Week 3 exercises

Brad McNeney

2019-01-17

More on matrices *vs* data frames

1. The following simulation function simulates n replicates of an explanatory variable X and a response variable $Y = \beta X + E$, where β is a regression coefficient between -1 and 1 and $E \sim N(0, 1)$ is random noise. Run the code chunk and then use the function to simulate one dataset of size $n = 1000$ and save the result in an object called `dd`.

```
simdat <- function(n) {
  beta <- runif(1,min=-1,max=1)
  x <- rnorm(n)
  y <- beta * x + rnorm(n)
  data.frame(x=x,y=y)
}
######3
dd <- simdat(1000)
```

2. Create a larger dataset by calling `simdat()` `N=500` times over and stacking the results. The larger dataset should have `500*1000` rows and 2 columns. Call your stacked dataset `bigd1`. To create the stacked dataset, initialize with `bigd1 <- NULL` and use a `for` loop to build up `bigd1` one layer at a time. Time this code using the `system.time()` function. An example use of `system.time()` to time an R command, e.g., `x <- rnorm(100000)` is:

```
system.time({
  x <- rnorm(100000) # Could put multiple lines of R code here
})
```

```
##    user  system elapsed
##   0.014   0.001   0.022
```

Use the first element of the output (`user` time) as your measure of execution time.

```
bigd1 <- NULL
for(i in range(500)){
  bigd1 <- append(bigd1,simdat(1000))
}
```

3. Repeat 2, but this time, instead of stacking the output of `simdat()`, coerce the output of `simdat()` to a matrix, and stack the matrices. Use `system.time()` to time your code and compare the timing from question (2).

4. Now build `bigd2` by (i) initializing an empty matrix of appropriate dimension, and (ii) looping 500 times and inserting simulated datasets of size $n = 1000$, coerced to matrices, into successive layers of `bigd2`. Time this code and compare the timing to that of part (3). You may find the following R function useful:

```r
layerInds <- function(layerNum,nrow) {
  ((layerNum-1)*nrow + 1):(layerNum*nrow)
}
# Example use:
inds <- layerInds(layer=1,nrow=1000)
range(inds)
```

```
## [1]    1 1000
```

Control flow

1. What type of vector does each of the following return?

```r
ifelse(TRUE, 1, "no")
```

```
## [1] 1
```

```r
ifelse(FALSE, 1, "no")
```

```
## [1] "no"
```

```r
ifelse(NA, 1, "no")
```

```
## [1] NA
```

2. Re-write the following using `switch`

```r
IQR_mid <- function(x) mean(quantile(x,c(.25,.75)))
cc <- function(x,method) {
  if(method=="mean") {
    mean(x)
  } else if(method=="median") {
    median(x)
  } else if(method=="IQR_mid") {
    IQR_mid(x)
  } else stop("centring method ",method," not implemented")
}
set.seed(123)
x <- c(-3,rnorm(100),1000)
cc(x,"mean")
```

```
## [1] 9.863143
```

```r
cc(x,"median")
```

```
## [1] 0.06175631
```

```r
cc(x,"IQR_mid")
```

```
## [1] 0.0993383
```

```r
try(cc(x,"cat"))
```

```
## Error in cc(x, "cat") : centring method cat not implemented
```

3. Rewrite the following function so that it uses a `while()` loop instead of the `for()` loop and `break` statement. Your while-approach will not require the `maxit` upper limit on the number of iterations.

```r
rtruncNormal <- function(thresh = 2, maxit=1000) {
  x<-NULL
  for(i in 1:maxit) {
    xnew <- rnorm(n=1)
    if(xnew>thresh) {
      break
    }
    x <- c(x,xnew)
  }
  x
}
set.seed(1234)
rtruncNormal()
```

```
##  [1] -1.20706575  0.27742924  1.08444118 -2.34569770  0.42912469  0.50605589
##  [7] -0.57473996 -0.54663186 -0.56445200 -0.89003783 -0.47719270 -0.99838644
## [13] -0.77625389  0.06445882  0.95949406 -0.11028549 -0.51100951 -0.91119542
## [19] -0.83717168
```

Functions

4. The following code chunk is typed into the R Console.

   - What is the output of the function call `f(5)`?
   - What is the enclosing environment of `f()`?
   - What is the enclosing environment of `g()`?
   - What search order does R use to find the value of `x` when it is needed in `g()`?

```r
x <- 1
f <- function(y) {
  g <- function(z) {
 (x+z)^2
  }
  g(y)
}
f(5)
```