Debugging in R

Jared Bennett

29 September, 2019

Debugging Tutorial

Learning Objectives

- How to get help online
- Common errors
- General habits for debugging
- `R` specific methods for debugging

0) Useful Links

- General advice for debugging
  - Efficient Debugging by Goldspink
  - Debugging for Beginners by Brody
- R specific debugging
  - Advanced R by Wickham
  - Debugging in Rstudio by Gadrow
- Material for this document
  - berkeley-scf by Paciorek

1) Getting Help Online

1.1) Mailing Lists and Online Forums

There are several mailing lists that have lots of useful postings. In general if you have an error, others have already posted about it.

- Simple web searches - *a la Google*
  - You may want to include "in R" or preface your question with "R yada yada yada"
- Stack overflow: R stuff will be tagged with 'R'
  - http://stackoverflow.com/questions/tagged/r
- R help special interest groups (SIG) such as r-sig-hpc (high performance computing), r-sig-mac (R on Macs), etc.
  - To search a SIG you might include the name of the SIG in the search string
- Rseek.org for web searches restricted to sites that have information on R
- R help: R mailing lists archive

Note: of course the various mailing lists are also helpful for figuring out how to do things, not just for fixing bugs. For example, this blog post has a guide to R based simply on Stack Overflow posts.

1.2) Asking Questions Online

If you've searched the archive and haven't found an answer to your problem, you can often get help by posting to the R-help mailing list or one of the other lists mentioned above. A few guidelines (generally relevant when posting to mailing lists beyond just the R lists):

- Search the archives and look through relevant R books or manuals first.
  - Advanced R by Hadley Wickham
- Boil your problem down to the essence of the problem, giving an example, including the output and error message
  - My first SO post
    - Notice the not-so-polite comments, see the remark below
  - My second SO question
- Say what version of R, what operating system and what operating system version you're using.
  - Provide `sessionInfo()` and `Sys.info()`. These show the current state of your machine
- Read the R mailing list posting guide.

The R mailing lists are a way to get free advice from the experts, who include some of the world's most knowledgeable R experts - seriously - members of the R core development team contribute frequently. The cost is that you should do your homework and that sometimes the responses you get **may be blunt**, along the lines of "read the manual". Chris considers it a pretty good tradeoff - where else do you get the foremost experts in a domain actually helping you?

2) Common Errors

- Parenthesis mis-matches
- `[[...]]` vs. `[...]`

```r
# example list
myList <- list("A"=1:10,
               "B"=11:20)

# one set
cat("Type: ", typeof(myList[1]), "\nLength: ", length(myList[1]), sep = "")
```

```
## Type: list
## Length: 1
```

```r
# two sets
cat("Type: ", typeof(myList[[1]]), "\nLength: ", length(myList[[1]]), sep = "")
```

```
## Type: integer
## Length: 10
```

- `==` vs. `=`
- Comparing real numbers exactly using `==` is dangerous because numbers on a computer are only represented to limited numerical precision.

```r
# exact comparison
1/3 == 4*(4/12 - 3/12)
```

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

```r
# approximate comparison
#  default tolerance is sqrt(.Machine$double.eps)
all.equal(target = 1/3 ,current = 4*(4/12 - 3/12))
```

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

- You expect a single value but execution of the code gives a vector

- You want to compare an entire vector but your code just compares the first value (e.g., in an if statement)
  - consider using `identical()` or `all.equal()`
- Silent type conversion when you don't want it, or lack of coercion where you're expecting it
  - eg., `read.csv()` and the `stringsAsFactors` argument
- Using the wrong function or variable name
- Giving unnamed arguments to a function in the wrong order
- In an if-else statement, the `else` cannot be on its own line (unless all the code is enclosed in `{}`) because R will see the `if` part of the statement, which is a valid R statement, will execute that, and then will encounter the `else` and return an error.
- Forgetting to define a variable in the environment of a function and having R, via lexical scoping, get that variable as a global variable from one of the enclosing environments. At best the types are not compatible and you get an error; at worst, you use a garbage value and the bug is hard to trace. In some cases your code may work fine when you develop the code (if the variable exists in the enclosing environment), but then may not work when you restart R if the variable no longer exists or is different.
  - Clear your environment before testing (`rm(list=ls());gc()`)
  - Restart `R` session and test
- R (usually helpfully) drops matrix and array dimensions that are extraneous. This can sometimes confuse later code that expects an object of a certain dimension.

```r
# 3x3 matrix
myMat <- matrix(data = 1:9, nrow = 3, ncol = 3)

# lost dimensions
dim(myMat[1, ])
```

```
## NULL
```

```r
# keep dimensions
dim(myMat[1, , drop = FALSE])
```

```
## [1] 1 3
```

3) `R`'s Debugging Tools

3.1) Tools

- `browser()`: pauses current execution, provides an interactive interpreter. You can now step through a function line-by-line to find errors.
- `debug(someFunc)`: sets a `browser()` statement at the first line of `someFunc`
  - `undebug(someFunc)` removes the `debug()` statement. Or close the `R` session
  - `debugonce(someFunc)` lets you debug only once, no need to run `undebug()`
- `options(error = recover)`: invokes a browser whenever an error occurs
- `trace()`: allows you to temporarily modify a function without saving the modifications

3.2) Example of Debugging

Code found here

```r
# Need MASS for cats data
# comes installed
#library(MASS)

gamma_est <- function(data) {
  # this fits a gamma distribution to a collection of numbers
  m <- mean(data)
  v <- var(data)
  s <- v/m
  a <- m/s
  return(list(a=a,s=s))
}

calc_var <- function(estimates){
  var_of_ests <- apply(estimates, 2, var)
  return(((n-1)^2/n)*var_of_ests)
}

gamma_jackknife <- function(data) {
  ## jackknife the estimation

  n <- length(data)
  jack_estimates = gamma_est(data[-1])
  for (omitted_point in 2:n) {
    jack_estimates = rbind(jack_estimates, gamma_est(data[-omitted_point]))
  }

  jack_var = calc_var(jack_estimates)

  return(sqrt(jack_var))
}

# jackknife gamma dist. estimates of cat heart weights
gamma_jackknife(MASS::cats$Hwt)
```

3.3) A More Involved Example: `logitBoot()`

- Load `data.csv`
- Fit a logistic regression model `y~x` called `mod` in the script provided.
- What is the std of the coefficient of `x` from `summary(mod)`?
- Now find the estimate of the same parameter now using bootstrap by simply calling `logitBoot()` as provided in the script.
- Why is this estimate so much larger?
- Use debugging tools to figure out the bug.

Code found here
Data found here

```r
myData <- read.csv('./data.csv')

logitBoot <- function(y, x, nBoot = 500) {
  set.seed(1)
  out <- sapply(seq_len(nBoot), myglm, y, x)
  boot_se <- sd(out)
  return(boot_se)
}

myglm <- function(i, y, x) {
  n <- length(y)
  ind <- sample(seq_len(n), n, replace = TRUE)
  out <- glm(y[ind]~x[ind], family='binomial')
  return(out$coef[2])
}

mod <- glm(y ~ x, data = myData, family = 'binomial')
summary(mod)
## note that the standard error for the regression coefficient is ~3

logitBoot(myData$y, myData$x)
```

3.3) Example of Defensive Programming

When writing functions, and software more generally, you'll want to warn the user or stop execution when there is an error and exit gracefully, giving the user some idea of what happened. Here are some things to consider:

- check function inputs and warn users if the code will do something they might not expect or makes particular choices
  - eg `assertthat`, `assertr`, and `checkmate` packages
- check inputs to `if` and the ranges in `for` loops
- provide reasonable default arguments
- document the range of valid inputs
- check that the output produced is valid
- stop execution based on checks and give an informative error message
- catch run-time erros using `try()` or `tryCatch()`.

```r
# methods comes installed
# unless you're running Rscript, then idk
#library(methods)
set.seed(0)
nCats <- 30
n <- 100
y <- rnorm(n)
x <- rnorm(n)
cats <- sample(1:nCats, n, replace = TRUE)
data <- data.frame(y, x, cats)

params <- matrix(NA, nrow = nCats, ncol = 2)

for (i in 1:nCats) {
  sub <- data[data$cats == i, ]
  fit <- try(lm(y ~ x, data = sub))
  if (!inherits(fit, "try-error"))
    params[i, ] = fit$coef
}
```

```
## Error in lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
```

```
##   0 (non-NA) cases
```

```
# view params
params
```

```
##                [,1]        [,2]
##  [1,] -0.4358199953  0.27748127
##  [2,]  1.1581183671  0.80912437
##  [3,] -0.3554216408  0.47883210
##  [4,]  0.1557150115  0.48844740
##  [5,]  1.3686786299 -0.93914607
##  [6,] -0.4253189229  0.83353130
##  [7,]  1.4554922258  1.70845976
##  [8,] -0.3658129573 -0.06813899
##  [9,]  0.0301244577 -0.08146689
## [10,] -0.4686109164 -2.05689759
## [11,]  2.2806852741 -1.66464111
## [12,]  0.2221364736  0.35794243
## [13,]  0.2072138141  0.83705334
## [14,] -0.2273286914          NA
## [15,]            NA          NA
## [16,]  0.5851340251 -0.67609092
## [17,] -0.4604747167 -0.12513299
## [18,]  1.7525542597  2.46613921
## [19,]  0.5986682621 -0.10533300
## [20,]  0.0341767117  0.16544258
## [21,]  0.4234918160 -0.25409147
## [22,] -0.6337088930 -2.37384473
## [23,] -0.3153278059  0.26485741
## [24,] -0.0008991281  1.39578005
## [25,]  0.0091840186  0.04655760
## [26,]  0.8239709421  0.41607748
## [27,] 14.0124163710 -6.91639295
## [28,]  0.2046168245 -0.42460241
## [29,] -0.5619773029 -0.82265332
## [30,] -0.9483964596  0.83635071
```

```
#pull out broken one
foo <- try(lm(y ~ x, data = data[data$cats == 15, ])) # why is this different for me?
```

```
## Error in lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##   0 (non-NA) cases
```

```
foo
```

```
## [1] "Error in lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) : \n  0 (non-NA) cases\n
## attr(,"class")
## [1] "try-error"
## attr(,"condition")
## <simpleError in lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...): 0 (non-NA) cases>
```

```
# check what  it is
class(foo)
```

```
## [1] "try-error"
```

```
inherits(foo, "try-error")
```

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