


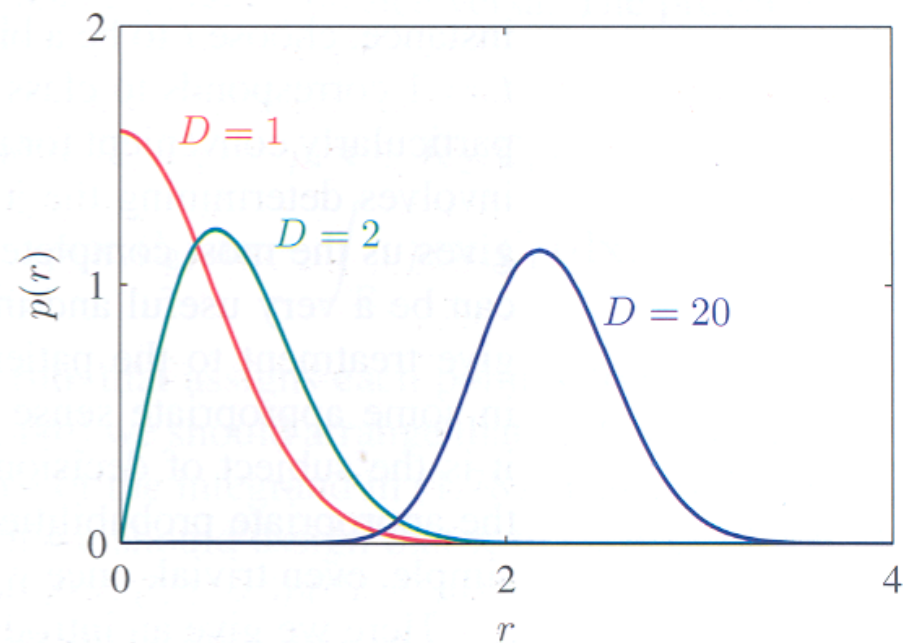
Notes on K-Means

- K-means is “hard” clustering—each point is in exactly one cluster
 - What you get is a function of starting “guess”
 - The error goes down with every iteration
 - This means you get a local minimum
 - Unfortunately, the dimension of the space is usually large, and there are lots of local maximum (standard problem!)
 - Dimensionality here is $K * \dim(\mathbf{x})$
 - Finding the global minimum for a real problem is very optimistic!
- you should be able to argue why this is true
- 

Clustering and dimensionality

- A simple clustering method that does **not** make sense in high dimensions
 - Partition space into hypercubes of edge size $1/K$
 - Put points into the appropriate cube
 - Most cubes do not get used (there are too many of them!)
- Real data lives in low dimensional manifolds (plus noise perturbations)
 - Most of a high dimensional space is not used
- A distance function takes two high dimensional points and maps them into a single number, which loses a lot of information about the points.
- Non-intuitive fact --- points in a cluster tend to be about the same distance away from the center

Figure 1.23 Plot of the probability density with respect to radius r of a Gaussian distribution for various values of the dimensionality D . In a high-dimensional space, most of the probability mass of a Gaussian is located within a thin shell at a specific radius.

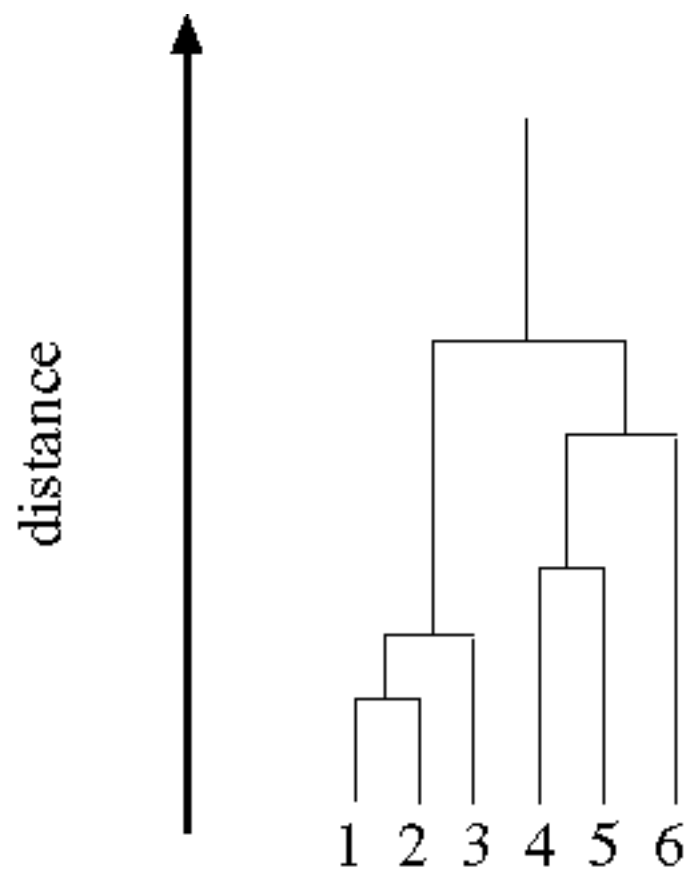


Clustering approaches

- Agglomerative clustering
 - initialize: every item is a cluster
 - attach item that is “closest” to a cluster to that cluster
 - repeat
- Divisive clustering
 - split cluster along best boundary
 - repeat
- Probabilistic clustering
 - define a probabilistic grouping model
 - pseudo-example is K-means
 - more principled example is Gaussian Mixture Model (GMM)

Simple agglomerative clustering

- Point-Cluster or Cluster-Cluster distance
 - single-link clustering (minimum distance from point to points in clusters or among pairs of points, one from each cluster)
 - complete-link clustering (maximum)
 - group-average clustering (average)
 - (terms are not important, but concepts are worth thinking about)
- Dendrograms
 - classic picture of output as clustering process continues



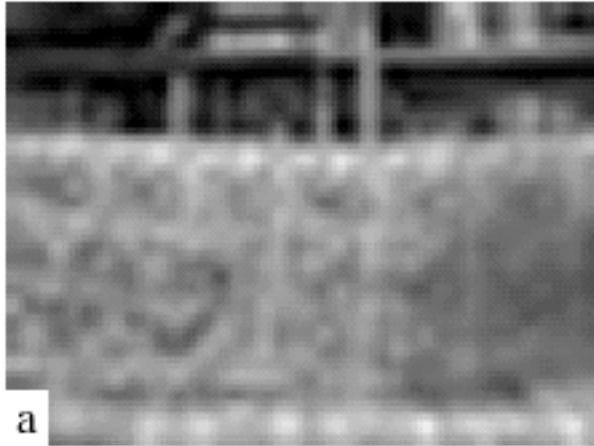
Grouping by Background Subtraction

- If we know what the background looks like, it is easy to identify “interesting bits”
- Applications
 - Is a person in their office
 - Tracking cars on a road
 - Surveillance
- Approach:
 - Use a moving average to estimate background image
 - Subtract from current frame
 - Large absolute values are interesting pixels
 - trick: use morphological operations to clean up pixels (remove “holes”)

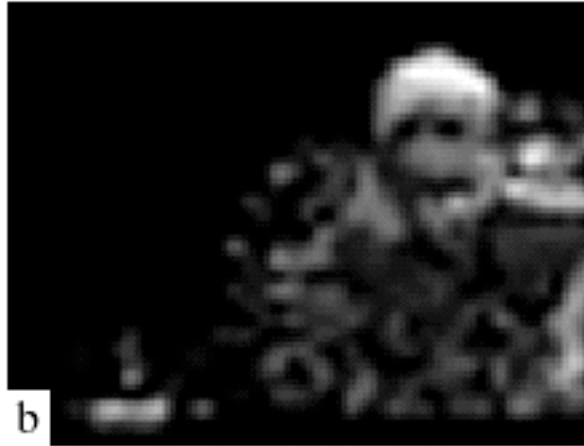
Grouping by Background Subtraction



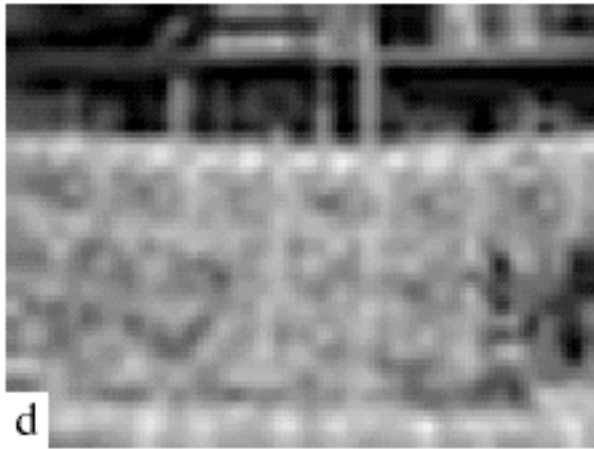
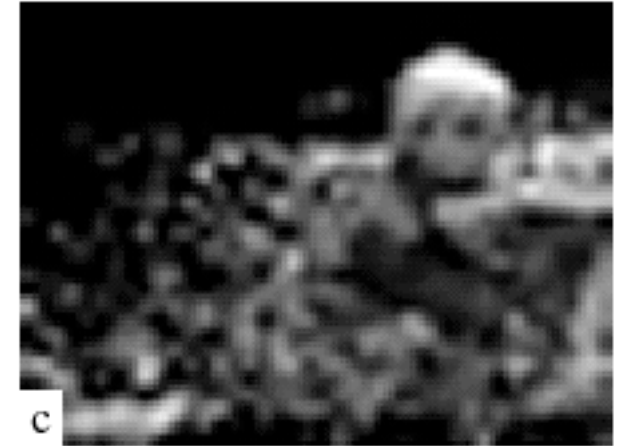
a) average of sequence



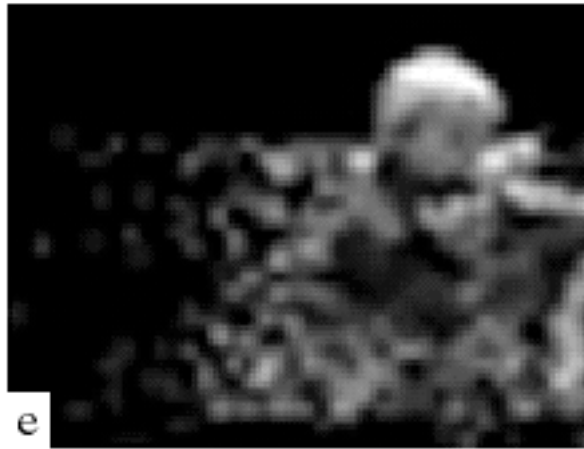
b) difference from background $> T_1$



c) difference from background $> T_2$



d) using EM (discussed later)



e) Foreground using EM (discussed later)

Grouping by Fitting to a Model

- Work with a parametric representation for “objects”
 - (e.g “line”, “ellipse”).
- 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 a given set of tokens best?
 - which of several objects gets which token? (**correspondence!**)
 - how many objects are there?

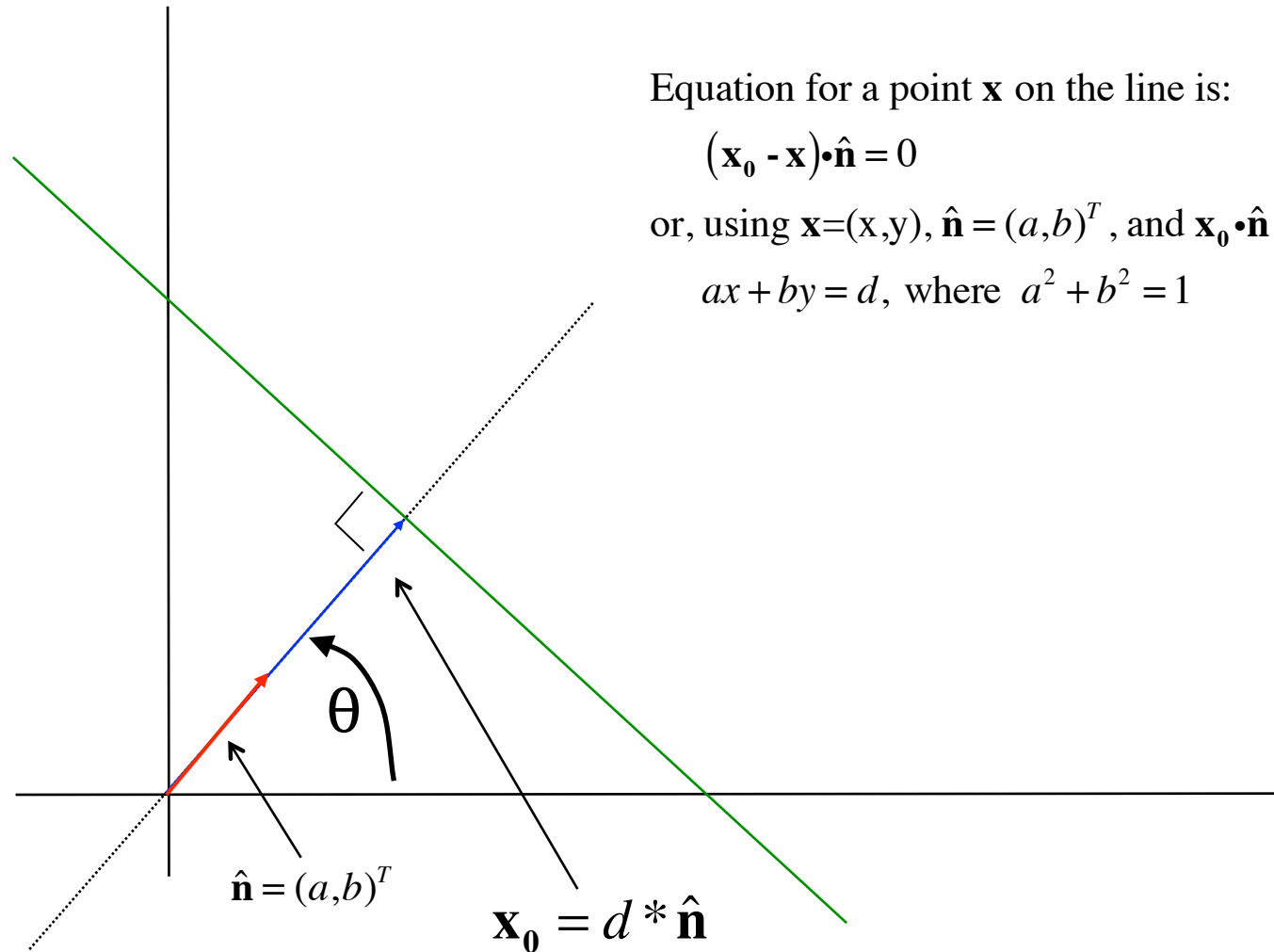
Line Equation (review)

Equation for a point \mathbf{x} on the line is:

$$(\mathbf{x}_0 - \mathbf{x}) \cdot \hat{\mathbf{n}} = 0$$

or, using $\mathbf{x}=(x,y)$, $\hat{\mathbf{n}} = (a,b)^T$, and $\mathbf{x}_0 \cdot \hat{\mathbf{n}} = d$,

$$ax + by = d, \text{ where } a^2 + b^2 = 1$$



Example: Hough Transform for lines

- A line is the set of points (x, y) such that

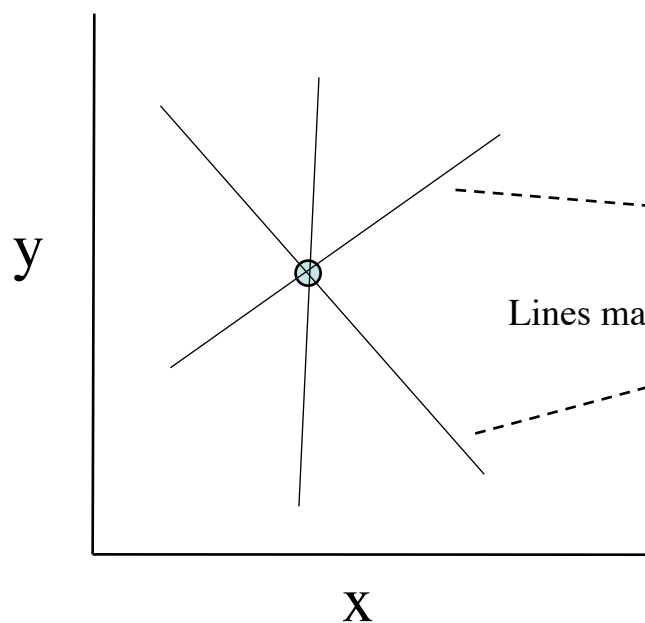
$$(\sin \theta)x + (\cos \theta)y + d = 0$$

- Different choices of θ , $d > 0$ give different lines
- For any (x, y) there is a family of lines through this point, given by

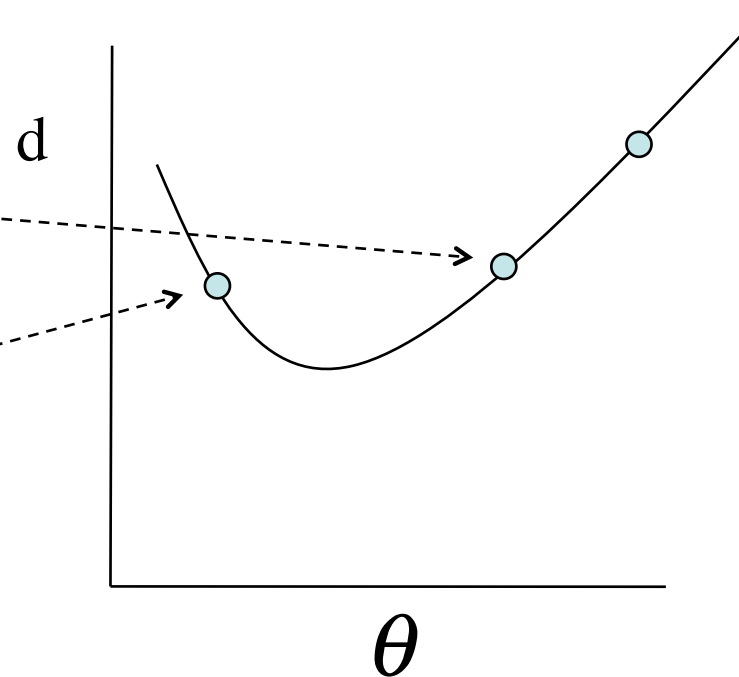
$$(\sin \theta)x + (\cos \theta)y + d = 0$$

- The choice of θ determines d (and vice versa). Thus the family of lines through (x, y) has **one** free parameter (intuitively true).

Data space



Parameter space



Lines map to point

$$(\sin \theta)x + (\cos \theta)y + d = 0$$

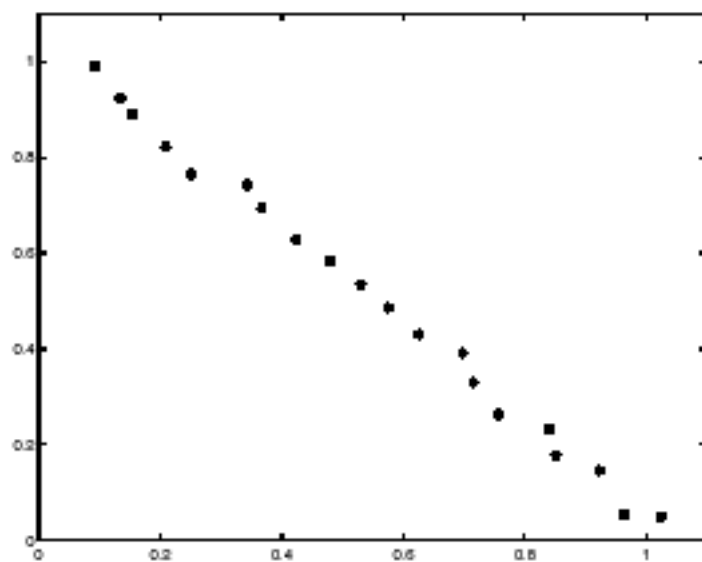
(Note: Curve is **not** accurate)

Example: Hough Transform for lines

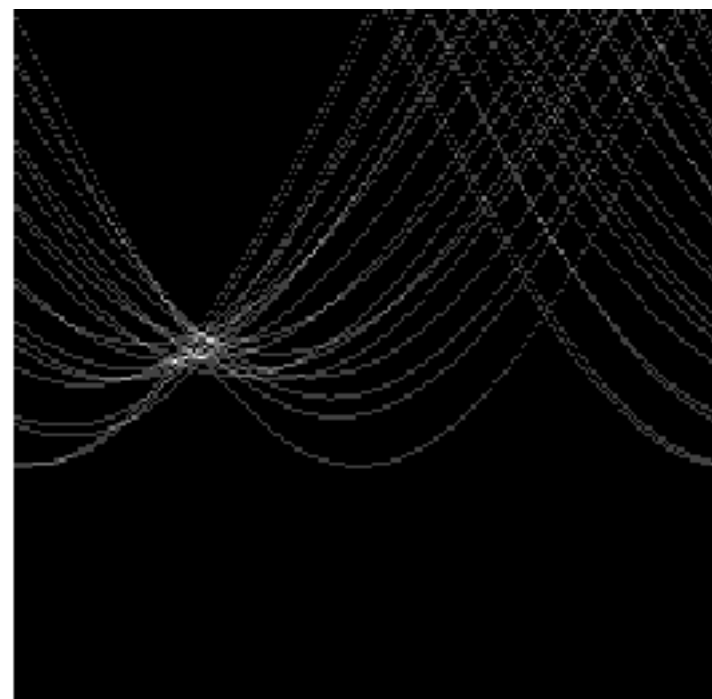
- Main idea: Each observed (x,y) votes for all (θ, d) satisfying

$$(\sin \theta)x + (\cos \theta)y + d = 0$$

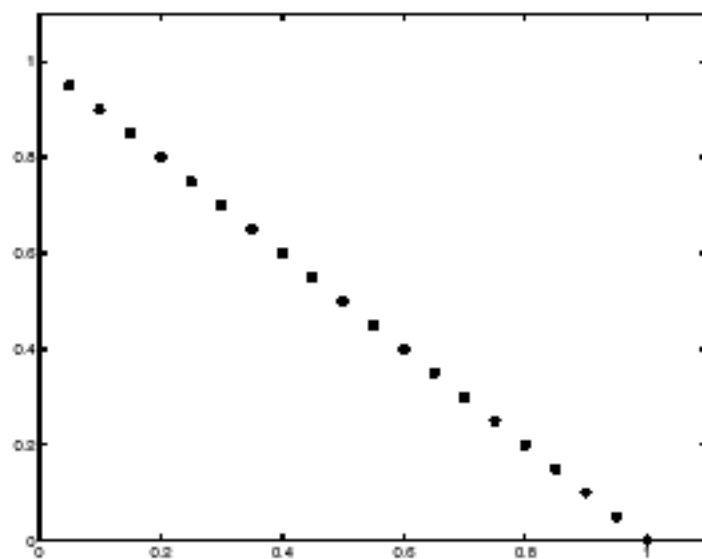
- Discretize the parameter space (θ, d) by an array
- Now each (x,y) leads to a bunch of votes (counts) in a (θ, d) grid (along the curve in the preceding slide).
- To find lines, let all edge points (x,y) vote, and look for (θ, d) cells with lots of votes.



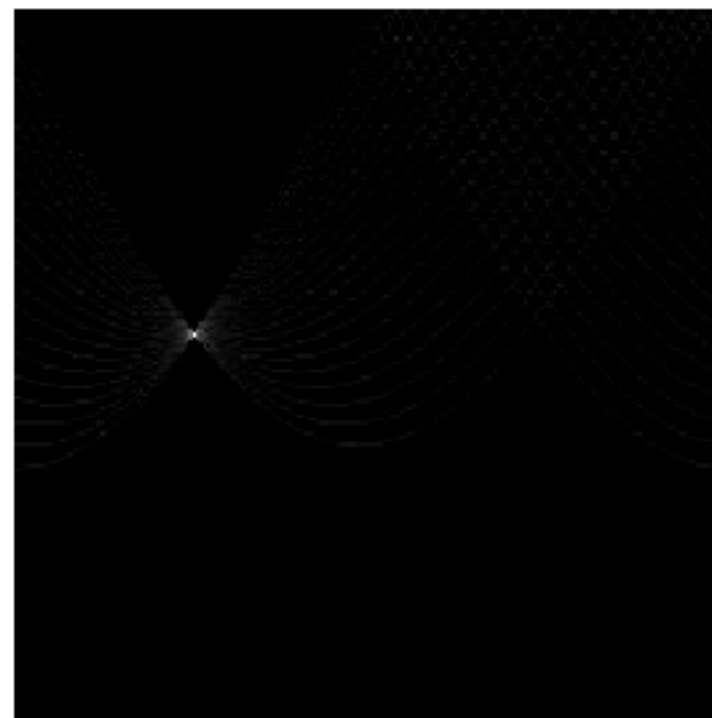
tokens



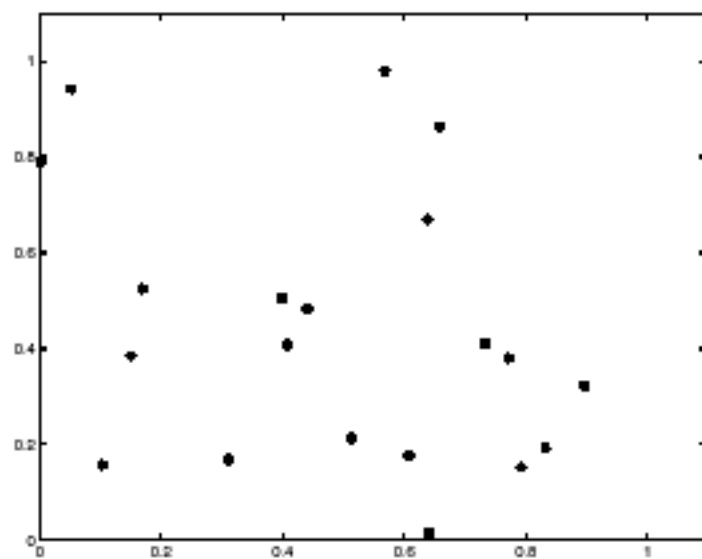
votes



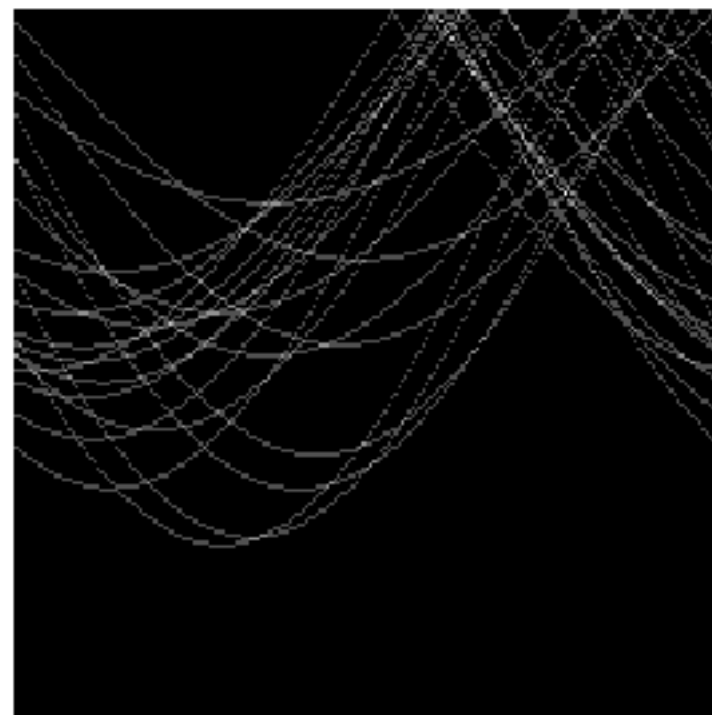
tokens



votes



tokens

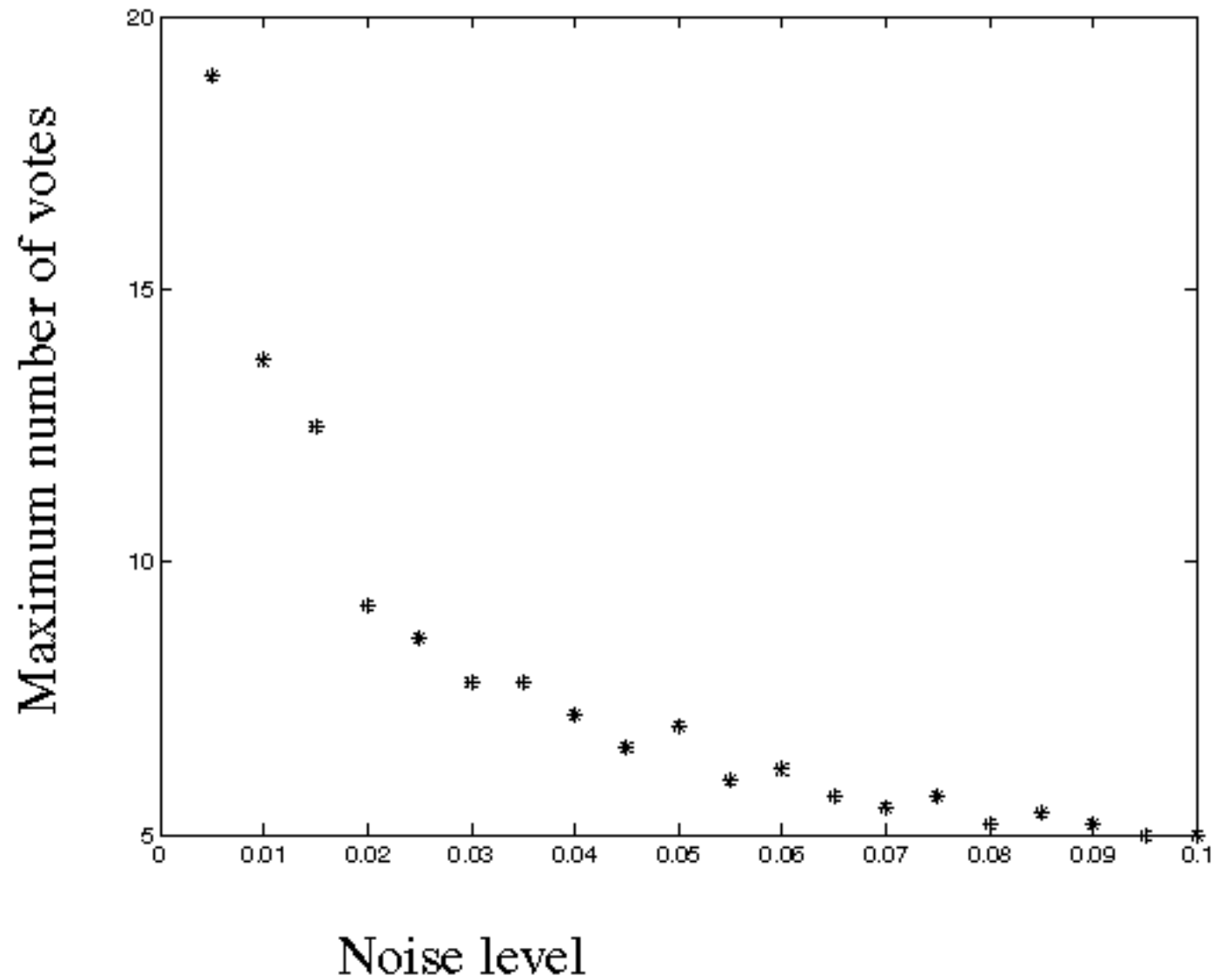


votes

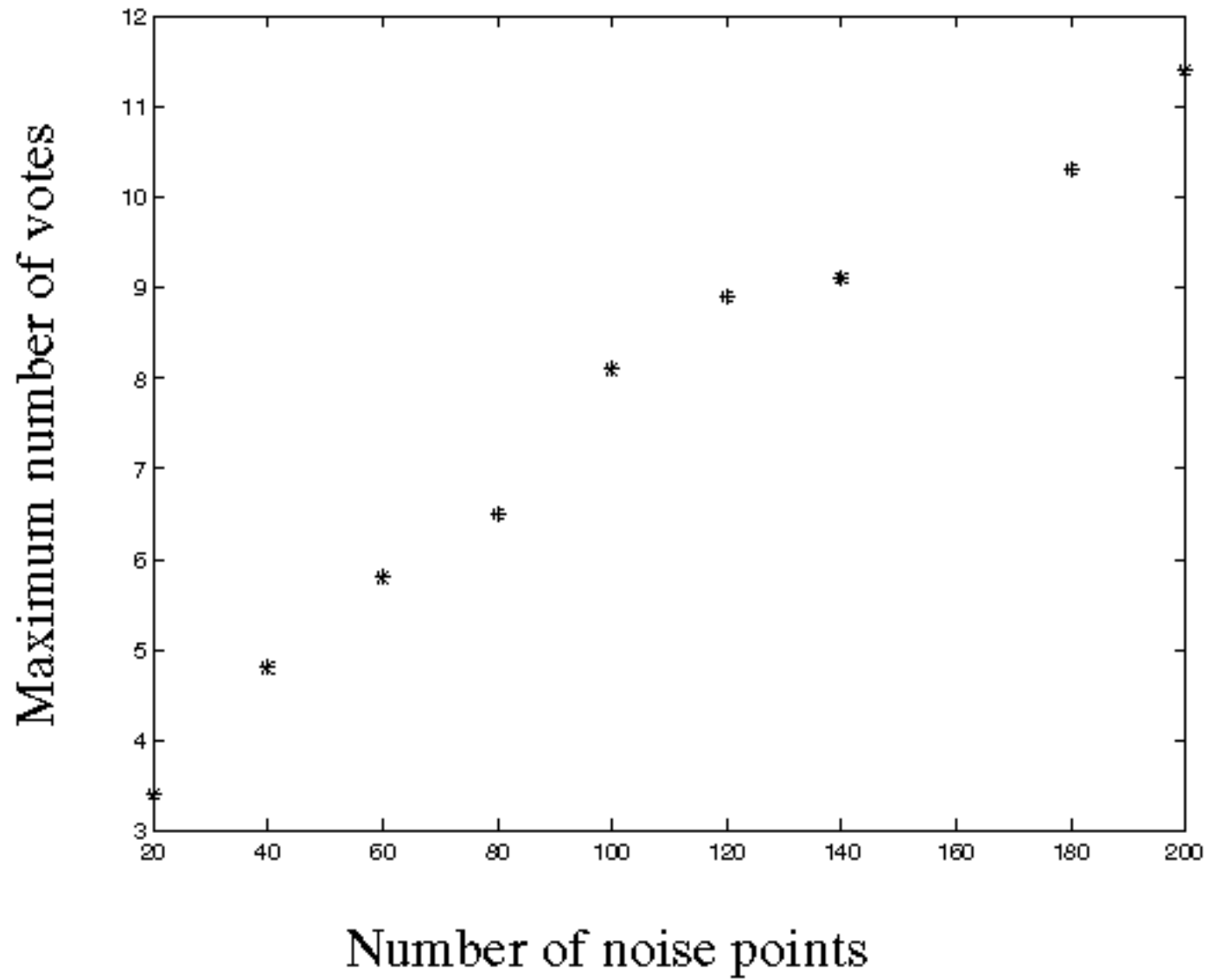
Difficulties with the Hough transform

- 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
- Hough transform is a useful idea, but it is not often satisfactory in practice, because problems with noise and cell size defeat it

Votes for a real line of 20 points versus noise



Votes for a line in a picture that does not have one



Line (model) fitting Challenges

- Robustness
 - Squared error grows rapidly as distance increases
 - Since large distance is unlikely given Gaussian assumption, this means that either the assumption or model is likely incorrect!
- How do we know whether a point is on the line?
 - Incremental line fitting
 - K-means line fitting
 - Probabilistic with missing data

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
```

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

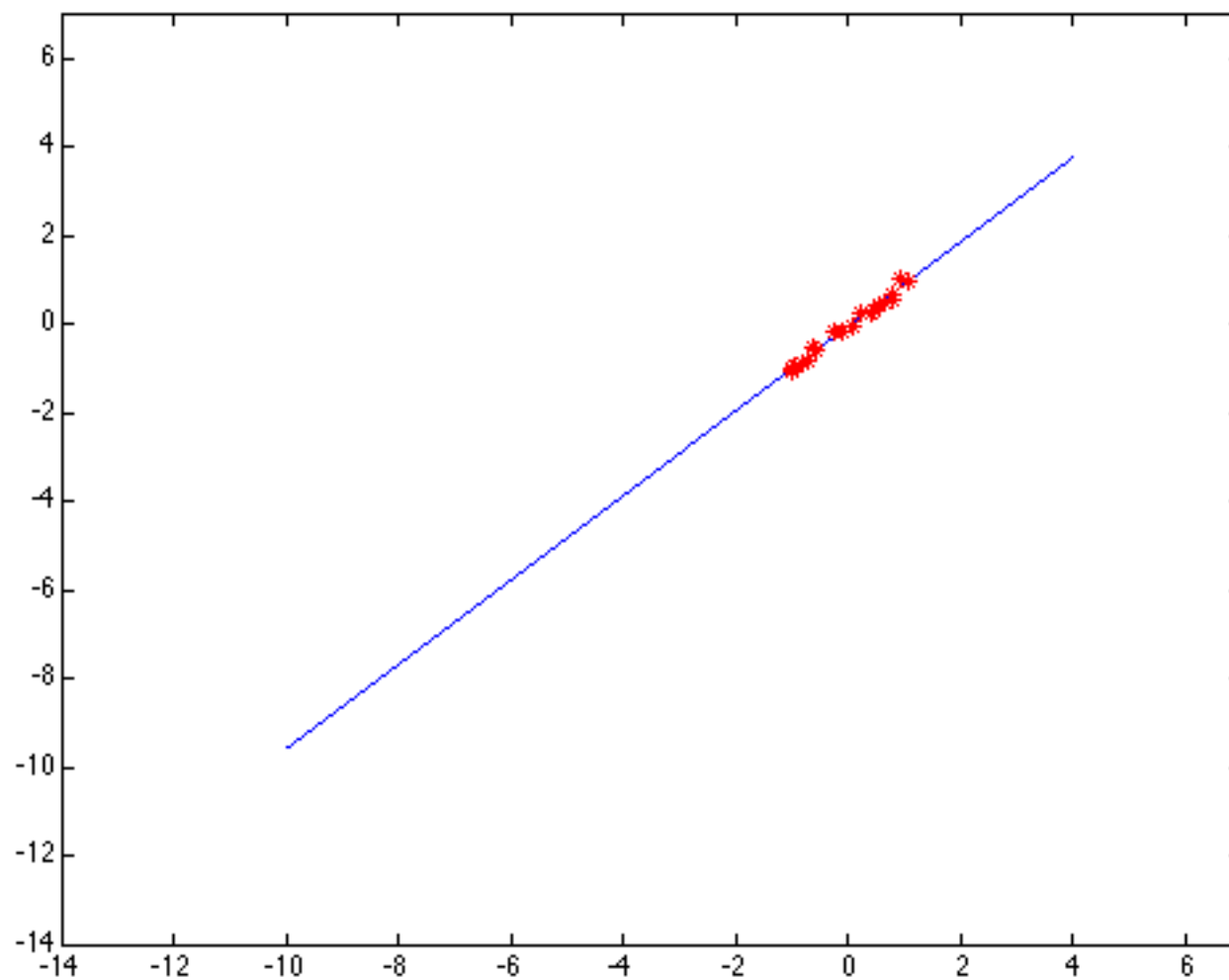
 Refit lines

end

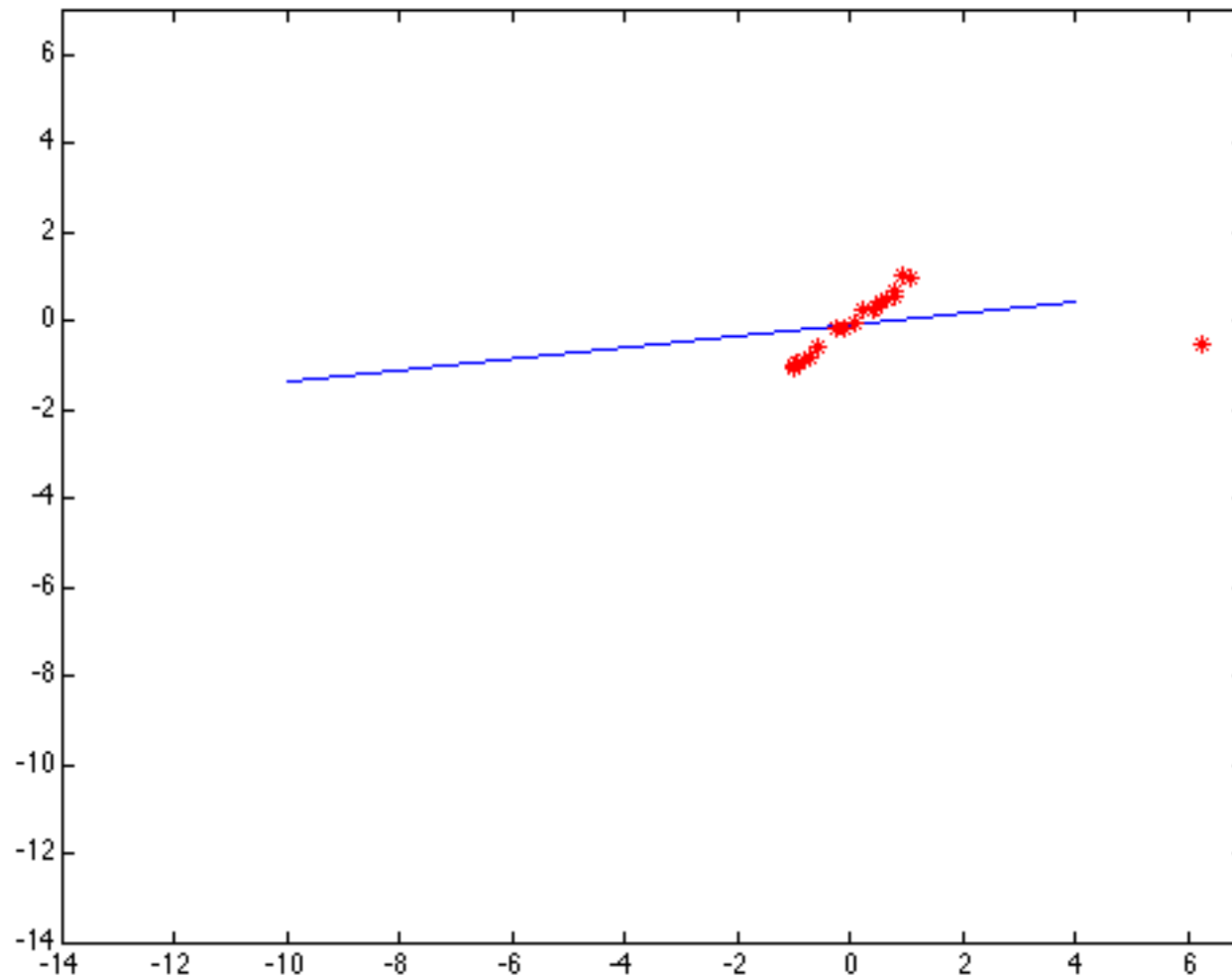
Robustness

- Squared error is a liability when model is wrong
 - One fix is to use an M-estimator
 - Square nearby, threshold far away
 - A solution that ignores outliers is RANSAC
 - Search for good points
 - Another fix is probabilistic grouping with a “noise” cluster
 - Often fit using Expectation-Maximization (covered later)

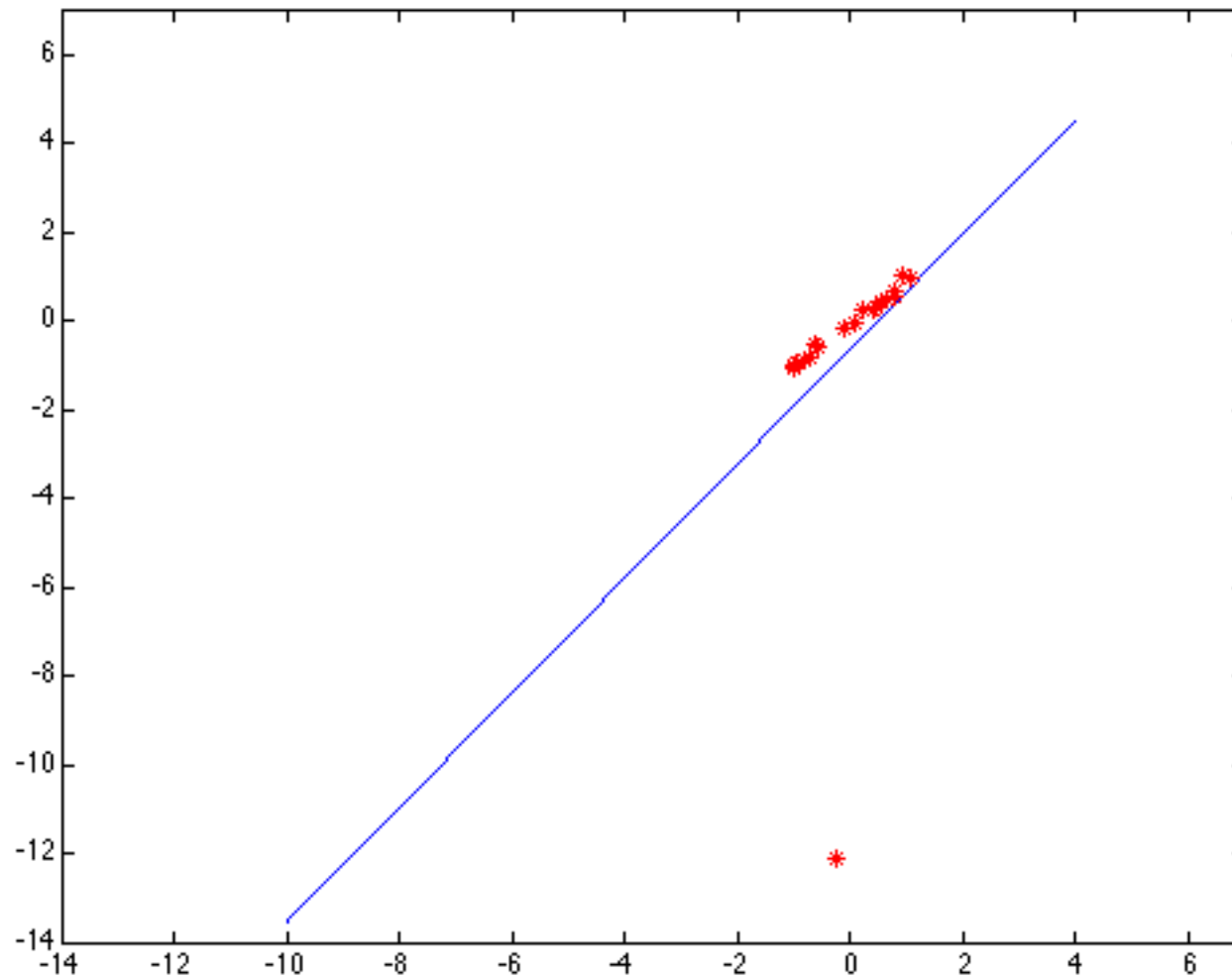
Least squares fit (good example)



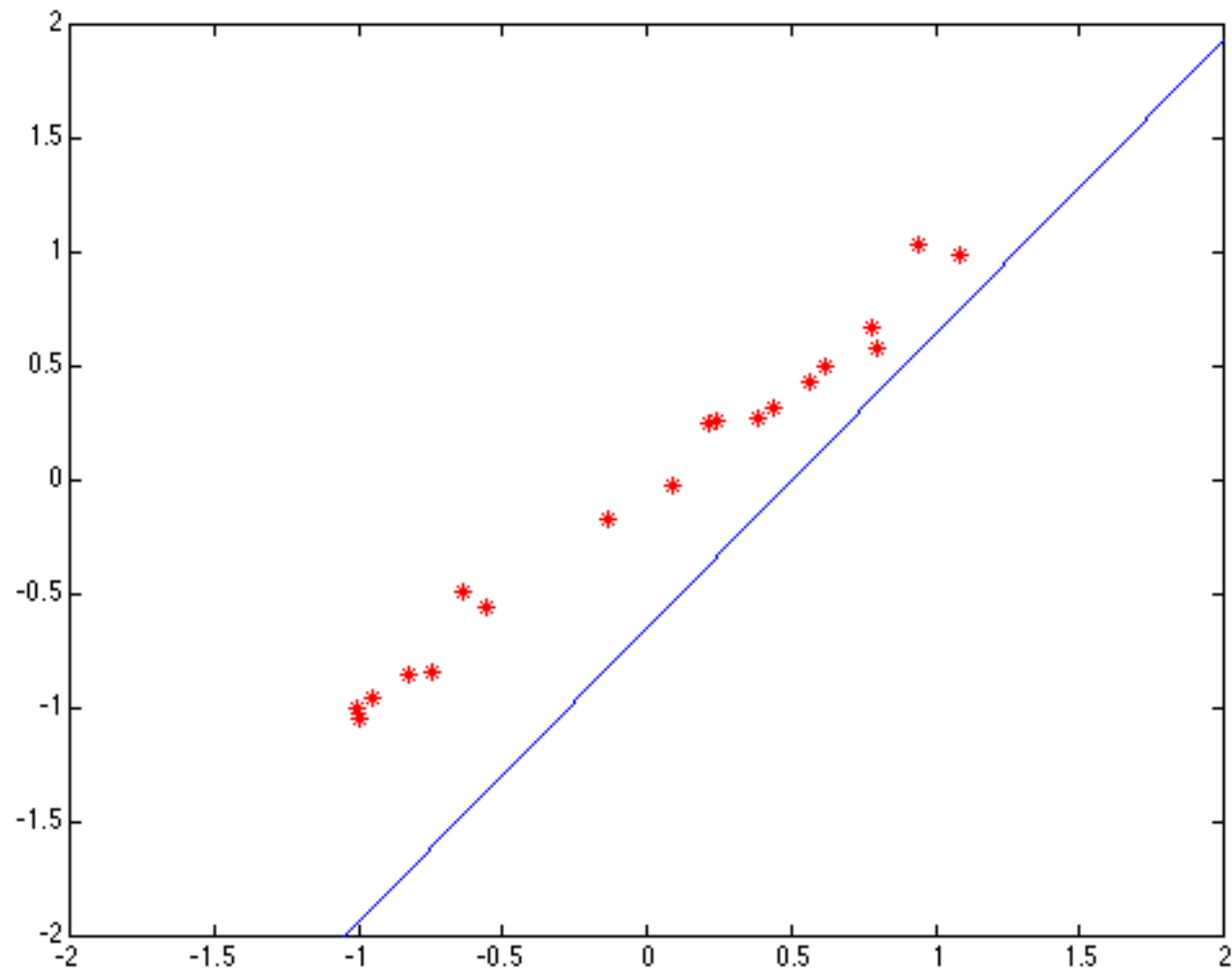
Least squares fit (destroyed by outlier)



Least squares fit (warped by outlier)



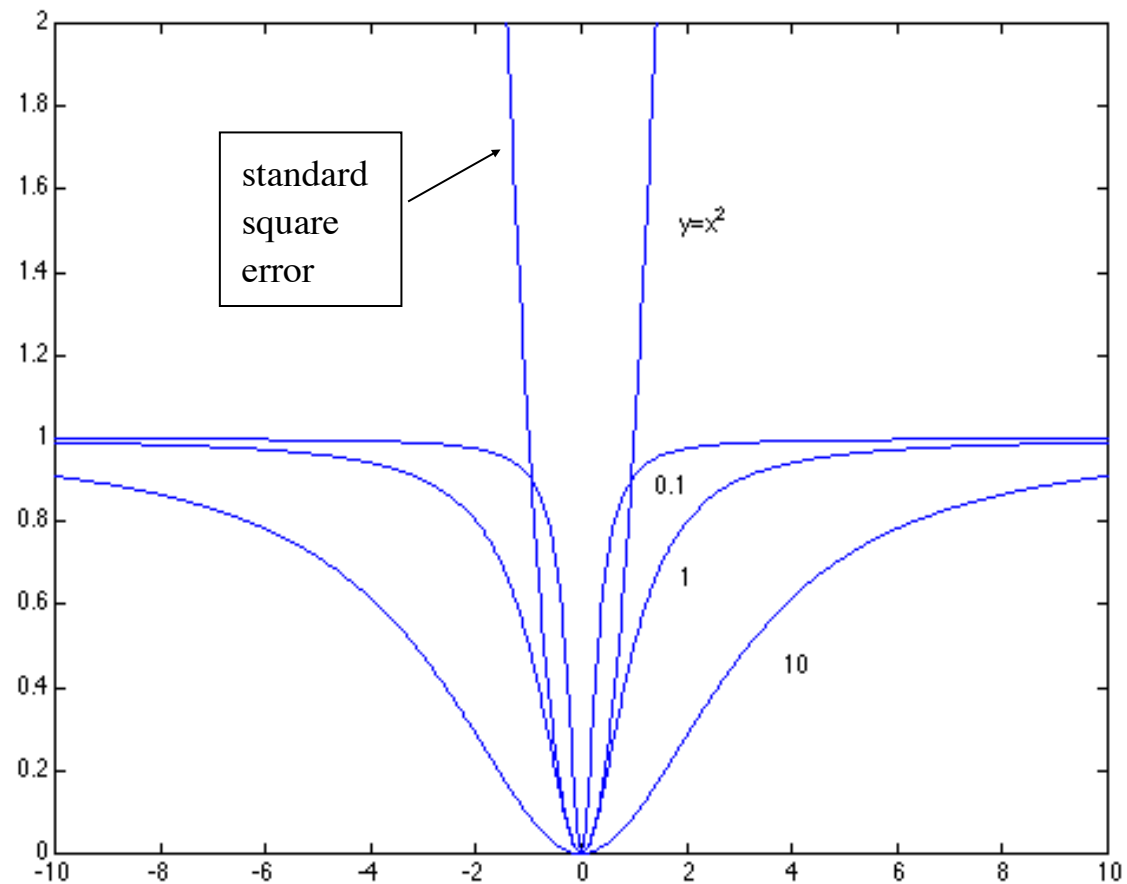
Least squares fit (previous slide zoom in)



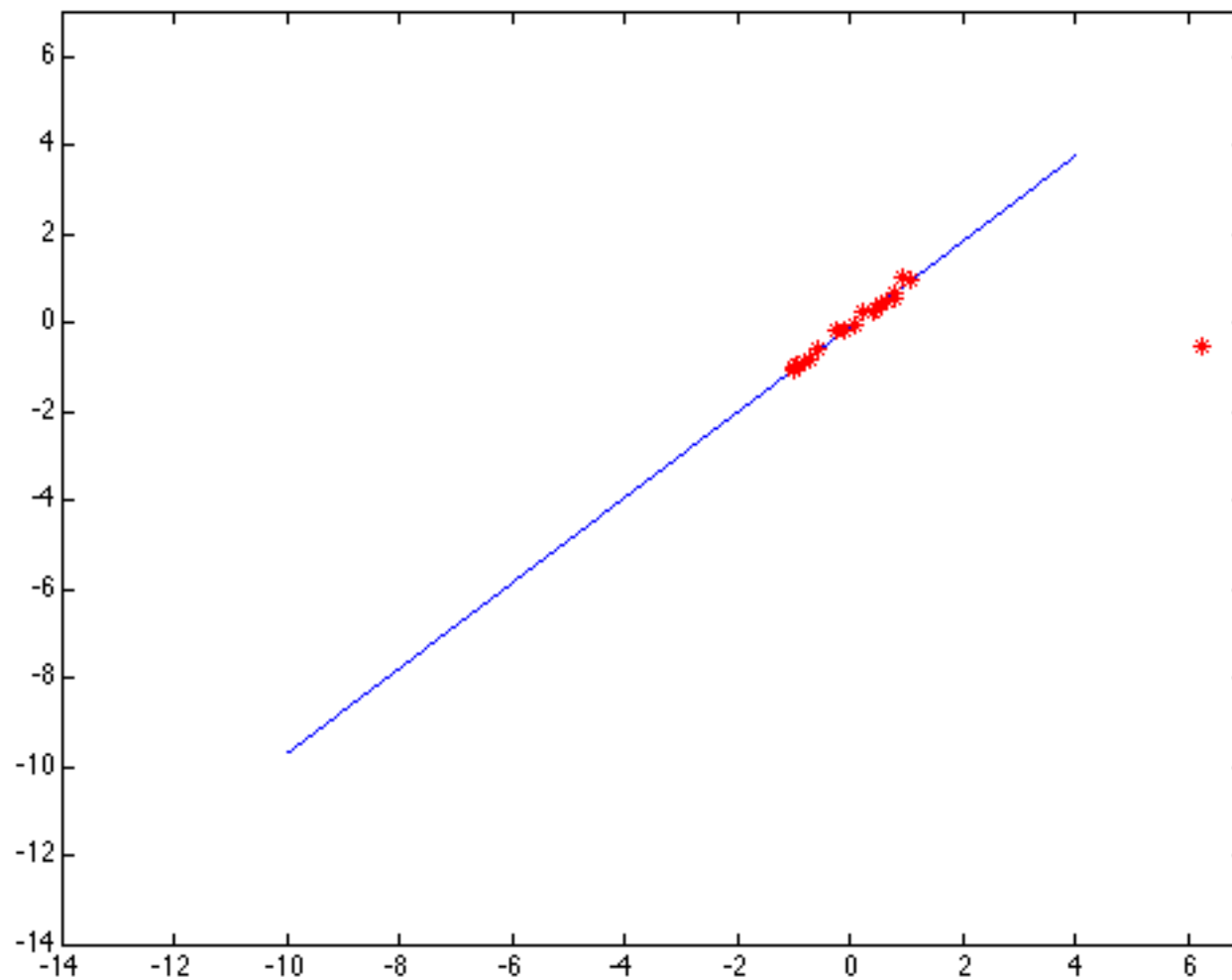
Example of a robust estimator. The effect of outliers are mitigated.
After a certain distance, errors count the same.

$$y = x^2 / (x^2 + s^2)$$

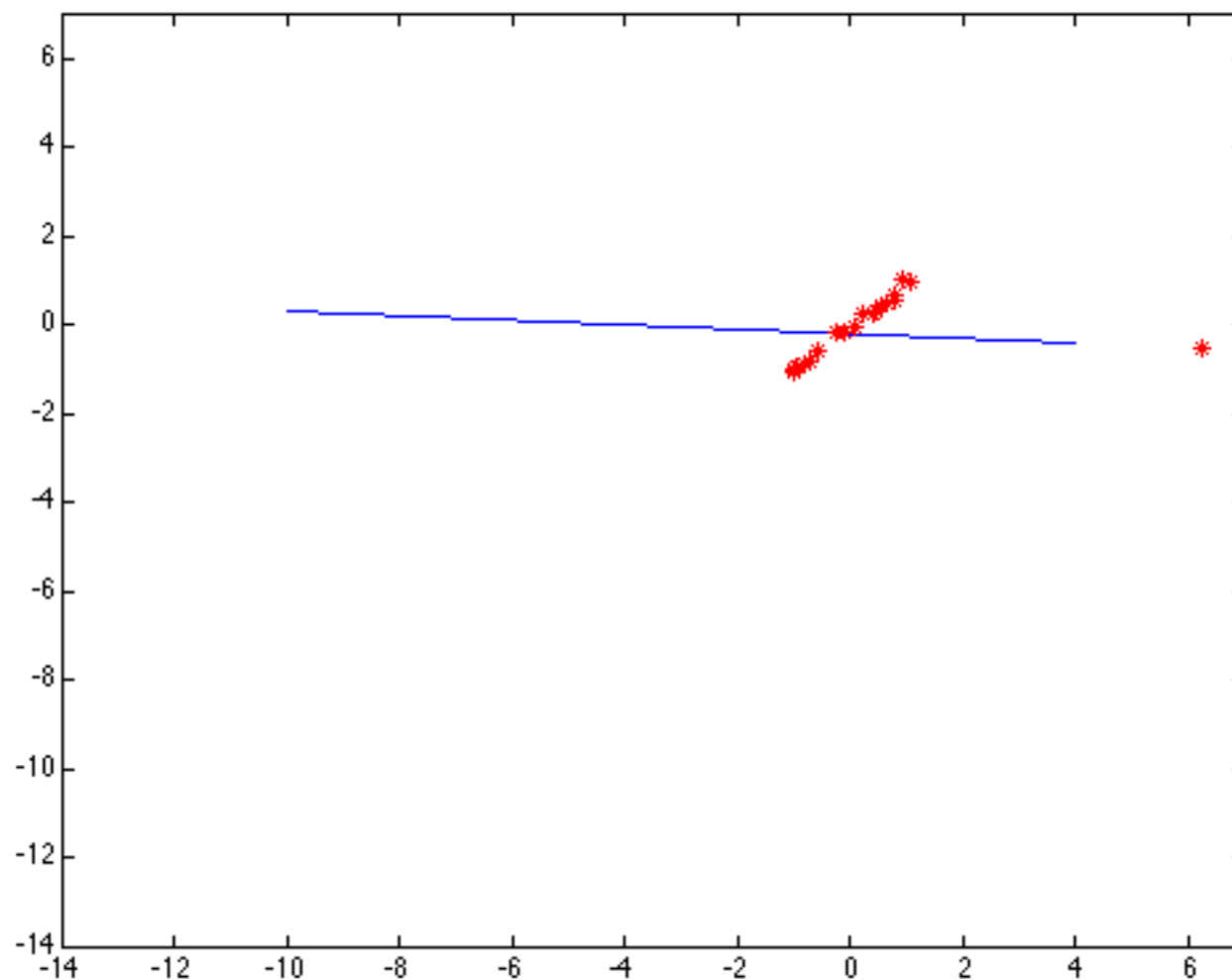
(Curve for three different
values of s shown)



Line fit with estimator with good choice for s

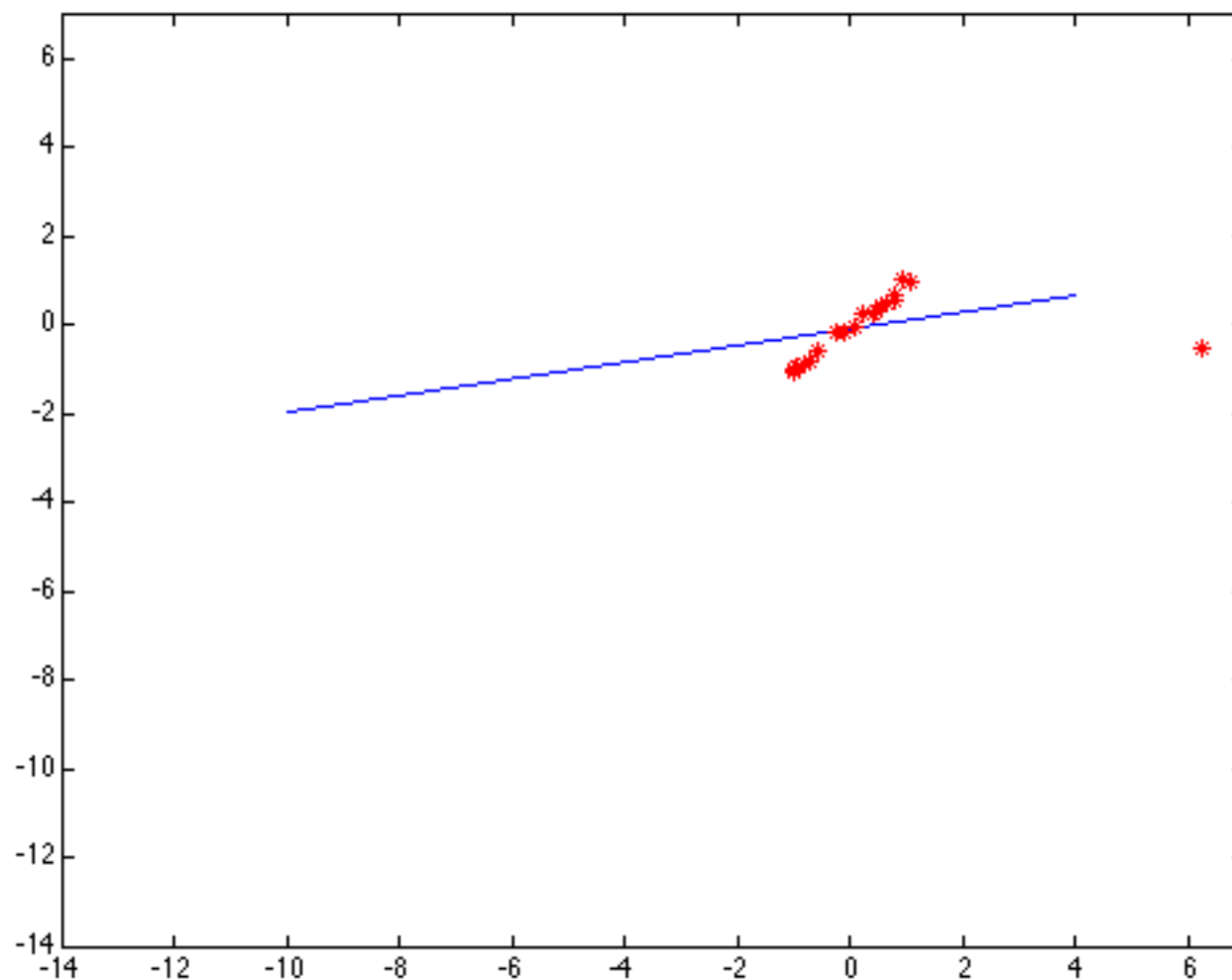


Line fit with estimator with choice for s that is too small



If s is too small, then the data is ignored too much

Line fit with estimator with choice for s that is too big



If s is too big, then we are back towards least squares

RANSAC

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

RANSAC

- How big a subset?
 - Smallest possible for the particular model (for a line, use 2 points)
- What does close mean?
 - Depends on the problem
 - Two strategies
 - Points within some fit threshold
 - Best $k\%$ points
- What is a good line?
 - One where the number of nearby points is so big it is unlikely to be all outliers (another threshold decision).

RANSAC

- How many iterations?
 - Often enough that we are likely to have a good model
 - Goes up with model complexity and belief about percentage of outliers
- Following notation from: <http://en.wikipedia.org/wiki/RANSAC>
 - Let w be the probability of getting an inlier.
 - Assume n points are needed for the model.
 - Suppose you want to be sure of a valid fit with probability, p .
 - Then the number of iterations, k , needed is:

$$k = \frac{\log(1 - p)}{\log(1 - w^n)}$$

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

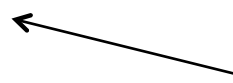
Test the distance from the point to the line
against t ; if the distance from the point to the line
is less than t , the point is close

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



Note use of
threshold