

# Lab 2: Flow Control and the Urban Economy – SOLUTIONS

36-350, Statistical Computing

Friday, 9 September 2011

1. Download the file, use `setwd()` to change the directory, and then

```
gmp <- read.table("gmp.dat")
```

The latest versions of R will manage the download for you:

```
gmp <- read.table("http://www.stat.cmu.edu/~cshalizi/statcomp/labs/02/gmp.dat")
```

2. 

```
pop <- gmp$gmp/gmp$pcgmp  
# or  
pop <- gmp[,2]/gmp[,3]
```

3. 

```
gmp <- data.frame(gmp,pop=pop) # or even data.frame(gmp,pop)  
# or  
gmp <- cbind(gmp,pop)
```

Use `colnames(gmp)` to check that these have correctly guessed the names of the columns from those of the variables being bound together (except for the very first command, which explicitly gave the name of the final column).

4. 

```
y0 <- 6611 # Not strictly necessary, could just have 6611 as a magic constant  
mean((gmp$pcgmp - y0*(gmp$pop^0.15))^2) # == 207057513
```
5. 

```
> a.sequence <- seq(from=0.10,to=0.15,by=0.005)  
> a.sequence # check the result:  
[1] 0.100 0.105 0.110 0.115 0.120 0.125 0.130 0.135 0.140 0.145 0.150
```
6. One way, using names and iterating directly along `a.sequence`:

```
mses <- vector(length=length(a.sequence))  
names(mses) = a.sequence  
for (a in a.sequence) {  
  mses[as.character(a)] <- mean((gmp$pcgmp - y0*(gmp$pop^a))^2)  
}
```

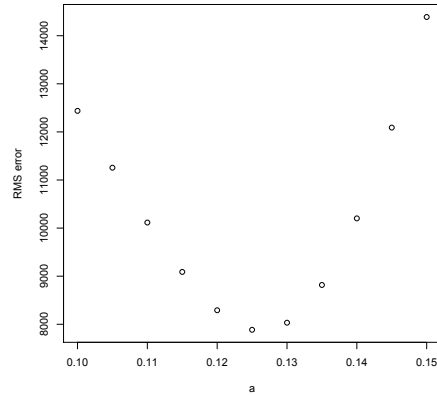


Figure 1: RMS error as a function of the parameter  $a$ , for selected  $a$ .

Another, using numerical indices:

```
mSES <- vector(length=length(a.sequence))
for (i in 1:length(a.sequence)) {
  mSES[i] <- mean((gmp$pcgmp - y0*(gmp$pop^a.sequence[i]))^2)
}
```

Yet another, using functions. We have not seen how to define our own functions yet, but we will next week, and then you may want to refer back to this:

```
powerlaw.mse <- function(a) { mean((gmp$pcgmp - y0*gmp$pop^a)^2) }
mSES <- sapply(a.sequence, powerlaw.mse)
```

7. `plot(x=a.sequence, y=sqrt(mSES), xlab="a", ylab="RMS error")`

See Figure 1.

The units of the RMS error are dollars per person (per year). The smallest error is obtained, visually, at  $a = 0.125$ ; this can be checked using `which.min()`, which gives the index of the smallest value in a vector:

```
> a.sequence[which.min(mSES)]
[1] 0.125
```

8. The code starts from an initial value of  $a$ , here 0.15, and finds the (approximate) derivative of the MSE with respect to  $a$ , by evaluating

$$\frac{MSE(a+h) - MSE(a)}{h}$$

where  $h$  is `deriv.step`, set to  $10^{-3}$ . It then subtracts this derivative, times `step.scale`, from  $a$  — this increases  $a$  when the derivative is negative and decreases it when it is positive, moving it in either case towards the smaller value of the MSE. It takes bigger steps when the magnitude of the derivative is large, and would do nothing if it was exactly zero. (Remember that the derivative is usually zero at a minimum.) The code stops either when it has made more than `maximum.iterations` ( $= 100$ ) steps, or when the derivative gets small ( $\leq 10^{-2}$ ). The derivative is initially set to infinity so that the `while` loop is executed the first time. When the loop ends, the code prepares a list which contains the file value of  $a$ , the number of iterations, and an indicator for whether it had converged or not.

This is a simple implementation of the minimization method called “gradient descent”.

EXERCISE: First re-write this as a `for` loop, and then remove the fiction of setting the derivative to infinity initially.

9. After running the code:

```
$a
[1] 0.1258166
```

```
$iterations
[1] 58
```

```
$converged
[1] TRUE
```

So the estimated minimizing value of  $a = 0.1258$ , at which point the RMS error is

```
> sqrt(mean((gmp$pcgmp-y0*gmp$pop^0.1258166)^2))
[1] 7867.909
```

or \$7868. Recall that by visually inspecting the plot we anticipated a minimum at  $a = 0.125$ , where the RMS error was \$7885. This matches very well, which is a good sanity check for our code.

10. Re-running the code exactly as before, except editing  $a$ , just before the loop, to exactly the old value, yields the output

```
$a
[1] 0.1258166
```

```
$iterations
[1] 1
```

```
$converged
[1] TRUE
```

This is because it enters the loop once, takes the numerical derivative, finds it smaller than the threshold, and stops. This value of  $a$  is in fact very slightly different from the one we started with (by about `stopping.deriv`  $\times$  `step.scale` =  $10^{-14}$ ).

Note that because the MSE changes very rapidly around the minimum, while `stopping.deriv` is quite small, even rounding to a few decimal points can give a derivative which is above threshold, and send us off for a few iterations. The stopping point, however, always ends up agreeing to the displayed precision of seven decimal places.