CHAPTER 14

Fitting Lines

An alternate master recipe to choose the "best" parametric model to explain or represent some data is to set up an optimization problem describing the "best" model, then search for a solution. In this chapter, I will concentrate on various applications of this approach to choosing a line to represent some tokens. These tokens could be edge points, or they could be the location of some interest points, or they could be other things entirely, but they are represented by their location.

Another very important case, dealt with in detail in Chapter **??**, might look different to you but isn't really. You are given a set of points $\mathbf{x}_i$ in one image and a set of points $\mathbf{y}_i$ in another, and must find a transformation $\mathcal{T}$ so that $\mathcal{T}(\mathbf{x}_i)$ is close to $\mathbf{y}_i$. Each of these problems is usually referred to as *fitting*.

14.1 FITTING JUST ONE LINE

Fitting just one line to a set of tokens is a good model problem. Assume that all the points that belong to a particular line are known, and the parameters of the line must be found.

14.1.1 Least Squares Fitting

There is a simple strategy for fitting lines, known as *least squares*. *Do not use this* - I am showing it only because it is traditional, and because the form of the derivation is repeated for other problems. This procedure has a long tradition and a substantial bias. You have likely seen this idea, but may not know why you should not use it. Represent a line as $y = ax + b$. There are N tokens, and the i'th token is at $\mathbf{x}_i = (x_{1,i}, x_{2,i})$. Now choose the line that best predicts the measured y coordinate for each measured x coordinate, and so minimizes

$$(1/2)\sum_i (x_{2,i} - a x_{1,i} - b)^2.$$

The term (1/2) very often appears in quadratic problems like this to absorb the 2 that appears when you differentiate. A certain looseness in notation is traditional (you might not see either), because the 2 appears in front of an expression that is equal to zero. Adopting the notation

$$\overline{u} = \frac{\sum u_i}{N}$$

the line is given by the solution to the problem

$$\begin{pmatrix} \overline{x_2^2} \\ \overline{x_2} \end{pmatrix} = \begin{pmatrix} \overline{x_1^2} & \overline{x_1} \\ \overline{x_1} & 1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix}.$$

If the errors are weighted with weight w_i corresponding to the i'th term, the cost becomes

$$(1/2)\sum_i w_i(x_{2,i} - ax_{1,i} - b)^2.$$

and, if you interpret

$$\overline{u} = \frac{\sum w_i u_i}{\sum w_i}$$

the expression above still works.

Although this is a standard linear solution to a classical problem, it's not much help in vision applications, because the model is an extremely poor one. The difficulty is that the measurement error is dependent on coordinate frame. Vertical offsets from the line count as errors, which means that near vertical lines lead to quite large values of the error and quite funny fits (Figure 14.1). In fact, the process is so dependent on coordinate frame that it doesn't represent truly vertical lines at all.

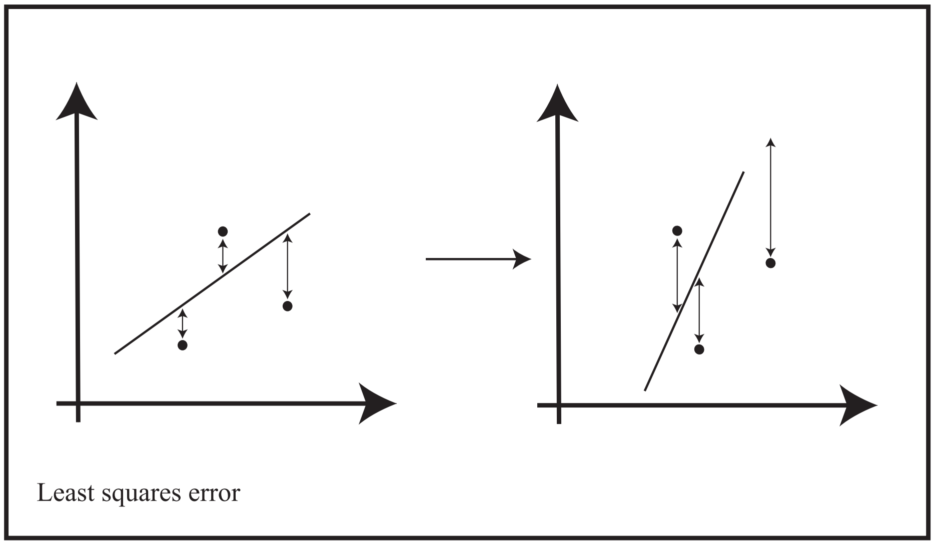

FIGURE 14.1: *Least squares finds the line that minimizes the sum of squared vertical distances between the line and the tokens (because it assumes that the error appears only in the y coordinate). This yields a (very slightly) simpler mathematical problem at the cost of a poor fit. Importantly, if the coordinate system rotates (say, the camera is rotated), the error between the rotated points and the rotated lines changes.*

14.1.2 Total Least Squares Fitting

Working with the actual distance between the token and the line (rather than the vertical distance) leads to a problem known as *total least squares*. Write $\mathbf{a} =$

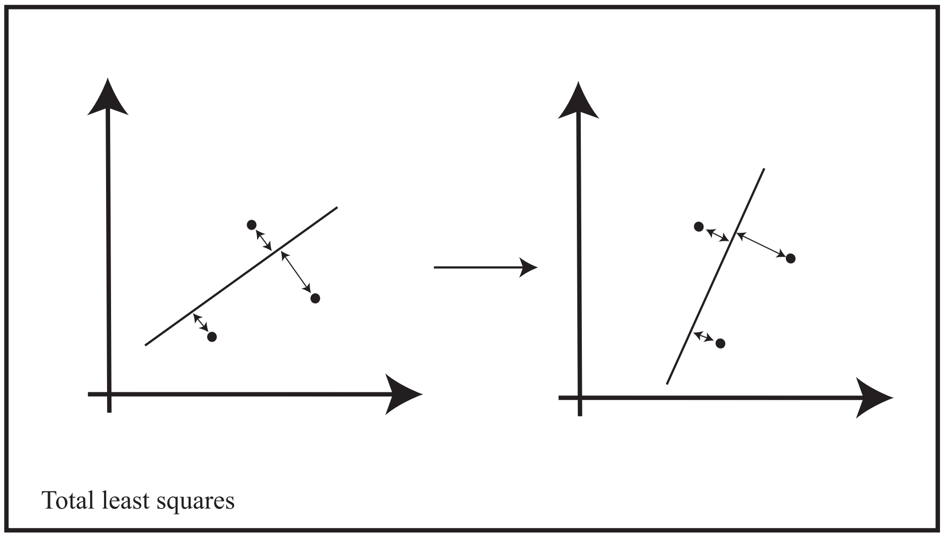

FIGURE 14.2: *Total least-squares finds the line that minimizes the sum of squared perpendicular distances between tokens and the line so near-vertical lines can be fitted without difficulty. If the coordinate system rotates (say, the camera is rotated), the error between the rotated points and the rotated lines does not change.*

$(a_1, a_2)^T$ and represent a line as the collection of points where $\mathbf{a}^T\mathbf{x}+c=0$. Require that $\mathbf{a}^T\mathbf{a}+c^2 \neq 0$, otherwise the equation is true for all points on the plane. Every line can be represented in this way.

Recall that for $s \neq 0$, the line given by $s(\mathbf{a}, c)$ is the same as the line represented by $(\mathbf{a}, c)$. Recall that the perpendicular distance from a point $\mathbf{x}$ to a line $(\mathbf{a}, c)$ is given by

$$\text{abs}(\mathbf{a}^T\mathbf{x}+c) \qquad \text{if} \qquad \mathbf{a}^T\mathbf{a}=1.$$

It is convenient to assume that each point has an associated weight w_i. In the case where you have no idea what the weights are, use $w_i = 1/N$. Notice that choosing $w_i = 1/N$ ensures that the fitting cost you compute does not depend on the number of points **exercises** . Recall

$$\overline{u} = \frac{\sum_i w_i u_i}{\sum_i w_i}$$

Minimizing the sum of perpendicular distances between points and lines requires minimizing

$$(1/2)\sum_i w_i(\mathbf{a}^T\mathbf{x}_i + c)^2,$$

subject to $\mathbf{a}^T\mathbf{a}=1$. Introduce a Lagrange multiplier λ; at a solution

$$\begin{pmatrix} \overline{x_1^2} & \overline{x_1x_2} & \overline{x_1} \\ \overline{x_1x_2} & \overline{x_2^2} & \overline{x_2} \\ \overline{x_1} & \overline{x_2} & 1 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ c \end{pmatrix} = \lambda \begin{pmatrix} 2a_1 \\ 2a_2 \\ 0 \end{pmatrix}.$$

This means that

$$c = -a_1\overline{x_1} - a_2\overline{x_2}.$$

Substitute this back to get the eigenvalue problem

$$\left(\begin{array}{cc} \overline{x_1^2} - \overline{x_1}\;\overline{x_1} & \overline{x_1x_2} - \overline{x_1}\;\overline{x_2} \\ \overline{x_1x_2} - \overline{x_1}\;\overline{x_2} & \overline{x_2^2} - \overline{x_2}\;\overline{x_2} \end{array}\right)\left(\begin{array}{c} a_1 \\ a_2 \end{array}\right) = \mu\left(\begin{array}{c} a_1 \\ a_2 \end{array}\right).$$

Because this is a 2D eigenvalue problem, two solutions up to scale can be obtained in closed form (for those who care, it's usually done numerically!). The scale is obtained from the constraint that $\mathbf{a}^T\mathbf{a} = 1$. The two solutions to this problem are lines at right angles; one maximizes the sum of squared distances and the other minimizes it (which is which is explored in exercises **exercises**).

Procedure: 14.1 *Fitting a Line using Weighted Total Least Squares*

Given N points $\mathbf{x}_i = (x_{i,1}, x_{i,2})$, and weights w_i associated with each point, find a line that minimizes the weighted total least squares error by forming

$$\begin{array}{rcl} \overline{x_1} & = & \frac{\sum_i w_i x_{1,i}}{\sum_i w_i} \\ \overline{x_2} & = & \frac{\sum_i w_i x_{1,i}}{\sum_i w_i} \\ \overline{x_1^2} & = & \frac{\sum_i w_i x_{1,i}^2}{\sum_i w_i} \\ \overline{x_1x_2} & = & \frac{\sum_i w_i x_{1,i}x_{i,2}}{\sum_i w_i} \\ \overline{x_2^2} & = & \frac{\sum_i w_i x_{2,i}^2}{\sum_i w_i} \end{array}$$

and

$$\mathcal{M} = \left(\begin{array}{cc} \overline{x_1^2} - \overline{x_1}\;\overline{x_1} & \overline{x_1x_2} - \overline{x_1}\;\overline{x_2} \\ \overline{x_1x_2} - \overline{x_1}\;\overline{x_2} & \overline{x_2^2} - \overline{x_2}\;\overline{x_2} \end{array}\right).$$

Write $\mathbf{e} = (e_1, e_2)^T$ for the unit eigenvector of $\mathcal{M}$ corresponding to the smallest eigenvalue. The line is

$$e_1x + e_2y - e_1\overline{x_1} - e_2\overline{x_2} = 0$$

14.1.3 Application: Surface Normals in Point Clouds

A *point cloud* is a set of points with no other structure. The idea is developed in greater detail in Chapter **??**. Usually, point clouds are obtained by sampling surfaces in some way – the points lie on objects in scenes, and are close enough that most points are on the same object as their closest neighbors. This means that you could associate a normal with each point fairly accurately by: finding a few (at least two) nearest neighbors; fitting a plane to the point and its neighbors; and using the normal of that plane.

You can find the nearest neighbors either by blank search (check the distance to every other point, and take the points with the smallest; which is slow) or by using the approximate nearest neighbors method of Section 10.3.1. Fitting a plane to points in 3D is very similar to fitting a line to points in 2D. You should use total least squares, just as in line fitting. Represent the plane as $\mathbf{a}^T\mathbf{x} + d = 0$; then the squared distance from a point $\mathbf{x}_i = (x_{1,i}, x_{2,i}, x_{i,3})$ to the plane will be $(\mathbf{a}^T\mathbf{x}+d)^2$ if $\mathbf{a}^T\mathbf{a} = 1$. **exercises** Now use the analysis above with small changes (the closed form expression for the three solutions is more complicated). The normal will then be **a** normalized to a unit vector.

Procedure: 14.2 *Fitting a Plane using Weighted Total Least Squares*

Given N points $\mathbf{x}_i = (x_{1,i}, x_{2,i}, x_{3,i})$, and weights w_i associated with each point, find a plane that minimizes the weighted total least squares error by forming

$$\begin{aligned}
\overline{x_1} &= \frac{\sum_i w_i x_{1,i}}{\sum_i w_i} \\
\overline{x_2} &= \frac{\sum_i w_i x_{1,i}}{\sum_i w_i} \\
\overline{x_3} &= \frac{\sum_i w_i x_{1,i}}{\sum_i w_i} \\
\overline{x_1^2} &= \frac{\sum_i w_i x_{1,i}^2}{\sum_i w_i} \\
\overline{x_1 x_2} &= \frac{\sum_i w_i x_{1,i} x_{i,2}}{\sum_i w_i} \\
\overline{x_2^2} &= \frac{\sum_i w_i x_{2,i}^2}{\sum_i w_i} \\
\overline{x_2 x_3} &= \frac{\sum_i w_i x_{2,i} x_{i,3}}{\sum_i w_i} \\
\overline{x_3^2} &= \frac{\sum_i w_i x_{i,3}^2}{\sum_i w_i}
\end{aligned}$$

and

$$\mathcal{M} = \begin{pmatrix} \overline{x_1^2} - \overline{x_1}\ \overline{x_1} & \overline{x_1 x_2} - \overline{x_1}\ \overline{x_2} & \overline{x_1 x_3} - \overline{x_1}\ \overline{x_3} \\ \overline{x_2 x_1} - \overline{x_2}\ \overline{x_1} & \overline{x_2^2} - \overline{x_2}\ \overline{x_2} & \overline{x_2 x_3} - \overline{x_2}\ \overline{x_3} \\ \overline{x_3 x_1} - \overline{x_3}\ \overline{x_1} & \overline{x_3 x_2} - \overline{x_3}\ \overline{x_2} & \overline{x_3^2} - \overline{x_3}\ \overline{x_3} \end{pmatrix}.$$

Write $\mathbf{e} = (e_1, e_2, e_3)^T$ for the unit eigenvector of $\mathcal{M}$ corresponding to the smallest eigenvalue. The plane is

$$e_1 x + e_2 y + e_3 z - e_1 \overline{x_1} - e_2 \overline{x_2} - e_3 \overline{x_3} = 0$$

Remember this: *Fit lines in 2D and planes in 3D with total least squares if there are no outliers. NEVER EVER use least squares without a good reason that you can articulate.*

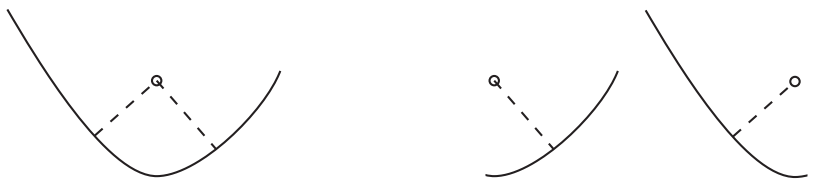

FIGURE 14.3: *There can be more than one point on a curve that looks as if it is closest to a token. This makes fitting curves to points very difficult. On the* **left**, *a curve and a token; dashed lines connect the token to the two points on the curve that could be closest. The dashed line from token to curve and the tangent to the curve are at right angles for each of the two points on the curve.* **Center** *and* **right** *show copies of part of the curve; for each, the closest point on the segment to the token is different, because part of the curve is missing. As a result, no local test can guarantee a point is closest – all candidates must be checked.*

14.1.4 Fitting Curved Shapes

Curves in 2D are different from lines in 2D. For every token on the plane, there is a unique, single point on a line that is closest to it. This is not true for a curve. Because curves curve, there might be more than one point on the curve that looks locally as though it is closest to the token (Figure 14.3). This means it can be very hard to find the smallest distance between a point and a curve. Similar effects occur for surfaces in 3D. If one ignores this difficulty, fitting curves is similar to fitting lines: minimize the sum of squared distances between the points and the curve as a function of the choice of curve. The exercises explore how to find the closest point on a curve (**exercises**), but generally these fitting problems are very badly behaved.

> **Remember this:** *Fitting curved shapes can be difficult because there may be many candidates for the closest point to the curved shape and because the distance can be hard to evaluate exactly. There are a number of approximate distances, but none is wholly satisfactory. Expect these fitting problems to be hard.*

14.2 FITTING ONE LINE IN THE PRESENCE OF OUTLIERS

Total least squares chooses a line by minimizing the sum of squared error terms. This is fine when no point is far from the line (Figure 14.4), but a single very bad data point can give errors that dominate those due to many good data points (Figure 14.5). This is because the square of a large number is very much larger than the square of a small number.

Outliers – data points that are far from the model, in this case, the line – are

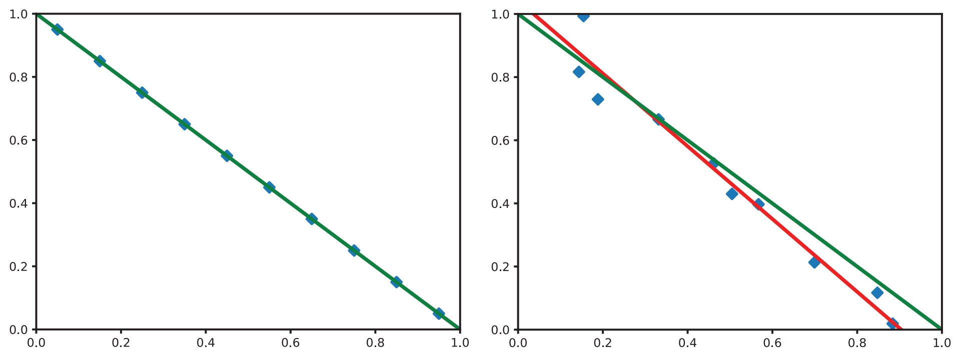

FIGURE 14.4: *Total least squares behaves well when all points lie fairly close to the line.* **Left***: A line fitted with total least squares to a set of points that actually lie on the line.* **Right***: The points have been perturbed by adding a small random term to both x and y coordinate, but the line fitted to the noisy points (***red***) is not that different from the original (***green***).*

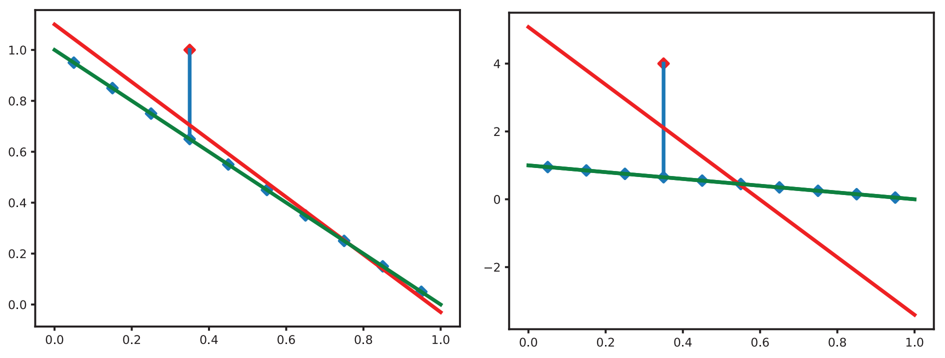

FIGURE 14.5: *Total least squares can misbehave when some points are significantly perturbed. Here the points are those of Figure 14.4, but one point has had its y coordinate replaced with a constant (new point in* **red***, joined to its original position with a line).* **Left:** *The constant is 1, and the line produced by total least squares has shifted significantly.* **Right:** *The constant is 4, and now the fitted line is completely wrong. In each case, the original line is in* **green** *(and passes close to the points), and the new line is* **red***.*

common. One way to encounter significant outliers is if you are fitting a line to a set of tokens that together form multiple lines – for example, a set of tokens on the edges of a square (Figure 14.9) – a line that fits some points (say, those on one side of the square) will have tokens far away from it (those on the other sides). Another source of outliers is errors in collecting or transcribing data points. Practical vision problems usually involve outliers.

Outliers can be handled by reducing the influence of distant points on the estimate. This is where the weights of the previous section come in useful. A natural strategy fits a line using a set of weights, then recomputes the weights using a large weight for points that are "trustworthy" and a small weight for points that are "suspicious". This is the basis of IRLS (iteratively reweighted least squares), a procedure that uses a robust cost function to organize the reweighting of points. Alternatively you could identify bad points and ignore them. For example, a small set of good points will identify the thing we wish to fit; other good points will agree, and the points that disagree are then bad points. This is the basis of RANSAC, an extremely important algorithm described in Section 14.3.

14.2.1 Robust Cost Functions

The weights are not known, but can be recovered using a *robust loss*, which reduces the cost of large errors. Typically, these cost functions "look like" squared error for small values, but grow more slowly for distant points.

Write θ for the parameters of the line and $r(\mathbf{x}_i, \theta)$ for a *residual*, which measures consistency between the point and the line. In this case, $r(\mathbf{x}_i, \theta)$ will always be $(\mathbf{a}^T\mathbf{x} + c)$, which is signed, so positive on one side of the line and negative on the other. Rather than minimizing the total least squares criterion, which is

$$(1/2)\sum_i (r(\mathbf{x}_i, \theta))^2$$

as a function of θ, choose the line by minimizing an expression of the form

$$\sum_i \rho(r(\mathbf{x}_i, \theta); \sigma),$$

for some appropriately chosen function ρ with a parameter σ. The total least squares error fits this recipe, using $\rho(u;\sigma) = u^2/2$. The trick is to make $\rho(u;\sigma)$ look like $u^2/2$ for smaller values of u, but ensure that it grows more slowly than $u^2/2$ for larger values of u.

The *Huber loss* uses

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

which is the same as $u^2/2$ for $-\sigma \leq u \leq \sigma$, switches to $|u|$ for larger (or smaller) σ, and has continuous derivative at the switch. The Huber loss is convex (meaning that there will be a unique minimum for our models) and differentiable, but is not smooth. The choice of the parameter σ (which is known as *scale*) has an effect on the estimate. You should interpret this parameter as the distance that a point can lie from the fitted function while still being seen as an *inlier* (anything that isn't even partially an outlier).

The *Pseudo Huber loss* uses

$$\rho(u;\sigma) = \sigma^2\left(\sqrt{1 + \left(\frac{u}{\sigma}\right)^2} - 1\right).$$

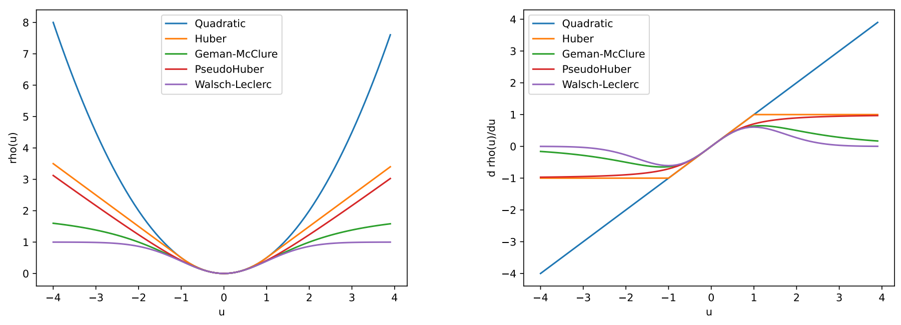

FIGURE 14.6: *Robust losses reduce the effect of large errors.* **Left***: the robust loss functions of the text, compared to a squared error loss. In each case, the scale variable σ is 1. Note how for each of the robust losses, the cost of large errors is reduced compared to quadratic losses.* **Right:** *the derivative for each of these loss functions; this derivative appears in the weight term in IRLS, and is sometimes referred to as the* influence function *of the loss. For losses like Huber and pseudo-Huber loss, large errors still have an influence (the derivative heads toward a non-zero constant); but for other losses, large errors can cause a data point to be completely discounted (the derivative is zero, so the weight in IRLS will be zero).*

A little fiddling with Taylor series reveals this is approximately $(u/\sigma)^2$ for $|u|/\sigma$ small, and linear for $|u|/\sigma$ big. This has the advantage of being differentiable.

The *Geman-McClure loss* uses

$$\rho(u;\sigma) = \frac{2\left(\frac{u}{\sigma}\right)^2}{\left(\frac{u}{\sigma}\right)^2 + 4} = \frac{2u^2}{u^2 + 4\sigma^2}$$

which is approximately $(1/2)(u/\sigma)^2$ for $|u|$ much smaller than σ, and close to $(1/2)\sigma^2$ for $|u|$ much larger than σ.

The *Welsch loss* (or *LeClerc loss*) uses

$$\rho(u;\sigma) = 1 - e^{\left[-\frac{1}{2}\left(\frac{u}{\sigma}\right)^2\right]}$$

notice this looks like $(1/2)(u/\sigma)^2$ for $|u|$ much smaller than σ, and close to 1 for larger u. All of these losses increases monotonically in $|u|$ (the absolute value is important here!), so it is always better to reduce the residual.

14.2.2 IRLS: Weighting Down Outliers

The line is chosen by minimizing

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

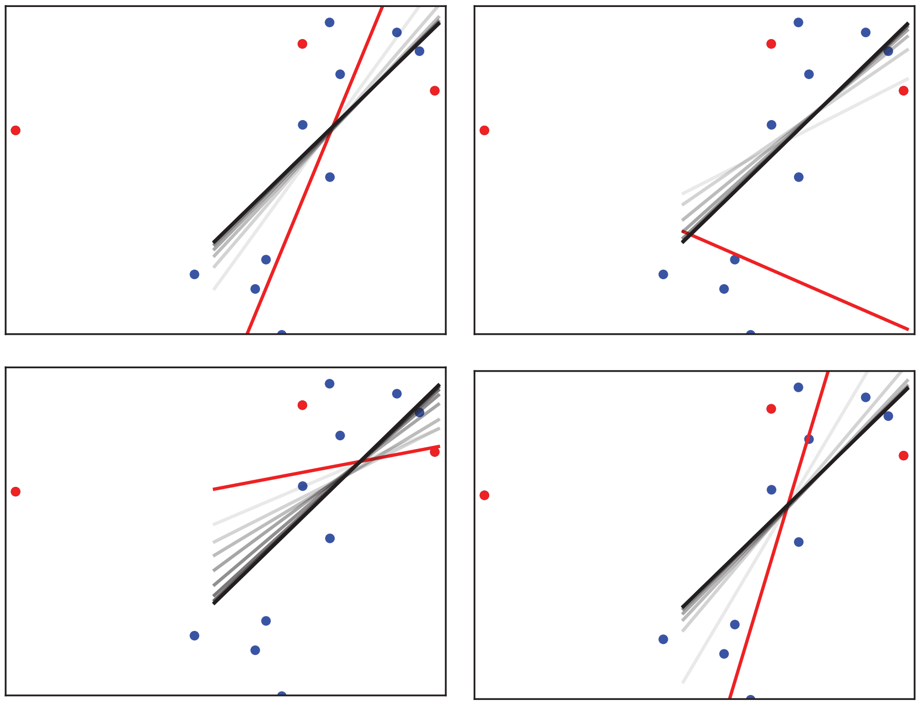

FIGURE 14.7: *Robust losses can control the influence of outliers.* **Blue** *points lie on a line, and have been perturbed by noise;* **red** *points are outliers. The* **red** *line shows a starting line, obtained by drawing a small random sample from the dataset, then fitting a line; the* **gray** *lines show iterates of IRLS applied to a Huber loss (later iterates are more opaque; scales are estimated as in the text). The procedure converges from a range of start points, some quite far from the "true" line. Notice how each start point results in the same line.*

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

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

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

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

Now $\left[(1/2) r(\mathbf{x}_i, \theta)^2\right]$ is the squared error used in total least squares. At the true minimum $\hat{\theta}$, writing

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

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

$$\sum_i w_i \nabla_\theta \left[(1/2) r(\mathbf{x}_i, \hat{\theta})^2\right] = \lambda \begin{pmatrix} 2a_1 \\ 2a_2 \\ 0 \end{pmatrix}.$$

will be true. In turn, if you knew the values the weights took at $\theta = \hat{\theta}$, then you could find the line with weighted total least squares. If you knew $\hat{\theta}$, you could estimate the weights. And you should be able to estimate a reasonable value for σ by observing points that are close to and far from the line.

This suggests the natural strategy: start with an estimate of the line, recover weights from that, then repeat:

- **Estimate the line** using the weights.
- **Estimate** σ using the new line and the points.
- **Re-estimate weights** using the new line and the new σ.

This procedure is an instance of a general recipe known as *iteratively reweighted least squares.* In this procedure, you should think of weights as a discount that is applied to the squared error – so weights are large for points that are close to the line, and small for points that are far. More instances of this recipe appear in Section **??**.

I have already dealt with estimating the line using the weights (Section 14.1.2) and the weights follow from the equation above. The parameter σ is often referred to as *scale.* In some cases, σ can be chosen from application logic. For example, if you are fitting lines to edges in pictures of buildings and you know that the pictures are fairly sharp and have little pixel noise, you can expect σ to be small. Alternatively, you can estimate σ from data. If the true line was known, then the values of $r(\mathbf{x}_i, \theta)$ would be "small" for inliers and "large" for outliers. Assuming there aren't too many outliers, then a fair and widely used estimate of scale at the n'th iteration is

$$\sigma^{(n)} = 1.4826 \text{ median}_i \left| r(x_i; \theta^{(n-1)}) \right| .$$

Procedure: 14.3 *Fitting a Line using Iteratively Reweighted Least Squares*

This procedure takes a set of N points $(x_{i,1}, x_{2,i})$ putatively lying on a line and obtains an estimate of that line.
Starting: Obtain an initial line $(a^{(1)}, b^{(1)}, c^{(1)})$ with $(a^{(1)})^2 + (b^{(1)})^2 = 1$ and an initial scale $\sigma^{(1)}$. One strategy to obtain an initial line is to draw a small set of points uniformly at random from the data, and apply total least squares. The scale may need to come from application considerations. Choose a robust cost function $\rho(u; \sigma)$ from Section 14.2.1 or somewhere else. Form an initial estimate of weights using

$$\begin{aligned} r_i^{(1)} &= a^{(1)} x_{1,i} + b^{(1)} x_{2,i} + c^{(1)} \\ w_i^{(1)} &= \left(\frac{d\rho}{du}\right) / \left(r_i^{(i)}\right) \end{aligned}$$

where the derivative is evaluated at $u = r_i^{(i)}$ and $\sigma^{(1)}$. Now use iterate three steps:

Estimate the line using weighted total least squares (Procedure 14.1) to obtain $(a^{(n+1)}, b^{(n+1)}, c^{(n+1)})$ and $r_i^{(n+1)}$ from $w_i^{(n)}$.

Estimate the scale, possibly using

$$\sigma^{(n)} = 1.4826 \text{ median}_i \, |r_i^{(n+1)}| \, .$$

Alternatively, use a fixed scale obtained using application considerations.

Re-estimate weights using

$$w_i^{(n+1)} = \left(\frac{d\rho}{du}\right) / \left(r_i^{(n)}\right)$$

where the derivative is evaluated at $u = r_i^{(n+1)}$ and $\sigma^{(n+1)}$.

Terminate iterations when either the change in the line is below a threshold or there have been too many.

14.2.3 Problems with Robust Losses

Each of these robust losses has a long record of being useful and helpful, but none is ideal. One important difficulty with IRLS one must start somewhere, and a bad start will result in a bad line. This can be controlled by using a number of randomly chosen lines to start – perhaps obtained by selecting some points at random, and fitting a line to them – then choosing the best line that results. As Figure 14.8

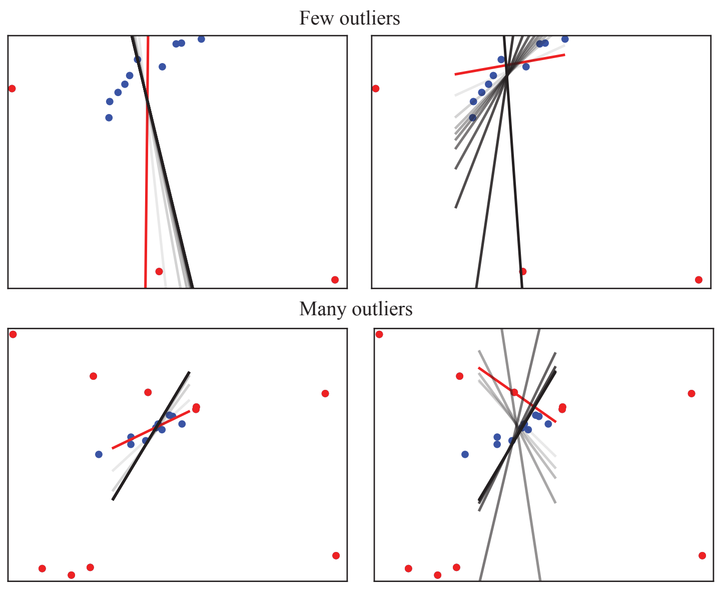

FIGURE 14.8: *Robust losses can fail, particularly when distant points still have some weight or if there are many outliers.* **Top left***: a bad start point leads to a bad line;* **top right***: on the same data set, quite a good start point still converges to a bad line. Here there are few outliers, but they are far from the data and they contribute a significant weight to the loss. When there are many outliers, this effect worsens. Because each outlier still contributes a signficant weight to the loss, even a good start fails (***bottom left***). A poor start (***bottom right***) also fails, and produces the same line as the good start – in fact, most starts end up close to this line. Again,* **blue** *points lie on a line, and have been perturbed by noise;* **red** *points are outliers; the* **red** *line shows a starting line, obtained by drawing a small random sample from the dataset, then fitting a line; the* **gray** *lines show iterates of IRLS (later iterates are more opaque).*

indicates, this isn't always a reliable approach. If the loss places some weight on distant points and the outliers are far away or there are many of them, then the iteration can settle on a bad line *even if* the initial line is good. The alternative – using a loss that wholly deemphasizes distant points, like the Welsch loss – means you may need to use an awful lot of start points. Robust losses are at their best when there are many data points and few outliers, and you have a reliable way of starting the fitting procedure.

Remember this: *IRLS fits lines or planes to points by iteratively estimating weights, estimating a scale, then computing a weighted total least squares fit. The weights come from a robust loss function. Robust loss functions look like squared error for small errors, and grow more slowly than the squared error for large errors. IRLS will usually encounter multiple local minima, and will usually need to be started with multiple random start points to find a good fit.*

14.3 RANSAC: SEARCHING FOR GOOD POINTS

An extremely powerful fitting algorithm, formalized by **?**, is based on two key ideas are: a fit on which many of the data points agree is likely right; and you can get this fit by randomly sampling data. The algorithm is usually called *RANSAC*, for RANdom SAmple Consensus.

RANSAC emerges naturally from an example. Imagine you must fit a line to a dataset that consists of about 50% outliers. Any pair of points yields a line. Repeat the following procedure: draw a pair of points uniformly and at random; pass a line through this pair; and use that line as a start point for (say) IRLS. About a quarter of these randomly drawn pairs will consist of two *inliers* (not outliers), and so should yield quite good estimates of the true line when passed to IRLS as a start point. Furthermore, you can tell when you have a reliable estimate of the true line – it should have many inliers.

You can tell how often you should draw pairs of points to have a high probability of seeing at least one reliable estimate of the true line. If the dataset is large, then a draw of k pairs will have a probability of 0.75^k of containing only bad pairs, so you can compute a k big enough to guarantee that you see at least one good pair with probability $1 - \epsilon$.

Procedure: 14.4 *Fitting a Line Using RANSAC*

This procedure takes a set of N points $(x_{1,i}, x_{2,i})$ putatively lying on a line and obtains an estimate of that line.
Start by choosing: the number of iterations required, k; the threshold used to identify a point that fits well, t; the number of nearby points required to assert a model fits well, d. Set up a collection of good fits, currently empty. Set $d = 3$ and the best current line to NULL.
Iterate until k iterations have occurred:

> Draw a sample of two distinct points from the data uniformly and at random, and determine the line through those two points.
>
> For each data point outside the sample, if the distance from the point to the structure is less than t, the point is close.
>
> If there are d or more points close to the line then there is a good fit. Refit the line using all these points and a robust loss. If the best current line is NULL, or the fitting error is better than the best fitting error, set d to the number of inliers, the best current line to the fitted line, and the best fitting error to the fitting error.

Report the best current line.

Warning: Note that in the version of RANSAC in the box, you must use a fitting error that is an average over points so that you can sensibly compare fits with different numbers of inliers. If, for example, you just sum the error over the points, you will incur a massive bias to fits with a small number of points and produce bad lines. The version of IRLS I gave averages error, and so is appropriate.

14.3.1 The Parameters for RANSAC

The number of points, n is always the smallest number you need to determine a fit. For lines, samples consist of pairs of distinct points drawn uniformly and at random from the dataset. The algorithm applies to much more general fitting, where sample contains the minimum number of points required to fit whatever you want to fit. For example, to fit planes in 3D, draw triples of points; to fit circles in 2D, draw triples of points, and so on (I will describe cases where you need to pick six points in Section 22.6).

The number of draws, k, is set by your desired probability of success and w, the fraction of inliers in the dataset. Usually, you do not know w, but you need only a reasonable estimate of this number. The expected value of the number of draws k required to get one good sample is given by

$$\begin{aligned} \mathrm{E}[k] &= 1P(\text{one good sample in one draw}) + \\ &\quad 2P(\text{one good sample in two draws}) + \dots \\ &= w^n + 2(1-w^n)w^n + 3(1-w^n)^2 w^n + \dots \\ &= w^{-n} \end{aligned}$$

(where the last step takes a little manipulation of algebraic series **exercises**). To be fairly confident that of seeing a good sample, you need to draw rather more than w^{-n} samples; a natural thing to do is to add a few – say f – standard deviations to this number. The standard deviation of k is

$$SD(k) = \frac{\sqrt{1-w^n}}{w^n}.$$

so you could draw

$$w^{-n}(1 + f\sqrt{1-w^n})$$

samples. An alternative approach to this problem is to look at a number of samples that guarantees a low probability z of seeing only bad samples. In this case

$$z = (1-w^n)^k,$$

which means that

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

Estimating the number of inliers: if w is unknown, it can be estimated from the fitting attempts. If you observe a sequence of A fitting attempts, of which g appear successful, then you can estimate $w = (g/A)^{1/n}$. You could start with a relatively low estimate of w, generate a sequence of attempted fits, and then improve the estimate of w. If you have more fitting attempts than the new estimate of w predicts, the process can stop. The problem of updating the estimate of w reduces to estimating the probability that a coin comes up heads or tails given a sequence of flips **exercises** .

The closeness threshold, t, tells you whether a point is close to the line or not (test the distance from point to line against this threshold). In general, specifying this parameter is part of the modeling process. Obtaining a value for this parameter is relatively simple. You generally need only an order of magnitude estimate, and the same value applies to many different experiments. The parameter is often determined by trying a few values and seeing what happens; another approach is to look at a few characteristic datasets, fitting a line by eye, and estimating the average size of the deviations.

14.3.2 Regimes that are Bad for RANSAC

Either calculation of the number of samples implies you need of the order of w^{-n} samples **exercises** . This becomes a problem if n is big or w is small (or both). These are regimes in which RANSAC is quite difficult to use successfully.

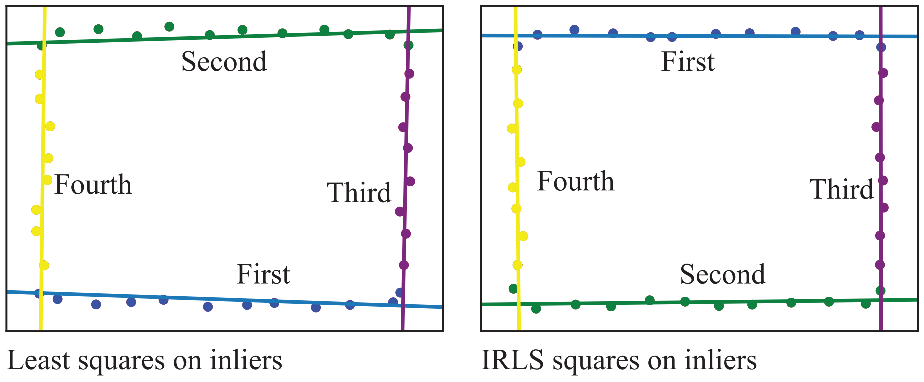

FIGURE 14.9: *Incremental RANSAC can successfully fit multiple lines to a noisy dataset. Points lie on the outline of a square, and have been perturbed by noise. I applied the strategy of Section 14.3.3 using a RANSAC line fitter. Colors show inliers and line for each round (in the order* **blue**, **green**, **purple**, **yellow***). On the* **left**, *the final fit to the inlying points is by total least squares. Because some inliers can be quite far from the line at the corners, the lines tend not to run close to the data points. On the* **right**, *the final fit uses 10 iterations of IRLS, with the Huber loss. The lines are now very close to the data points.*

If there are very few inliers (which isn't the same as a small fraction of inliers), the calculations are a poor guide to what will happen, because they assume that samples drawn are independent of one another. If there are very few inliers then good samples will tend to be correlated because they are likely to share some inliers. In this case a more delicate calculation is required to use RANSAC successfully [].

14.3.3 Fitting Many Lines with RANSAC

Here is a natural strategy for fitting k lines to a set of tokens. Start with a working set of all tokens. For k rounds, find the best line fitting the working set of tokens, then keep this line and take the tokens that are close to it out of the working set. It is difficult to make this strategy work using IRLS. Because there are many lines, there must be many outliers for each line. IRLS can respond quite badly to large numbers of outliers (Figure 14.8). RANSAC is a natural choice. As Figure 14.9 shows, it is quite important to use a robust loss in the line fit step.

Remember this: *RANSAC finds a good line in a pool of data by a form of randomized search. One repeatedly draws samples of pairs of points at random, passes a line through them, finds nearby points, and refits the line to those nearby points. The right line is one which has the best fitting error. You can fit circles, spheres, planes and so on with RANSAC if you can (a) determine how many points to draw and (b) fit the object to the points. RANSAC is reliable and well behaved outside regimes where there are very few inliers or where you must draw many points to fit.*

14.4 YOU SHOULD

14.4.1 remember these definitions:

14.4.2 remember these facts:

Use total least squares to fit lines and planes	249
Avoid fitting curved shapes if you can	250
IRLS fits lines or planes to points using a robust loss	258
RANSAC is the go-to algorithm for fitting in the face of outliers	262

14.4.3 remember these procedures:

Fitting a Line using Weighted Total Least Squares	247
Fitting a Plane using Weighted Total Least Squares	249
Fitting a Line using Iteratively Reweighted Least Squares	256
Fitting a Line Using RANSAC	259

14.4.4 use these resources:

14.4.5 be able to:

- Apply a Hough transform to find lines.
- Apply a simple voting method for instance classification and detection.
- Explain why voting is helpful and why censoring votes can be useful.

EXERCISES

QUICK CHECKS

14.1. Why does the family of lines given by $(u, v, 1)$ omit any line through the origin?

14.2. A plane is given by $\mathbf{a}^T\mathbf{x} + d = 0$; show the squared distance from a point $\mathbf{x}_i = (x_{1,i}, x_{2,i}, x_{i,3})$ to the plane is $(\mathbf{a}^T\mathbf{x} + d)^2$ if $\mathbf{a}^T\mathbf{a} = 1$.

14.3. Why is it fairly obvious that there should be local minima for a line fit using a robust loss?

14.4. Section 14.3.1 has:"The problem of updating the estimate of w reduces to estimating the probability that a coin comes up heads or tails given a sequence of flips" Explain.

14.5. A hyperplane in 4D is a set of points (x_1, x_2, x_3, x_4) such that $ax_1 + bx_2 + cx_3 + dx_4 + e = 0$ for some (a, b, c, d, e). You want to use RANSAC to fit a hyperplane to 4D data. How many points are there in a sample?

14.6. Which takes fewer samples: use RANSAC to fit a line to a collection of points that contains 20% outliers, with a probability of $1e-5$ that you see a good sample at least once; or use RANSAC to fit a plane to a collection of points that contains 20% outliers, with a probability of $1e-5$ that you see a good sample at least once? Why?

14.7. What will happen if you use RANSAC to fit a line to a dataset that contains no outliers?

14.8. What will happen if you use RANSAC to fit a line to a dataset that contains only outliers?

LONGER PROBLEMS

14.9. You wish to minimize

$$(1/2)\sum_i w_i(\mathbf{a}^T\mathbf{x}_i + c)^2,$$

subject to $\mathbf{a}^T\mathbf{a} = 1$.

(a) Show that, at a solution,

$$c = -a_1\overline{x_1} - a_2\overline{x_2}$$

and

$$\begin{pmatrix} \overline{x_1^2} - \overline{x_1}\ \overline{x_1} & \overline{x_1x_2} - \overline{x_1}\ \overline{x_2} \\ \overline{x_1x_2} - \overline{x_1}\ \overline{x_2} & \overline{x_2^2} - \overline{x_2}\ \overline{x_2} \end{pmatrix}\begin{pmatrix} a_1 \\ a_2 \end{pmatrix} = \mu\begin{pmatrix} a_1 \\ a_2 \end{pmatrix}.$$

(b) Show that, although the condition for a solution above is homogenous in a_1, a_2, there are only two possible solutions.

(c) Show that the two possible values of μ can be recovered in closed form.

(d) Show that the two possible solutions are lines at right angles (remember the eigenvectors of a symmetric matrix are orthogonal).

(e) Show that one possible solution maximizes the sum of squared distances and the other minimizes it.

14.10. A curve on the plane is given in implicit form by $f(\mathbf{x}) = f(x_1, x_2) = 0$.

(a) Show that, at a point on the curve $\mathbf{x}$, the unit normal to the curve is given by

$$\frac{\nabla_{\mathbf{x}} f(\mathbf{x})}{\sqrt{\nabla_{\mathbf{x}} f(\mathbf{x})^T \nabla_{\mathbf{x}} f(\mathbf{x})}}$$

as long as the denominator is not zero.

(b) Show that, at a point on the curve $\mathbf{x} = (x_1, x_2)^T$, the vector

$$\begin{bmatrix} \frac{\partial f}{\partial x_2} \\ -\frac{\partial f}{\partial x_1} \end{bmatrix}$$

is tangent to the curve (recall a tangent will be at right angles to the curve).

(c) Show that if $\mathbf{x}$ is the closest point on the curve to some other point $\mathbf{p}$, then $\mathbf{p} - \mathbf{x}$ is normal to the curve.

(d) Write $\mathbf{x}$ for the closest point on the curve to some other point $\mathbf{p}$. Show that $\mathbf{x}$ must satisfy the two equations

$$f(\mathbf{x}) = 0 \text{ and } (\mathbf{p} - \mathbf{x})^T \begin{bmatrix} \frac{\partial f}{\partial x_2} \\ -\frac{\partial f}{\partial x_1} \end{bmatrix} = 0$$

(e) Check that the curve $x_1^2 + 4y^2 - 4 = 0$ is an ellipse. Set up these equations for this curve. Show that there could be as many as four real solutions to the equations solved by the closest point for this curve. How would you find the actual closest point?

14.11. Write $S = 1 + 2a + 3a^2 + 4a^3 + \ldots$.

(a) Show that $S - aS = 1 + a + a^2 + \ldots$.

(b) Show that

$$S = \frac{1}{1 - a^2}$$

(c) Now show that

$$\begin{aligned} \mathrm{E}[k] &= 1P(\text{one good sample in one draw}) + \\ &\quad 2P(\text{one good sample in two draws}) + \ldots \\ &= w^n + 2(1 - w^n)w^n + 3(1 - w^n)^2 w^n + \ldots \\ &= w^{-n} \end{aligned}$$

14.12. This exercise sets up total least squares for circles in the plane. Write $\mathbf{c}$ for the unknown center of the circle and r for its unknown radius.

(a) A circle is a special curve, in that it is straightforward to find the closest point on a circle to a given point $\mathbf{p}$. Show how to obtain this point by joining the center of the circle to $\mathbf{p}$.

(b) Show that the square of the shortest distance from $\mathbf{x}_i$ to the circle is

$$\left[r \frac{(\mathbf{x}_i - \mathbf{c})}{\sqrt{(\mathbf{x}_i - \mathbf{c})^T (\mathbf{x}_i - \mathbf{c})}} + (\mathbf{c} - \mathbf{x}_i) \right]^T \left[r \frac{(\mathbf{x}_i - \mathbf{c})}{\sqrt{(\mathbf{x}_i - \mathbf{c})^T (\mathbf{x}_i - \mathbf{c})}} + (\mathbf{c} - \mathbf{x}_i) \right]$$

(c) Use the results of the previous exercise to set up the cost function to fit a circle to a set of points in the plane using total least squares. Don't look for a closed form solution; you would need a numerical optimization procedure to solve this.

(d) How would you set up IRLS in this case?

14.13. A circle in the plane is given by $(x_1 - a)^2 + (x_2 - b)^2 - c^2 = 0$.

(a) What are the radius and center of this circle?

(b) You can write this circle as $x_1^2 + x_2^2 - 2ax_1 - 2bx_2 - d = 0$. Show that $d > a^2 + b^2$.

(c) Show that any circle passing through the three points $(x_{1,1}, x_{2,1})^T$, $(x_{1,2}, x_{2,2})^T$, $(x_{1,3}, x_{2,3})^T$ must solve

$$\begin{bmatrix} 2x_{1,1} & 2x_{2,1} & 1 \\ 2x_{1,2} & 2x_{2,2} & 1 \\ 2x_{1,3} & 2x_{2,3} & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ d \end{bmatrix} = \begin{bmatrix} x_{1,1}^2 + x_{2,1}^2 \\ x_{1,2}^2 + x_{2,2}^2 \\ x_{1,3}^2 + x_{2,3}^2 \end{bmatrix}.$$

(d) Use the results of the previous sub exercises to explain how to use RANSAC to fit circles on the plane.

PROGRAMMING EXERCISES

14.14. This exercise explores fitting lines to points in the plane with total least squares.

(a) Set up a code to produce a simulated dataset for line fitting by generating samples of t uniformly and at random in the range $[-1, 1]$; turning these samples into points on the line $x_1 - x_2 = 0$ by forming $(t, t)^T$; then generating samples of $\mathbf{x} = (x_1, x_2)^T$ as $(t, t)^T + \xi$, where ξ is a normal random variable with mean $\mathbf{0}$ and covariance $\sigma^2\mathcal{I}$.

(b) Set up a code to fit a line with total least squares. Check your code by ensuring that, when you pass it points on some arbitrary line, it returns the coefficients of that line.

(c) Write δ for the dot product between $(1/\sqrt{2})(1, -1)^T$ (which is the normal of the line in the first sub exercise) and the unit normal of the line your code produced. Now investigate the relationship between δ and σ for $N = 10$ using simulations. Since δ is a random variable, you should plot the mean and the standard deviation of δ (obtained by lots of fitting) as a function of σ.

(d) What happens to δ if you fix σ at a fairly large value and increase N?

14.15. This exercise explores fitting lines to points in the plane with RANSAC and IRLS.

(a) Set up a code to produce a simulated dataset for line fitting. Your code should accept w (fraction of good points), N (number of points), σ (scale of gaussian noise) and b (scale of the outliers). You will need to produce about wN good points and about $(1 - w)N$ outliers. Do this by, for each point you simulate, drawing u uniformly and at random in the range $[0, 1]$. If $u > w$, produce an outlier, otherwise, produce an inlier. Generate samples on the line by generating samples of t uniformly and at random in the range $[-1, 1]$; turning these samples into points on the line $x_1 - x_2 = 0$ by forming $(t, t)^T$; then generating samples of $\mathbf{x} = (x_1, x_2)^T$ as $(t, t)^T + \xi$, where ξ is a normal random variable with mean $\mathbf{0}$ and covariance $\sigma^2\mathcal{I}$. Generate outliers as points obtained uniformly and at random in $[-k, k] \times [-k, k]$.

(b) Set up a code to fit a line to a set of points using RANSAC and total least squares. I've always found this sort of code easier to write than to figure out what an API is up to, but your mileage may vary. Check your code by ensuring that, when you pass it points on some arbitrary line, it returns the coefficients of that line.

(c) Write δ for the dot product between $(1/\sqrt{2})(1,-1)^T$ (which is the normal of the line in the first sub exercise) and the unit normal of the line your code produced. Now investigate the relationship between δ, σ and w. Since δ is a random variable, you should plot the mean and the standard deviation of δ (obtained by lots of fitting) as a function of σ. You might expect that sometimes δ is very bad because of an unlucky choice of outliers, so you should likely use a boxplot for δ. How often is δ very bad for reasonable w?

14.16. Use the results of previous exercises to build a code that can fit circles to points in the plane with total least squares. You should check the behavior of your code using gaussian noise only.

14.17. Use the results of previous exercises to build a code that can fit circles to points in the plane with IRLS. You should check the behavior of your code using gaussian noise and outliers.

14.18. Use the results of previous exercises to build a code that can fit circles to points in the plane with RANSAC. You should check the behavior of your code using gaussian noise and outliers.