Fitting

Computer Vision
CS 543 / ECE 549
University of Illinois
Fitting lines to point sets

In general, fit a function from a given function class to samples from the function
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 (not covered)
Simple example: Fitting a line
Least squares line fitting

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

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

\[
E = \sum_{i=1}^{n} \left( \begin{bmatrix} x_i & 1 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} - y_i \right)^2 = \left\| \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} - \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \right\|^2 = \|Ap - y\|^2
\]

\[
\]

\[
\frac{dE}{dp} = 2A^T Ap - 2A^T y = 0
\]

\[
A^T Ap = A^T y \Rightarrow p = (A^T A)^{-1} A^T y
\]

Matlab: \( p = A \backslash y \);

Modified from S. Lazebnik
Problem with “vertical” least squares

• Not rotation-invariant
• Fails completely for vertical lines
Total least squares

If \((a^2 + b^2 = 1)\) then

Distance between point \((x_i, y_i)\) and line \(ax + by + c = 0\) is \(|ax_i + by_i + c|\)

proof:  
http://mathworld.wolfram.com/Point-LineDistance2-Dimensional.html
Total least squares

If \((a^2+b^2=1)\) then
Distance between point \((x_i, y_i)\) and line \(ax+by+c=0\) is \(|ax_i + by_i + c|\)

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

\[
E = \sum_{i=1}^{n} (ax_i + by_i + c)^2
\]
Total least squares

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

\[
E = \sum_{i=1}^{n} (ax_i + by_i + c)^2
\]

\[
\frac{\partial E}{\partial c} = \sum_{i=1}^{n} 2(ax_i + by_i + c) = 0
\]

\[
E = \sum_{i=1}^{n} (a(x_i - \bar{x}) + b(y_i - \bar{y}))^2 = \frac{1}{n} \left( A^T A \right) p = p^T A^T A p
\]

\[
\text{minimize } p^T A^T A p \quad \text{s.t. } p^T p = 1 \quad \Rightarrow \quad \text{minimize} \frac{p^T A^T A p}{p^T p}
\]

Solution is eigenvector corresponding to smallest eigenvalue of \(A^T A\)

# Recap: Two Common Optimization Problems

<table>
<thead>
<tr>
<th>Problem statement</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{minimize} \quad |Ax - b|^2 )</td>
<td>( x = (A^T A)^{-1} A^T b )</td>
</tr>
<tr>
<td>least squares solution to ( Ax = b )</td>
<td>( x = A \backslash b ) (matlab)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Problem statement</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>minimize ( x^T A^T A x ) s.t. ( x^T x = 1 )</td>
<td>([v, \lambda] = \text{eig}(A^T A))</td>
</tr>
<tr>
<td>( \text{minimize} \frac{x^T A^T A x}{x^T x} )</td>
<td>( \lambda_1 &lt; \lambda_{2..n} : x = v_1 )</td>
</tr>
<tr>
<td>non-trivial lsq solution to ( Ax = 0 )</td>
<td></td>
</tr>
</tbody>
</table>
Recap: Fitting Lines

- a. fit $y = mx + b$
- b. fit $ax + by + c = 0$

Solution involves:
- 1. Eigen vector
- 2. Pseudo-inverse

- A. $a \rightarrow 1$, $b \rightarrow 2$
- B. $a \rightarrow 2$, $b \rightarrow 1$
- C. $a \rightarrow 1$, $b \rightarrow 1$
- D. $a \rightarrow 2$, $b \rightarrow 2$
Least squares (global) optimization

Good
• Clearly specified objective
• Optimization is easy

Bad
• May not be what you want to optimize
• Sensitive to outliers
  – Bad matches, extra points
• Doesn’t allow you to get multiple good fits
  – Detecting multiple objects, lines, etc.
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 least squares (to deal with outliers)

General approach:

\[
\text{minimize } \sum_{i} \rho(u_i(x_i, \theta); \sigma) \quad u^2 = \sum_{i=1}^{n} (y_i - mx_i - b)^2
\]

\(u_i(x_i, \theta)\) – residual of \(i^{th}\) point w.r.t. model parameters \(\theta\)
\(\rho\) – robust function with scale parameter \(\sigma\)

The robust function \(\rho\)

- Favors a configuration with small residuals
- Constant penalty for large residuals

The diagram illustrates the behavior of \(\rho\) for different values of \(\sigma\). For small residuals, \(\rho\) provides a constant penalty, while for large residuals, it decreases to zero.
Robust Estimator

1. Initialize: e.g., choose $\theta$ by least squares fit and $\sigma = 1.5 \cdot \text{median}(\text{error})$

2. Choose params to minimize: $\sum_i \frac{\text{error}(\theta, \text{data}_i)^2}{\sigma^2 + \text{error}(\theta, \text{data}_i)^2}$
   - E.g., numerical optimization

3. Compute new $\sigma = 1.5 \cdot \text{median}(\text{error})$

4. Repeat (2) and (3) until convergence
Choosing the scale: Just right

The effect of the outlier is minimized

Slide from L. Lazebnik
Choosing the scale: Too small

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

Slide from L. Lazebnik
Choosing the scale: Too large

Behaves much the same as least squares

Slide from L. Lazebnik
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 (not covered)
Hough Transform: Outline

1. Create a grid of parameter values

2. Each point votes for a set of parameters, incrementing those values in grid

3. Find maximum or local maxima in grid
Hough transform


Given a set of points, find the curve or line that explains the data points best

\[ y = mx + b \]

Slide from S. Savarese
Hough transform

Slide from S. Savarese
Hough transform


Issue: parameter space \([m,b]\) is unbounded...

Use a polar representation for the parameter space

\[
x \cos \theta + y \sin \theta = \rho
\]
Hough transform - experiments
Hough transform - experiments

Noisy data

Need to adjust grid size or smooth
Hough transform - experiments

Issue: spurious peaks due to uniform noise
1. Image $\rightarrow$ Canny
2. Canny $\rightarrow$ Hough votes

Slide from D. Hoiem
3. Hough votes $\rightarrow$ Edges

Find peaks and post-process
Hough transform example
Finding circles \((x_0, y_0, r)\) using Hough transform

- Fixed \(r\)
- Variable \(r\)
Hough transform for circles

image space

Hough parameter space

\[(x, y) + r \nabla I(x, y)\]

\[(x, y) - r \nabla I(x, y)\]

Slide from L. Lazebnik
Incorporating image gradients

- When we detect an edge point, we also know its gradient orientation.
- How does this constrain possible lines passing through the point?

- Modified Hough transform:
  - For each edge point \((x,y)\)
    \[
    \theta = \text{gradient orientation at } (x,y) \\
    \rho = x \cos \theta + y \sin \theta \\
    H(\theta, \rho) = H(\theta, \rho) + 1
    \]
    end

\[
\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \end{bmatrix}
\]

\[
\theta = \tan^{-1} \left( \frac{\partial f}{\partial y} / \frac{\partial f}{\partial x} \right)
\]
Hough transform conclusions

Good
- Robust to outliers: each point votes separately
- Fairly efficient (much faster than trying all sets of parameters)
- Provides multiple good fits

Bad
- Some sensitivity to noise
- Bin size trades off between noise tolerance, precision, and speed/memory
  - Can be hard to find sweet spot
- Not suitable for more than a few parameters
  - grid size grows exponentially

Common applications
- Line fitting (also circles, ellipses, etc.)
- Object instance recognition (parameters are position/scale/orientation)
- Object category recognition (parameters are position/scale)
RANSAC

(RANdom SAmple Consensus):

Fischler & Bolles in ‘81.

Algorithm:

1. **Sample** (randomly) the number of points required to fit the model
2. **Solve** for model parameters using samples
3. **Score** by the fraction of inliers within a preset threshold of the model

Repeat 1-3 until the best model is found with high confidence
Algorithm:

1. **Sample** (randomly) the number of points required to fit the model (#=2)
2. **Solve** for model parameters using samples
3. **Score** by the fraction of inliers within a preset threshold of the model

**Repeat** 1-3 until the best model is found with high confidence
RANSAC

Line fitting example

Algorithm:

1. **Sample** (randomly) the number of points required to fit the model (#=2)
2. **Solve** for model parameters using samples
3. **Score** by the fraction of inliers within a preset threshold of the model

Repeat 1-3 until the best model is found with high confidence
RANSAC

Line fitting example

$N_I = 6$

Algorithm:
1. **Sample** (randomly) the number of points required to fit the model (#=2)
2. **Solve** for model parameters using samples
3. **Score** by the fraction of inliers within a preset threshold of the model

Repeat 1-3 until the best model is found with high confidence
RANSAC

Algorithm:

1. **Sample** (randomly) the number of points required to fit the model (#=2)
2. **Solve** for model parameters using samples
3. **Score** by the fraction of inliers within a preset threshold of the model

Repeat 1-3 until the best model is found with high confidence
How to choose parameters?

- **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: $e$)

- **Number of sampled points** $s$
  - Minimum number needed to fit the model

- **Distance threshold** $\delta$
  - Choose $\delta$ so that a good point with noise is likely (e.g., prob=0.95) within threshold
  - Zero-mean Gaussian noise with std. dev. $\sigma$: $t^2=3.84\sigma^2$

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

<table>
<thead>
<tr>
<th>$s$</th>
<th>5%</th>
<th>10%</th>
<th>20%</th>
<th>25%</th>
<th>30%</th>
<th>40%</th>
<th>50%</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>11</td>
<td>17</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td>9</td>
<td>11</td>
<td>19</td>
<td>35</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>5</td>
<td>9</td>
<td>13</td>
<td>17</td>
<td>34</td>
<td>72</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>6</td>
<td>12</td>
<td>17</td>
<td>26</td>
<td>57</td>
<td>146</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>7</td>
<td>16</td>
<td>24</td>
<td>37</td>
<td>97</td>
<td>293</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>8</td>
<td>20</td>
<td>33</td>
<td>54</td>
<td>163</td>
<td>588</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>9</td>
<td>26</td>
<td>44</td>
<td>78</td>
<td>272</td>
<td>1177</td>
</tr>
</tbody>
</table>
RANSAC conclusions

Good
• Robust to outliers
• Applicable for larger number of objective function parameters than Hough transform
• Optimization parameters are easier to choose than Hough transform

Bad
• Computational time grows quickly with fraction of outliers and number of parameters
• Not as good for getting multiple fits (though one solution is to remove inliers after each fit and repeat)

Common applications
• Computing a homography (e.g., image stitching)
• Estimating fundamental matrix (relating two views)
Fitting Summary

• Least Squares Fit
  – closed form solution
  – robust to noise
  – not robust to outliers

• Robust Least Squares
  – improves robustness to noise
  – requires iterative optimization

• Hough transform
  – robust to noise and outliers
  – can fit multiple models
  – only works for a few parameters (1-4 typically)

• RANSAC
  – robust to noise and outliers
  – works with a moderate number of parameters (e.g., 1-8)