Statistics Live

An Introduction to Incorporating Simulation in Undergraduate Psychology Courses

Dr. Matthew Sigal
Simon Fraser University

Slides available at: www.matthewsigal.com/#talks

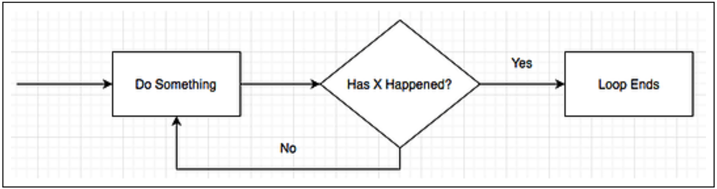
Presented at AMOM, June 2020

SIMON FRASER UNIVERSITY

Statistics Live!

Primary Goals

- Briefly discuss "Simulation"
  - What? Why? How?
  - Simulations in undergraduate courses?
- The How and the How To
  - What are *shiny* apps?
  - An interactive example
  - Future dashboards



Simulation

Simulation: The process of using statistical models and distributional parameters to generate random (but plausible) data.

Monte Carlo Simulation Studies

MCSS are **experiments** that use simulation to **generate** random data and estimate or **analyse** the behavior of other statistics across many *conditions*.

This is repeated over many *iterations* and results are **summarized** for dissemination.

Putting the Central Limit Theorem to Work

Given a population parameter ψ, let $\hat{\psi} = f(D)$ be the associated sample estimate, which is a function of data input D.

Theoretical CLT: given an *infinite number* of randomly sampled datasets D_i of size n, ψ can be recovered as the mean of all $f(D_i)$s.

Putting the Central Limit Theorem to Work

Given a population parameter ψ, let $\hat{\psi} = f(D)$ be the associated sample estimate, which is a function of data input D.

Theoretical CLT: given an *infinite number* of randomly sampled datasets D_i of size n, ψ can be recovered as the mean of all $f(D_i)$s.

MCSS: Generate a large (but finite!) number of datasets ("replications", R) to obtain a sample approximation of the population parameter ($\tilde{\psi}$):

$$\tilde{\psi} = \frac{f(D_1) + f(D_2) + \cdots + f(D_R)}{R}$$

Further...

- Further, the sampling error of ψ can be approximated by finding the standard deviation of all $f(D_i)$ sets:

$$SE(\tilde{\psi}) = \sqrt{\frac{[f(D_1) - \tilde{\psi}]^2 + \cdots + [f(D_R) - \tilde{\psi}]^2}{R}},$$

...which is interpreted as *the standard deviation of a statistic under a large number of random samples* — an empirically obtained estimate of the standard error that does not require or assume an infinite number of samples.

- While this seems reasonable for explaining concepts like the standard error of the mean, this holds for virtually *any* statistic and data generating mechanism (Mooney, 1997).

The General Structure

1. **Generate** a dataset with n values according to some probability density function (e.g., normal, log-normal, binomial, χ^2, etc.).
2. **Analyse** the generated data by finding the statistic of interest and store this value for later use.
3. Repeat steps 1 and 2 R times. Once complete, **summarise** the set of stored values with an appropriate statistic (e.g. mean, standard deviation).

Manipulate!

Once this structure is built, all sorts of things can be manipulated: generating distribution, sample size, number of replications, degree of heterogeneity of variance, and so on.

Conducting MCSS: An Introduction

Let's say Georgie is interested in the ability of a sample mean ($\overline{x}$) to recover μ and if the CLT approximation for the standard error is reasonable, given three different sample sizes. How can this be run?

Simulation Design

- Choice of generating distribution: *normal*
- Values of interest: *the mean*, *the standard error*
- Manipulation of interest: *sample size* (e.g., 5, 30, 60)

Georgie's First Simulation: Setup

```
# Design
R <- 5000   # set 5,000 replications
mu <- 10    # set mu to 10
sigma <- 2   # set standard deviation to 2
N <- c(5, 30, 60)  # set 3 sample size conditions

# Results
res <- matrix(0, R, 3) # create a null matrix
                       # (with R rows, and 3 columns)
                       # to store output.
colnames(res) <- N # name columns (5, 30, 60)

head(res, n = 2)
```

```
##      5 30 60
## [1,] 0  0  0
## [2,] 0  0  0
```

Georgie's First Simulation: Replications

```r
set.seed(77)  # Set seed to make analysis replicable
for(i in N){  # i = 5/30/60, across the 3 iterations
   for(r in 1:R){  # 1:R creates a vector 1,2,3,...,R
     dat <- rnorm(n = i, mean = mu, sd = sigma)
        # generate random data from a normal
        # distribution with set mean and sd
     res[r, as.character(i)] <- mean(dat)
        # return mean of dat and put it in res on row
        # r and in either column 5, 30, or 60.
   }
}
```

```
##             5     30     60
## [1,] 10.957 10.112 10.075
## [2,] 10.649 10.010  9.903
```

Georgie's First Simulation: Summarise

```
# summarise by calculating mean for each column
apply(res, 2, mean)
```

```
##      5     30     60
## 10.002 10.001 10.002
```

```
# summarise by calculating s for each column
apply(res, 2, sd)
```

```
##     5    30    60
## 0.889 0.368 0.258
```

Georgie's Observations

- μ was recovered well regardless of n.
- Sampling variability of the estimates decreased as n increased.
- Empirical SEs can be compared against CLT ($\sigma/\sqrt{n}$):
  - 0.894, 0.365, and 0.258

Conducting MCSS: A WARNING

ABORT

While "for loops" are useful for introducing simulation designs they **should not** be used if at all possible:

- Setup mixes generate and summarise steps
- For loops become increasingly complex as the design expands (nested loops)
- Objects can be easily overwritten accidentally
- Design change might require overhaul of entire loop structure
- Deciphering and debugging for loops is hell

Conducting MCSS: What to look for in Software

What we want. . .

- An overarching philosophy for structuring MCSS that clearly delineates the **generate**, **analyse**, and **summarise** steps.
- A structure that can be expanded as needed for various designs.
- Convenience features, e.g.:
  - Resample non-convergent results
  - Support parallel computation
  - Save/restore results in case of power failures
  - Explicit tools for debugging

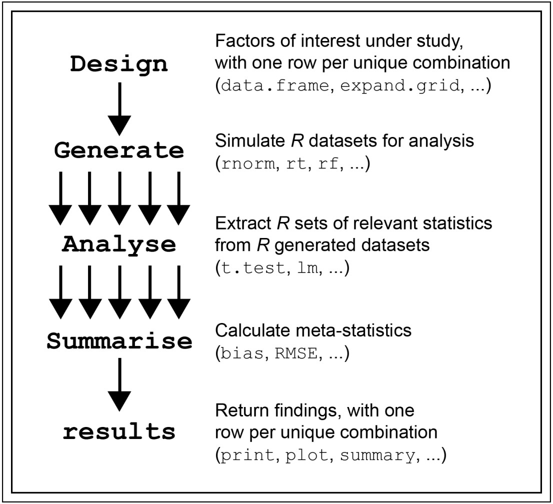
Conducting MCSS: My Recommendation

Highly recommended: `SimDesign` in `R` (Chalmers, 2018):

```r
install.packages("SimDesign")
library(SimDesign)
```

What does SimDesign provide?

SimDesign makes explicit reference to G.A.S.:



This structure can be applied to any simulation study!

It is. . . by Design

The "design" of a simulation study is typically a (fully-crossed) set of factors. `SimDesign` uses a `tibble` to store this:

```r
Design <- createDesign(sample_size = c(5, 30, 60))
Design
```

```
## # A tibble: 3 x 1
##   sample_size
##         <dbl>
## 1           5
## 2          30
## 3          60
```

Benefits:

- `Design` will be accessed sequentially (top to bottom), so it is easy to see what parameters are being passed and when.
- Rows of `Design` can be filtered, just as you would subset any other data.
- Columns can be added to incorporate other factors!

createDesign()

Add another variable to create fully-crossed design object:

```r
Design <- createDesign(sample_size = c(30, 60, 120),
                       distribution = c('norm', 'chi'))
Design
```

```
## # A tibble: 6 x 2
##   sample_size distribution
##         <dbl> <chr>
## 1          30 norm
## 2          60 norm
## 3         120 norm
## 4          30 chi
## 5          60 chi
## 6         120 chi
```

Generate This!

`Generate()` is a function that has only 1 required input: `condition` (a single row from `Design`) and uses parameters from that row to prepare a single dataset:

```
Generate <- function(condition, fixed_objects = NULL) {
  dat <- rnorm(n = condition$sample_size, mean = 10, sd = 2)
  dat
}
```

- Note the use of `condition$` to access variables from `Design`.
- Use `if()` statements if needed (e.g., for generating distribution).

Analyse That!

The purpose of `Analyse()` is to calculate and store all statistics of interest from each iteration.

For example, if we are only interested in the mean:

```
Analyse <- function(condition, dat, fixed_objects = NULL) {
  ret <- mean(dat)
  ret
}
```

This code will be called R times for each row of the `Design` matrix and can be used to return multiple values, if needed.

Then Summarise!

`Summarise()` is where we compute meta-statistics such as means, standard deviations, degree of bias, root mean-square error (RMSE), detection rates, and so on.

```
Summarise <- function(condition, results, fixed_objects = NULL) {
  c_mean <- mean(results)
  c_se <- sd(results)
  ret <- c(mu = c_mean, se = c_se)  # create a named vector
  ret
}
```

For each row of the design matrix, `SimDesign` will return the mean and standard error of the R replications as well as the number of replications, computation time, and a summary of any warnings that occurred.

`runSimulation()`

The final step is to pass the objects to `runSimulation()`:

```
results <- runSimulation(design=Design, replications = 5000,
    generate=Generate, analyse=Analyse, summarise=Summarise)
```

- Useful optional arguments:
  - `seed`: Set a random value seed for reproducability.
  - `save`: Save results to an external file.
  - `parallel`/`ncores`: Use parallel processing.
  - `debug`: Set to jump inside a running simulation (via `browser()`). Options include: `error`, `all`, `generate`, `analyse`, `summarise`.

See Sigal and Chalmers (2016) for more details.

How to make it interactive?

- `Shiny` (Chang et al., 2020) is an `R` package for coding interactive applets.
- Applets can be made to be incredibly user-friendly!
- Variety of **inputs**: action buttons, checkboxes, text fields, sliders.
- Can render a variety of **outputs**: plots, text, tables, user interface elements.

Shiny Apps

Traditionally, two files:

ui.R

- Script that defines the *user interface* of your app

server.R

- Code to process everything displayed in your app

Possible to put everything in one file:

app.R

```
library(shiny)
ui <- ...
server <- ...
shinyApp(ui = ui, server = server)
```

Hosting Shiny apps

- On your own computer:
  - Put your app's `ui.R` and `server.R` files in the same folder
  - Start `R` and load package with `library(shiny)`
  - Run your app with `runApp()` or RStudio's button
- Online:
  - shinyapps.io
  - Host on a Shiny server, like the one provided through the SFU Research Computing Group at www.rcg.sfu.ca/services/shiny/

Shiny + Simulation

1. For **teaching demonstrations**, I recommend coding a `shiny` app from scratch.

- Use a template and create a new app for each topic.
- Inputs should highlight primary pedagogical goals.

2. For teaching **Monte Carlo simulation studies**, I recommend using the `SimShiny()` function from `SimDesign` to create an app template based upon working MCSS code then edit as needed.

Teaching: The Central Limit Theorem… Before

	n	mean	s
n1.sample1	1	99.74	–
n1.sample2	1	88.24	–
n1.sample3	1	119.90	–
n2.sample1	2	85.78	15.19
n2.sample2	2	115.30	20.73
n2.sample3	2	96.36	4.88
n10.sample1	10	99.31	13.89
n10.sample2	10	93.53	16.07
n10.sample3	10	111.12	10.37
n25.sample1	25	101.13	15.16
n25.sample2	25	97.91	15.18
n25.sample3	25	105.70	12.37
n1000.sample1	1000	100.45	14.87
n1000.sample2	1000	99.90	15.29
n1000.sample3	1000	100.04	15.52

Teaching: The Central Limit Theorem… After

The Central Limit Theorem and Sampling Error

This applet generates random normally-distributed data based upon the parameters in the left-hand column. Try to find a sample size where the known population parameters are recovered!

Known Population Mean:

100

Known Population Standard Deviation:

15

Number of Samples:

1

Sample Size (n):

- ○ 1
- ● 2
- ○ 3
- ○ 10
- ○ 30
- ○ 50
- ○ 100

Seed Number:

1983

Sample	Sample_Size	Mean	Std_Dev	Min	Max
1	2	93.99	8.13	88.24	99.74

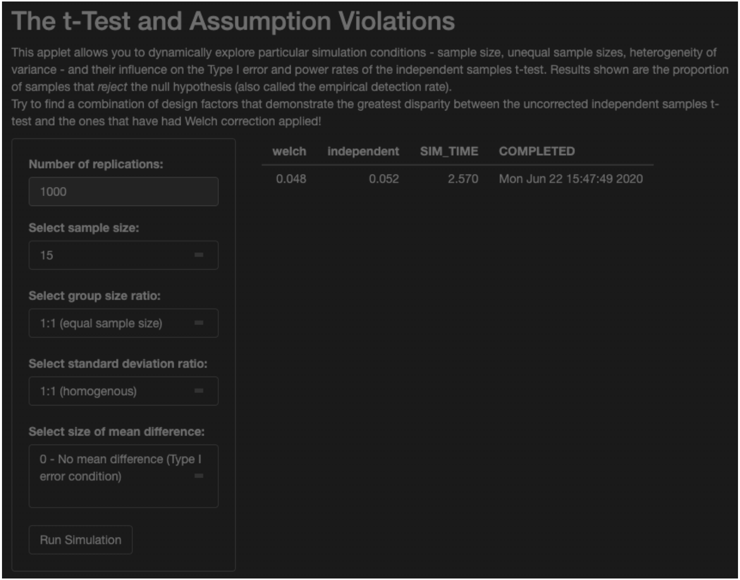
Overall, the mean of the 1 sample mean(s) was 93.995 and the average deviation was 8.131.

Sample 1

sample	values
1	99.74
1	88.24

https://shiny.rcg.sfu.ca/u/msigal/CLT/

Teaching: Monte Carlo Simulation Studies



https://shiny.rcg.sfu.ca/u/msigal/SIM/

Future Dashboards

Many topics in the undergraduate psychology curriculum could benefit from interactive applets. For example:

- Demonstrate the properties of statistical distributions using different sample sizes
- Demonstrate the influence of sample size/heterogeneity of variance on type I error rates and power
- Evaluate the bias and efficiency of estimators

Future Dashboards

Many topics in the undergraduate psychology curriculum could benefit from interactive applets. For example:

- Demonstrate the properties of statistical distributions using different sample sizes
- Demonstrate the influence of sample size/heterogeneity of variance on type I error rates and power
- Evaluate the bias and efficiency of estimators

But why?

- Allows students to "see it for themselves". They can play with various parameters and see the impact on results
- Provides a foundational understanding of simulation and simulation-based research than can be expanded on during a QM related degree
- Underlying code can be shared (e.g., via a GitHub repo) so keen students can also learn some `R` at the same time!

References I

Chalmers, P. (2018). *SimDesign: Structure for Organizing Monte Carlo Simulation Designs*. R package version 1.11, https://CRAN.R-project.org/package=SimDesign.

Chang, W., Cheng, J., Allaire, J., Xie, Y., and McPherson, J. (2020). *shiny: Web Application Framework for R*. R package version 1.4.0.2.

Mooney, C. Z. (1997). *Monte Carlo Simulations*. Sage, Thousand Oaks, CA.

Sigal, M. J. and Chalmers, R. P. (2016). Play it again: Teaching statistics with Monte Carlo simulation. *Journal of Statistics Education*, 24(3):136–156.