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]

1 Graduate Group in Biostatistics, University of California, Berkeley **2** Department of Statistics, University of California, Berkeley ¶ Corresponding author * These authors contributed equally.

DOI: N/A

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: 01 January 1970

Summary

Data science simulation studies occupy an important role in data science research as a means to gain insight into new and existing statistical methods. Whether as a means 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, simulations serve as statistical sandboxes that open a path toward otherwise inaccessible discoveries. Yet creating high-quality simulation studies typically involves a number of repetitive and error-prone coding tasks, such as implementing data-generating processes (DGPs) and statistical methods, sampling from these DGPs, parallelizing computation of simulation replicates, summarizing metrics, and visualizing, documenting, and saving results. While this administrative overhead is necessary to reach the end goals of a given data science simulation, it is not sufficient, as the data scientist must navigate a number of important judgment calls such as the choice of data settings, baseline statistical methods, associated parameters, and evaluation metrics for scientific relevancy. The scientific context varies drastically from one study to the next while the simulation scaffolding remains largely similar; yet simulation code repositories often lack the flexibility to easily allow for reuse in novel settings or even simple extension when new questions arise in the original context.

simChef addresses the need for an intuitive, extensible, and reusable framework for data science simulations. Drawing substantially from the Predictability, Computability, and Stability (PCS) framework (Yu & Kumbier, 2020), simChef empowers data scientists to focus their attention toward the scientific best practices encompassed by PCS by removing many of the administrative burdens of simulation design with an intuitive tidy grammar of data science simulations and automated interactive R Markdown documentation.

The Journal of Open Source Software

A powerful grammar of data science simulations

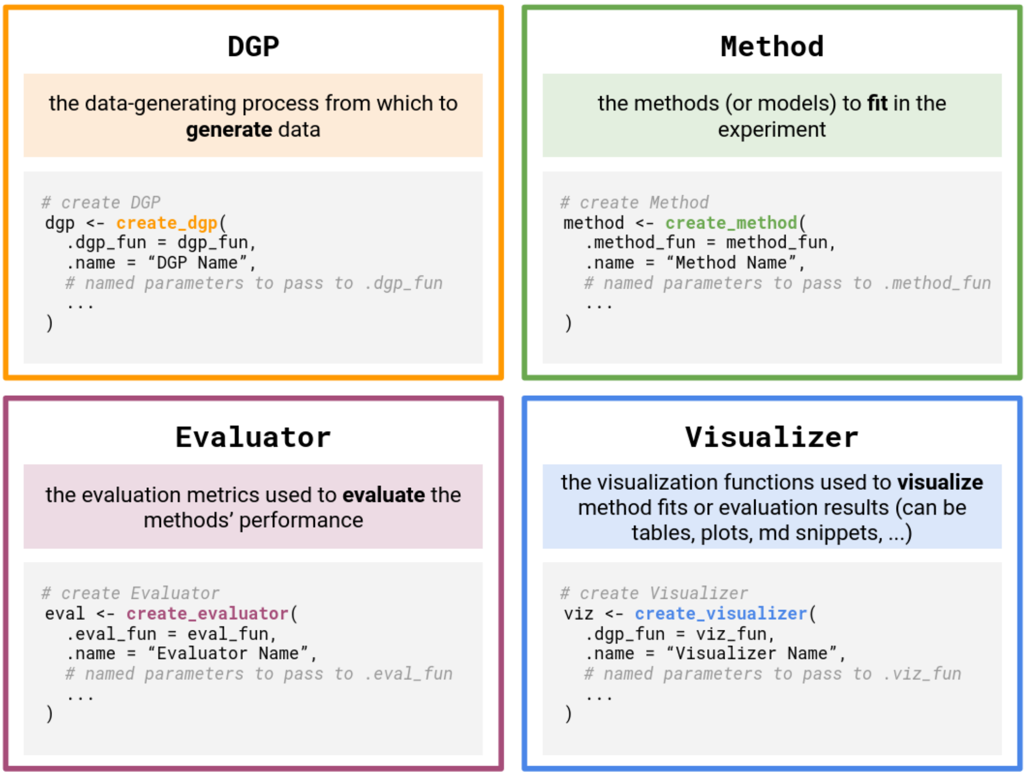

Figure 1: `simChef` provides four classes which implement distinct simulation objects in an intuitive and modular manner: `DGP`, `Method`, `Evaluator`, and `Visualizer`.

Inspired by the tidyverse (Wickham et al., 2019), `simChef` develops an intuitive grammar of simulation studies:

```
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,
    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)
```

Internally, `simChef` provides a modular conceptualization of data science simulations using four `R6` (Chang, 2022) classes, portrayed Figure 1: `DGP`, `Method`, `Evaluator`, and `Visualizer`. Users create or reuse custom functions (`dgp_fun`, `method_fun`, `eval_fun`, and `viz_fun` above) aligned with their scientific goals. The custom functions are then optionally parameterized and encapsulated in one of the corresponding classes via a `create_*` method together with optional constant parameters (e.g., `sd` above).

A fifth `R6` class, `Experiment`, serves as a concrete implementation of the user's intent to answer a specific scientific question. The `Experiment` stores references to the first four objects along with the `DGP` and `Method` parameters that should be varied and combined during the simulation run. Parameters that are common across the users functions can be added jointly (as is the case for the `n` parameter to `dgp_fun1` and `dgp_fun2` above) and can have arbitrary data type (such as `scalar_valued_param` and `vector_valued_param` to `method_fun`).

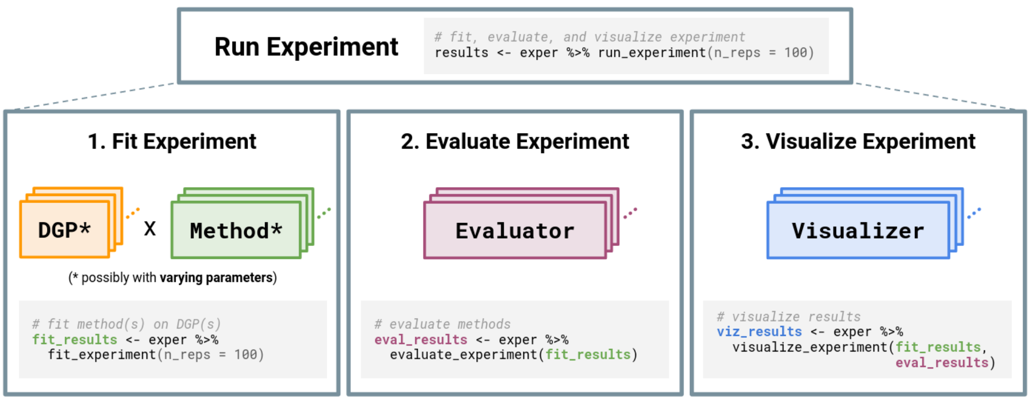

Figure 2: The `Experiment` class handles relationships between the four classes portrayed in Figure 1. Experiments may have multiple `DGP` and `Method` objects, which are combined across the Cartesian product of their varying parameters (represented by `\*`). Once computed, each `Evaluator` and `Visualizer` take in simulation replicates, while `Visualizer` additionally receives evaluation summaries.

The `Experiment` class flexibly handles the computation of simulation replicates in parallel using `future` (Bengtsson, 2021). 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` (Figure 2). 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 first call to `run_experiment` in the above example, 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. 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. Thus, there are a total of 108 DGP-method-parameter combinations in the experiment, each of which is replicated 100 times. Figure 3 provides a detailed schematic of the `run_experiment` workflow, along with the expected inputs to and outputs from user-defined functions.

Users can also 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 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.

Automated documentation in an interactive R Markdown template 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 interactive R Markdown document, either for iterative design and analysis of the simulation or to provide a high-quality overview that can be easily shared. We provide an example of the simulation documentation at this link and corresponding source code is available on GitHub at PhilBoileau/simChef-case-study.

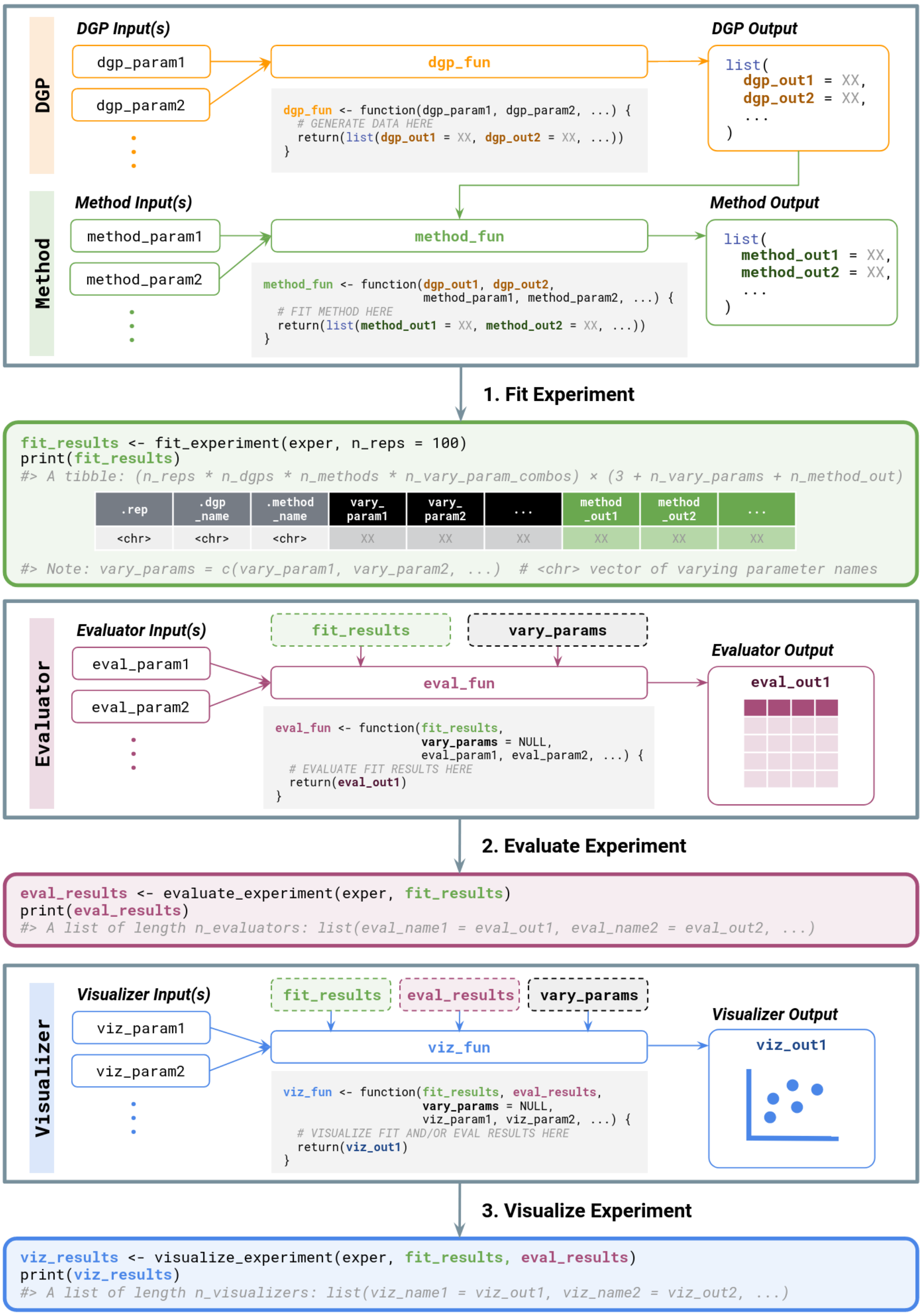

Figure 3: Detailed schematic of the `run_experiment` workflow.

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

Chang, W. (2022). *R6: Encapsulated classes with reference semantics*.

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