Carolyn Cotterman
Stat 243
December 10, 2010

Assignment 4

1 Introduction

For this assignment, I wrote a subroutine in C that will sweep the columns of a matrix. I then created a function in R that accesses this C code such that an R user can use it to sweep a matrix. Next, I demonstrated my sweep function's abilities by performing a regression that uses it. Finally, I coded and tested a C program that will perform a non-linear regression.

My source code and output pertaining to this assignment are contained in the following documents:

- HW4_main.c
- HW4_functions.c
- HW4_header.h
- beatonsweep.c
- beatonsweepr.c
- bsweep.r
- HW4_R_code.r
- hw4_R_out.txt
- hw4_C_out.txt

2 Sweep

The "Beaton sweep" algorithm is useful in many contexts of statistical analysis. For example, by enabling us to find the inverse of a square matrix, it can be used in a linear regression context to solve for $\hat{\beta}$, where $\hat{\beta} = (X'X)^{-1}X'Y$.[1] Sweeping all of the columns of a square matrix yields the inverse of the original matrix. So, we could simply apply the Beaton sweep algorithm to columns 1 thru p of $X'X$ in order to find $(X'X)^{-1}$, and then multiply by $X'Y$ in order to obtain our regression coefficients.

[1] As usual, X is an nxp matrix containing a value for each of our p explanatory variables for each of our n observations, Y is an nx1 matrix containing an outcome value for each of our n observations, and $\hat{\beta}$ is a px1 matrix containing the beta coefficients for each explanatory variable.

However, there is an even better way to use Beaton sweep for solving a regression problem. This method readily yields not only estimates for the coefficients, but also provides the coefficients' standard errors and the residual sum of squares (RSS).[2] Here's how it works: if we create a matrix, $X^* = [X : Y]$ (that is, matrices X and Y are placed side-by-side to create X^*), then

$$X^{*1}X^* = \begin{bmatrix} X'X & X'Y \\ Y'X & Y'Y \end{bmatrix} \tag{1}$$

Now, if we sweep columns 1 thru p of the matrices above, then we get

$$\begin{bmatrix} (X'X)^{-1} & (X'X)^{-1}X'Y \\ -Y'X(X'X)^{-1} & Y(I - X(X'X)^{-1}X')Y \end{bmatrix} = \begin{bmatrix} \frac{1}{\sigma^2}V(\hat{\beta}) & \hat{\beta} \\ -\hat{\beta}' & RSS \end{bmatrix} \tag{2}$$

I wrote a C function that implements the Beaton sweep algorithm on specified columns of a square matrix. To test my code, I demonstrated that sweeping all the columns of a matrix yielded the inverse of my original matrix, as obtained from R (and sweeping a second time yeilds the original matrix). In the next section, I will describe the process by which I was able to call my C function from R, and demonstrate its use by solving a regression problem in the manner described above.

3 Writing Functions in R

I wrote a function in R that calls my sweep subroutine (i.e., "beatonsweep.c") using the ".c method" for calling C functions from R. This process was more frustrating than I had anticipated due to some of Window's funky defaults and non-intuitive interface, but I was eventually able to get it to work, and will (hopefully) be able to do this again in the future without the hitches of doing it for the first time. For prosperity's sake, here are some notes on the process (for those of us dumb enough to use a Windows machine):

1. Create a wrapper for beatonsweep. Call this wrapper "beatonsweepr.c".

   Note: The.C method requires that all of the arguments passed to the function be pointers. But, if we include a wrapper, then we don't need to modify the beatonsweep() function.

2. Create the shared object that R will use to access the function. Call this shared object "bsweep.dll". To create this shared object, do the following:

   (a) Install rtools.exe (see http://www.murdoch-sutherland.com/Rtools/) if not already installed.

   (b) Open Window's command prompt (i.e., cmd.exe) by searching for"cmd" from the start menu. (It seems that you will need to use cmd.exe to create the shared object — cygwin, as far as I can tell, will not work.)

[2]Note that we can then find an estimates for the mean square of our residuals, $\hat{\theta}^2$, since $\hat{\theta}^2 = \frac{RSS}{n-p}$.

(c) Change the evironmental variables (if necessary) for cmd.exe. (By default, the Window's command prompt will not search the appropriate directories to find where R and the other relevent tools are located.)

(d) Place beatonsweep.c and beatonsweepr.c in the working directory of the command prompt.

(e) Type the following into the command prompt to create bsweep.dll:
R CMD SHLIB beatonsweepr.c beatonsweep.c -o bsweep.dll

3. Load the shared object (i.e., bsweep.dll) into an R session by typing the following (if bsweep.dll is located in C:/Users/Carolyn):

   dyn.load('C:/Users/Carolyn/bsweep.dll')

4. Create (and run) an R function definition program (e.g., "bsweep.r"). This is R code that should ensure that the correct types of objects will get passed from R to C, and back.

After following the above procedure, I then got to experience the beauty of C from R. Specifically, I ran a regression using my Beaton sweep algorithm and compared its results with those obtained from R's linear regression function. To do this, I first created an X matrix and Y matrix for testing purposes. Then, using R functions for matrix manipulations, I computed $X^{*1}X^{*}$, as in equation 1. Next, I applied the Beaton sweep algorithm to columns 1 thru p, generating a matrix containing values for the regression coefficients and for RSS (as in equation 2). With just a little bit of manipulation, I was then able to uncover the standard errors of $\hat{\beta}$. To my great relief, I found that my regression results using Beaton sweep were identical to those obtained using R's lm command (phew!)

4 Non-Linear Regression

I will now discuss how to run a non-linear regression using the Gauss-Newton algorithm with Hartley's Modification. Following this discussion, I will provide a breif overview of my program that implements this method.

As before, we will assume that n is the number of observations and p is the number of parameters in our regression problem. Our challenge is to minimize the sum of squares (i.e., the sum of the squares of the residuals). That is, we wish to find the value of θ (which is a vector with p values) that minimizes:

$$F(\theta) = \frac{1}{2}\sum_{k=1}^{n}(y_k - h(x_k, \theta))^2 \tag{3}$$

where h is some function that we will assume is known (i.e., will be provided). Well, it turns out that there may be no closed-form solution to this problem. Instead, we will use an iterative technique to identify $\tilde{\theta}$ – the optimal value of θ.

If we define the residuals, r_i, for each observation i as

$$r_i = y_i - h(x_i, \theta) \tag{4}$$

then our challenge can be described as one in which we wish to minimize $r'r$. To develop a method for minimizing $r'r$, it is a good idea to first define the nxp Jacobian matrix, J, as

$$\left(J(\tilde{\theta}_{n\times p})\right)_{kj} = \frac{\partial F^{(k)}(\tilde{\theta})}{\partial \tilde{\theta}_j} \tag{5}$$

After a bunch of matrix manipulations and calculus, we find that we can estimate $\tilde{\theta}$ through an iterative process in which we calculate a new value, $\tilde{\theta}_{i+1}$, according to the following equation:

$$\tilde{\theta}_{i+1} = \tilde{\theta}_i - (J'J)^{-1}J'r \tag{6}$$

With this method, we would continue to calculate new values for $\tilde{\theta}$ until convergence is reached according to some pre-established stopping rule.

A simple stopping rule could consist of stopping when $|F(\tilde{\theta}_i) - F(\tilde{\theta}_{i-1})| < \epsilon$ for m successive iterations. However, this stopping rule can be fooled. A better rule is to stop when

$$|\tilde{\theta}_{ij} - \tilde{\theta}_{i+1,j}| \leq \epsilon_1 \cdot (|\tilde{\theta}_{ij}| + \epsilon_2) \qquad for \qquad j = 1, ..., p \tag{7}$$

where

$$\epsilon_1 = \sqrt{\text{machine epsilon}} \qquad and \qquad \epsilon_2 = 10\epsilon_1$$

For simple models, equation 6, along with a sensible stopping rule, is sufficient for finding $\tilde{\theta}$. However, we must develop a method for dealing with situations in which divergence occurs (i.e., when the sum of squared residuals is larger for $\tilde{\theta}_{i+1}$ than it is for $\tilde{\theta}_i$). One such method is called the "Hartley's Modification" method. This method involves "step-halving," which means that we would calculate the next value, $\tilde{\theta}_i$ as:

$$\tilde{\theta}_{i+1} = \tilde{\theta}_i - s(J'J)^{-1}J'r \tag{8}$$

where s represents the length of the "shift vector," and may be successively halved until the new value, $\tilde{\theta}_{i+1}$, generates an RSS lower than $\tilde{\theta}_i$ generates.

I programmed the method, described above, for running non-linear regressions. Note that there are multiple ways to find $(J'J)^{-1}J'r$ in equation 6. I chose to think of this term as the resulting $\hat{\beta}$ from running a linear regression in which J stands in for the X matrix and r acts as the Y vector. I was therefore able to use the Gram-Schmidt linear regression program that I wrote for assignment 3 to find $(J'J)^{-1}J'r$.

Finally, note that the Gauss-Newton algorithm requires starting values for $\tilde{\theta}$. I obtained reasonable starting values by using R to conduct what is known as a "grid search." This procedure involved evaluating the function at many different parameter value combinations, and choosing the parameter combination that minimized the sum of squared residuals. (Please see R code for details.)

I tested my non-linear regression code using data files prob1.txt and prob2.txt, taken from the s243 samples directory.

5 Conclusion

This assignment was a challenging culmination of the semester's material. I employed some of my old tricks, developed during previous assignments, while learning new ones, such as how to run C code from R, and how to pass a function as a parameter to another function. I hope that my statistical programming education will not end here, for as much as I learned this semester, there is still much, much more that I do not know and would like to know.