`simChef`: High-quality data science simulations in R

James Duncan [1*¶], **Tiffany Tang** [2*], **Corrine F. Elliott** [2], **Philippe Boileau** [1], **and Bin Yu**[1,2,3,4]

1 Graduate Group in Biostatistics, University of California, Berkeley **2** Department of Statistics, University of California, Berkeley **3** Department of Electrical Engineering and Computer Sciences, University of California, Berkeley **4** Center for Computational Biology, University of California, Berkeley ¶ Corresponding author * These authors contributed equally.

Summary

`simChef` is an R package that empowers data science practitioners to rapidly plan, carry out, and summarize statistical simulation studies in a flexible, efficient, and low-code manner. Drawing substantially from the Predictability, Computability, and Stability (PCS) framework (Yu & Kumbier, 2020), `simChef` emphasizes the scientific best practices encompassed by PCS by removing many of the administrative burdens of simulation design through: (1) an intuitive tidy grammar of data science simulations; (2) powerful abstractions for distributed simulation processing backed by `future` (Bengtsson, 2021); and (3) automated generation of interactive R Markdown simulation documentation, situating results next to the workflows needed to reproduce them. Taken together, `simChef`'s capabilities overcome many of the design, computational, and reproducibility hurdles inherent in nearly every data science simulation study.

Statement of need

Data science simulation studies occupy an important role in scientific research as a means to gain insight into new and existing statistical methods. Simulations serve as statistical sandboxes that open a path toward otherwise inaccessible discoveries. For example, they can be used to establish comprehensive benchmarks of existing procedures for a common task; to demonstrate the strengths and weaknesses of novel methodology applied to synthetic and real-world data; or to probe the validity of a theoretical analysis.

Creating high-quality simulation studies typically involves a number of repetitive and error-prone coding tasks: implementing data-generating processes (DGPs) and statistical methods; sampling from these DGPs; parallelizing computation of simulation replicates; summarizing metrics; visualizing, documenting, presenting, and saving results; and so on. While this

administrative overhead is necessary, it is not sufficient for scientific understanding. Data scientists must navigate a number of important judgment calls such as the choice of DGPs, baseline statistical methods, associated parameters, and evaluation metrics for scientific relevancy.

While the scientific context may vary drastically from one study to the next, the simulation scaffolding remains largely similar. Yet simulation code repositories often lack reusability, both for novel settings and when new questions arise in the original context. simChef addresses the need for an intuitive, extensible, and reusable framework for data science simulations, allowing data science practitioners to focus their energies on scientific questions by reducing the burdens of parameterization, parallelization, and documentation.

Core abstractions of data science simulations

At its core, simChef breaks down a simulation experiment into four modular components (Figure 1), each implemented as an R6 class (Chang, 2022):

- DGP: the data-generating processes from which to *generate* data
- Method: the methods (or models) to *fit* in the experiment
- Evaluator: the evaluation metrics used to *evaluate* the methods' performance
- Visualizer: the visualization functions used to *visualize* outputs from the method fits or evaluation results (can be tables, plots, or even R Markdown snippets to display)
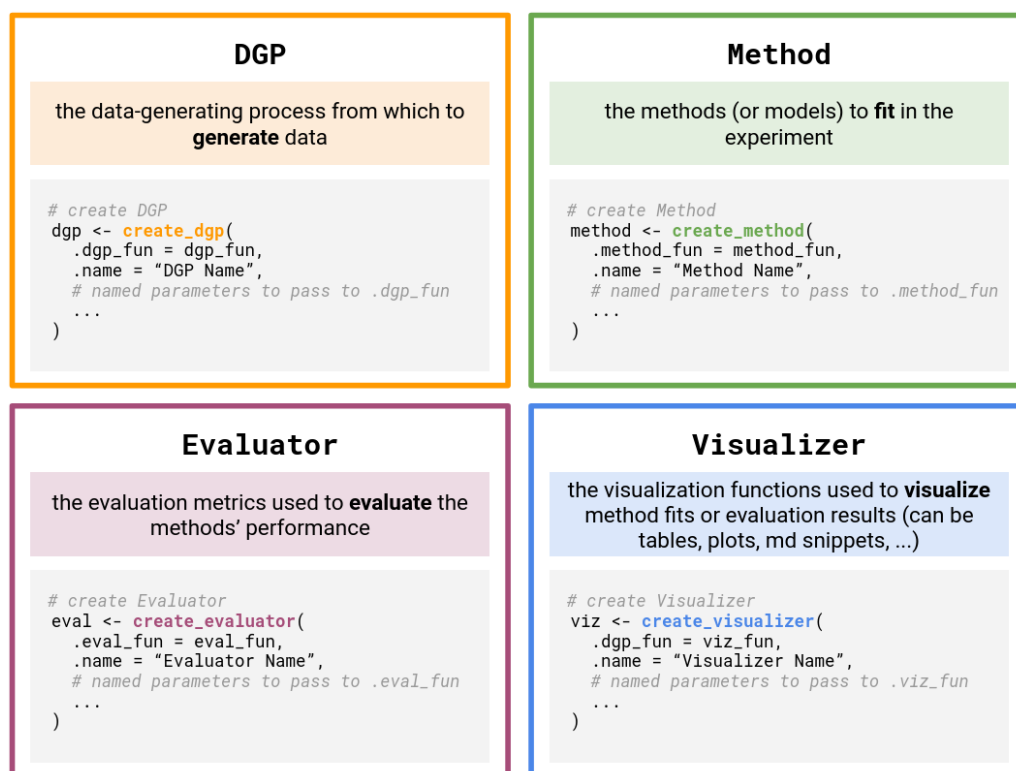

Figure 1: Overview of the four core components in a simChef Experiment. simChef provides four classes that implement distinct simulation objects in an intuitive and modular manner: DGP, Method, Evaluator, and Visualizer. Using these classes, users can easily build a simChef Experiment using reusable, customizable functions (i.e., dgp_fun, method_fun, eval_fun, and viz_fun). Optional named parameters can be set in these custom functions via the ... arguments in the create_*() methods.

Using these classes, users can create or reuse custom functions (i.e., dgp_fun, method_fun,

Duncan et al. (1970). simChef: High-quality data science simulations in R. *Journal of Open Source Software*, ¿VOL?(¿ISSUE?), ¿PAGE? 2
https://doi.org/N/A.

eval_fun, and `viz_fun` in [Figure 1](#)) aligned with their scientific goals. The custom functions then can be parameterized and encapsulated in one of the corresponding classes via a `create_*` method, together with optional named parameters (see [Figure 1](#)).

A fifth R6 class, `Experiment`, unites the four components above and serves as a concrete implementation of the user's intent to answer a specific scientific question. Specifically, the `Experiment` stores references to the DGP(s), Method(s), Evaluator(s), and Visualizer(s) along with the DGP and `Method` parameters that should be varied and combined during the simulation run.
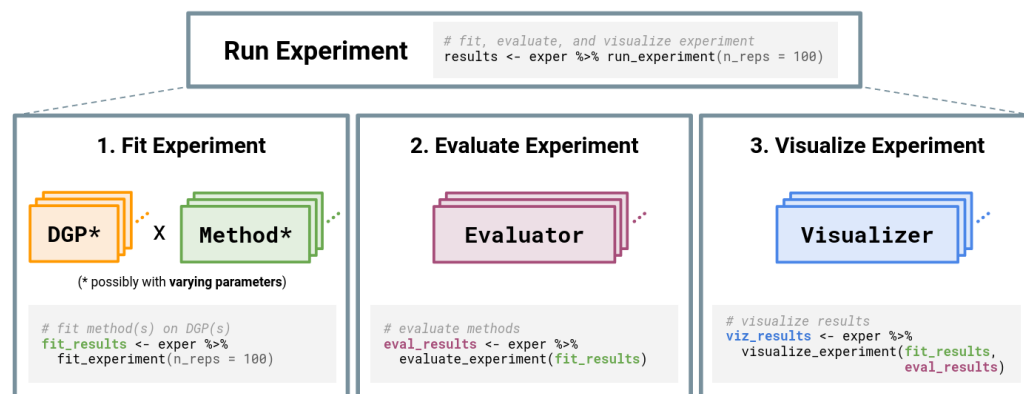
Figure 2: Overview of running a `simChef` `Experiment`. The `Experiment` class handles relationships among the four classes portrayed in [Figure 1](#). Experiments may have multiple DGPs and Methods, which are combined across the Cartesian product of their varying parameters (represented by *). Once computed, each `Evaluator` and `Visualizer` takes in the fitted simulation replicates, while `Visualizer` additionally receives evaluation summaries.

A powerful grammar of data science simulations

Inspired by the tidyverse ([Wickham et al., 2019](#)), `simChef` develops an intuitive grammar for running simulation studies using the aforementioned R6 classes. We provide an illustrative example usage next.

```r
library(simChef)

dgp1 <- create_dgp(dgp_fun1, "my_dgp1", sd = 0.5)
dgp2 <- create_dgp(dgp_fun2, "my_dgp2")
method <- create_method(method_fun, "my_method")
eval <- create_evaluator(eval_fun)
viz <- create_vizualizer(viz_fun)

exper <- create_experiment(dgp_list = list(dgp1, dgp2)) %>%
  add_method(method) %>%
  add_vary_across(
    list(dgp1, dgp2),
    n = c(1e2, 1e3, 1e4)
  ) %>%
  add_vary_across(
    dgp2,
    sparse = c(FALSE, TRUE)
  ) %>%
  add_vary_across(
    method,
```

Duncan et al. (1970). simChef: High-quality data science simulations in R. *Journal of Open Source Software*, ¿VOL?(¿ISSUE?), ¿PAGE? 3
https://doi.org/N/A.

```
    scalar_valued_param = c(0.1, 1.0, 10.0),
    vector_valued_param = list(c(1, 2, 3), c(4, 5, 6)),
    list_valued_param = list(list(a1=1, a2=2, a3=3),
                             list(b1=3, b2=2, b3=1))
  ) %>%
  add_evaluator(eval) %>%
  add_viz(viz)

future::plan(multicore, workers = 64)

results <- exper %>%
  run_experiment(n_reps = 100, save = TRUE)

new_method <- create_method(new_method_fun, 'my_new_method')

exper <- exper %>%
  add_method(new_method)

results <- exper %>%
  run_experiment(n_reps = 100, use_cached = TRUE)

init_docs(exper)
render_docs(exper)
```

In the example usage, DGP(s), Method(s), Evaluator(s), and Visualizer(s) are first created via create_*(). These simulation objects can then be combined into an Experiment using either create_experiment() and/or add_*().

In an Experiment, DGP(s) and Method(s) can also be varied across one or multiple parameters via add_vary_across(). For instance, in the example Experiment, there are two DGP instances, both of which are varied across three values of n and one of which is additionally varied across two values of sparse. This effectively results in nine distinct configurations for data generation (i.e., 3 variations on dgp1 + 3x2 variations on dgp2). For the single Method in the experiment, we use three values of scalar_valued_param, two of vector_valued_param, and another two of list_valued_param, giving 12 distinct configurations. Hence, there are a total of 9x12 = 108 DGP-method-parameter combinations in the Experiment.

Thus far, we have simply instantiated an Experiment object (akin to creating a recipe for an experiment). To compute and run the simulation experiment, we next call run_experiment with the desired number of replicates. As summarized in Figure 2, running the experiment will (1) *fit* each Method on each DGP (and for each of the varying parameter configurations), (2) *evaluate* the experiment according to the given Evaluator(s), and (3) *visualize* the experiment according to the given Visualizer(s). Furthermore, the number of replicates per combination of DGP, Method, and parameters specified via add_vary_across is determined by the n_reps argument to run_experiment. Because replication happens at the per-combination level, the effective total number of replicates in the Experiment depends on the number of DGPs, methods, and varied parameters. In the given example, there are 108 DGP-method-parameter combinations, each of which is replicated 100 times. To reduce the computational burden, the Experiment class flexibly handles the computation of simulation replicates in parallel using the future package (Bengtsson, 2021). Figure 3 provides a detailed schematic of the run_experiment workflow, along with the expected inputs to and outputs from user-defined functions.
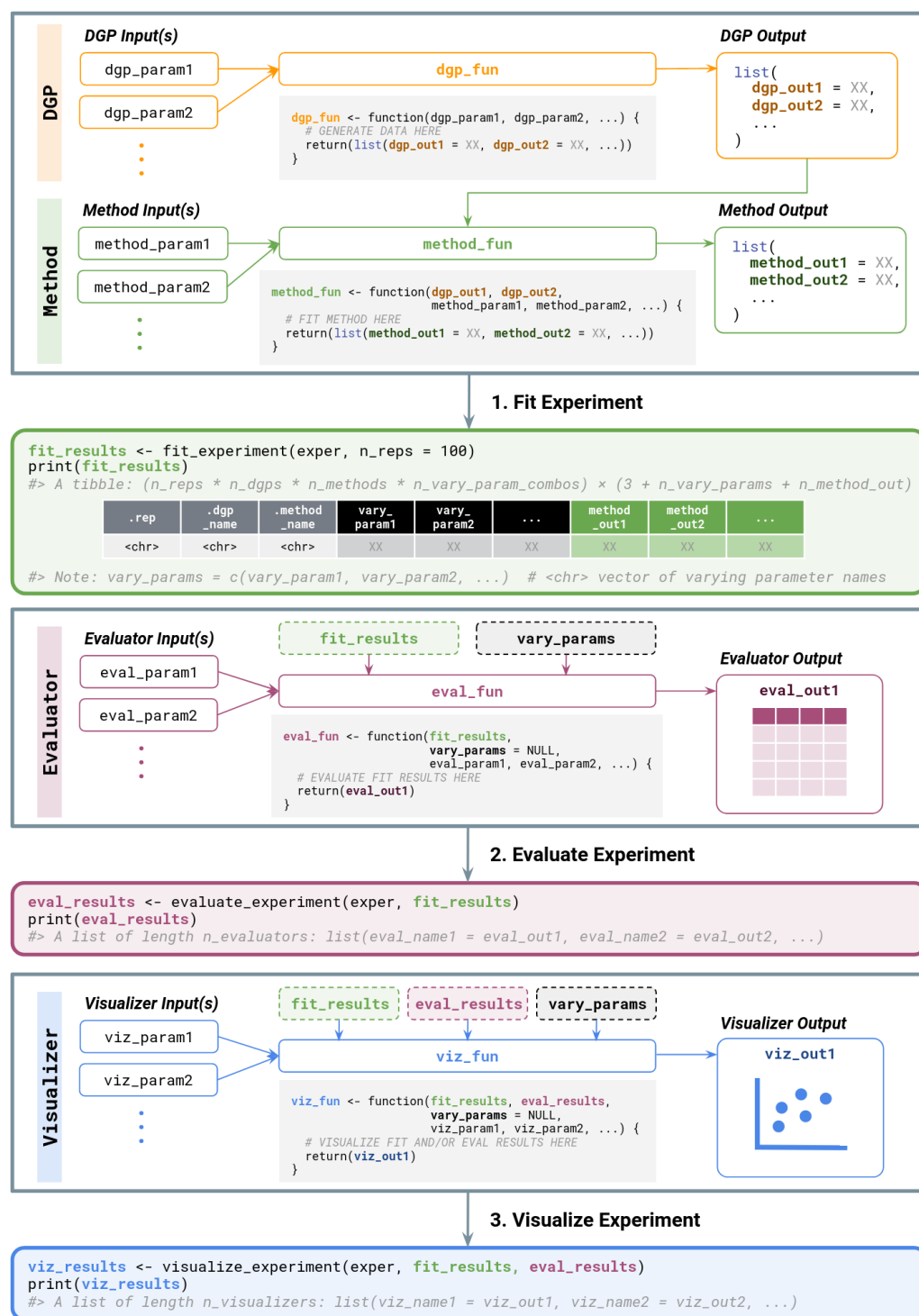
Figure 3: Detailed schematic of the `run_experiment` workflow using `simChef`. Expected inputs to and outputs from user-defined functions are also provided.

Additional Features

In addition to the ease of parallelization, `simChef` enables caching of results to further alleviate the computational burden. Here, users can choose to save the experiment's results to disk by

passing `save = TRUE` to `run_experiment`. Once saved, the user can add new `DGP` and `Method` objects to the experiment and compute additional replicates without re-computing existing results via the `use_cached` option. Considering the example above, when we add `new_method` and call `run_experiment` with `use_cached = TRUE`, simChef finds that the cached results are missing combinations of `new_method`, existing DGPs, and their associated parameters, giving nine new configurations. Replicates for the new combinations are then appended to the cached results.

`simChef` also provides users with a convenient API to automatically generate an R Markdown document. This documentation gathers the scientific details, summary tables, and visualizations side-by-side with the user's custom source code and parameters for data-generating processes, statistical methods, evaluation metrics, and plots. A call to `init_docs` generates empty markdown files for the user to populate with their overarching simulation objectives and with descriptions of each of the `DGP`, `Method`, `Evaluator`, and `Visualizer` objects included in the `Experiment`. Finally, a call to `render_docs` prepares the R Markdown document, either for iterative design and analysis of the simulation or to provide a high-quality overview that can be shared easily. We provide an example of the simulation documentation [here](). Corresponding R source code is available on [GitHub]().

Related R packages

A number of existing R packages and projects address needs related to `simChef`'s functionality. At a higher level of abstraction, the `batchtools` package ([Lang et al., 2017]()) includes concepts for "problems", "algorithms", and "experiments", similar to `simChef`'s `DGP`, `Method`, and `Experiment` objects, respectively, but less tailored to the specific needs of data science simulation experiments. Additionally, `batchtools` provides a number of utilities for shared-memory and distributed memory computations, including for interacting with high-performance computing cluster schedulers such as Slurm and Torque. `simChef` is able to leverage these utilities for distributed computations via the backends provided by the `future.batchtools` package which is part of the `future` ecosystem of R packages ([Bengtsson, 2021]()). Whereas `batchtools` is a general tool for distributed mapping operations, `simChef` specializes in data science simulations and provides additional functionality tailored to that setting including its `tidy` grammar of simulation experiments, the `Evaluator` and `Visualizer` concepts, and automated documentation capabilities discussed above.

Like `simChef`, many existing packages specifically aim to simplify the process of creating simulation experiments by reducing coding burden through helpful abstractions, distributed computing helpers, and preset methods for generating, computing, and summarizing simulation replicates. Of particular note are the following:

- `SimDesign` ([Chalmers, 2020]()) focuses on Monte Carlo simulation experiments and provides a function `runSimulation` that accepts user-defined `generate`, `analyse`, and `summarise` functions, with support for distributed computation via the `parallel` base R package and `future`.
- `simulator` ([Bien, 2016]()) provides a `tidy` grammar of simulation experiments and highly modular helpers for evaluating and managing simulation outputs, relying on the `parallel` package for distributed computation.
- `simpr` ([Brown, 2023]()) defines a `tidy` simulation framework for generating data, fitting models, varying parameters, and aggregating simulation results with user-defined and `purr`-style functions. In addition, it support distributed computations backed by the `future` framework.
- `SimEngine` ([Kenny & Wolock, 2023]()) defines and executes simulation 'levels' (parameters to vary) and 'scripts' (functions to execute a single simulation replicate). It manages the definition and execution of simulations and calculates summary statistics, with support for distributed computations in coordination with high-performance computing cluster schedulers.

A third category of related packages are those that share conceptual similarities `simChef` in terms of providing helpful abstractions for the design and analysis of simulation experiments, but at a finer level of detail than `simChef` intends. For example, the package `DeclareDesign` (Blair et al., 2019) provides various `declare_*` functions for defining and evaluating statistical research questions, with an emphasis on the social sciences. The package `infer` (Couch et al., 2021) provides a `tidy` API for statistical inference, providing the ability to specify random variables and their relationships, define a null hypothesis, generate data under that hypothesis, and calculate distributions of statistics based on that hypothesis. Both of these packages and many of the packages below could be employed in a user's `DGP`, `Method`, `Evaluator`, or `Visualizer` and deployed via an `Experiment` to carry out a large-scale simulation with automated documentation in harmony with `simChef`.

Finally, many packages provide a small number of well-tailored helper functions for specific data-generating processes and simulation settings, with or without distributed computation. In no particular order these include: `simitation` (Shilane et al., 2023), `simhelpers` (Joshi & Pustejovsky, 2024), `simTool` (Scheer, 2020), `parSim` (Epskamp, 2023), `rsimsum` (Gasparini, 2018), `simsalapar` (Hofert & Mächler, 2016), `tidyMC` (Linner et al., 2022), `MonteCarloSEM` (Orcan, 2021), `simMetric` (Parsons, 2022), and `simmer` (Ucar et al., 2019). To our knowledge, no single existing package includes `simChef`'s combination of conceptual modularity, `tidy` grammar, computational flexibility, simulation workflow management, and automated documentation.

Discussion

While `simChef`'s core functionality focuses on computability (C) – encompassing efficient usage of computational resources, ease of user interaction, reproducibility, and documentation – we emphasize the importance of predictability (P) and stability (S) in data science simulations. The principal goal of `simChef` is to provide a tool for data scientists to create simulations that incorporate predictability (through fit to real-world data) and stability (through sufficient exploration of uncertainty) in their simulations. In future work, we intend to provide tools that can be flexibly tailored to a user's particular scientific needs and further these goals through automated predictability and stability summaries and documentation.

Acknowledgements

References

Bengtsson, H. (2021). A Unifying Framework for Parallel and Distributed Processing in R using Futures. *The R Journal*, *13*(2), 208. https://doi.org/10.32614/RJ-2021-048

Bien, J. (2016). *The Simulator: An Engine to Streamline Simulations*. https://arxiv.org/abs/1607.00021

Blair, G., Cooper, J., Coppock, A., & Humphreys, M. (2019). Declaring and Diagnosing Research Designs. *American Political Science Review*, *113*(3), 838–859.

Brown, E. (2023). *simpr: Flexible 'Tidyverse'-Friendly Simulations*. https://statisfactions.github.io/simpr/

Chalmers, M. C., R. Philip AND Adkins. (2020). Writing Effective and Reliable Monte Carlo Simulations with the SimDesign Package. *The Quantitative Methods for Psychology*, *16*(4), 248–280. https://doi.org/10.20982/tqmp.16.4.p248

Chang, W. (2022). *R6: Encapsulated Classes with Reference Semantics*. https://r6.r-lib.org

Couch, S. P., Bray, A. P., Ismay, C., Chasnovski, E., Baumer, B. S., & Çetinkaya-Rundel, M. (2021). infer: An R package for tidyverse-friendly statistical inference. *Journal of Open Source Software*, *6*(65), 3661. https://doi.org/10.21105/joss.03661

Epskamp, S. (2023). *parSim: Parallel Simulation Studies*. https://CRAN.R-project.org/package=parSim

Gasparini, A. (2018). rsimsum: Summarise results from Monte Carlo simulation studies. *Journal of Open Source Software*, *3*(26), 739. https://doi.org/10.21105/joss.00739

Hofert, M., & Mächler, M. (2016). Parallel and Other Simulations in R Made Easy: An End-to-End Study. *Journal of Statistical Software*, *69*(4), 1–44. https://doi.org/10.18637/jss.v069.i04

Joshi, M., & Pustejovsky, J. (2024). *simhelpers: Helper Functions for Simulation Studies*. https://meghapsimatrix.github.io/simhelpers/index.html

Kenny, A., & Wolock, C. (2023). *SimEngine: A Modular Framework for Statistical Simulations in R*. https://CRAN.R-project.org/package=SimEngine

Lang, M., Bischl, B., & Surmann, D. (2017). batchtools: Tools for R to work on batch systems. *Journal of Open Source Software*, *2*(10), 135. https://doi.org/10.21105/joss.00135

Linner, S., Moreira Lara, I., & Lehmann, K. (2022). *tidyMC: Monte Carlo Simulations Made Easy and Tidy*. https://github.com/stefanlinner/tidyMC

Orcan, F. (2021). MonteCarloSEM: An R Package to Simulate Data for SEM. *International Journal of Assessment Tools in Education*, *8*(3), 704–713.

Parsons, R. (2022). *simMetric: Metrics (with Uncertainty) for Simulation Studies that Evaluate Statistical Methods*. Queensland University of Technology. https://doi.org/10.25912/RDF_1665114451679

Scheer, M. (2020). *simTool: Conduct Simulation Studies with a Minimal Amount of Source Code*. https://CRAN.R-project.org/packages=simTool

Shilane, D., Budugutta, S., & Bansal, M. (2023). *simitation: Simplified Simulations*. https://CRAN.R-project.org/package=simitation

Ucar, I., Smeets, B., & Azcorra, A. (2019). simmer: Discrete-event simulation for R. *Journal of Statistical Software*, *90*(2), 1–30. https://doi.org/10.18637/jss.v090.i02

Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L. D., François, R., Grolemund, G., Hayes, A., Henry, L., Hester, J., Kuhn, M., Pedersen, T. L., Miller, E., Bache, S. M., Müller, K., Ooms, J., Robinson, D., Seidel, D. P., Spinu, V., … Yutani, H. (2019). Welcome to the Tidyverse. *Journal of Open Source Software*, *4*(43), 1686. https://doi.org/10.21105/joss.01686

Yu, B., & Kumbier, K. (2020). Veridical data science. *Proceedings of the National Academy of Sciences*, *117*(8), 3920–3929. https://doi.org/10.1073/pnas.1901326117