Prof. David Draper
Department of
Applied Mathematics and Statistics
University of California, Santa Cruz

AMS 132: Discussion Section 2

(All computer operations in this course will be described for the `Windows` operating system (OS); if you use a different OS, you'll need to figure out (with the help of the TA) how to do the same things in your OS environment. All mouse clicks in this course are left-clicks unless otherwise indicated.)

This discussion section involves you working in real time with `R`, so if you haven't already done so:

- On your laptop, point your favorite browser at `https://cran.r-project.org/` .

- `R` is available for 3 OSs): `Linux`, `Mac OS X`, and `Windows`. If you're running some other OS on your laptop, you'll either need to run `R` inside an emulator of one of those 3 OSs or work with another AMS 132 student in your discussion section whose laptop is `R`-compatible.

  Assuming that you can proceed on your machine, click on `Download R for` ______, where ______ is your OS. As noted above, I'll describe the next steps for `Windows`; for `Linux` or `Mac OS X`, get your Discussion Section TA or somebody else in your discussion section who's computer-savvy to help you.

  When you get to the `R for Windows` page, click on `Install R for the first time`. As of this writing, this will take you to the `R-3.3.2 for Windows (32/64 bit)` page, at which you can click on `Download R 3.3.2 for Windows`. In the lower left-corner of your browser a new icon will appear that says `R-3.2.2-win.exe`; wait for the download to complete into this icon, and then left-click once on it. You'll be asked if you want to let the `R` program install itself on your machine; say `Yes`. You'll now be led through a long series of small pop-up windows; you should reply `OK` and/or `Next` whenever it's waiting for you to choose something. Eventually you'll get to a window called `Installing`; `R` will now install, and the installation process will be completed when you get to the window that says `Completing the R for Windows 3.3.2 Setup Wizard`, at which point you click on `Finish` and `R` should now be successfully installed on your machine. On your desktop you should now see two new icons called `Ri386 3.3.2` and `Rx64 3.3.2`; you should use the first of these icons if your laptop only supports 32-bit computing and the second if you're 64-bit compatible. If you don't know which applies to your laptop, double-left-click on the `Rx64` icon; if you're not 64-bit compatible, an error message will occur, but the other icon should work for you.

  Once you've started an `R` session, you'll typically want to resize the window to take up less than 100% of your screen. In the `R Console` window, type `demo( graphics )` and hit `Enter` repeatedly, stopping before each new `Enter` to look at the demo results — this will show you some of the graphical descriptive capabilities in `R`. When the demo is finished, you'll get a nearly blank line that begins with `>`; this is the command prompt in `R`, and you can now begin doing some actual computations.

  Every `R` session ends with the `q( )` command, at which point `R` will ask you whether you want to save the `R` workspace you've created; if you have code that will completely and quickly recreate the entire session (as in this discussion section), you typically don't need to save the workspace. In later discussion sections we'll talk about how to save and restore `R` workspaces.)

Many people feel that the graphical user interface (GUI) supplied by `R` is less than terrific. After having installed `R`, if you want you can go to

`https://www.rstudio.com/products/rstudio/download/`

and download an open-source freeware package called `RStudio` that has a better GUI for `R`; your TA can show you how to do this, and how to use `RStudio`.

In what follows, for simplicity, I'm going to tell you how to run `R` in the same way that I do in class: with an `R` session filling half of the screen and a text file (using `Notepad`) filling the other half, and with you copying and pasting back and forth from one window to the other. (At least one of the TAs will tell you that this is somewhat clumsy and old-school when compared to `RStudio`; he's right.)

1. In the Kaiser Intensive Care Unit (ICU) case study from class, the statistical model was

$$(Y_i \mid \theta) \overset{\text{IID}}{\sim} \text{Bernoulli}(\theta) \quad (\text{for } i = 1, \ldots, n)\,, \tag{1}$$

with $n = 112$ and $\boldsymbol{Y} = (Y_1, \ldots, Y_n)$, and we found it useful to keep track of the sample sum $S = \sum_{i=1}^n Y_i$ and the sample mean $\bar{Y} = \frac{1}{n} \sum_{i=1}^n Y_i$; recall that — in the actual data set $\boldsymbol{y} = (y_1, \ldots, y_n)$ — S took on the value $s = 4$, leading to an observed sample mean of $\bar{y} = \frac{1}{n} \sum_{i=1}^n y_i = \frac{4}{112} \doteq 0.0357$. Since the mean $E(Y_i \mid \theta)$ of a Bernoulli(θ) random variable is θ, we agreed in class that an intuitively reasonable estimate of θ is the sample mean: $\hat{\theta} = \bar{Y}$. Neyman's confidence interval (CI) logic then gave us the large-sample approximate $100(1 - \alpha)\%$ CI

$$\hat{\theta} \pm \Phi^{-1}\Big(1 - \frac{\alpha}{2}\Big) \sqrt{\frac{\hat{\theta}(1 - \hat{\theta})}{n}}\,, \tag{2}$$

in which Φ is the cumulative distribution function (CDF) of the standard normal distribution; for example, for a 95% nominal confidence level, $\alpha = 0.05$ and $\Phi^{-1}(0.975) \doteq 1.96$, as you can verify with the `R` command `qnorm( 0.975 )`.

I wrote some `R` code that suggested that interval (2) was not *well-calibrated* in this problem, by which I mean that the *actual* percentage of time (across repeated-sampling data sets simulated from model (1)) that the interval (2) included the true θ at *nominal* confidence level 95% was less than 95%. We then worked out a possible small-sample improvement to interval (2), by noticing that the repeated-sampling distribution of the sample mean $\hat{\theta}$ with a value of n as small as 112 and a true θ near 0 was not very close to the normal curve suggested by the Central Limit Theorem (CLT: because the Theorem only says what will happen for large n).

One way to understand why the CLT doesn't provide a very good approximation in this case study is to note that θ lives in $(0, 1)$ but the normal curve lives on the entire real line $\mathbb{R}$. This suggested trying to find a *transformed* version of θ that lives on the whole real line, which led us to the *logit* transform $\hat{\eta} \triangleq \log\left(\frac{\hat{\theta}'}{1 - \hat{\theta}'}\right)$, in which $\theta' \triangleq \frac{s + \frac{1}{2}}{n + 1}$ nudges θ away from 0 and 1 (to keep from dividing by 0 or trying to take the log of 0). Using the Delta Method we showed that $\hat{\eta}$ was approximately normally distributed with approximate mean and variance

$$E(\hat{\eta}) \doteq \eta \quad \text{and} \quad V(\hat{\eta}) \doteq \frac{1}{n\, \hat{\theta}'(1 - \hat{\theta}')}\,. \tag{3}$$

Using Neyman's logic on η gives the approximate $100(1-\alpha)\%$ CI

$$\hat{\eta} \pm \Phi^{-1}\Big(1-\frac{\alpha}{2}\Big)\,\widehat{SE}(\hat{\eta}) = \hat{\eta} \pm \frac{\Phi^{-1}\big(1-\frac{\alpha}{2}\big)}{\sqrt{n\,\hat{\theta}'(1-\hat{\theta}')}}\,, \tag{4}$$

and by back-transforming the endpoints of this interval, using the inverse logit transformation

$$\eta = \log\left(\frac{p}{1-p}\right) \quad \text{if and only if} \quad p = \frac{e^{\eta}}{1+e^{\eta}} = \frac{1}{1+e^{-\eta}}\,, \tag{5}$$

we finally got the (hopefully improved) small-sample approximate $100(1-\alpha)\%$ CI

$$\left(\frac{1}{1+e^{-(\eta-A)}}, \frac{1}{1+e^{-(\eta+A)}}\right)\,, \tag{6}$$

in which $A = \frac{\Phi^{-1}\big(1-\frac{\alpha}{2}\big)}{\sqrt{n\,\hat{\theta}'(1-\hat{\theta}')}}$.

I've written some new `R` code that implements the two approximate CIs (2) and (6). Go to the course web page

`https://ams132-winter17-01.courses.soe.ucsc.edu/`

and click on the file called

Kaiser case study calibration R code

Your screen will fill with a text file. Right click inside your browser and you'll get a menu; choose `Save as...` . A new window will appear called `Save As`; on my laptop the default place to save is the `Desktop` (on yours it might be `Documents`, or somewhere else). Pay attention to where the system is about to save this file and click `Save`. Find this text file and double-left-click on it to open it; it will look like the following:

```
################################################################
#
# checking the calibration of the large-sample
# and small-sample intervals in the Kaiser ICU case study
# with a large number of simulation replications,
# to get definitive calibration answers

rm( list = ls( ) ) # this removes everything in the R workspace
                   # so that you can start with a clean slate

N <- 8561          # population size

n <- 112           # sample size

# this is a probability calculation, so in visualizing
# what's going on it's helpful to create the population data set

population <- c( rep( 1, 306 ), rep( 0, 8255 ) )
```

```
print( population.theta <- mean( population ) )

# 0.03574349

n.null <- 0

n.negative.large.sample <- 0

n.negative.small.sample <- 0

n.hit.large.sample <- 0

n.hit.small.sample <- 0

M.calibration.large.sample <- 1000000

seed <- 1

set.seed( seed )

# i ran the code twice, with seed = 1 and seed = 2;
# results are given below

alpha <- 0.05        # confidence level 100 ( 1 - alpha ) %

system.time(

  for ( m in 1:M.calibration.large.sample ) {

    y.star <- sample( population, n, replace = T )

#   another way to do this would be to sample directly
#   from the Bernoulli distribution with the rbinom function
#   (R doesn't have a built-in rbernoulli function, but
#   you can use the fact that a Bernoulli random variable
#   is just a binomial random variable in which only 1
#   success/failure trial occurs): in place of the
#
#     y.star <- sample( population, n, replace = T )
#
#   command above you could just say
#
#   y.star <- rbinom( n, 1, 306 / 8561 )
#
#   (with this approach you don't have to create a
#   population data set at all)
```

```
s.star <- sum( y.star )

if ( s.star == 0 ) {

  n.null <- n.null + 1

}

theta.hat <- s.star / n

se.theta.hat <- sqrt( theta.hat * ( 1 - theta.hat ) / n )

large.sample.confidence.interval <-
  c( theta.hat - qnorm( 1 - alpha / 2 ) * se.theta.hat,
  theta.hat + qnorm( 1 - alpha / 2 ) * se.theta.hat )

if ( large.sample.confidence.interval[ 1 ] < 0 ) {

  n.negative.large.sample <- n.negative.large.sample + 1

}

if ( ( large.sample.confidence.interval[ 1 ] <
  population.theta ) & ( large.sample.confidence.interval[ 2 ] >
  population.theta ) ) {

  n.hit.large.sample <- n.hit.large.sample + 1

}

theta.hat.prime <- ( s.star + 0.5 ) / ( n + 1 )

eta.hat <- log( theta.hat.prime / ( 1 - theta.hat.prime ) )

se.eta.hat <- sqrt( 1 / ( n * theta.hat.prime *
  ( 1 - theta.hat.prime ) ) )

eta.interval <- c( eta.hat - qnorm( 1 - alpha / 2 ) * se.eta.hat,
  eta.hat + qnorm( 1 - alpha / 2 ) * se.eta.hat )

small.sample.confidence.interval <- 1 /
  ( 1 + exp( - eta.interval ) )

if ( small.sample.confidence.interval[ 1 ] < 0 ) {

  n.negative.small.sample <- n.negative.small.sample + 1

}
```

```
    if ( ( small.sample.confidence.interval[ 1 ] <
      population.theta ) & ( small.sample.confidence.interval[ 2 ] >
      population.theta ) ) {

      n.hit.small.sample <- n.hit.small.sample + 1

    }

  }

)

#  user  system elapsed
# 29.36    0.00   29.38

# on my laptop 1,000,000 simulation iterations takes
# about 30 seconds of clock time

print( paste( 'seed =', seed, '; proportion of all-0 data sets =',
  signif( n.null / M.calibration.large.sample, digits = 4 ) ) )

# "seed = 1 ; proportion of all-0 data sets = 0.01685"
# "seed = 2 ; proportion of all-0 data sets = 0.01698"

dbinom( 0, n, population.theta )

# 0.01696559

( 1 - population.theta )^n

# 0.01696559

print( paste( 'seed =', seed,
  '; proportion of large-sample negative intervals =',
  signif( n.negative.large.sample / M.calibration.large.sample,
  digits = 4 ) ) )

# "seed = 1 ; proportion of large-sample negative intervals = 0.412"
# "seed = 2 ; proportion of large-sample negative intervals = 0.4118"

print( paste( 'seed =', seed, '; large-sample interval hit rate =',
  signif( n.hit.large.sample / M.calibration.large.sample,
  digits = 4 ) ) )

# "seed = 1 ; large-sample interval hit rate = 0.9058"
# "seed = 2 ; large-sample interval hit rate = 0.9056"
```

```
print( paste( 'seed =', seed,
  '; proportion of small-sample negative intervals =',
  signif( n.negative.small.sample / M.calibration.large.sample,
  digits = 4 ) ) )

# "seed = 1 ; proportion of small-sample negative intervals = 0"
# "seed = 2 ; proportion of small-sample negative intervals = 0"

print( paste( 'seed =', seed, '; small-sample interval hit rate =',
  signif( n.hit.small.sample / M.calibration.large.sample,
  digits = 4 ) ) )

# "seed = 1 ; small-sample interval hit rate = 0.9521"
# "seed = 2 ; small-sample interval hit rate = 0.9518"
```

(a) Study this `R` code, and work with your TA to resolve any questions you may have about how it works. Your goal is to completely understand how every command and function does what it does, because later this quarter you'll be writing your own `R` code and you need to know how things work.

(b) Start an `R` session. Copy the entire block of `R` code in your text file and paste it into your `R` window (after pasting, you may need to hit `Enter` to get all of it into `R`). The code will begin running, and `R` will show you that it's computing with a little wheel of death; on my laptop with M = 1,000,000 simulation replications the code takes about 30 clock seconds to run. You should get *exactly* the same results I did with `seed = 1` (that's the point of setting a random number seed: the output of the simulation is "random," but (to promote replicability) when the code is run twice with the same random number seed it will yield *exactly* the same output).

I ran the code twice, with `seed = 1` and `seed = 2`, to get some idea of the Monte Carlo accuracy with M = 1,000,000. Pick a few more values of `seed` of your own choosing, edit them into the text window at the appropriate place (one by one), and rerun the code to see what kind of Monte Carlo noise you get. Try varying the α value to get different nominal confidence levels, to go along with the $\alpha = 0.05$ value in the code; I would suggest $\alpha = 0.2$, $\alpha = 0.1$ and $\alpha = 0.01$. Complete a table like the following:

CI Coverage ($n = 112$, $\theta \cdot 0.0357$)

	Large-Sample	Small-Sample
Nominal	Actual	Actual
80%		
90%		
95%	90.6%	95.2%
99%		

What conclusions do you draw about the calibration of the large-sample intervals with $n = 112$ and a true (population) θ value of about 0.0357? How about the small-sample intervals? (If you have time you can also try different values of `n` (e.g., even smaller than 112) and `population.theta` (e.g., even closer to 0).)