The Journal of Open Source Software

Empirical: A scientific software library for research, education, and public engagement

Anya Vostinar[4], **Alexander Lalejini**[7], **Charles Ofria**[1,2,3], **Emily Dolson**[1,2,3], **and Matthew Andres Moreno**[4,5]

1 BEACON Center for the Study of Evolution in Action **2** Computer Science and Engineering, Michigan State University **3** Ecology, Evolutionary Biology, and Behavior, Michigan State University **4** Computer Science, Carleton College **5** Ecology and Evolutionary Biology, University of Michigan **6** Center for the Study of Complex Systems, University of Michigan **7** Computer Science, Grand Valley State University

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor:

Submitted: 14 December 2023
Published: unpublished

Summary

Empirical is a C++ library designed to promote open science and facilitate the development of scientific software that is efficient, reliable, and easily distributable to researchers and non-experts alike. Specifically, the library sets out to fulfill the following goals:

1. **Utility:** Empirical tools streamline common scientific computing tasks such as configuration, end-to-end data management, and mathematical manipulations.
2. **Efficiency:** Empirical implements general-purpose data structures and algorithms that emphasize computational efficiency to support scientific computing workloads.
3. **Reliability:** Empirical provides sophisticated debug-mode instrumentation including audited memory management and safety checked versions of standard library containers.
4. **Distributability:** Empirical is highly portable, uses common data formats, and facilitates compile-to-web app development with object-oriented bindings for Emscripten/WebAssembly GUI elements, all with the goal of building broadly accessible scientific software.

Statement of Need

High quality open-science tools improve code quality, scientific rigor, and ease of replication or extension for scientific software. Empirical's debugging suite combats C++ programming pitfalls, such as iterator invalidation, memory leakage, and out-of-bounds indexing. Throughout, library design achieves both performance and safety through compile-time toggling of checks for undefined or incorrect behavior.

Unfortunately, in practice, scientific software is often difficult to obtain, install, or use. Modern web-based interfaces give computational research the potential to better embody open science objectives by empowering easier and more complete access (Woelfle et al., 2011). Empirical leverages modern web technology to provide browser-based interactive interfaces for C++ source code.

Empirical Features

Better Code for Scientific Software

Empirical components are subjected to structured code review, unit testing with coverage tracking, and other best practices detailed in our documentation. Effort invested into optimization of the library's utilities enables developer-users to more easily produce safe and efficient software, especially for new developers. We provide a template project that streamlines laying out crosscompilation boilerplate.

As an example of Emprical's utility, the library provides a configuration framework that includes utilities to

- create documented configuration parameters with default values in a single line of C++ code,
- adjust parameters via configuration files, command line flags, URL query parameters, or in-browser GUIs,
- perform on-the-fly configuration adjustments, and
- support independent configuration subsystems.

High-quality software needs a robust, inclusive, and diverse community of users and contributors. Our development practices reflect this priority.

Realizing the Promise of Emscripten-based Web UIs

Educational editions of scientific software promote classroom learning and citizen science. The Emscripten compiler enables an existing native codebase to additionally compile to the web (Zakai, 2011). Browser-based delivery can yield particularly effective public-facing apps due to easy access and compelling interfaces.

Empirical amplifies Emscripten by fleshing out its interface for interaction with browser elements. DOM elements are bound to corresponding C++ objects (e.g., `emp::Button` manages a `<button>` and `emp::Canvas` manages a `<canvas>`) and are easily manipulated from within C++. Empirical also packages collections of prefabricated web widgets (e.g., configuration managers or collapsible data displays). These tools simplify generating a mobile-friendly, web-based GUI.

A live demo of Empirical widgets, presented alongside their source C++ code, is available here.

Runtime Efficiency

WebAssembly's runtime efficiency — achieving 50% to 90% of native performance (Jangda et al., 2019) — has driven adoption in web development (Haas et al., 2017) and enabled new possibilities for browser-based scientific computation. For example, Avida-ED leverages WebAssembly to incorporate sophisticated agent-based evolution models into classroom activities.

More broadly, Empirical provides optimized tools for performance-critical tasks. For example, `emp::BitArray` and `emp::BitVector` are faster drop-in replacements for their standard library equivalents (`std::bitset` and `std::vector<bool>`) with extensive additional functionality. More fundamentally, Empirical's header-only design prioritizes ease of use and runtime performance, albeit at the cost of longer compilation times.

Debugging

Although performant, C++'s permissiveness to out-of-bounds indexing or memory managment errors can emperil validity of generated data and analyses. Standard library vendors — like `libstdc++`, `libc++`, and `stl` — provide some runtime safety features, but these are incomplete and poorly documented[1]. Empirical supplements vendor offerings with debug mode standins for standard library containers and even raw pointers that can identify memory leaks and invalid memory access.

[1] For example, neither GCC 10.3 nor Clang 12.0.0 detect `std::vector` iterator invalidation when appending to a `std::vector` happens to fall within existing allocated buffer space (GCC live example; Clang live example). Clang 12.0.0's sanitizers also fail to detect this iterator invalidation (live example).

Developers typically compensate for C++'s missing guardrails with external toolchains like Valgrind, GDB, and sanitizers. Although mature, such tooling suffers substantial limitations[2], particularly for WASM compiled with Emscripten. Although Emscripten provides some sanitizer support and other debugging features, Empirical's safety features offset remaining limitations, such as the lack of a steppable debugger.

Outlook and Future Plans

Empirical remains under active development. Current priorities include web-friendly refinements (e.g., file management, rich text handling) and additional step-by-step tutorials for new users. That said, Empirical has largely converged to API stability, and releases are archived on Zenodo for those who depend on them (Ofria et al., 2020).

Empirical already underlies major projects within digital evolution, artificial life, and genetic programming. To benefit the broader scientific software and open science community, we look forward to welcoming new collaborations and supporting a wider collection of end-users.

Related Software Packages

Several projects pursue objectives related to Empirical's.

RepastHPC

RepastHPC, accessible at https://repast.github.io/, is a C++ modeling framework targeted to high-performance computing (Collier & North, 2013; North et al., 2013). A Java-based counterpart, Repast Simphony, provides interactive GUI support.

Boost C++ Libraries

Boost C++ Libraries, available at https://www.boost.org/, implement a broad portfolio of software components. However, Boost lacks tools for web-based GUI, configuration management, or data management tailored to scientific software.

Emscripten

Emscripten provides cross-compilation from C++ to WebAssembly and available at https://emscripten.org/ (Zakai, 2011). Empirical furnishes a complementary high-level interface to Emscripten intrinsics.

Cheerp

Cheerp, another C++ to WebAssembly compiler, is available at https://leaningtech.com/cheerp/. Like Emscripten, Cheerp provides primariliy low-level APIs for browser interaction.

Non-C++ Comparable Software

- TinyGo
- WebIO
- GWT
- yew
- Pyodide (Droettboom & Developers, 2021)
- Shiny (Chang et al., 2020)

[2] For example, neither GCC 10.3 nor Clang 12.0.0 detect `std::vector` iterator invalidation when appending to a `std::vector` happens to fall within existing allocated buffer space (GCC live example; Clang live example). Clang 12.0.0's sanitizers also fail to detect this iterator invalidation (live example).

Projects Using the Software

- Aagos (Gillespie et al., 2018): model to test impact of environmental change on genetic architecture evolution.
- conduit (Moreno et al., 2020): library for best-effort communication in high-performance computing.
- Dishtiny (Moreno & Ofria, 2019): agent-based model to studdy major transitions in evolution.
- ecology in evolutionary computation explorer (Dolson & Ofria, 2018): interactive visualization of ecological interaction networks in evolutionary computation.
- Symbulation (Vostinar, 2017): agent-based model for evolution of parasitism, mutualism, and commensalism.
- SignalGP Moreno et al. (2021): an event-driven genetic programming substrate.
- Model of cancer evolution on an oxygen gradient.

Acknowledgements

This research was supported in part by NSF grants DEB-1655715 and DBI-0939454, by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-1424871, by Michigan State University through the computational resources provided by the Institute for Cyber-Enabled Research, and by the Eric and Wendy Schmidt AI in Science Postdoctoral Fellowship, a Schmidt Futures program. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

Chang, W., Cheng, J., Allaire, J., Xie, Y., & McPherson, J. (2020). *Shiny: Web application framework for r*. https://CRAN.R-project.org/package=shiny

Collier, N., & North, M. (2013). Parallel agent-based simulation with repast for high performance computing. *SIMULATION*, *89*(10), 1215–1235. https://doi.org/10.1177/0037549712462620

Dolson, E., & Ofria, C. (2018). Ecological theory provides insights about evolutionary computation. *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 105–106. https://doi.org/10.1145/3205651.3205780

Droettboom, M., & Developers, P. (2021). *Shiny: Web application framework for r*. https://pyodide.org/

Gillespie, L., Dolson, E., Lalejini, A., & Ofria, C. (2018). Changing environments drive the separation of genes and increased evolvability in NK-inspired landscapes. *Late Breaking Abstract at The 2018 Conference on Artificial Life*.

Haas, A., Rossberg, A., Schuff, D. L., Titzer, B. L., Holman, M., Gohman, D., Wagner, L., Zakai, A., & Bastien, J. (2017). Bringing the web up to speed with WebAssembly. *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 185–200. https://doi.org/10.1145/3062341.3062363

Jangda, A., Powers, B., Berger, E. D., & Guha, A. (2019). Not so fast: Analyzing the performance of webassembly vs. Native code. *2019 {USENIX} Annual Technical Conference ({USENIX}{ATC} 19)*, 107–120. ISBN: 9781939133038

Lalejini, A., & Ofria, C. (2018). Evolving event-driven programs with SignalGP. *Proceedings of the Genetic and Evolutionary Computation Conference on - GECCO '18*, 1135–1142. https://doi.org/10.1145/3205455.3205523

Moreno, M. A., & Ofria, C. (2019). Toward Open-Ended Fraternal Transitions in Individuality. *Artificial Life*, *25*(2), 117–133. https://doi.org/10.1162/artl_a_00284

Moreno, M. A., Papa, S. R., Lalejini, A., & Ofria, C. (2021). *SignalGP-lite: Event driven genetic programming library for large-scale artificial life applications*. arXiv. https://doi.org/10.48550/ARXIV.2108.00382

Moreno, M. A., rodsan0, perryk12, & Badger, C. (2020). *mmore500/conduit: Initial release* (Version v0.1.0). Zenodo. https://doi.org/10.5281/zenodo.4118608

North, M. J., Collier, N. T., Ozik, J., Tatara, E. R., Macal, C. M., Bragen, M., & Sydelko, P. (2013). Complex adaptive systems modeling with repast simphony. *Complex Adaptive Systems Modeling*, *1*(1), 1–26. https://doi.org/10.1186/2194-3206-1-3

Ofria, C., Moreno, M. A., Dolson, E., Lalejini, A., rodsan0, Fenton, J., perryk12, Jorgensen, S., hoffmanriley, grenewode, & al., et. (2020). *Devosoft/empirical*. https://doi.org/10.5281/zenodo.2575606

Vostinar, A. E. (2017). *Suicide, signals, and symbionts: Evolving cooperation in agent-based systems*. Michigan State University. ISBN: 978-0-355-07992-0

Woelfle, M., Olliaro, P., & Todd, M. H. (2011). Open science is a research accelerator. *Nature Chemistry*, *3*(10), 745–748. https://doi.org/10.1038/nchem.1149

Zakai, A. (2011). Emscripten: An LLVM-to-JavaScript compiler. *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion*, 301–312. https://doi.org/10.1145/2048147.2048224