Utopia: A Comprehensive and Collaborative Modeling Framework for Complex and Evolving Systems

Lukas Riedel [1,2,3], **Benjamin Herdeanu** [1,4], **Harald Mack** [1], **Yunus Sevinchan** [1], **and Julian Weninger**[5]

1 Institute of Environmental Physics, Heidelberg University, Germany **2** Interdisciplinary Center for Scientific Computing, Heidelberg University, Germany **3** Heidelberg Graduate School of Mathematical and Computational Methods for the Sciences, Heidelberg University, Germany **4** Heidelberg Graduate School for Physics, Heidelberg University, Germany **5** Department of Biochemistry, University of Geneva, Switzerland

DOI: 10.21105/joss.02165

Software

- Review
- Repository
- Archive

Editor: Arfon Smith

Reviewers:

- @platipodium
- @Shibabrat
- @szhorvat

Submitted: 07 March 2020
Published: 02 September 2020

Summary

Utopia is a modeling framework that supports the entire workflow of computational scientists in the field of complex and evolving systems. It is designed to facilitate collaborative research and flexible model development while maintaining high individual freedom in implementation and analysis. Utopia includes a C++ library for model implementations and data writing and a Python frontend for simulation control, data analysis, and plotting. A basic set of models is distributed alongside the framework.

Applications of Utopia span a wide range both in terms of the target audience and features relevant in each domain: For researchers, Utopia offers a rich toolkit for model implementation, data analysis, and generation of simulation data—up to and including large-scale simulations on distributed, high-performance computing hardware. In a teaching context, students can focus on the investigation of readily provided models using their own machines, e.g. by performing easily available sensitivity analysis. Furthermore, by developing new library functionality, collaborators can easily share new features and thus enhance Utopia's applicability for all users of the framework. This makes Utopia a valuable tool for both research in and teaching of complex and evolving systems.

Utopia is available as Docker image from DockerHub[1] or can be compiled from source.

Research of Complex Systems

Many physical, environmental, and socio-cultural questions are studied in the field of complex and evolving systems (Holland, 2006; Levin, 2003). These systems feature a hierarchic, self-organized structure with non-linear interactions between their compartments and often exhibit emergent macroscopic properties. We call such systems "evolving" when the dimension and structure of their state space or the nature of their internal interactions can change under varying external forcings, in contrast to systems that merely traverse a static state space volume.

Given their high-dimensional, non-linear, and oftentimes open and fundamentally interconnected nature, laboratory-scale versions of complex and evolving systems often do not exist and investigating them experimentally is difficult. This is why they are typically studied through heuristic computer models, in search of an abstract understanding of their fundamental processes. As model constants and effective parameters remain unknown, researchers sample the usually

[1]https://hub.docker.com/r/ccees/utopia

high-dimensional parameter spaces of these models to find qualitatively distinguishable dynamic regimes. This produces high-dimensional output data and therefore requires an intuitive and convenient way of loading, analyzing, and visualizing such data.

Consequently, the software for implementing and analyzing computational models becomes the main source of scientific insight. It is therefore required to be not only performant but also reliable. At the same time, complicated models necessarily demand sophisticated software and large amounts of code that needs to be produced collaboratively by multiple researchers if a critical project size is exceeded. To verify and maintain its functionality, the software must be comprehensibly tested. Insights gained from the implemented models need to be easily communicated among peers. In recent years, software engineering workflows have therefore been adopted in the field of computational science. This facilitated satisfying the aforementioned demands in scientific software development, helped to avoid redundant re-implementations and enabled efficient, collaborative development of research software in large groups of researchers (Storer, 2017).

Representations of complex and evolving systems are often created using variations of three main modeling techniques: cellular automata (CA), network models, and agent-based models (ABM). CA are grid-based methods with update rules depending on the current states of grid cells and their respective cell neighborhoods and are used to investigate the emergence of macroscopic features from cell-local interactions (Chopard et al., 2002; Wolfram, 1983). In contrast, network models focus on heterogeneous and possibly dynamic interaction structures between entities (Albert & Barabási, 2002; Boccaletti et al., 2006). Finally, the constituents of ABMs are heterogeneous individuals with dynamic behavior in possible interplay with their environment. Among applications in nearly every scientific field, ABMs are also used to model traffic and population dynamics as well as biological evolution (Macal, 2016).

We summarize the typical workflow of a computational scientist investigating complex and evolving systems via these modeling techniques in a computer model *lifecycle* consisting of four stages:

1. Conceptualization of the research question
2. Implementation of a computer model
3. Generation of simulation data
4. Analysis of the data and extraction of results

These are intrinsically linked and insights from one stage might necessitate a re-iteration of a previous stage. A comprehensive modeling framework should support researchers and facilitate software engineering workflows through all of these stages.

State of the Art

Several open-source frameworks for modeling complex and evolving systems in different programming languages are readily available (Cardinot et al., 2019; Masad & Kazil, 2015; Vahdati, 2019). NetLogo is an especially popular scientific software environment with a similar premise as Utopia (Wilensky, 1999). As a framework for agent-based models, it provides a graphical user interface for an immediate and easily accessible investigation of simulations and its own programming language to support the development of models by scientists with little programming experience. It also ships with a model library including numerous models from several fields of research.

In our perception, however, none of these frameworks were able to support all stages of the aforementioned model lifecycle without compromising either flexibility or performance. Several frameworks also would not support the software engineering procedures we deemed crucial for collaboratively developing models in our research group. With Utopia, we focus on the generation and evaluation of multi-dimensional data in parameter space scans and granting programmers the entire feature range of the C++ and Python programming languages by providing slim and open interfaces and an extensive function library. With this orientation, we hope that Utopia serves as a valuable addition to the existing range of modeling frameworks that aim to investigate complex and evolving systems.

Feature Overview

We devised Utopia as a single framework supporting the four stages of a computer model's lifecycle. Utopia models consist of the C++ model implementation, customized analysis and plotting functions, model tests, and configuration files defining default parameters.

Conceptualizing a model (stage 1) requires both a set of common building blocks and routines, and adequate freedom to explore and investigate new scenarios. Utopia supplies a library for building models that defines a common language for all its users through its nomenclature and functional interplay. This facilitates creating computational representations of abstract concepts. While models may be integrated into the framework itself, its entire functionality is available when using Utopia as a regular dependency for independent model development. Moreover, Utopia can be used with any computer model that either conforms to its configuration data input and simulation data output schemes directly or for that appropriate interfaces are added.

For implementing a specific model (stage 2), Utopia provides a `Model` base class and a template library for common operations and effortless framework integration. We call this the Utopia *backend* which is written in modern C++ and employs the C++17 standard revision. The base class provides the scaffolding for implementing dynamics and integrates into the Utopia modeling workflow by providing the interfaces for configuration input and data output as well as for coupling several models into one. Recurring tasks like building grids and networks, managing CA and ABMs, and applying state update rules on entities of any model type can be handled conveniently using the Utopia `core` library. This library uses Armadillo as high-level linear algebra backend (Sanderson & Curtin, 2018, 2016). Furthermore, the `dataIO` library provides versatile capabilities for storing hierarchically nested simulation data in HDF5 files (The HDF Group, 1997). It wraps around the HDF5 C library and supports numerous complicated data output tasks that can be defined and controlled via configuration files.

The Utopia *frontend* manages the configuration and execution of simulations (stage 3). It is implemented as the `utopya` Python package which parses user input to generate a configuration, invokes the model executables in parallel, and reports on the progress throughout a simulation run. Simulations can be controlled both from an interactive Python session or via a command line interface. An important user experience concept of the frontend is to provide reasonable default parameters on all levels while allowing to specify custom values where desired. This is achieved by using a hierarchical, dictionary-like configuration structure and multiple sets of default configuration values, that can be updated by the user. With this structure, parameters can be supplied both via the Python interface or YAML configuration files. Utopia utilizes the `paramspace` Python package for conveniently defining parameter sweeps in this hierarchical structure (Sevinchan, 2019).

For handling multi-dimensional, hierarchical data (stage 4), the frontend interfaces with the `dantro` Python package to load, process, and visualize the HDF5 data generated by model simulations (Sevinchan, Herdeanu, & Traub, 2020). Like the simulation runs, data processing and the creation of plots and animations are specified using a set of configuration parameters which can be supplied in Python code or via YAML configuration files. Utopia provides an extensive and extensible set of universal and model-specific plot functions for simulation data plots of a single time step, timelines of model characteristics, as well as visualizations of parameter space scans.

Utopia aims to increase model reliability by providing easy means of implementing tests alongside a model. These tests can be C++ unit tests utilizing the Boost Test library or Python-based tests on the simulation output that assert the expected macroscopic behavior. All tests are carried out as part of an automated test suite in the GitLab CI/CD, and thus provide rapid feedback to model developers. This helps to address the practical aspects of testing scientific software (Kanewala & Bieman, 2014).

Application in Research and Teaching

More than 20 Bachelor, Master, and Ph.D. students in our research group have employed Utopia as the sole software framework for their modeling-based research projects so far. Current PhD projects use Utopia to investigate the feedback between environment and evolving populations with CA and ABMs, the evolution of ecological interaction networks, the emergence of cooperation in dynamic social interaction networks, and the development of geometric and polarity properties of the basilar papilla in agent-based vertex models.

Utopia has also been used for teaching complex and evolving environmental systems in the M.Sc. Physics curriculum at the Department of Physics and Astronomy, Heidelberg University. In seminars, students were able to implement their own models over the course of one semester, even if they started with very limited programming experience. Additionally, student exercises that involved investigating models discussed in lectures have been conducted with Utopia.

Showcase

In this section, we demonstrate the capabilities and the typical workflow of Utopia with the `ForestFire` model that simulates forest fires with a lightning probability and an immediate burn-down of connected tree clusters (Drossel & Schwabl, 1992). It implements a cellular automaton with two states, `empty` and `tree`, and a simple set of update rules for each simulated iteration:

- If a cell is `empty`, it is set to `tree` with a growth probability.
- If a cell is `tree`, it may spontaneously ignite with a lightning probability and is then set to `empty`.
- If a cell is ignited, all cells in its neighborhood with state `tree` are unconditionally ignited. This rule is then recursively applied to all other ignited cells.

The following configuration file is used to perform a parameter scan of the model. It globally defines the number of iteration steps and the spatial resolution of the model. The sweep is performed over three different lightning probabilities and ten different random number generator seeds. Additionally, we set ten different tree densities as initial condition. The sweep over these values is coupled to the sweep over the random number generator seeds. In total, 30 simulations are executed.

```yaml
# run-cfg.yml
---
# Tell Utopia to perform a parameter sweep
perform_sweep: true

# Configure the parameter space
parameter_space:
  # Total iteration steps
  num_steps: 1000

  # Sweep over random number generator seeds
  seed: !sweep
    default: 42
    range: [10]

  # ForestFire model configuration
```

```yaml
ForestFire:
  # Set the grid resolution
  cell_manager:
    grid:
      resolution: 512

    # Set initial tree density
    cell_params:
      # Sweep over 10 different values
      # Couple this sweep to the sweep over 'seed'
      p_tree: !coupled-sweep
        target_name: seed
        default: 0.3
        linspace: [0.3, 0.4, 10]

  # Sweep over lightning probability
  p_lightning: !sweep
    default: 1.0e-4
    values:
      - 1.0e-3
      - 1.0e-4
      - 1.0e-5
```

The simulations are conveniently executed via the Utopia command line interface,

```
utopia run ForestFire run-cfg.yml
```

This performs the parameter sweep, stores the simulation data, and immediately runs the default analysis and plotting scripts. Custom plotting configurations can be supplied via the `--plots-cfg` option, and already generated simulation data can be re-evaluated with the command

```
utopia eval ForestFire
```

Among the standard plots for this model are snapshots of the cellular automaton state or visualizations of connected tree clusters, as displayed in Figure 1. These single plots can also be joined to movie clips by changing one configuration line. Additionally, Figure 2 depicts the output of an analysis and plotting script for a parameter scan of the model and displays a time-series of the mean tree densities for the different lightning probabilities.

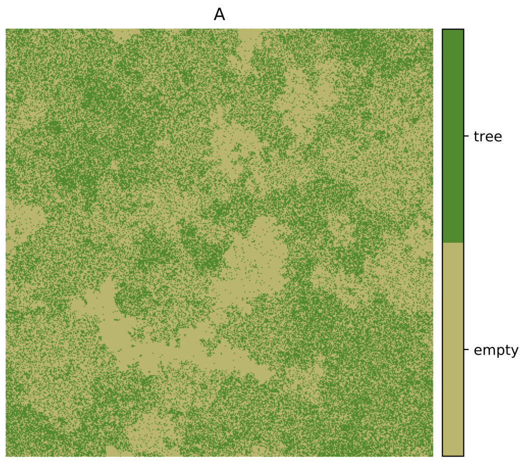

Figure 1: Snapshot of a single `ForestFire` simulation after 1000 iterations. **A**: The current state of the cellular automaton. Cells with state `tree` are colored green, cells with state `empty` are colored ocher. **B**: Visualization of connected tree clusters, where neighboring cells with state `tree` are shown in the same color and `empty` cells are white.

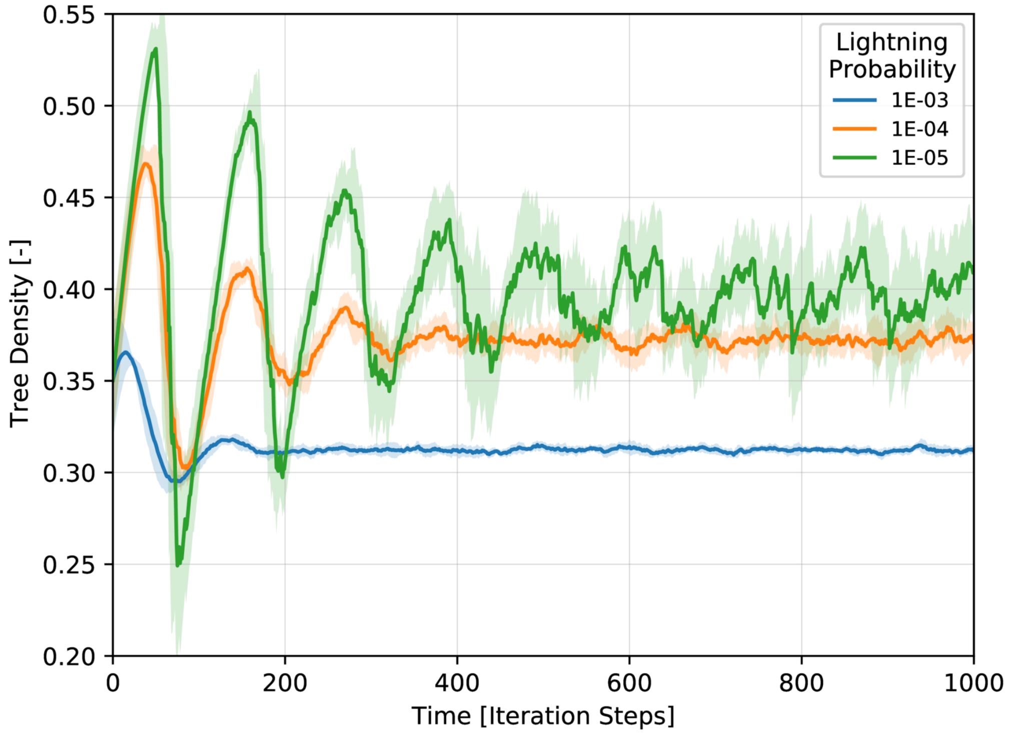

Figure 2: Parameter space time-series of the `ForestFire` simulation. The line plots depict the mean tree density calculated over all simulations with the same lightning probability. The shaded areas denote the respective standard deviation.

Acknowledgements

We gratefully acknowledge the contributions by Narek Baghumian, Lorenzo Biasi, Unai Fi-scher Abaigar, Thomas Gaskin, Leonhard Holtmann, Christian Kobrow, Fabian Krautgasser, Daniel

Lake, Hendrik Leusmann, Peter Manshausen, Robert Rauschen, Soeren Rubner, Laila Pilar Schmidt, Simeon Schreib, Lukas Siebert, Sebastian Stezura, Jeremias Traub, and Josephine Westermann to the inception, development, and maintenance of Utopia. We also extend our gratitude to Kurt Roth for the guiding discussions that lead to its development.

Related Work

A conference contribution on how our research group profited from developing Utopia as a collaborative framework for computational models of complex and evolving systems was published in the proceedings of the International Conference on Computational Science (ICCS) 2020 (Sevinchan, Herdeanu, Mack, et al., 2020). An article presenting the design and features of Utopia in full detail is in preparation (Herdeanu et al., 2020).

References

Albert, R., & Barabási, A.-L. (2002). Statistical mechanics of complex networks. *Reviews of Modern Physics*, *74*(1), 47–97. https://doi.org/10.1103/RevModPhys.74.47

Boccaletti, S., Latora, V., Moreno, Y., Chavez, M., & Hwang, D.-U. (2006). Complex networks: Structure and dynamics. *Physics Reports*, *424*(4), 175–308. https://doi.org/10.1016/j.physrep.2005.10.009

Cardinot, M., O'Riordan, C., Griffith, J., & Perc, M. (2019). Evoplex: A platform for agent-based modeling on networks. *SoftwareX*, *9*, 199–204. https://doi.org/10.1016/j.softx.2019.02.009

Chopard, B., Dupuis, A., Masselot, A., & Luthi, P. (2002). Cellular automata and lattice boltzmann techniques: An approach to model and simulate complex systems. *Advances in Complex Systems*, *05*(02n03), 103–246. https://doi.org/10.1142/S0219525902000602

Drossel, B., & Schwabl, F. (1992). Self-organized critical forest-fire model. *Phys. Rev. Lett.*, *69*, 1629–1632. https://doi.org/10.1103/PhysRevLett.69.1629

Herdeanu, B., Mack, H., Riedel, L., & Sevinchan, Y. (2020). *Utopia: A comprehensive modeling framework for complex and evolving systems*. In preparation.

Holland, J. H. (2006). Studying complex adaptive systems. *Journal of Systems Science and Complexity*, *19*(1), 1–8. https://doi.org/10.1007/s11424-006-0001-z

Kanewala, U., & Bieman, J. M. (2014). Testing scientific software: A systematic literature review. *Information and Software Technology*, *56*(10), 1219–1232. https://doi.org/10.1016/j.infsof.2014.05.006

Levin, S. (2003). Complex adaptive systems: Exploring the known, the unknown and the unknowable. *Bulletin of the American Mathematical Society*, *40*(1), 3–19. https://doi.org/10.1090/S0273-0979-02-00965-5

Macal, C. M. (2016). Everything you need to know about agent-based modelling and simulation. *Journal of Simulation*, *10*(2), 144–156. https://doi.org/10.1057/jos.2016.7

Masad, D., & Kazil, J. (2015). Mesa: An agent-based modeling framework. In K. Huff & J. Bergstra (Eds.), *Proceedings of the 14th Python in Science Conference* (pp. 51–58). https://doi.org/10.25080/Majora-7b98e3ed-009

Sanderson, C., & Curtin, R. (2018). A user-friendly hybrid sparse matrix class in C++. In J. H. Davenport, M. Kauers, G. Labahn, & J. Urban (Eds.), *Mathematical software – ICMS 2018* (pp. 422–430). Springer International Publishing. https://doi.org/10.1007/978-3-319-96418-8_50

Sanderson, C., & Curtin, R. (2016). Armadillo: A template-based C++ library for linear algebra. *Journal of Open Source Software*, *1*(2), 26. https://doi.org/10.21105/joss.00026

Sevinchan, Y. (2019). *Paramspace, Version 2.3.1*. Python Package Index (PyPI). https://pypi.org/project/paramspace/

Sevinchan, Y., Herdeanu, B., Mack, H., Riedel, L., & Roth, K. (2020). Boosting group-level synergies by using a shared modeling framework. In V. V. Krzhizhanovskaya, G. Závodszky, M. H. Lees, J. J. Dongarra, P. M. A. Sloot, S. Brissos, & J. Teixeira (Eds.), *Computational Science – ICCS 2020* (Vol. 12143, pp. 442–456). Springer International Publishing. https://doi.org/10.1007/978-3-030-50436-6_32

Sevinchan, Y., Herdeanu, B., & Traub, J. (2020). dantro: A Python package for handling, transforming, and visualizing hierarchically structured data. *Journal of Open Source Software*, *5*(52), 2316. https://doi.org/10.21105/joss.02316

Storer, T. (2017). Bridging the Chasm: A Survey of Software Engineering Practice in Scientific Programming. *ACM Computing Surveys (CSUR)*, *50*(4). https://doi.org/10.1145/3084225

The HDF Group. (1997). *Hierarchical Data Format, Version 5*. http://www.hdfgroup.org/HDF5/

Vahdati, A. (2019). Agents.jl: Agent-based modeling framework in Julia. *Journal of Open Source Software*, *4*(42), 1611. https://doi.org/10.21105/joss.01611

Wilensky, U. (1999). *NetLogo*. Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL. http://ccl.northwestern.edu/netlogo/

Wolfram, S. (1983). Statistical mechanics of cellular automata. *Reviews of Modern Physics*, *55*(3), 601–644. https://doi.org/10.1103/RevModPhys.55.601