Lecture 20

Least Squares Fitting: Noisy Data

Very often data has a significant amount of noise. The least squares approximation is intentionally well-suited to represent noisy data. The next illustration shows the effects noise can have and how least squares is used.

Traffic flow model

Suppose you are interested in the time it takes to travel on a certain section of highway for the sake of planning. According to theory, assuming up to a moderate amount of traffic, the time should be approximately

$$T(x) = ax + b$$

where b is the travel time when there is no other traffic, and x is the current number of cars on the road (in hundreds). To determine the coefficients a and b you could run several experiments which consist of driving the highway at different times of day and also estimating the number of cars on the road using a counter. Of course both of these measurements will contain *noise*, i.e. random fluctuations.

We could simulate such data in MATLAB as follows:

```
x = 1:.1:6;
T = .1*x + 1;
Tn = T + .1*randn(size(x));
plot(x,Tn,'.')
```

The data should look like it lies on a line, but with noise. Click on the `Tools` button and choose `Basic fitting`. Then choose a `linear` fit. The resulting line should go through the data in what looks like a very reasonable way. Click on `show equations`. Compare the equation with $T(x) = .1x + 1$. The coefficients should be pretty close considering the amount of noise in the plot. Next, try to fit the data with a spline. The result should be ugly. We can see from this example that **splines are not suited to noisy data**.

How does MATLAB obtain a very nice line to approximate noisy data? The answer is a very standard numerical/statistical method known as *least squares*.

Linear least squares

Consider in the previous example that we wish to fit a line to a lot of data that does not exactly lie on a line. For the equation of the line we have only two free coefficients, but we have many data points. We can

not possibly make the line go through every data point, we can only wish for it to come reasonably close to as many data points as possible. Thus, our line must have an error with respect to each data point. If $\ell(x)$ is our line and $\{(x_i, y_i)\}$ are the data, then

$$e_i = y_i - \ell(x_i)$$

is the error of ℓ with respect to each (x_i, y_i). To make $\ell(x)$ reasonable, we wish to simultaneously minimize all the errors: $\{e_1, e_2, \ldots, e_n\}$. There are many possible ways one could go about this, but the standard one is to try to minimize the *sum of the squares* of the errors. That is, we denote by $\mathcal{E}$ the sum of the squares:

$$\mathcal{E} = \sum_{i=1}^{n} (y_i - \ell(x_i))^2 = \sum_{i=1}^{n} (y_i - ax_i - b)^2 . \tag{20.1}$$

In the above expression x_i and y_i are given, but we are free to choose a and b, so we can think of $\mathcal{E}$ as a function of a and b, i.e. $\mathcal{E}(a, b)$. In calculus, when one wishes to find a minimum value of a function of two variables, we set the partial derivatives equal to zero:

$$\begin{aligned} \frac{\partial \mathcal{E}}{\partial a} &= -2 \sum_{i=1}^{n} (y_i - ax_i - b)\, x_i = 0 \\ \frac{\partial \mathcal{E}}{\partial b} &= -2 \sum_{i=1}^{n} (y_i - ax_i - b) = 0. \end{aligned} \tag{20.2}$$

We can simplify these equations to obtain

$$\begin{gathered} \left(\sum_{i=1}^{n} x_i^2 \right) a + \left(\sum_{i=1}^{n} x_i \right) b = \sum_{i=1}^{n} x_i y_i \quad \text{and} \\ \left(\sum_{i=1}^{n} x_i \right) a + nb = \sum_{i=1}^{n} y_i. \end{gathered} \tag{20.3}$$

Thus, the whole problem reduces to a 2 by 2 linear system to find the coefficients a and b. The entries in the matrix are determined from simple formulas using the data. The process is quick and easily automated, which is one reason it is very standard.

We could use the same process to obtain a quadratic or higher polynomial fit to data. If we try to fit an n degree polynomial, the software has to solve an $n \times n$ linear system, which is easily done. This is what MATLAB's basic fitting tool uses to obtain an n degree polynomial fit whenever the number of data points is more than $n + 1$.

Drag coefficients

Drag due to air resistance is proportional to the square of the velocity, i.e. $d = kv^2$. In a wind tunnel experiment the velocity v can be varied by setting the speed of the fan and the drag can be measured directly (it is the force on the object). In this and every experiment some random noise will occur. The following sequence of commands replicates the data one might receive from a wind tunnel:

```
v = 0:1:60;
d = .1234*v.^2;
dn = d + .4*v.*randn(size(v));
plot(v,dn,'*')
```

The plot should look like a quadratic, but with some noise. Using the tools menu, add a quadratic fit and enable the "show equations" option. What is the coefficient of x^2? How close is it to 0.1234?

Note that whenever you select a polynomial in MATLAB with a degree less than $n-1$ MATLAB will produce a least squares fit.

You will notice that the quadratic fit includes both a constant and linear term. We know from the physical situation that these should not be there; they are remnants of noise and the fitting process. Since we know the curve should be kv^2, we can do better by employing that knowledge. For instance, we know that the graph of d versus v^2 should be a straight line. We can produce this easily:

```
z = v.^2;
plot(z,dn,'*')
```

By changing the independent variable from v to $z = v^2$ we produced a plot that looks like a line with noise. Add a linear fit. What is the linear coefficient? This should be closer to 0.1234 than using a quadratic fit.

The second fit still has a constant term, which we know should not be there. If there was no noise, then at every data point we would have $k = d/v^2$. We can express this as a linear system `z'*k = dn'`, which is badly overdetermined since there are more equations than unknowns. Since there is noise, each point will give a different estimate for k. In other words, the overdetermined linear system is also inconsistent. When MATLAB encounters such systems, it automatically gives a least squares solution of the matrix problem, i.e. one that minimizes the sum of the squared errors, which is exactly what we want. To get the least squares estimate for k, do

```
k = z'\dn'
```

This will produce a number close to .1234.

Note that this is an application where we have physical knowledge. In this situation extrapolation would be meaningful. For instance we could use the plot to find the predicted drag at 80 mph.

Exercises

20.1 Find two tables of data in an engineering textbook or engineering website. Plot each (with "*" at data points) and decide if the data is best represented by a polynomial interpolant, spline interpolant, or least squares fit polynomial. Label the axes and include a title. Turn in the best plot of each. Indicate the source and meaning of the data.

20.2 The following table contains information from a chemistry experiment in which the concentration of a product was measured at one minute intervals.

T	0	1	2	3	4	5	6	7
C	3.033	3.306	3.672	3.929	4.123	4.282	4.399	4.527

Plot this data. Suppose it is known that this chemical reaction should follow the law: $c = a - b\exp(-0.2t)$. Following the example in the notes about the drag coefficients, change one of the variables so that the law is a linear function. Then plot the new variables and use the linear fit option to estimate a and b. What will be the eventual concentration? Finally, plot the graph of $a - b\exp(-0.2t)$ on the interval [0,10] (as a solid curve), along with the data (using `'*'`).