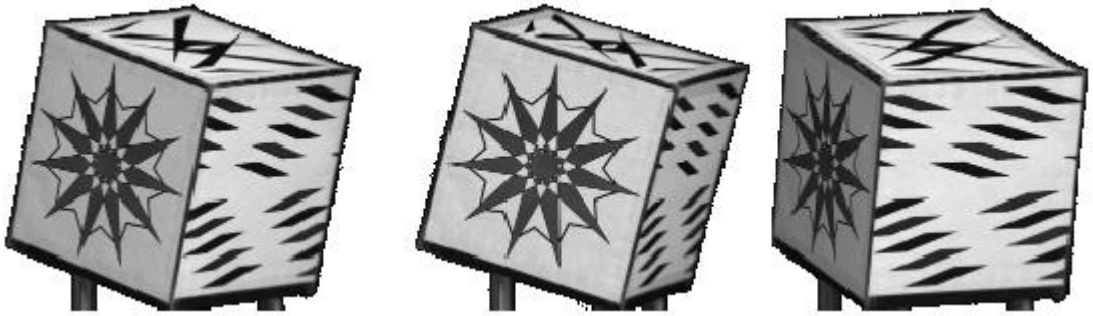
Figure from “Representing Images with layers,”, by J. Wang and E.H. Adelson, IEEE Transactions on Image Processing, 1994, c 1994, IEEE

If we use multiple frames to estimate the appearance of a segment, we can fill in occlusions; so we can re-render the sequence with some segments removed.

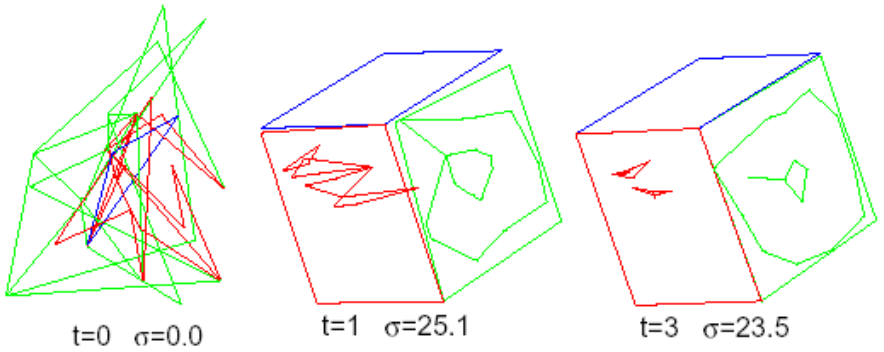
Figure from “Representing Images with layers,”, by J. Wang and E.H. Adelson, IEEE Transactions on Image Processing, 1994, c 1994, IEEE

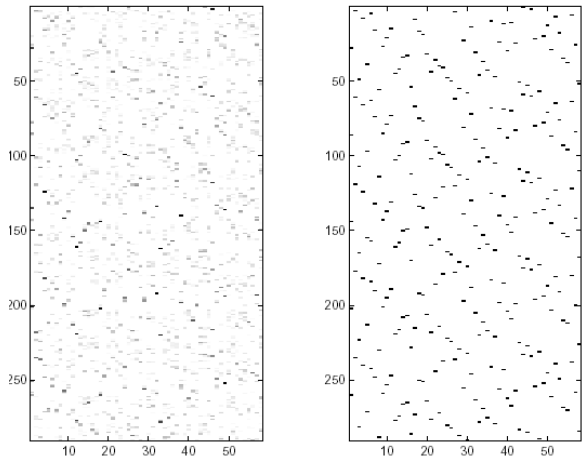
Structure from motion without correspondences

Dellaert et al.00



EM
MCMC
Chain Flipping



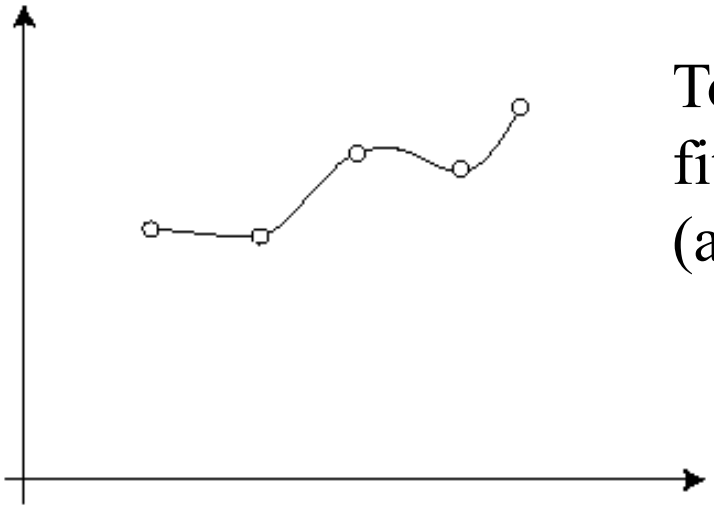


Some generalities

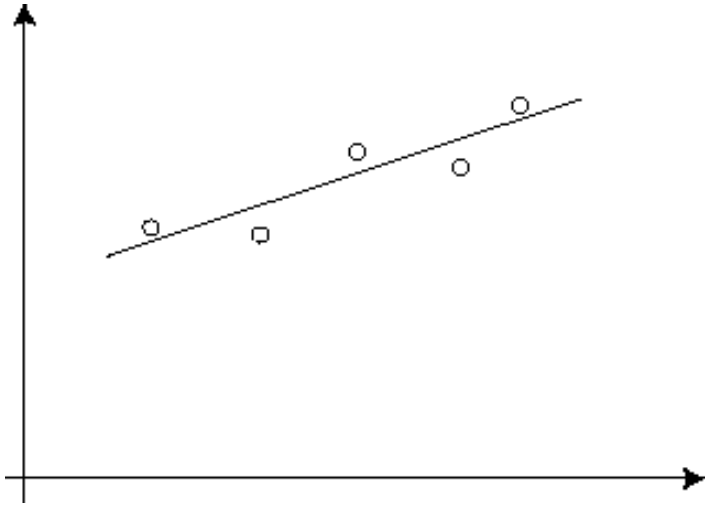
- Many, but not all problems that can be attacked with EM can also be attacked with RANSAC
  - need to be able to get a parameter estimate with a manageably small number of random choices.
  - RANSAC is usually better
- Didn't present in the most general form
  - in the general form, the likelihood may not be a linear function of the missing variables
  - in this case, one takes an expectation of the likelihood, rather than substituting expected values of missing variables
  - Issue doesn't seem to arise in vision applications.

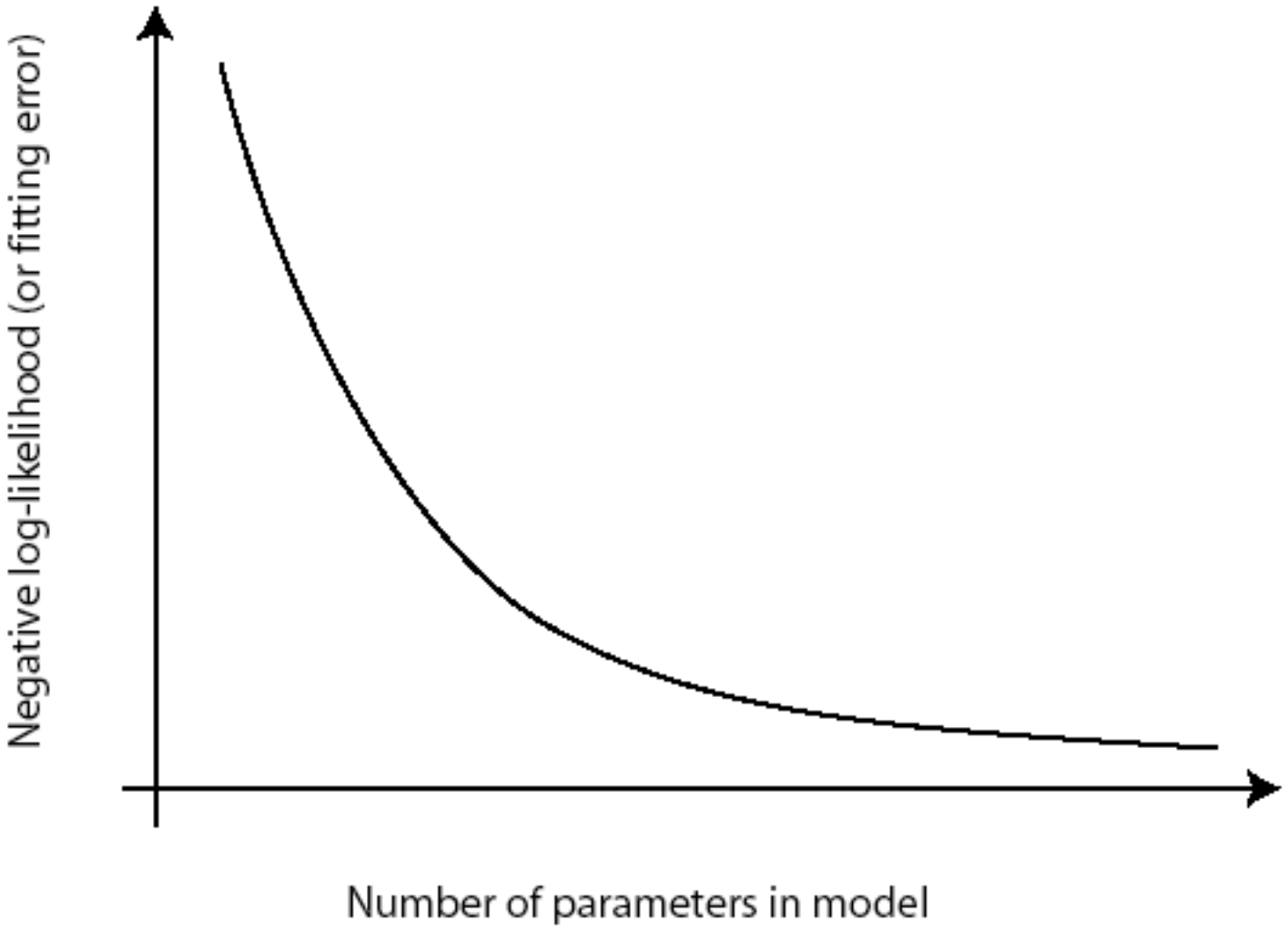
Model Selection

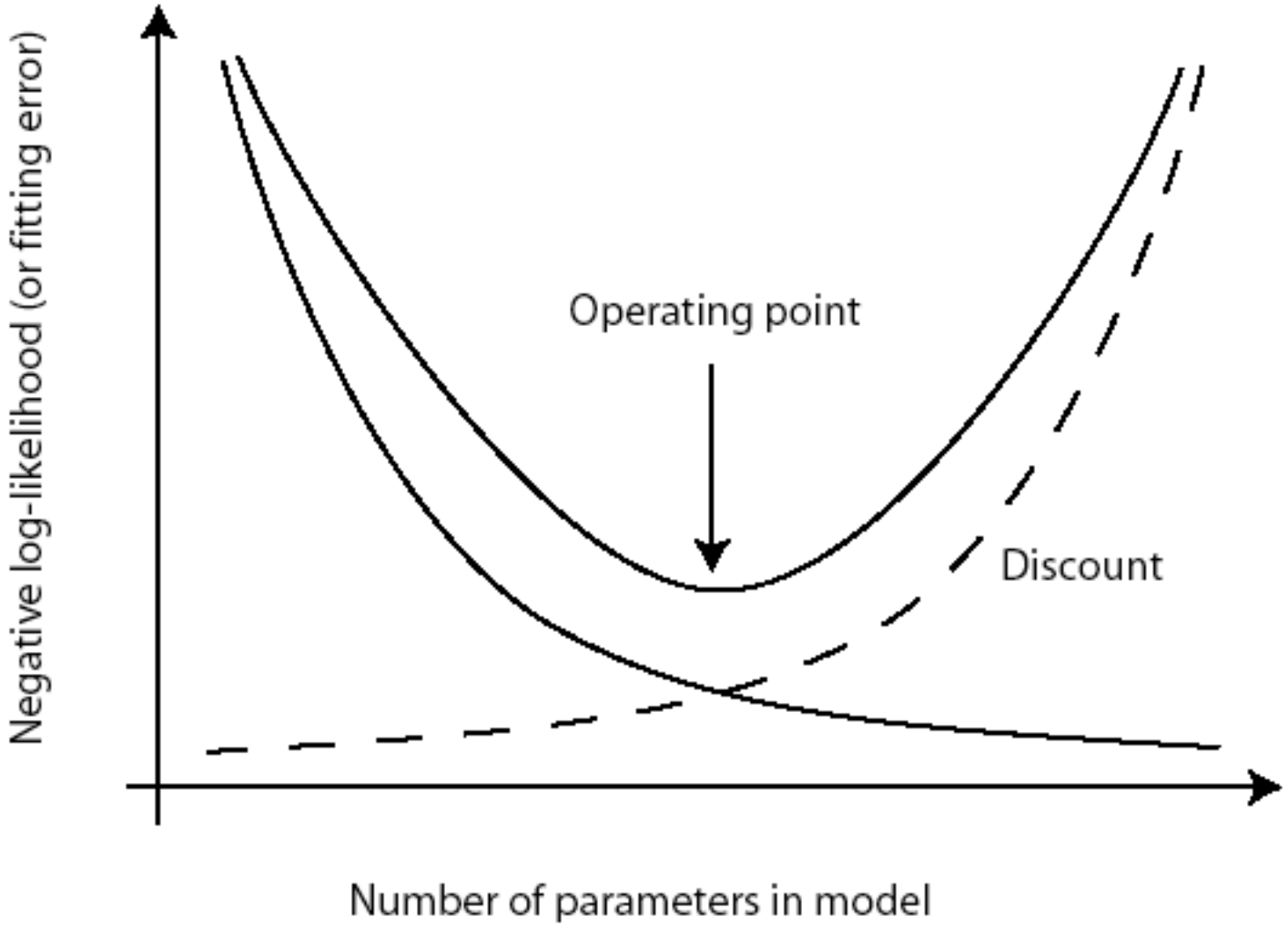
- We wish to choose a model to fit to data
  - e.g. is it a line or a circle?
  - e.g g is this a perspective or orthographic camera?
  - e.g. is there an aeroplane there or is it noise?
- Issue
  - In general, models with more parameters will fit a dataset better, but are poorer at prediction
  - This means we can't simply look at the negative log-likelihood (or fitting error)



Top is not necessarily a better fit than bottom
(actually, almost always worse)




Negative log-likelihood (or fitting error)
Number of parameters in model



We can discount the fitting error with some term in the number of parameters in the model.

Discounts

- AIC (an information criterion)
  - choose model with smallest value of

$$-2L\left(D;\theta^{*}\right)+2p$$

  - p is the number of parameters
- BIC (Bayes information criterion)
  - choose model with smallest value of

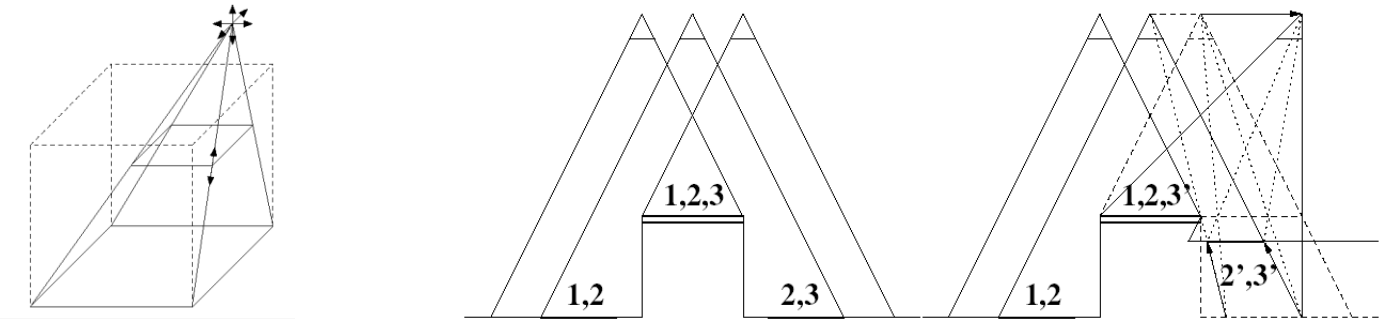
$$-2L\left(D;\theta^{*}\right)+p\log N$$

  - N is the number of data points
- Minimum description length
  - same criterion as BIC, but derived in a completely different way

Example: Dominant planes in USfM

(Pollefeys et al., ECCV' 2002)

Two or three views only sharing points in a plane cause problem for <u>uncalibrated</u> structure from motion



Solution:

- Detect problem through <u>model selection</u>
- Delay computation of ambiguous parameters until more information is available

GRIC: Geometric Robust Information Criterion

(Torr et al., ICCV'98)

(inspired from Kanatani's GIC)

$$\text{GRIC} = \sum \rho(e_i^2) + (\underbrace{nd\ln(r)}_{\text{structure complexity}} + \underbrace{k\ln(rn)}_{\text{model complexity}})$$

$$\rho(e^2) = \min\left(\frac{e^2}{\sigma^2}, \underbrace{2(r-d)}_{\text{co-dimension}}\right) .$$

robust!

(otherwise outliers will make you get the most expensive model every time, i.e. the model is never good enough for them)

the residuals e_i

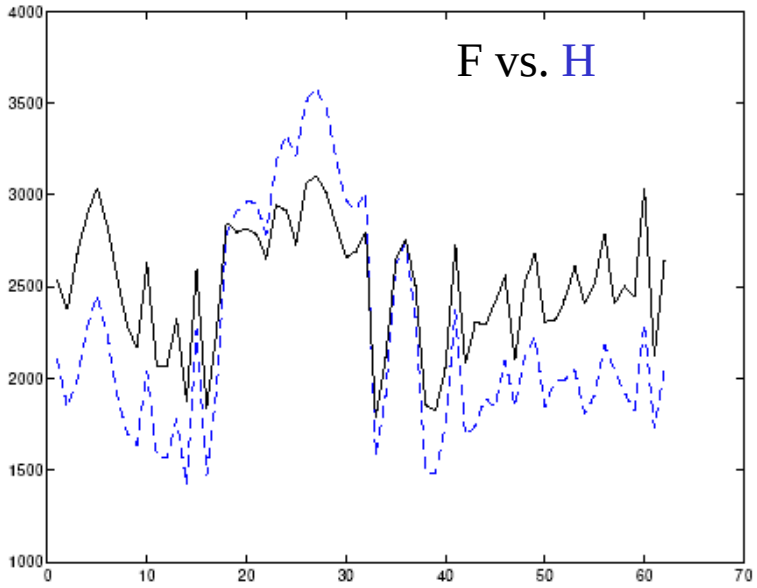
the standard deviation of the measurement error σ

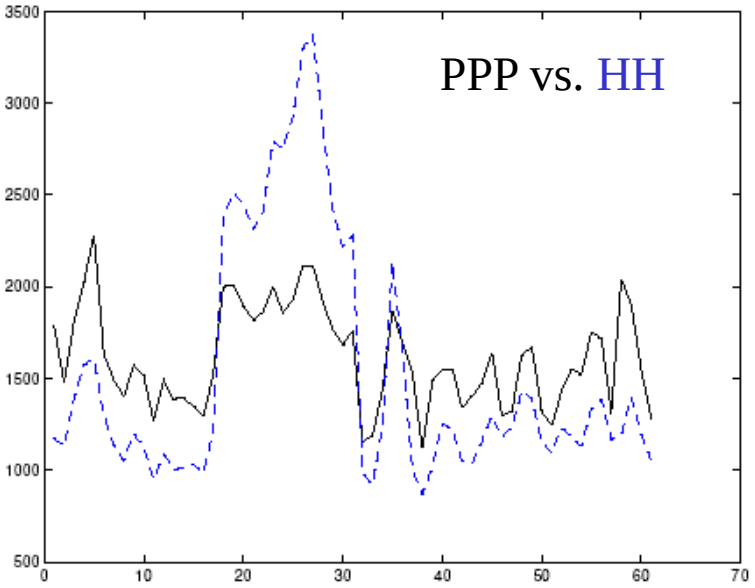
number of n of inliers plus outliers

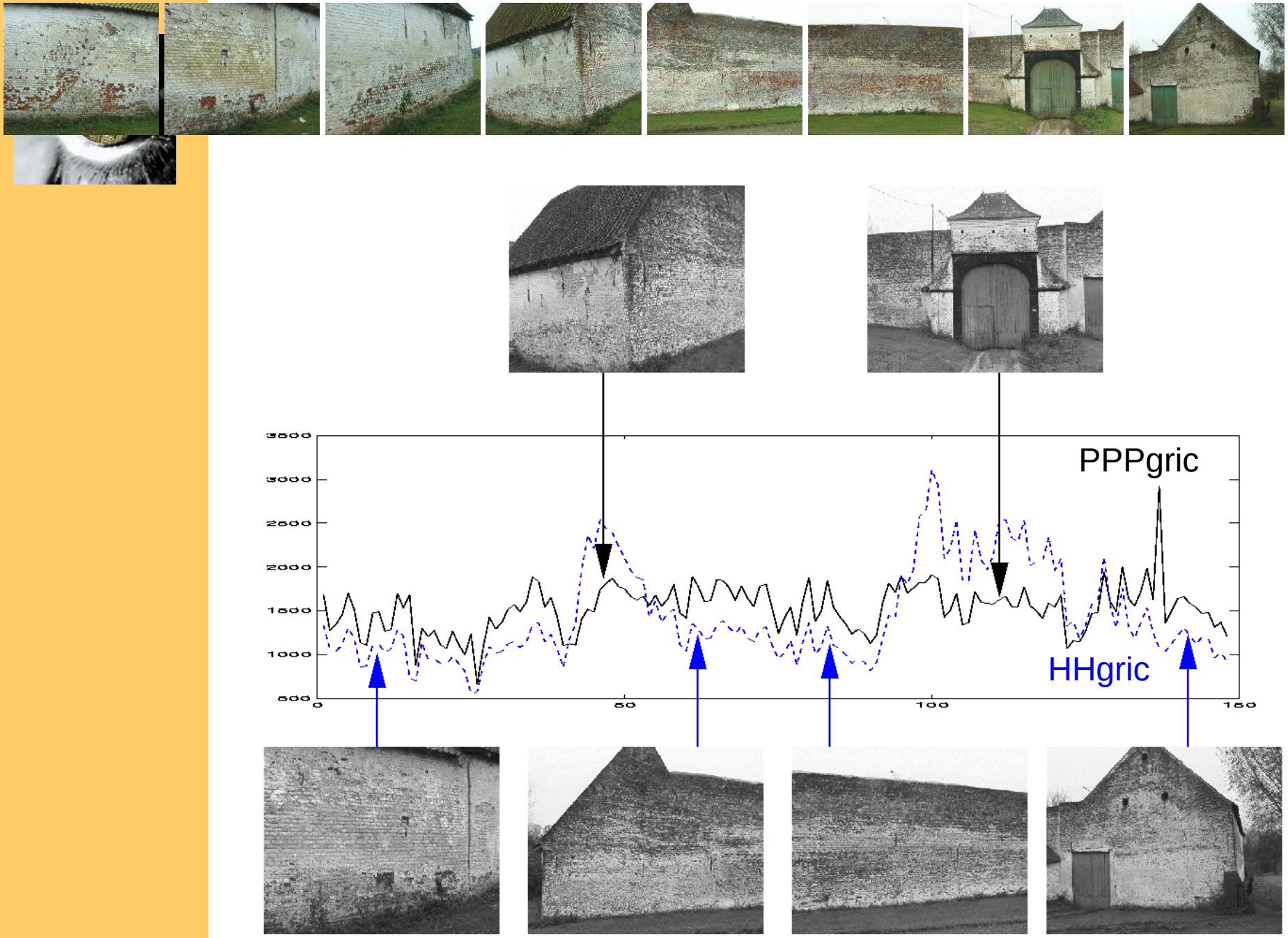
dimension d of the structure (2D or 3D)

the dimension of the data r. (6 for points in 3 views)

number k of motion model parameters (11x3-15 (PPP) or 8x2(HH))


F vs. H
4000
3500
3000
2500
2000
1500
1000
0
10
20
30
40
50
60
70

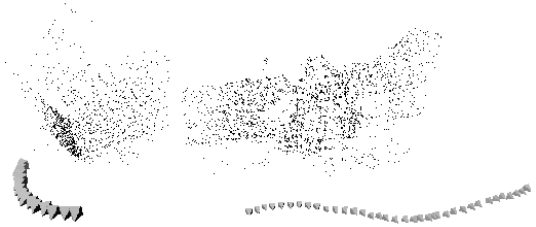

PPP vs. HH
3500
3000
2500
2000
1500
1000
500
0
10
20
30
40
50
60
70

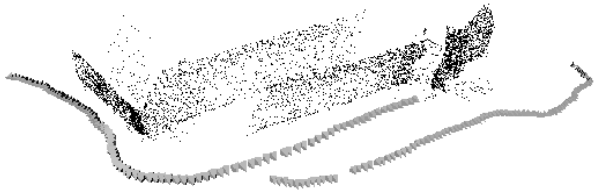

PPPgric
HHgric
3500
3000
2500
2000
1500
1000
500
0
50
100
150

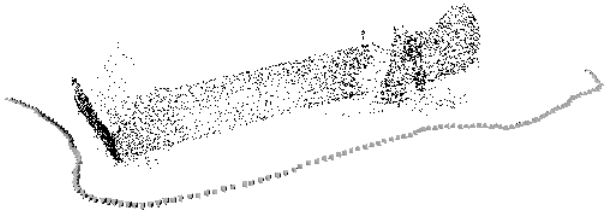
Farmhouse 3D models

Strategy:

- Detect problem using model selection
- Reconstruct planes and 3D separately
- Perform self-calibration
- Assemble reconstruction
- compute poses (for planar parts)
- Refine complete reconstruction (bundle adjustment)







Cross-validation

- Split data set into two pieces, fit to one, and compute negative log-likelihood on the other
- Average over multiple different splits
- Choose the model with the smallest value of this average

- The difference in averages for two different models is an estimate of the difference in KL divergence of the models from the source of the data

Model averaging

- Very often, it is smarter to use multiple models for prediction than just one
- e.g. motion capture data
  - there are a small number of schemes that are used to put markers on the body
  - given we know the scheme S and the measurements D, we can estimate the configuration of the body X

- We want

$$P(X \mid D) = P(X \mid S_1, D)P(S_1 \mid D) + P(X \mid S_2, D)P(S_2 \mid D) + P(X \mid S_3, D)P(S_3 \mid D)$$

- If it is obvious what the scheme is from the data, then averaging makes little difference
- If it isn't, then not averaging underestimates the variance of X --- we think we have a more precise estimate than we do.

Next week:
Image Segmentation