Computational Skills for Biostatistics I: Lecture 2

Amy Willis, Biostatistics, UW

October 5, 2017

Housekeeping

- The high bar for Homework 1 was met
- Individual comments coming soon via Github Classroom

Pop quiz

What is the distribution of the median of 51 exponentially-distributed random variables with rate $= 1$?

Pop quiz

What is the distribution of the median of 51 exponentially-distributed random variables with rate $= 1$?

- No idea? Me neither!
- How could we use computing power to help us?

Avoiding math with computers

To understand the distribution of the median of 51 exponentially-distributed random variables with rate $= 1$, we can

- Draw 51 Exp(1) random variables, calculate their median
- Do this again, and again, and again...

We can use the collection of medians to calculate summary statistics, draw histograms, do hypothesis testing...

Avoiding math with computers…

… and learning how to write loops in the process

```
simulations <- 10000
many_medians <- rep(NA, simulations)
set.seed(171005)
for (i in 1:simulations) {
  my_sample <- rexp(n = 51, rate = 1)
  many_medians[i] <- median(my_sample)
}
```

Avoiding math with computers

```r
mean(many_medians) # actually: 0.70286
```

```
## [1] 0.7012355
```

```r
var(many_medians) # actually: 0.01978
```

```
## [1] 0.01985761
```

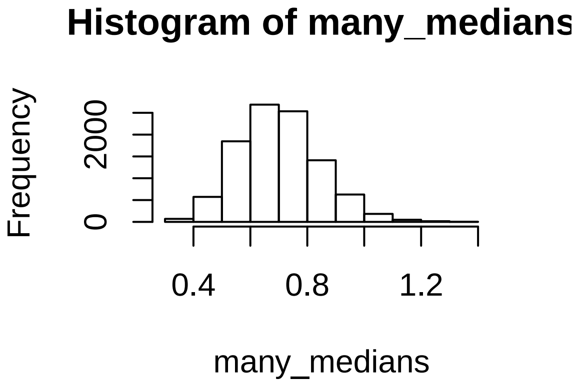
We just calculated the moments of an intractable distribution using computing!

Avoiding math with computers

We could work out almost anything about the sample median in this way!

The distribution of the median of 51 Exp(1) random variables:

```r
hist(many_medians)
```



Reproducible simulations

```r
set.seed(9)
rexp(4)
```

```
## [1] 1.403092 1.479229 1.255778 1.170410
```

```r
rexp(4)
```

```
## [1] 0.337385913 0.005871764 0.897366012 0.971816242
```

```r
set.seed(9)
rexp(4)
```

```
## [1] 1.403092 1.479229 1.255778 1.170410
```

A note on history

Insert funny story about book of random numbers

Structure of a for loop

`for()` loops are not terrible, but watch out:

- First make an empty object of the correct dimension (e.g. vector, matrix, data frame) and *then* fill it in
- Don't forget to store the output of each iteration!
- For large loops and objects, growing the output is a big slowdown
- This is because of the way that memory is handled in R **

A special set up

The only use of the index `i` was for storage.

```
simulations <- 10000
many_medians <- rep(NA, simulations)
set.seed(171005)
for (i in 1:simulations) {
  my_sample <- rexp(n = 51, rate = 1)
  many_medians[i] <- median(my_sample)
}
```

A special set up

Since we are merely doing the same thing again and again, let's use a new function to take care of all of the admin

```
set.seed(171005)
many_medians <- replicate(simulations,
                          median(rexp(n = 51, rate = 1)))
```

The second argument to `replicate()` is the expression you want replicated

Loop indices

The index of our loop (`i`) does not need to be a vector

```
str(airquality) # a built-in dataset
```

```
## 'data.frame':    153 obs. of  6 variables:
##  $ Ozone  : int  41 36 12 18 NA 28 23 19 8 NA ...
##  $ Solar.R: int  190 118 149 313 NA NA 299 99 19 194 ..
##  $ Wind   : num  7.4 8 12.6 11.5 14.3 14.9 8.6 13.8 20.
##  $ Temp   : int  67 72 74 62 56 66 65 59 61 69 ...
##  $ Month  : int  5 5 5 5 5 5 5 5 5 5 ...
##  $ Day    : int  1 2 3 4 5 6 7 8 9 10 ...
```

Loop indices

The index of our loop (`i`) does not need to be a vector

```
for (month in unique(airquality$Month)) {
  print(mean(airquality$Ozone[airquality$Month == month],
             na.rm = TRUE)) # prints but doesn't store
}
```

```
## [1] 23.61538
## [1] 29.44444
## [1] 59.11538
## [1] 59.96154
## [1] 31.44828
```

Loop indices

A better way using by()

```
by(airquality$Ozone, list(month = airquality$Month),
   mean, na.rm = TRUE)
```

```
## month: 5
## [1] 23.61538
## ------------------------------------------------------------
## month: 6
## [1] 29.44444
## ------------------------------------------------------------
## month: 7
## [1] 59.11538
## ------------------------------------------------------------
## month: 8
## [1] 59.96154
## ------------------------------------------------------------
## month: 9
## [1] 31.44828
```

- "Break the data into subsets by month, then calculate the mean Ozone level for each month, omitting missing values"

Looping over subsets: `by()`

```
by(airquality$Ozone, list(month = airquality$Month),
   mean, na.rm = TRUE)
```

- First argument (`data`): variable to be analysed
- Second argument (`INDICES`): list of subsets. Could be multiple variables: `list(month = airquality$Month, toohot = airquality$Temp > 85)`
- Third argument (`FUN`) is the analysis function to use on the subsets
- Any other arguments (e.g. `na.rm=TRUE`) are used as additional arguments to the analysis function

Looping over subsets: by()

- Output is an object of class by, which has its own print method, print.by()
- The implementation of print for objects of class by is kind of annoying: use unclass() to get rid of it

```
ozone_summary <- by(airquality$Ozone,
                    list(month = airquality$Month),
                    mean, na.rm = TRUE)
unclass(ozone_summary) # one option
```

```
## month
##        5        6        7        8        9
## 23.61538 29.44444 59.11538 59.96154 31.44828
## attr(,"call")
## by.default(data = airquality$Ozone, INDICES = list(montl
##     FUN = mean, na.rm = TRUE)
```

Looping over variables: apply()

```r
apply(X=airquality, MARGIN=2, FUN=mean, na.rm=TRUE)
```

```
##      Ozone    Solar.R       Wind       Temp      Month
##  42.129310 185.931507   9.957516  77.882353   6.993464
```

- X: an array, usually a matrix or data frame
- MARGIN: the direction. MARGIN = 1 applies the function to each row, MARGIN = 2 applies the function to each column.
- FUN: the function to be applied
- Any other arguments to be passed to FUN

Looking over variables: apply()

Ad-hoc functions can be defined inline:

```
apply(airquality, 2,
      function(x) { c(mean = mean(x, na.rm = TRUE),
                      sd = sd(x, na.rm = TRUE))})
```

```
##         Ozone   Solar.R     Wind     Temp    Month
## mean 42.12931 185.93151 9.957516 77.88235 6.993464 15.80
## sd   32.98788  90.05842 3.523001  9.46527 1.416522  8.86
```

(but it's generally better to define them externally)

Passing arguments through to other functions

```
mean_and_sd <- function(x, ...) { c(mean = mean(x, ...),
                                    sd = sd(x, ...)) }
apply(airquality, 2, mean_and_sd, na.rm = TRUE)
```

```
##         Ozone   Solar.R     Wind     Temp    Month
## mean 42.12931 185.93151 9.957516 77.88235 6.993464 15.80
## sd   32.98788  90.05842 3.523001  9.46527 1.416522  8.86
```

Debugging code with ellipses can be tricky! Be cautious…

by()-ing more

Applying our own functions using by()

```
by(airquality, list(toohot = airquality$Temp > 85),
   function(subset) { round(apply(subset, 2, mean_and_sd),
                            digits = 2) })
```

```
## toohot: FALSE
##      Ozone Solar.R  Wind  Temp Month   Day
## mean    NA      NA 10.59 74.50  6.83 16.30
## sd      NA      NA  3.41  7.78  1.49  8.58
## ------------------------------------------------------------
## toohot: TRUE
##      Ozone Solar.R Wind  Temp Month   Day
## mean    NA      NA 7.73 89.74  7.56 14.06
## sd      NA      NA 3.01  3.18  0.93  9.74
```

git

- To download all new material to your local copy, go to your materials folder and type `git pull`
  - This will give you lecture 2 and homework 2
- The standard workflow for adding a new file or updating an old one

```
git pull
git add homework2-response.pdf
git commit -a -m 'question 2 part b response'
git push
```

- You must have a git repository set up already to do this (e.g. with `git init` or `git clone ...`)

Coming soon

- Homework 2 due next Thursday at 2 p.m.
    - Submission via github classroom
    - Same instructions as homework 1 – but don't overwrite homework 1!
- Homework 1 feedback coming soon
- Next week: pipe operators!