Introduction to the R Statistical Computing Environment
R Programming: Exercises 3

John Fox
(McMaster University)
ICPSR

2018

1. *Debugging Functions:* A file with the bugged functions given below is available for download from the lecture-series web site. Use the debugging tools in R and RStudio to locate and fix the errors.

(a) If one of your own functions written for an earlier programming exercise still doesn't work properly, try to debug it.

(b) A function to calculate running medians (from problem 1 in Programming Exercises 1):

```
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)]
    }
```

(c) A bugged version of the `fib1` function for computing Fibonnaci numbers (from problem 3 in Programming Exercises 1):

```
fib1_bugged <- function(n){  # bugged!
    if (n <= 2) return(1)
    last.minus.1 <- 1
    last.minus.2 <- 1
    for (i in 3:n){
        last.minus.1 <- last.minus.1 + last.minus.2
        last.minus.2 <- last.minus.1
    }
    last.minus.1
}
```

(d) A function to calculate binary logistic-regression estimates by the Newton-Raphson algorithm (from problem 3 in Programming Exercises 2).

```
lregNR <- function(X, y, max.iter=10, tol=1E-6, verbose=FALSE){ # bugged
    # X is the model matrix
    # y is the response vector of 0s and 1s
    # max.iter is the maximum number of iterations
    # tol is a convergence criterion
    # verbose: show iteration history?
    X <- cbind(1, X)  # add constant
    b <- previous.b <- rep(0, ncol(X))  # initialize coefficients
    it <- 1  # initialize iteration counter
    while (it <= max.iter){
        if (verbose) cat("\niteration = ", it, ": ", b)
        p <- 1/(1 + exp(-X %*% b))
        V <- diag(p * (1 - p))
        var.b <- solve(t(X) %*% V %*% X)
        b <- b + var.b %*% t(X) %*% (y - p)  # update coefficients
        if (max(abs(b - previous.b)/(abs(previous.b) + 0.01*tol)) < tol) break
        previous.b <- b  # update previous coefficients
        it <- it + 1  # increment counter
    }
    if (verbose) cat("\n")  # newline
    if (it > max.iter) warning("maximum iterations exceeded")
    list(coefficients=as.vector(b), var=var.b, iterations=it)
}
```

To test this function (and the one in the next problem), you can use the following commands:

```
library(car) # for Mroz data set
Mroz$lfp <- with(Mroz, ifelse(lfp == "yes", 1, 0)) # create 0/1 dummy variables
Mroz$wc <- with(Mroz, ifelse(wc == "yes", 1, 0))
Mroz$hc <- with(Mroz, ifelse(hc == "yes", 1, 0))
mod.mroz <- with(Mroz, lregNR(cbind(k5, k618, age, wc, hc, lwg, inc), lfp))
```

(e) A function to calculate binomial logistic-regression estimates by iteratively reweighted least-squares (from problem 4 in Programming Exercises 2) :

```
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)
    }
```

2. *Profiling Functions:* Profile your recursive Fibonacci function `fib0` (from Problem 3 in Programming Exercises 1) to figure out why it takes so long to compute large Fibonacci numbers. *Hard:* Can you figure out how to improve the efficiency of the function dramatically while still doing the computation recursively?