

Math 131/135/194, Fall 2004

Applied Calculus

Profs. Kaplan & Flath

Macalester College

Lab 5: Fitting Functions using Optimization

Goal of this lab

To gain experience with using minimization in order to find parameters that bring a functional form as close as possible to observed data.

Software

You will need to use the same package, `optim.r`, that was used in Lab 4 on Optimization. Remember to “source” this package each time you start R.

In addition to the `evalAtGrid`, `contour`, and `nlm`, you will use the `linearErrorFun` function.

Function fitting

Optimization is extremely important throughout science, technology, and the economy. Many of the ways we describe the action of nature involve optimization of theoretical objective functions such as fitness or potential energy. One application of optimization that is of particular importance to us is *function fitting*.

The setting for function fitting is this: we have a set of observed data consisting of some explanatory variables and a corresponding response variable. We have already seen some examples of this in previous labs. In the US Census data we used the year as the explanatory variable; the response variable was the population. In the ball-bouncing data the bounce number was the explanatory variable and the bounce height the response variable.

In fitting a function, we seek to construct a model of the relationship between the explanatory and response variables. The function has some unknown parameters that can be adjusted to make the function as close as possible to the relationship implied by the observed data. Suppose we make several measurements, $i = 1, 2, \dots, N$ of the explanatory variable t_i and the corresponding response variable y_i . For example, in the census data from Lab 2, $t_1 = 1790$ and $y_1 = 3.929$ million people. There were 22 censuses from 1790 to 2000, so $N = 22$. More vocabulary: each of the paired measurements indexed with the subscript i is called a *case*.

The function that we fit in Lab 2 was the solution to the

logistic growth model

$$f(t_i, A, m, b) = \frac{A}{1 + e^{-(b+mt_i)}}$$

The input variable is t_i . The function’s parameters are A , b , and m .

Back in Lab 2, we used the letter t , without a subscript, to denote the explanatory variable, time. But now we will be a bit more formal in accounting for each of the individual measurements. When we write t_i , the subscript i refers to a particular census; t_i is the time at which the census denoted by i occurred.

It is conventional to use the letter x to denote the explanatory variable, even when this variable is time. But, rather than relying on alphabetical conventions to determine which variables are explanatory and which ones response, it’s best to figure this out by careful reading and thought about the problem.

If the model were perfect, the observed population would be exactly equal to the output of the model function when the input is set to the corresponding value of the explanatory variable. That is, once we know the parameters A , m , and b , to find the population in the i th census, y_i , we just need to plug these parameters and the value of t_i :

$$y_i = f(t_i, A, m, b). \quad (1)$$

In practice, however, models are rarely perfect and our observations contain errors or random components not included in the model. (Indeed, models are far from perfect and, next semester, we will spend a lot of time and effort answering the question, “Does this model tell us anything at all about the observed data?”) There are many ways to include the imperfection of models when relating them to data. The most important one in practice is to include an additive error term in Eq. 1, giving

$$y_i = f(t_i, A, m, b) + \epsilon_i. \quad (2)$$

The quantity ϵ_i is called the model *error* or model *residual*.

Note that there is a subscript on ϵ . This means that each case — in our example, each of the censuses from 1790 through year 2000 — has its own residual. No matter how crumbly is the model $f(t_i, A, m, b)$, we can always find some

ϵ_i that pairs the explanatory value t_i to the corresponding response value y_i . To find the actual ϵ_i , we just compute:

$$\epsilon_i = y_i - f(t_i, A, m, b). \quad (3)$$

When we **fit a function**, we seek to find the values of the arguments — in this case the parameters A , m , and b , that make the residuals ϵ_i as small as possible. (If it were possible to make the residuals exactly equal to zero, then the model function would describe the relationship between t_i and y_i perfectly.)

There are many ways to quantify the ideal of “making the residuals small.” The most important — but not the only one — is to minimize the sum of squares of the residuals. That is, minimize the objective function $g(\cdot)$

$$g(A, m, b) = \sum_{i=1}^N \epsilon_i = \sum_{i=1}^N (y_i - f(t_i, A, m, b))^2 \quad (4)$$

Since t_i and y_i are our observed data, their values are fixed for each i . Thus, although t_i and y_i appear symbolically in Eq. 4, they really have fixed numerical values and so the objective function g takes as arguments only the parameters of the model function.

This sort of objective function is called a **least-squares** objective. The process of finding the optimal parameters is called **least-squares fitting**. The idea of least-squares fitting is about 200 years old, and associated with the mathematician Gauss.

We will illustrate the ideas of fitting functions by constructing a very simple linear function $f(x) = mx + b$ with parameters m and b .

- Pick some values of m and b that you like. (Try to keep them reasonably small, say in the range -100 to 100 . For example, suppose $a = 2$ and $b = -5$.)
- Pick 5 to 10 values of x and enter them into R, storing them in a variable called x . For example, $x = c(1, 2, 4, 8, 10)$.
- By hand, for each of the values of x , compute $mx + b$. But do the computation in a sloppy way, introducing a bit of error (but making them approximately right). Enter these sloppy values of y into a variable called y . For example, $y = c(-2, -1, 4, 10, 15)$. A plot of y against x should show points that are almost on a line, but not quite.

Now, using only the data in x and y , we are going to try to find the values of m and b . (In practice, we wouldn't know the values of a and b that generated the data or even that a model of the form $y = mx + b$ is a correct model.) One way to do this is to guess values of m and b , compute the residuals from the model, and then see if we can make a better guess. We can do this by hand:

```
> mguess = 1
> bguess = 0
> modelvals = mguess*x + bguess
> residuals = y - modelvals
> sum(residuals^2)
[1] 47
```

Now we repeat the process with a different guess

```
> mguess = 1
> bguess = 2
> modelvals = mguess*x + bguess
> residuals = y - modelvals
> sum(residuals^2)
[1] 63
```

Our first guess produced smaller residuals; it's a better guess.

If we continued this process for a long time, we might find the very best guesses for m and b , the ones that produce the smallest residuals. This would be tedious, in part because of all the typing involved in the above, in part because we don't have a fast, systematic way to improve the guesses.

We can address the typing issue by a bit of computer programming, which your professors have done for you. We have written a function `linearErrorFun` that makes it easy to compute the size of the residuals. To use it, you pass as arguments the x and y data. The returned value will be a function that allows you to specify guesses for b and m and receive back the sum of squared residuals:

```
> f = linearErrorFun(x,y)
> f(c(0,1))
[1] 47
> f(c(2,1))
[1] 63
```

Notice that the parameters m and b are being passed as a vector, in the order b then m . It's also critical to notice that `linearErrorFun(x,y)` is being used to create another function, called `f` in this example, and that `f` is used to find the sum of square residuals.

- **1** Make a contour plot of the values of `f` for a range of parameters m and b . You will do this by setting up two variables that have a range of m values and of b values using commands like this

```
> mrange = seq(-10,10,length=100)
> brange = seq(-10,10,length=100);
> z = evalAtGrid(brange,mrange,f)
> contour(brange,mrange,z)
```

It might be that this contour plot is hard to interpret. If you want to focus on the contours near the minimum, you can give an optional argument to `contour` so that it draws in particular contour lines. For example,

```
> contour(brange,mrange,z,  
  levels=c(50,100,200,500,1000))
```

Even simpler, you might want to make a contour plot of the logarithm of z .

- 2 If your contour plot is nicely drawn, it should

be obvious where the argmin of f is. `nlm` will find the argmin very quickly. Use `nlm`, with starting guess $c(0,0)$, to find the best values of m and b .

The `nlm` approach can be used to fit many sorts of functions, for example, the logistic function fitted to the census data. We will do this in class.