Lecture Notes: Nonlinear Model Fitting

The idea of root finding can be applied to nonlinear model fitting. This is done by writing an error function and finding its minimum, which is where its derivative is zero. In conceptual form, it works by writing an error function like the one used for linear model fitting:

$$E = \sum (\text{data} - \text{model}) \tag{1}$$

Taking the derivative:

$$\frac{\partial E}{\partial \ \text{unknown(s)}} = 0 \tag{2}$$

Starting with an initial guess, an iterative technique can be used to move towards the zero crossing.

For example, suppose we want to fit data like that shown in figure 1. We can select a model of the form $y = ax^2$ where a is the unknown. For this model, the fitting problem is linear, and can be solved using the normal equations. Do not confuse which part of the function needs to be linear, or you will do a lot of extra work for no reason!

Suppose however we want to fit a data that looks like that shown in figure 2. In this case, the data looks like an exponential. We can select a model of the form $y = e^{ax}$ but this model is nonlinear in the unknown a. Therefore, we write an error function as follows:

$$E = \sum_{i=1}^{N} (y_i - e^{ax_i})^2 \tag{3}$$

We take the partial derivative with respect to the unknown:

$$\begin{aligned} \frac{\partial E}{\partial a} &= \sum_{i=1}^{N} 2(y_i - e^{x_i a})(-e^{x_i a})(x_i) && (4) \\ &= \sum_{i=1}^{N} -2x_i(y_i - e^{x_i a})(e^{x_i a}) && (5) \\ & && (6) \end{aligned}$$

We wish to minimize this error, which is where this function is equal to zero:

$$\sum_{i=1}^{N} -2x_i(y_i - e^{x_i a})(e^{x_i a}) = 0 \tag{7}$$

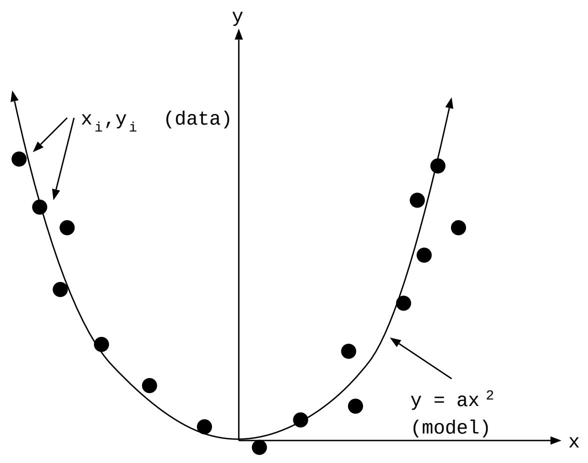

Figure 1: Fitting a quadratic function to data.

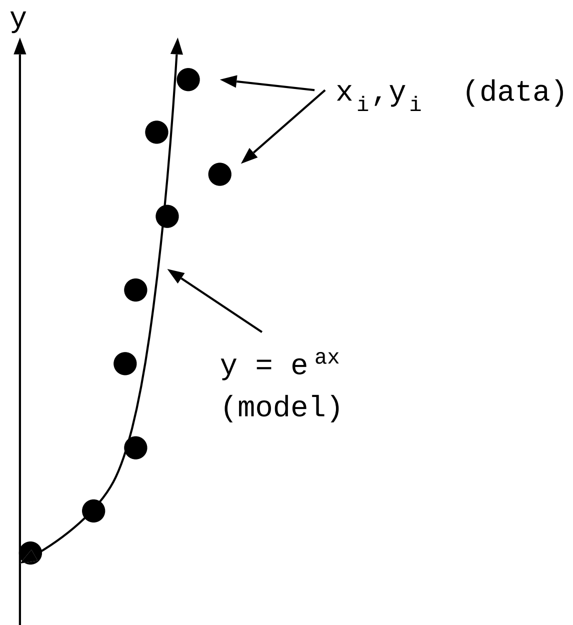

Figure 2: Fitting an exponential function to data.

Since x_i is a constant with respect to a, this can be simplified to

$$\sum_{i=1}^{N}(y_i - e^{x_i a})(e^{x_i a}) = 0 \tag{8}$$

To solve this problem iteratively using the root finding approach, let

$$\begin{aligned} f(a) &= \sum_{i=1}^{N}(y_i - e^{x_i a})(e^{x_i a}) && (9) \\ &= \sum_{i=1}^{N}(y_i e^{x_i a} - e^{2x_i a}) && (10) \\ & && (11) \end{aligned}$$

The derivative is

$$f'(a) = \sum_{i=1}^{N}(x_i y_i e^{x_i a} - 2x_i e^{2x_i a}) \tag{12}$$

Now we take an initial guess, say, $a = 4.0$. Through successive iterations of the following we converge to the solution:

$$a_{n+1} = a_n - \frac{f(a_n)}{f'(a_n)} \tag{13}$$

A program solving this problem will be demonstrated in class.

For a second example, suppose we wish to fit a model of the form $y = sin(ax)$ to some data, where a is the unknown. We can write an error function as follows:

$$E = \sum_{i=1}^{N}(y_i - sin(ax_i))^2 \tag{14}$$

We take the partial derivative with respect to the unknown:

$$\begin{aligned} \frac{\partial E}{\partial a} &= \sum_{i=1}^{N} 2(y_i - sin(x_i a))(-cos(x_i a))(x_i) && (15) \\ &= \sum_{i=1}^{N} -2x_i(y_i - sin(x_i a))cos(x_i a) && (16) \\ & && (17) \end{aligned}$$

We wish to minimize this error, which is where this function is equal to zero:

$$\sum_{i=1}^{N} -2x_i(y_i - sin(x_i a))cos(x_i a) = 0 \tag{18}$$

Since x_i is a constant with respect to a, this can be simplified to

$$\sum_{i=1}^{N}(y_i - sin(x_i a))cos(x_i a) = 0 \tag{19}$$

To solve this problem iteratively using the root finding approach, let

$$\begin{aligned} f(a) &= \sum_{i=1}^{N}(y_i - sin(x_i a))cos(x_i a) && (20)\\ &= \sum_{i=1}^{N}(y_i cos(x_i a) - sin(x_i a)cos(x_i a)) && (21)\end{aligned}$$

The derivative is

$$\begin{aligned} f'(a) &= \sum_{i=1}^{N}(y_i(-sin(x_i a))x_i - [cos(x_i a)x_i cos(x_i a) + sin(x_i a)(-sin(x_i a))x_i]) && (22)\\ &= \sum_{i=1}^{N}\left(-x_i y_i sin(x_i a) - x_i cos^2(x_i a) + x_i sin^2(x_i a)\right) && (23)\end{aligned}$$

After taking an initial guess, successive iterations of the following converge to a solution:

$$a_{n+1} = a_n - \frac{f(a_n)}{f'(a_n)} \tag{24}$$

A program solving this problem will be demonstrated in class.

This approach can also be extended to multidimensional problems. In practice, the most popular techniques for obtaining these types of solutions are the Levenberg-Marquardt algorithm, the Gauss-Newton algorithm, and variations of these that depend on the problem conditions. For example, the MATLAB function `nlinfit()` uses the Levenberg-Marquardt algorithm. Most mathematical and statistical programming environments operate similarly.