Lab 4: Introduction to Monte Carlo Simulation

EDP 380C.28: Simulation in R

Fall 2022 – Unique: 12104

Brian T. Keller, Ph.D.
bk@utexas.edu
University of Texas at Austin
September 26, 2022

Objectives

- Understand the slight distinction between methods and simulations.
- Learn about evaluation criteria for Monte Carlo simulations.
- Learn to measure uncertainty in Monte Carlo simulations.

---

1 Introduction

There are two distinct concepts involving the term Monte Carlo: ***Monte Carlo methods*** and ***Monte Carlo simulations***. Monte Carlo methods apply to a class of mathematical methods mainly used to solve problems involving numerical optimizations, numerical integration, and generating draws from probability distributions. In contrast, Monte Carlo simulations represent an approximation of a "natural" (and potentially hypothetical) stochastic process. By stochastic process, we mean a sequence of states determined by a random event.

One key difference between Monte Carlo methods and simulations is that Monte Carlo methods generally apply a probabilistic "game of chance" to solve a nonprobabilistic problem. To illustrate, suppose we want to compute the numerical constant π. We can define π based on the following definite integral.

$$\pi = 2\int_{-1}^{1}\sqrt{1-x^2}\,dx \;\approx\; 3.1415926\ldots$$

Instead of solving the mathematical expression, we can devise a game of chance to evaluate and solve π. Consider a square with a circle placed inside of it. The ratio of the circle and square's area is equal to $\pi/4$ (the area of the circle is π, and the area of the square is 4). We can leverage this fact to solve for π by setting up a game of chance to compute this ratio. Suppose you devised a board representing a square board with a dart board perfectly placed inside it. We can then randomly throw darts at the board and count the number of darts that fall in the circle versus the square to compute an empirical ratio of the areas. Figure 1 presents this setup with 100 randomly thrown darts that landed within this board. In Figure 1 we have 75 darts landing within the circle, resulting in the following ratio.

$$\frac{\pi}{4} \approx \frac{75}{100}$$

$$\pi \approx 0.75 \times 4 \;=\; 3$$

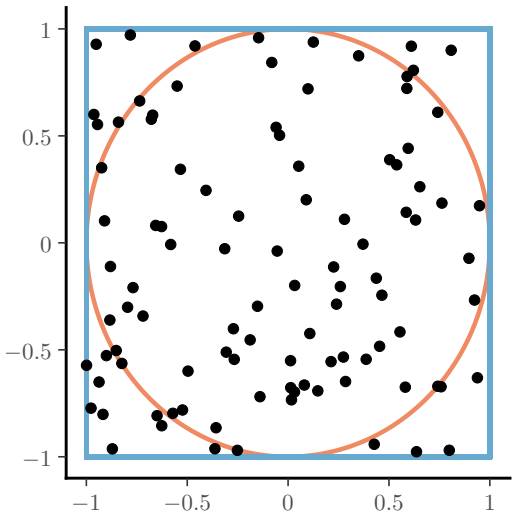

Figure 1: Illustration of Monte Carlo simulation to estimate π.

Although this is an imprecise approximation, we could hypothetically continue the game of chance until we reach some sort of convergence (i.e., adding a new dart does not greatly change our estimate). Note that there is no specific link to computers in using this method; however, the advances in modern-day computing make this approach more feasible to obtain rather precise estimates. Table 1 provides the Monte Carlo estimate of π as we increase the number of darts.

Table 1: Estimate of π as a function of number of simulated darts.

# **Draws**	**Estimate** ($\hat{\pi}$)	**Diff** ($\pi - \hat{\pi}$)
10	3.20000000	−0.05840735
100	3.12000000	0.02159265
1e3	3.17200000	−0.03040735
1e4	3.14520000	−0.00360735
1e5	3.13356000	0.00803265
1e6	3.13856800	0.00302465
1e7	3.14167960	−0.00008695

1.a In R:

Use Monte Carlo integration to estimate π.

(1) Set the seed to `19827`.

(2) Draw 1×10^7 random pairs that represent coordinates of dart throws. Note, it is faster to draw all `x` and then all `y` instead of drawing them as a single pair.

(3) Count the number of times that the points fall within the circle by evaluating the following function: `function(x, y) x^2 + y^2 <= 1`.

To accomplish this use the `Map()` function in R.

Monte Carlo methods are a useful mathematical tool to solve many statistical problems we cannot analytically solve. In later Labs, we will discuss them in more detail and how they apply to statistics. The primary focus of the remainder of this lab will be on Monte Carlo simulations, and we will devote the remainder of the lab to discussing their conception, implementation, and analysis.

2 Monte Carlo Simulations

A classical Monte Carlo simulation was designed by mathematician Stanisław Marcin Ulam, one of the founders of Monte Carlo (and who Stan software is named after). As described in his autobiography (*Adventures of a Mathematician*; Ulam, 1976), he devised a simulation to solve the probability of winning a game of solitaire. In a standard 52-card deck, there are $52!$ possible shuffle combinations ($\approx 8 \times 10^{67}$). Instead of solving this problem analytically, we could devise a computer simulation that randomly shuffles a simulated deck, construction of the card piles, and simulates the perfect completion of the game. Taking the average of the result over many replications would provide an estimate of the probability of success that converges to the true probability as the number of samples goes to infinity (i.e., law of large numbers).

In the context of this course, we will primarily focus on ***methodological*** Monte Carlo simulations. A methodological Monte Carlo simulation employs Monte Carlo simulations primarily evaluate the Frequentist properties of estimators, violations of modeling assumptions, and provide checks of the implementations for statistical algorithms. At their heart, these simulations are no different than the solitaire example we have already described. First, we must develop a conceptual model of the stochastic process we are attempting to evaluate. Second, we use this model to generate hypothetical random states (e.g., new shuffles of a deck, a sample data set). Third, we evaluate each sample based on preset evaluation criteria (e.g., a winnable game of solitaire, statistical estimator, discrepancy measures). Fourth, we average across the evaluation criteria to obtain a Monte Carlo estimate of the evaluation criteria. Each step of the process is essential and requires careful planning to implement the method correctly, and each step should be checked to ensure the implementation is without errors.

2.1 Conceptual Model and Generating States

One approach to developing a conceptual model would be to use resampling methods to sample from a known data set ***with replacement***. However, the results from this approach would be limited to only a data set with the same exact characteristics in the population. Alternatively, we can specify probabilistic models to represent processes that occur naturally. For example, with the solitaire example, we could sample uniformly without replacement to create a random permutation of all 52 cards. Alternatively, we can use statistical models that are the "true" data generating model as a starting point.

For example, consider Figure 2. In Lab 3 we used this process to generate data for a multiple regression. Suppose we are interested in evaluating the effects of violating several assumptions of linear regression across various conditions. First, let investigate violations to homoscedasticity by generating data with heteroskedastic residuals. The functional specification for this model is as follows.

$$f\left(X_1, X_2\right) \Rightarrow \quad X_1, X_2 \sim \mathcal{N}_2\left(\boldsymbol{\mu}_{\mathbf{X}}, \boldsymbol{\Sigma}_{\mathbf{X}}\right) \tag{1}$$

$$f\left(Y \mid X_1, X_2\right) \Rightarrow \quad Y \sim \mathcal{N}\left(\beta_0 + \mathbf{X}\boldsymbol{\beta}, \sigma_{e_i}\right) \tag{2}$$

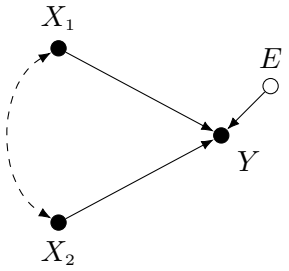

Figure 2: Graph of Data Generating Process for Two Predictors

For the purpose of this simulation, suppose the residual variance for each individual observation ($\sigma^2_{e_i}$) follows that of a gamma distribution with shape of $\frac{\bar{\sigma}^2_e}{2}$ and rate of 2, where $\bar{\sigma}^2_e$ is the average residual variance across the entire population.

$$\sigma^2_{e_i} \sim \text{Gamma}\left(\frac{\bar{\sigma}^2_e}{2}, 2\right) \tag{3}$$

The appropriate model to analyze this data in the population is a weighted least squares (WLS). Although we will not go into detail about this model, we can estimate this model if we know the variances in R by using the `weights` input in the `lm()` function. The weights are analogous to the inverse of the variances in the population (i.e., $w_i = 1/\sigma^2_{e_i}$).

2.a In R:

(1) Look in the directory labeled `scripts` for the files labeled `rmvnorm.R` and `ols_regression.R`. In these files, add the `rmvnorm()` function you implemented in Lab 2 and the `ols_regression` function you implemented in Lab 3.

Make sure to fill out the documentation based on the skeleton provided in the comments.

(2) Create a function to generate data based on Figure 2 and equations (1), (2), and (3). This function should map onto that of Method 1 in Lab 3 but modifying for the heteroskedasticity. The solution to $\bar{\sigma}^2_e$ is given by the following equation.

$$\bar{\sigma}^2_e = \boldsymbol{\beta}'\boldsymbol{\Sigma}_{\mathbf{X}}\boldsymbol{\beta}\left(\frac{1}{\overline{R}^2} - 1\right)$$

Make sure to include the weights in the data frame output.

(3) Create a function that generates one replicated data set and analyzes that data set with your `ols_regression` function and R's WLS regression function (i.e., use the `lm()` function with the `weights` input). This function should output the following structure: `list(ols, wls)` where each element of the list ought to include β_0, β_1, β_2, and residual standard deviation.

2.2 Evaluation Criteria

For methodological simulations, we typically evaluate an estimator ($\hat{\theta}$) based on good Frequentist properties. From a Frequentist perspective this often means some of the following "desirable" properties.

(1) $\hat{\theta}$ is a consistent estimator: $n \to \infty, \hat{\theta}_n \to \theta$

(2) $\hat{\theta}$ is unbiased: $\mathrm{E}\left(\hat{\theta}\right) = \lim_{n\to\infty} \sum_{i=1}^{n} \left(\frac{\hat{\theta}_i}{n}\right) = \theta$
Note: Unbiased is not an invariant property to sample size—e.g., logistic regression is a consistent estimator but biased in small sample sizes.

(3) The estimated sample variance should be consistent estimator of the sampling variance of $\hat{\theta}$. Said differently, the standard error estimate ought to be a consistent estimate of the true standard deviation of the sampling distribution.

(4) Interval estimates (e.g., confidence intervals) should contain the true value with proportion following $1 - \alpha$.

(5) Estimators ought to be efficient—i.e., as small as possible standard errors.

Most other evaluations can be considered a measure that is one of or combines multiple of the above criteria. This section will discuss common measures of discrepancy and other evaluation criteria often used in methodological simulations.

Measures of Bias

Typically, we do not produce measures of consistency itself because this would require an infinite sample size and are shown via asymptotic derivations. Instead, we evaluate measures of bias to see how estimators perform with small and large samples across multiple replications. We will discuss three measures: raw bias, proportion/percent bias, and standardized bias.

$$\begin{aligned} \text{Raw Bias} &= \mathrm{E}\left(\hat{\theta}\right) - \theta \\ &= \frac{1}{n}\sum_{i=1}^{n} \hat{\theta}_i - \theta \end{aligned} \tag{4}$$

Raw bias is the measure of the average discrepancy between the parameter estimate ($\hat{\theta}$) and the "true" parameter (θ). This quantity is often not evaluated in methodological simulations, but other quantities are computed based on it and a denominator to scale it.

Proportion bias (or its percent form) represents the amount of raw bias found divided by the absolute value of the population parameter estimate.

$$\text{Proportion Bias} = \frac{\text{Raw Bias}}{|\theta|} \tag{5}$$

This is a popular estimate that is often reported in most methodological simulations, and rules of thumb for acceptable ranges for finite samples generally range from ± 0.05 to ± 0.1. A major disadvantage of this metric is that it is not invariant to different linear scaling of parameters. For example, if we were to set a parameter to be one higher and saw the same raw bias, we would see

a reduction in proportion bias. Said differently, if we have a raw bias of 1 (i.e., $\hat{\theta} = \theta + 1$), the limit of proportion bias as θ goes to infinity equals zero.

$$\lim_{\theta \to \infty} \frac{(\theta + 1) - \theta}{|\theta|} = 0$$

Therefore, in theory, we could always arbitrarily increase θ in the population, and the proportion bias will always appear to be zero. Another poor property of proportion bias is that it is undefined when $\theta = 0$.

This leads us to standardized bias, where we use the sampling variance of the estimator to scale raw bias into a meaningful metric.

$$\text{Standardized Bias} = \frac{\text{Raw Bias}}{\sqrt{\text{Var}\left(\hat{\theta}\right)}} \tag{6}$$

This measure is invariant to the scale of θ and provides an interpretation of discrepancy in terms of standard error units. Generally, an acceptable range for standardized bias may be between ± 0.5 which means that our average estimate falls within half a standard error of our true value. Unless we know it analytically, generally we use an empirical estimate of $\text{Var}\left(\hat{\theta}\right)$ based on the simulation's results (i.e., taking the standard deviation of the parameter estimates across replications).

In addition to the previous three measures there is also the measure of ***mean squared error (MSE)***. The mean squared error is taking the average of the squared deviation between a parameter estimate and the true value,

$$\text{MSE} = \text{E}\left(\left[\hat{\theta} - \theta\right]^2\right) \tag{7}$$

An alternative specification is to take the square root of MSE to obtain the root mean square error (RMSE). Another important property is that the MSE is equivalent to the variance of an estimator plus its squared raw bias.

$$\text{MSE} = \text{Var}\left(\hat{\theta}\right) + \left[\text{E}\left(\hat{\theta}\right) - \theta\right]^2$$

This relationship leads to what some view as a strength of MSE and others view as a weakness. MSE conflates the two measures into a single number summary which can represent the fact that ultimately we are interested in two quantities for an estimator, unbiased and efficient. Trying to find an estimator that optimizes both is known as the ***bias–variance problem***. Often it is the case that we can find estimators that are highly precise (i.e., efficient) but biased (e.g., regularized regression has superior MSE compared to OLS regression but is not unbiased). Finally, another use of MSE is to comprise ratios and compare different methods. These MSE ratios can be helpful to benchmark competing methods to some "Gold Standard" estimator (i.e., the ratio's denominator).

Measures of Statistical Inference

In addition to bias measures, Monte Carlo simulations often evaluate the Frequentist properties of the statistical inferences. Typically, we can evaluate this via Type 1 error, Type 2 error, and coverage—all three a straightforward to evaluate. To evaluate Type 1 error, we must generate data specifically from the null distribution (i.e., no effect) of the hypothesis test. We can then evaluate how often we improperly reject the null hypothesis at a given α level. Alternatively, we can evaluate Type 2 error rates by generating data and evaluating how often we fail to reject the null

hypothesis. Similarly, we can construct the interval estimates for coverage and evaluate if the interval contains the true parameter estimate the appropriate number of times. As a rule of thumb, acceptable coverage for 95% intervals is between 0.925 and 0.975.

2.b In R:

(1) Construct function factories for proportion bias, standardized bias, and MSE. See code example below.

(2) Construct and run a simulation (replications $= 200$) that looks at proportion bias, standardized bias, and MSE for OLS and WLS estimators.

- Use the data generation functions you developed in the previous section to generate and analyze the data. Set the seed to `12345` for the simulation.
- Use the following population parameters: $\beta_0 = 100$, $\beta_1 = \beta_2 = 1$, (average) $\overline{R}^2 = 0.3$, $\sigma_1 = \sigma_2 = 1$, $\rho_{12} = 0.3$, $\mu_1 = \mu_2 = 0$, $\beta_0 = 100$, and $n = 100$.
- Use the function factories you made to compute each of the three quantities.

Make sure to construct the parameters set-ups in R similarly to previous labs.

Answer:

Construct the following table for OLS and WLS regression. Include the printout of the two tables labeled in a comment in the code.

	Pop.	Est.	P. Bias	SD Bias	MSE
b0	—	—	—	—	—
b1	—	—	—	—	—
b2	—	—	—	—	—
SD(e)	—	—	—	—	—

R Code Example for function factory

```
# Set up for a generic function factory
factory <- function(pop_value) {
    force(pop_value}
    function(data) {
        # implement computation using pop_value
    }
}
```

2.3 Characterizing Uncertainty in Monte Carlo Simulations

Just like estimates from a finite sample of observations have measures of uncertainty (i.e., standard errors), simulation with finite number of replications have uncertainty in their evaluation criteria. Said differently, Monte Carlo simulations provide ***estimates*** of the evaluation criteria that have uncertainty associated with them. Only as the number of replications in the simulation goes to infinity will it converge to the "truth." To illustrate this fact, consider Figure 3. Figure 3 illustrates

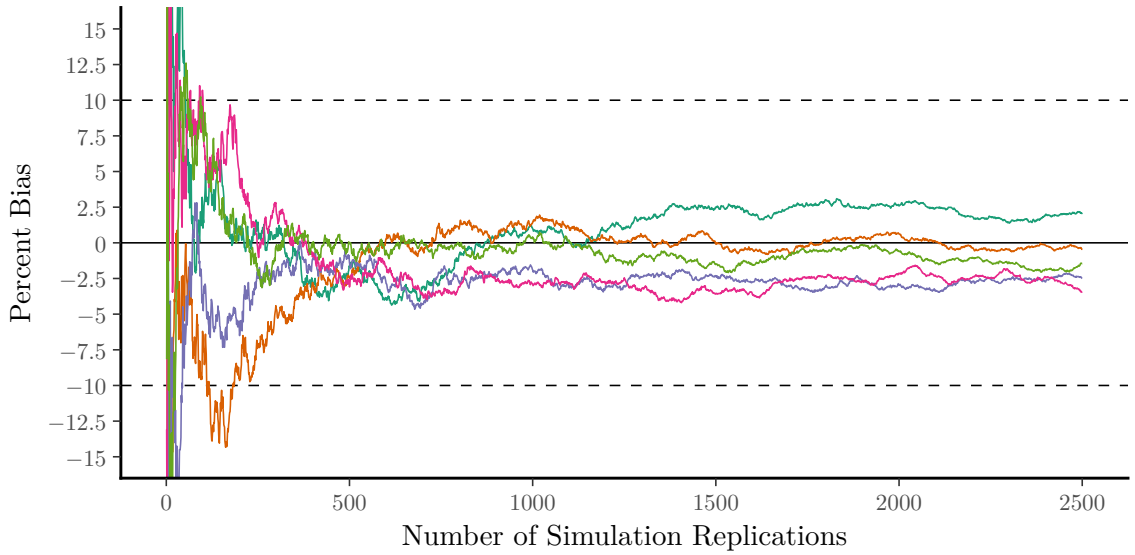

Figure 3: Running Monte Carlo estimate of Percent Bias for Five Simulations.

five differently seeded Monte Carlo simulations evaluating the percent bias of a single predictor regression with $n = 20$. The lines represent the current value of percent bias as the number of replications in each simulation increases (up to 2500 replications). As we can see, the final results from the simulation range from approximately -3% to 2%, with four of the five simulations below 0%.

Moreover, we can see how despite sampling 2500 replications, we still have not converged to being close to 0%. This illustrates why including measures of uncertainty is necessary. Instead of reporting a mere point estimate, we should include an interval that quantifies a range of plausible values. While there are analytical formulas that attempt to calculate the standard error from Monte Carlo simulations, it is much more practical to use numerical techniques to estimate these quantities empirically. Furthermore, in my experience, these give a more accurate estimate of the uncertainty.

The first, most brute force, and impractical way to obtain an estimate of Monte Carlo standard error is to run another Monte Carlo simulation. Said differently, we could treat each Monte Carlo simulation as a single replication of n enveloping Monte Carlo simulation and look at the empirical variance of those estimates. Figure 4 illustrates this concept graphically. As you might expect, this is a very impractical solution because extensive Monte Carlo simulations can often take weeks to months to run.

A second more practical approach is to use resampling techniques to estimate intervals or standard errors, namely the bootstrap. A ***naive bootstrap*** is a simple process that will sample with replacement from the simulation replications. The bootstrap has the following steps for n simulation replications:

(1) Sample a size of n from the original replications with replacement to generate a new sample.

(2) Evaluate the criteria on this new sample (e.g., compute parameter bias for the new sample).

(3) Repeat this for a desired number of bootstrap samples to obtain an empirical sampling distribution of the evaluation criteria selected.

(4) Compute measures of uncertainty (e.g., standard deviation, quantile intervals).

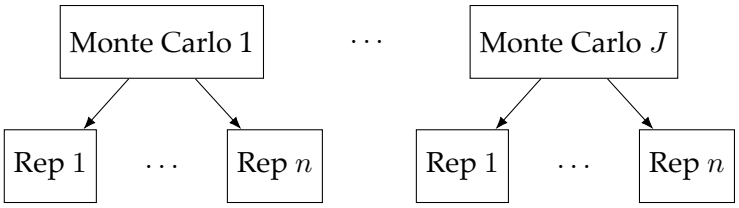

Figure 4: Monte Carlo Simulation within a Monte Carlo Simulation

The naive bootstrap essentially creates a Monte Carlo simulation, as discussed in the previous paragraph; however, it has the advantage that we do not need to analyze and obtain new parameter estimates for each replication. Instead, we only need to obtain evaluation measures from the already analyzed parameter estimates. As you might expect, the bootstrap's estimate of the standard error also has a standard error associated with it, and hypothetically we could continue this process again.

Finally, a nonprobabilistic resampling technique is the ***jackknife***. The jackknife predates the bootstrap and is not used as often, but it offers a straightforward calculation for estimates of the Monte Carlo standard error for our simulation. The jackknife obtains a resampling estimate by obtaining each evaluation criteria's "leave-one-out" estimate. Said differently, we obtain the evaluation criteria (e.g., percent bias) by dropping one observation and averaging across all estimates. More concretely, the jackknife has the following steps for n simulation replications:

(1) Repeat the following steps for $i = 1$ to n.

   (a) Drop the i^{th} observation from the data set.

   (b) Evaluate the criteria on this new sample (e.g., compute parameter bias for the new sample).

(2) Compute the mean of all the previous estimates to obtain $\hat{\theta}_{\text{jack}}$ where θ is ***evaluation criteria*** (not the parameter estimates themselves).

(3) Compute the jackknife estimate of the variance with the following equation.

$$\text{Var}\left(\hat{\theta}_{\text{jack}}\right) = \frac{n-1}{n}\sum_{i=1}^{n}\left(\theta_i - \hat{\theta}_{\text{jack}}\right)^2 \tag{8}$$

(4) Take the square root of $\text{Var}\left(\hat{\theta}_{\text{jack}}\right)$ to obtain the standard deviation of the Monte Carlo sampling error.

The advantage of the jackknife estimate is that it follows a known formula for the evaluation measure's sampling variance. Therefore, we are not required to quantity any additional uncertainty as we theoretically would need to with a finite number of bootstrap samples.

2.c In R:

(1) Create a bootstrap function of the following form:

```
bootstrap <- function(data, func, n)
```

where `data` is a vector of parameter estimates, `func` is an evaluation function of choice (e.g., percent bias function generated), and `n` is the number of bootstrap samples.

(2) Create a jackknife function of the following form:

```
jackknife <- function(data, func)
```

where `data` is a vector of parameter estimates and `func` is an evaluation function of choice (e.g., percent bias function generated).

Answer:

Use the two functions to compute estimated Monte Carlo standard errors for all measures in 2.b (proportion bias, standardized bias, and MSE) for OLS only. The output should be in a list with each element corresponding to the following table.

	Bootstrap MCSE	Jackknife MCSE
b0	—	—
b1	—	—
b2	—	—
SD(e)	—	—

Include the printout of the two tables labeled in a comment in the code.

3 Running Your Own Monte Carlo Simulation

In this section, you are tasked with setting up and running your own simulation to evaluate the effects of a residual correlation between X_1 and the outcome. Consider Figure 5, which will be the model you use to generate the data. You will fit the following misspecified model.

$$\begin{aligned} y_i &= \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + e_i \\ e_i &\sim \mathcal{N}\left(0, \sigma_e\right) \end{aligned} \tag{9}$$

You will evaluate the above model based on proportion bias, standardized bias, and MSE for all values of interest (i.e., β_0, β_1, β_2, σ_e, and R^2). Although we have not generated data from this model before, we have gone over every tool you need to set it up and create the data generating function. Consider writing out the functional notation, as I have done in previous examples for this new example. Breaking this problem down will be the key to solving every parameter. ***Hint:*** You will need to use R^2 to determine the total variance of E.

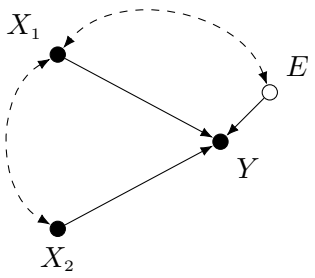

Figure 5: Data Generating Process for Two Predictors with Correlated Error

3.a In R:

- Create a data generating function and empirically verify each population parameter (i.e., use a large sample $n = 100000$ and verify each parameter set in the model). Include the output of this verification in your code.
- Set the model parameters to following: $\beta_0 = 100$, $\beta_1 = \beta_2 = 1$, $R^2 = 0.4$, $\sigma_1 = \sigma_2 = 1$, $\rho_{12} = 0.3$, $\mu_1 = \mu_2 = 0$, and $n = 100$.
- You will evaluate the correlation between X_1 and E under three conditions: 0.1, 0.3, and 0.8.
- Set the seed to whatever you like.

Answer:

Construct three tables (one for each evaluation criteria) that compare the values of interest (i.e., β_0, β_1, β_2, σ_e, and R^2) across the three conditions. Include the output from these tables as comments in the code. The tables should be labeled and have the parameter labels on the rows and residual correlation conditions on the columns.