Getting Functions from Data
M250 Class Notes

D. Hundley

November 1, 2003

1 Introduction

In *empirical modeling*, we are given a data set, and we want to obtain a functional form for it. This is in contrast to what we have done in the course up to this point, where we obtain a function from first principles- for example, by modeling acceleration from Newton's second law, we get velocity and position.

There are a couple of ways to proceed at this point:

- **Parametric Modeling:** We have a function in mind that has some unknown parameters, and we determine the parameters from the data. We briefly looked at an example of this- If the data looks linear, we might assume a model function of the form $y = mx + b$, where m and b are chosen by using the data.
- **Nonparametric Modeling:** No function is given *a priori*. You may have seen an example of this in Calculus. There is a theorem that says if one has a periodic continuous function, then it can always be written as an infinite combination of sines and cosines,

$$f(x) = a_0 + \sum_{k=1}^{\infty} a_k \sin(kx) + b_k \cos(kx)$$

  where a_k and b_k can be found via integration[1].

  In this form, the model equation has enough parameters to fit *any* periodic continuous function, rather than a particular function.

As an additional classification under both Parametric and Nonparametric modeling is whether the model is *linear* or *nonlinear* in the unknowns. For example, in parametric modeling, if the data looks quadratic we might have the model function:

$$f(x) = ax^2 + bx + c$$

This function is nonlinear in x, but the unknown parameters are a, b, c. Therefore, this is a linear parametric modeling problem. On the other hand, if your model function is given by:

$$f(x) = A\mathrm{e}^{cx} + B\mathrm{e}^{-dx}$$

then the model is *nonlinear* in the unknown parameters c, d (but is linear in A, B). As you might guess, linear problems are much easier to solve, and we'll use some linear algebra to do it. Nonlinear problems will require nonlinear optimization (unless it is possible to transform it to a linear problem).

In this section, we will focus on the parametric models, and we'll start with linear problems.

[1] For more information, you can search the web for *Fourier Analysis*. We'll talk about this in detail next semester.

2 Parametric Modeling

In parametric modeling, we have a model function that we assume will be a good fit to the data. For example,

$$f(x) = mx + b \text{ the unknowns are } m, b$$

or, when we did biological modeling,

$$f(x) = Ax^b \text{ the unknowns are } A, b$$

It is possible to have more than 1 domain variable. For example,

$$f(x, y) = ax^2 + by^2 + cxy + dx + ey + f \text{ the unknowns are } a, b, c, d, e, f$$

is a general quadratic in two variables.

In general, we will be given a model function which we can write as:

$$y = f(x_1, x_2, \ldots, x_n; w_1, w_2, \ldots, w_m)$$

where $x_1, \ldots, x_n$ are the input variables (the variables for the domain), and $w_1, w_2, \ldots, w_m$ are the parameters that we will find by using the data.

3 An Example in Detail: The Line of "Best" Fit

We can state the general problem:

Given a set n of ordered pairs, $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$
Find the line of "best" fit, $y = mx + b$.

Ideally, this means that:

$$\begin{array}{ll} y_1 & = mx_1 + b \\ y_2 & = mx_2 + b \\ \vdots & \vdots \\ y_n & = mx_n + b \end{array} \quad \text{or} \quad \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

If the data lies *exactly* on a line, then these equations are very redundant- we only need two points to find the equation for a line. Therefore, we assume that the points do not lie exactly on a line, which implies that there is some error between the data point itself, (x_i, y_i) and where the data point *would be* if the data point was without error, $(x^*, mx^* + b)$.

To simplify things for our first time out, let's assume that all of the error is in measuring y, not in x. In that case, the data point without error should be $(x_i, mx_i + b)$.

There are many ways to measure the error. For example, we might make the error on the i^{th} data point as the absolute value of the difference between the desired output, y_i, and the model output, $mx_i + b$,

$$|y_i - (mx_i + b)|$$

The problem with the absolute value function is that it is not differentiable at its minimum. An attractive alternative is to square the error,

$$(y_i - (mx_i + b))^2$$

where the overall error is the sum of the individual errors:

$$E(m, b) = (y_1 - (mx_1 + b))^2 + \ldots + (y_n - (mx_n + b))^2 = \sum_{i=1}^{n} (y_i - (mx_i + b))^2$$

IMPORTANT: Notice that the error function E is a function only of m and b. The data has been entered in for x_i, y_i. For example, suppose our data set looks like:

x	-1	0	1
y	-4.9	-1.8	1.2

Then the error function is:

$$E(m,b) = (-4.9-(-m+b))^2 + (-1.8-(0+b))^2 + (1.2-(m+b))^2 = 28.69 - 12.2m + 11b + 2m^2 + 3b^2$$

which we can plot in Figure 1.

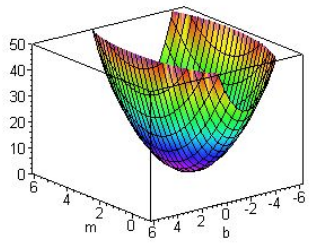

Figure 1: A plot of the error function for the line of best fit

We also note that the error function is quadratic in the unknowns m and b (with positive coefficients). Therefore, the error function we constructed will always have a global minimum (and no maximums), and the global minimum will occur at the critical point. Therefore, to find values for m and b, we need to set the gradient of E to zero. Doing this, we will obtain two equations in the two unknowns, m and b.

Now the goal is to minimize the error. Since E is quadratic in m, b (with positive coefficients), there will be a global minimum where the gradient is zero. Solving these equations will give us two equations in two unknowns:

$$\frac{\partial E}{\partial m} = 2\sum_{i=1}^{n}(y_i - (mx_i+b))(-x_i) = -2\sum_{i=1}^{n} x_i y_i + 2m\sum_{i=1}^{n} x_i^2 + 2b\sum_{i=1}^{n} x_i$$

$$\frac{\partial E}{\partial b} = 2\sum_{i=1}^{n}(y_i - (mx_i+b))(-1) = -2\sum_{i=1}^{n} y_i + 2m\sum_{i=1}^{n} x_i + 2b\sum_{i=1}^{n} 1$$

Setting these to zero, we have the following system (all sums have an index $i = 1 \ldots n$):

$$\begin{bmatrix} \sum x_i^2 & \sum x_i \\ \sum x_i & n \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} = \begin{bmatrix} \sum x_i y_i \\ \sum y_i \end{bmatrix}$$

which we can solve directly using either Cramer's Rule or the inverse of a 2×2 matrix,

$$m = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{n\sum x_i^2 - (\sum x_i)^2}, \quad b = \frac{-\sum x_i \sum x_i y_i + \sum x_i^2 \sum y_i}{n\sum x_i^2 - (\sum x_i)^2}$$

This method is implemented when you use Matlab's Data Fitting tool, or we could implement it directly using a short function. In the next section, we'll see the linear algebra version which is very short.

3.1 A Summary of Parametric Modeling

As we've seen, when performing parametric modeling, you must make two choices: (1) The form of the function that you'll use, and (2) A choice of the error function. Once the error has been chosen, we will attempt to minimize it.

3.2 Other Choices for "Best" Line

Here we examine other choices for fitting a line to data. The choice depends primarily on your assumptions about the error.

3.2.1 The Median-Median Line

The sum of squares error will be relatively sensitive to data points that have unusually large error. If the data has a lot of error, you might choose the median-median line (the median of a data set is not sensitive to unusually large errors). This method is also appropriate if there is error both in x and in y.

To find the median-median line, we:

- Separate the data into three groups according to x. Let n be the number of data points. If n is divisible by 3, we'll have three equal groups. If $n/3$ has a remainder of 1, let the middle group have the extra data point, and if there is a remainder of 2, the two outer groups will have the extra data points.
- Let (a_1, b_1) be the median of the first group of data (in x and y), and let (a_2, b_2) be the median of the second group. Let $y = mx + b$ be the line that passes through these two points- that is, $y - b_1 = \frac{b_2 - b_1}{a_2 - a_1}(x - a_1)$
- Shift the line 1/3 of the way towards the middle median point.

This method has been implemented in the function file `medmedline.m` on the class website. Input the $n \times 2$ array of data, and the output will be an $n \times 3$ data for the median-median line (third column is the set of residual errors), as well as the slope m and intercept b. The function will also sort the data, so it may return the data in a different order than what was input (unless the domain values are already in ascending order).

3.2.2 Other Error Measures

- Rather than take the error as a vertical measure, we might define the error as the distance between the data point and an orthogonal projection to the line. This would be appropriate if there are errors in both x and y.
- We can also change the sum of squares error to be:

$$E(m, b) = \max_i |y_i - mx_i - b|$$

  In this case, the optimization problem becomes a linear program. See the text for more details.

- Similarly, if we take the error as:

$$E(m, b) = \sum_{i=1}^{n} |y_i - mx_i - b|$$

  then again the optimization problem could be written as a linear program (very much like the rocket launch objective function).

4 Fitting with other Models

We can always translate a linear fitting problem into an easy linear algebra problem. With the line of best fit, we saw that we could write our equations as:

$$A\mathbf{c} = \mathbf{y}$$

where the i^{th} row of A was given by using the i^{th} data point, $[1 \quad x_i]$, and the vector c is the vector containing the unknown slope and intercept.

The matrix A, not being square, is not invertible. But, if we multiply both sides of the equation by A^T, we get:

$$\left(A^T A\right) \mathbf{c} = A^T \mathbf{y}$$

We will assume that the 2×2 matrix is invertible, and the solution will be given by:

$$\mathbf{c} = \left(A^T A\right)^{-1} A^T \mathbf{y}$$

If you've had linear algebra, you might recall that this indeed the "least squares" solution to the original equation. If we were to compute what these matrices look like, we'd come up with the same solution that we did by using Calculus, and setting the partial derivatives of our error function to zero.

Similarly, if we wish to model data using a polynomial of degree k, then the matrix equation would look the same, except that the i^{th} row of A would be:

$$A_i = [1 \quad x_i \quad x_i^2 \ \ldots \ x_i^k]$$

You might also recall that we can fit a set of $n+1$ points exactly using a polynomial of degree n. In this case, the matrix A would be $n+1 \times n+1$, and inverting it directly would give the interpolating[2] polynomial. We could also use *Lagrange Polynomials*. We won't discuss them here; see your text for more information. Lagrange polynomials play a big role in Numerical Analysis, where we use them to construct numerical algorithms for differentiation and integration.

NOTE: Using a high degree polynomial is a bad idea; it can oscillate wildly between data points. When fitting data with noise, one must always balance how accurate the model is versus the smoothness of the fitting function. In statistics, this is the balance between "bias" and "variance". In machine learning, this is a balance between "memorization" and "generalization". If your data has zero noise, then you might consider interpolating functions as described in the next section. Next semester we will be looking at this problem in detail as we build models from $\mathbb{R}^n$ to $\mathbb{R}^m$.

As a last example, suppose we have data that we want to model using:

$$y = A\sin(x) + B\cos(x) + C\sin(3x) + D$$

The matrix equation would look the same, except that the matrix A would have rows:

$$A_i = [\ \sin(x_i) \quad \cos(x_i) \quad \cos(3x_i) \quad 1\ ]$$

with $\mathbf{c} = [\ A\ B\ C\ D\ ]^T$. This is why it is a linear problem, even though the model functions themselves are not linear. However, the model function:

$$y = A\sin(k_1 x) + B\cos(k_2 x) + C\sin(k_3 x) + D$$

would be a nonlinear fitting problem. In fact, in such a case, we would probably use something called *Fourier analysis*.

We'll use Matlab's "Basic Fitting" tools to try some problems out.

[2]To interpolate means to go through the data points *exactly*, with no error.

5 The Importance of the Error

Before going on to the error-free modeling situation, we want to emphasize here the importance of checking the residual errors (the Basic Fitting GUI provides this as an option).

When we are fitting a curve through data, we will generally assume (implicitly if not explicitly) that the errors are *random*. Without going into details about what kinds of random distributions you might see (normal, uniform, etc.), when you plot the residual error using a "good fit", you should not see any structure in the residuals. Seeing structure in the residuals means that there was structure in the data that has not been modeled using your function.

For example, consider the figure below, where a data set has (mistakenly) been modeled using a line. We clearly see that there is structure in the residuals error- this needs to be incorporated into the model.

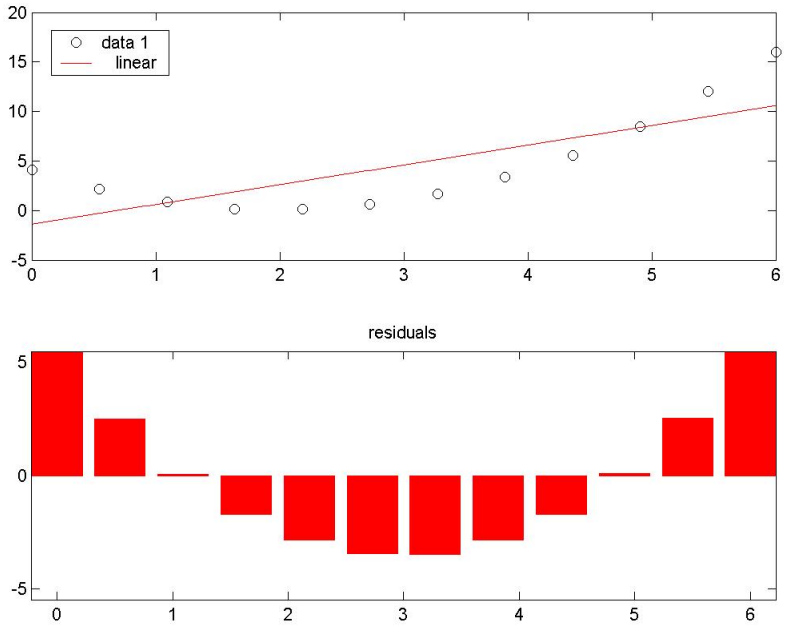

Figure 2: In checking the residuals, we are looking for randomness. In this case, there is definite structure that needs a closer look- we should modify the model function.

6 Error Free Modeling using Cubic Splines

To interpolate data means that we want to enforce a restriction that our model function must pass, without error, through the given data. Such problems come up frequently in Computer Assisted Drafting (CAD), or computer animation for example, where the data is *exact*.

Closely related to cubic splines are $B-splines$ and NURBS (Non-uniform rational B-spline). These curves are also piecewise defined cubics, but you can interactively change the shapes by using "control points" (you might have seen the related Bezier curves in the past, which also uses control points). These topics are integral to the study of CAD design and animation.

6.1 The Simplest Solution

We could use a piecewise linear function to perform the interpolation. In this case, we get a continuous function, but it is probably not differentiable at the knots (a "knot" in interpolation language is a data point). See the class notes for the details.

On the other extreme, we could use a complicated function to interpolate all the data, but we will normally not have any control over the size of the derivatives (and thus, we will have no control over the shape of the curve).

6.2 The Cubic Spline

As a balance between smoothness and simplicity, the cubic spline technique is among the most popular choices. Here, we will use piecewise cubic functions. In what follows, we will see that by choosing cubics, we can enforce continuity of the function and continuity of the first two derivatives.

Definition: A cubic spline on the data $\{(x_i, y_i)\}_{i=1}^{n}$ is a piecewise defined function, where the function on the j^{th} interval is given by:

$$S_j(x) = c_0 + c_1 x + x_2 x^2 + c_3 x^3, \quad x_j \le x < x_{j+1}$$

Note that the domain is a half-closed interval. There are $n-1$ functions S_j that make up the full model function. The last interval with right endpoint x_n will be covered by $S_{n-1}(x)$. You might check that in this problem, the output of a cubic spline routine will be a set of $4(n-1) = 4n-4$ coefficients, $\left\{(c_0^j, c_1^j, c_2^j, c_3^j\right\}_{j=1}^{n-1}$.

The following conditions on these polynomials will uniquely define the set of coefficients:

1. Interpolation: $S_j(x_j) = y_j$, for $j = 1, 2, \ldots, n$. Furthermore, $S_{n-1}(x_n) = y_n$. This gives n equations.
2. At all interior points, the functions match up. This gives $n-2$ equations:
$$S_j(x_{j+1}) = S_{j+1}(x_{j+1})$$
3. At all interior points, the derivatives match. This gives another $n-2$ equations:
$$S_j'(x_{j+1}) = S_{j+1}'(x_{j+1})$$
4. At all interior points, the second derivatives match. This gives another $n-2$ equations:
$$S_j''(x_{j+1}) = S_{j+1}''(x_{j+1})$$

   At this point, you might check that we currently have $4n-6$ equations for our $4n-4$ unknowns. We need to define two more conditions, and this is where there are some standard choices.

5. (a) The natural spline: $S_1''(x_1) = 0$ and $S_{n-1}''(x_n) = 0$.
   (b) The clamped spline: Define the first derivative at the first and last point.
   (c) The "Not-a-knot" condition: The third derivative is continuous at x_2 and x_{n-1}. Since the third derivative of S_j is $6c_3^{(j)}$, this means:
   $$c_3^{(1)} = c_3^{(2)}, \quad c_3^{(n-2)} = c_3^{(n-1)}$$
   This is the default condition for Matlab's `spline` command.