Introduction to the R Statistical Computing Environment
R Programming: Exercises

John Fox
(McMaster University)
ICPSR Summer Program

2010

1. *A challenging problem:* Iterated weighted least squares (IWLS) is a standard method of fitting generalized linear models to data. As described in Section 5.12 of the *R Companion*, the IWLS algorithm applied to binomial logistic regression proceeds as follows:

   (a) Set the regression coefficients to initial values, such as $\boldsymbol{\beta}^{(0)} = \mathbf{0}$ (where the superscript 0 indicates start values).

   (b) At each iteration t calculate the current fitted probabilities $\boldsymbol{\mu}$, variance-function values $\boldsymbol{\nu}$, working-response values $\mathbf{z}$, and weights $\mathbf{w}$:

   $$\begin{aligned}\mu_i^{(t)} &= [1+\exp(-\eta_i^{(t)})]^{-1}\\ v_i^{(t)} &= \mu_i^{(t)}(1-\mu_i^{(t)})\\ z_i^{(t)} &= \eta_i^{(t)} + (y_i-\mu_i^{(t)})/v_i^{(t)}\\ w_i^{(t)} &= n_i v_i\end{aligned}$$

   Here, n_i represents the binomial denominator for the ith observation; for binary data, all of the n_i are 1.

   (c) Regress the working response on the predictors by weighted least squares, minimizing the weighted residual sum of squares

   $$\sum_{i=1}^{n} w_i^{(t)}(z_i^{(t)} - \mathbf{x}_i'\boldsymbol{\beta})^2$$

   where $\mathbf{x}_i'$ is the ith row of the model matrix.

   (d) Repeat steps 2 and 3 until the regression coefficients stabilize at the maximum-likelihood estimator $\widehat{\boldsymbol{\beta}}$.

   (e) Calculate the estimated asymptotic covariance matrix of the coefficients as

   $$\widehat{\mathcal{V}}(\widehat{\boldsymbol{\beta}}) = (\mathbf{X}'\mathbf{W}\mathbf{X})^{-1}$$

   where $\mathbf{W} = \text{diag}\{w_i\}$ is the diagonal matrix of weights from the last iteration and $\mathbf{X}$ is the model matrix.

Problem: Program this method in R. The function that you define should take (at least) three arguments: The model matrix `X`; the response vector of observed proportions `y`; and the vector of binomial denominators `n`. I suggest that you let `n` default to a vector of `1`s (i.e., for binary data, where `y` consists of `0`s and `1`s), and that you attach a column of `1`s to the model matrix for the regression constant so that the user does not have to do this when the function is called.

Programming hints:

- Adapt the structure of the example developed in Section 8.5.1 of the *R Companion* (but note that the example is for *binary* logistic regression, while the current exercise is to program the more general *binomial* logit model).
- Use the `lsfit` function to get the weighted-least-squares fit, calling the function as `lsfit(X, z, w, intercept=FALSE)`, where `X` is the model matrix; `z` is the current working response; and `w` is the current weight vector. The argument `intercept=FALSE` is needed because the model matrix already has a column of `1`s. The function `lsfit` returns a list, with element `$coef` containing the regression coefficients. See `help(lsfit)` for details.
- One tricky point is that `lsfit` requires that the weights (`w`) be a *vector*, while your calculation will probably produce a *one-column matrix* of weights. You can coerce the weights to a vector using the function `as.vector`.
- Return a list with the maximum-likelihood estimates of the coefficients, the covariance matrix of the coefficients, and the number of iterations required.
- You can test your function on the `Mroz` data in the `car` package, being careful to make all of the variables numeric (as in Section 8.5.1). You might also try fitting a binomial (as opposed to binary) logit model.

2. *Another challenging problem (though perhaps somewhat less so):* A matrix is said to be in *reduced row-echelon form* when it satisfies the following criteria:

   (a) All of its nonzero rows (if any) precede all of its zero rows (if any).

   (b) The first entry (from left to right) — called the *leading entry* — in each nonzero row is 1.

   (c) The leading entry in each nonzero row after the first is to the right of the leading entry in the previous row.

   (d) All other entries are 0 in a column containing a leading entry.

   A matrix can be put into reduced row echelon form by a sequence of *elementary row operations*, which are of three types:

   (a) Multiply each entry in a row by a nonzero constant.

   (b) Add a multiple of one row to another, replacing the other row.

   (c) Exchange two rows.

   Gaussian elimination is a method for transforming a matrix to reduced row-echelon form by elementary row operations. Starting at the first row and first column of the matrix, and proceeding down and to the right:

(a) If there is a 0 in the current row and column (called the *pivot*), if possible exchange for a lower row to bring a nonzero element into the pivot position; if there is no nonzero pivot available, move to the right and repeat this step. If there are no nonzero elements anywhere to the right (and below), then stop.

(b) Divide the current row by the pivot, putting a 1 in the pivot position.

(c) Proceeding through the other rows of the matrix, multiply the pivot row by the element in the pivot column in another row, subtracting the result from the other row; this zeroes out the pivot column.

Consider the following example:

$$\begin{bmatrix} -2 & 0 & -1 & 2 \\ 4 & 0 & 1 & 0 \\ 6 & 0 & 1 & 2 \end{bmatrix}$$

Divide row 1 by -2:

$$\begin{bmatrix} 1 & 0 & 0.5 & -1 \\ 4 & 0 & 1 & 0 \\ 6 & 0 & 1 & 2 \end{bmatrix}$$

Subtract 4 $\times$ row 1 from row 2:

$$\begin{bmatrix} 1 & 0 & 0.5 & -1 \\ 0 & 0 & -1 & 4 \\ 6 & 0 & 1 & 2 \end{bmatrix}$$

Subtract 6 $\times$ row 1 from row 3:

$$\begin{bmatrix} 1 & 0 & 0.5 & -1 \\ 0 & 0 & -1 & 4 \\ 0 & 0 & -2 & 8 \end{bmatrix}$$

Multiply row 2 by -1:

$$\begin{bmatrix} 1 & 0 & 0.5 & -1 \\ 0 & 0 & 1 & -4 \\ 0 & 0 & -2 & 8 \end{bmatrix}$$

Subtract 0.5 $\times$ row 2 from row 1:

$$\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & -4 \\ 0 & 0 & -2 & 8 \end{bmatrix}$$

Add 2 $\times$ row 2 to row 3:

$$\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & -4 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

The matrix is now in reduced row-echelon form. The rank of a matrix is the number of nonzero rows in its reduced row-echelon form, and so the matrix in this example is of rank 2.

Problem: Write an R function to calculate the reduced row-echelon form of a matrix by elimination.

Programming hints:

- When you do “floating-point” arithmetic on a computer, there are almost always rounding errors. One consequence is that you cannot rely on a number being exactly equal to a value such as 0. When you test that an element, say x, is 0, therefore, you should do so within a tolerance — e.g., $|x| < 1 \times 10^{-6}$.
- The computations tend to be more accurate if the absolute values of the pivots are as large as possible. Consequently, you can exchange a row for a lower one to get a larger pivot even if the element in the pivot position is nonzero.

3. *A less difficult problem:* Write a function to compute *running medians*. Running medians are a simple smoothing method usually applied to time-series. For example, for the numbers 7, 5, 2, 8, 5, 5, 9, 4, 7, 8, the running medians of length 3 are 5, 5, 5, 5, 5, 5, 7, 7. The first running median is the median of the three numbers 7, 5, and 2; the second running median is the median of 5, 2, and 8; and so on. Your function should take two arguments: the data (say, `x`), and the number of observations for each median (say, `length`). Notice that there are fewer running medians than observations. How many fewer?

4. *Another less difficult problem:* In response to a question on the `r-help` email list, the following function (slightly edited here) was posted:

```
riffle <- function (a, b) {
    # Interleave a and b, starting with a, without repeating
    x <- NULL
    count <- 1
    for (i in 1:max(length(a), length(b))) {
        if (i <= length(a)) {
            x[count] <- a[i]
            count <- count + 1
        }
        if (i <= length(b)) {
            x[count] <- b[i]
            count <- count + 1
        }
    }
    x
}
```

This function works as follows:

```
> riffle(1:10, 50:55)
 [1]  1 50  2 51  3 52  4 53  5 54  6 55  7  8  9 10
```

Write a similar function but without using a loop. Try to measure which function is more efficient for very large vectors. Can you improve the efficiency of the function employing the loop without eliminating the loop? Which function is easier to understand?

5. *Debugging Functions:* Here are "solutions" to the first three programming problems, but each function has a bug (or two) that either causes it to fail or, possibly only in certain circumstances, to give the wrong answer. In each case, find the bug(s) and fix the function. A file with the bugged functions is available for download from the course web site.

(a) A function to calculate logistic-regression estimates by iteratively reweighted least-squares:

```
lregIWLS <- function(X, y, n=rep(1,length(y)), maxIter=10, tol=1E-6){ # bugged!
    # X is the model matrix
    # y is the response vector of observed proportion
    # n is the vector of binomial counts
    # maxIter is the maximum number of iterations
    # tol is a convergence criterion
    X <- cbind(1, X)  # add constant
    b <- bLast <- rep(0, ncol(X))  # initialize
    it <- 1  # iteration index
    while (it <= maxIter){
        if (max(abs(b - bLast)/(abs(bLast) + 0.01*tol)) < tol)
            break
        eta <- X %*% b
        mu <- 1/(1 + exp(-eta))
        nu <- as.vector(mu*(1 - mu))
        w <- n*nu
        z <- eta + (y - mu)/nu
        b <- lsfit(X, z, w, intercept=FALSE)$coef
        bLast <- b
        it <- it + 1  # increment index
        }
    if (it > maxIter) warning('maximum iterations exceeded')
    Vb <- solve(t(X) %*% diag(w) %*% X)
    list(coefficients=b, var=Vb, iterations=it)
    }
```

(b) A function to compute the reduced row-echelon form of a matrix by Gaussian elimination:

```
reducedRowEchelonForm <- function(A){
    n <- nrow(A)
    m <- ncol(A)
    i <- j <- 1
    while (i <= n && j <= m){
        while (j <= m){
            currentColumn <- A[,j]
            currentColumn[1:n < i] <- 0
            # find maximum pivot in current column at or below current row
            which <- which.max(abs(currentColumn))
            pivot <- currentColumn[which]
            if (pivot == 0) { # check for 0 pivot
                j <- j + 1
                next
            }
        if (which > i) A[c(i, which),] <- A[c(which, i),]  # exchange rows
        A[i,] <- A[i,]/pivot            # pivot
        row <- A[i,]
        A <- A - outer(A[,j], row)      # sweep
        A[i,] <- row                    # restore current row
        j <- j + 1
        break
        }
    i <- i + 1
    }
 # 0 rows to bottom
zeros <- which(apply(A[,1:m], 1, function(x) all(x == 0)))
if (length(zeros) > 0){
    zeroRows <- A[zeros,]
    A <- A[-zeros,]
    A <- rbind(A, zeroRows)
    rownames(A) <- NULL
    }
A
}
```

Note that this function returns the right answer for the matrix used as an example in Problem 2,

$$\begin{bmatrix} -2 & 0 & -1 & 2 \\ 4 & 0 & 1 & 0 \\ 6 & 0 & 1 & 2 \end{bmatrix}$$

whose reduced row-echelon form is

$$\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & -4 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

but gives the wrong answer for the matrix

$$\begin{bmatrix} 16 & 2 & 3 & 13 \\ 5 & 11 & 10 & 8 \\ 9 & 7 & 6 & 12 \\ 4 & 14 & 15 & 1 \end{bmatrix}$$

whose correct row-echelon form is

$$\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & -3 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

(c) A function to calculate running medians:

```
runningMedian <- function(x, length=3){   # bugged!
#   x: a numeric vector
#   length: the number of values for each running median, defaults to 3
n <- length(x)
X <- matrix(x, n, length)
for (i in 1:length) X[1:(n - i + 1), i] <- x[-(1:(i - 1))]
apply(X, 1, median)[1:(n - length + 1)]
}
```

6. *Object-Oriented Programming:* Recall the function `lreg.default`, which returns an object of class `lreg`.

   (a) Write methods for the generic functions `coef`, `vcov`, and `deviance`, to return the logistic-regression coefficients, their covariance matrix, and the residual deviance for objects of this class.

   (b) More ambitiously, write a method for the generic function `anova` to compare two objects of class `lreg` by a likelihood-ratio test, supposing that one object represents a model that is properly nested within the other. Allow the larger model to be given either first or second, and try to check that the models are in fact nested.

   (c) Convert your logistic-regression function from Exercise 1 into an S3 generic function with `default` and `formula` methods.