The Journal of Open Source Software

CellPyLib: A Python Library for working with Cellular Automata

Luis M. Antunes[1]

1 Department of Chemistry, University of Reading, Whiteknights, Reading RG6 6DX, United Kingdom

DOI: 10.21105/joss.03608

Software

- Review
- Repository
- Archive

Editor: Mehmet Hakan Satman

Reviewers:

- @blsqr
- @szhorvat
- @martibosch

Submitted: 30 July 2021
Published: 10 November 2021

Summary

Cellular Automata (CA) are discrete dynamical systems with a rich history (Ilachinski, 2001). Introduced by John von Neumann and Stanislaw Ulam in the 1940s (Von Neumann, 1951), CA have continued to fascinate, as their conceptual simplicity serves as a powerful microscope that allows us to explore the nature of computation and complexity, and the origins of emergence. Far from being an antiquated computational model, investigators are utilizing CA in novel and creative ways, such as the incorporation with Deep Learning (Mordvintsev et al., 2020; Nichele & Molund, 2017). Popularized and investigated by Stephen Wolfram in his book *A New Kind of Science* (Wolfram, 2002), CA remain premier reminders of a common theme in the study of the physical world: that simple systems and rules can give rise to remarkable complexity. They are a laboratory for the study of the origins of the complexity we see in the world around us.

`CellPyLib` is a Python library for working with CA. It provides a concise and simple interface for defining and analyzing 1- and 2-dimensional CA. The CA can consist of discrete or continuous states. Neighbourhood radii are adjustable, and in the 2-dimensional case, both Moore and von Neumann neighbourhoods are supported. With `CellPyLib`, it is trivial to create Elementary CA, and CA with totalistic rules, as these rules are provided as part of the library. Additionally, the library provides a means for creating asynchronous and reversible CA. Finally, an implementation of C. G. Langton's approach for creating CA rules using the lambda value is provided, allowing for the exploration of complex systems, phase transitions and emergent computation (Langton, 1990).

Utility functions for plotting and viewing the evolved CA are also provided. These tools make it easy to visualize the results of CA evolution, and include the option of creating animations of the evolution itself. Moreover, utility functions for computing the information-theoretic properties of CA, such as the Shannon entropy and mutual information, are included.

Statement of need

The Python software ecosystem is lacking when it comes to Cellular Automata. A web search reveals that while there are some projects dedicated to the simulation of CA, most are not general-purpose, focusing only on certain CA systems, and are generally missing a substantial test suite, hindering their future extensibility and maintainability. In short, there appears to be a dearth of robust and flexible libraries for working with CA in Python.

Currently, many scientists choose Python as their main tool for computational tasks. Though researchers can choose to implement CA themselves, this is error-prone, as there are some subtleties when it comes to correctly handling issues such as boundary conditions on periodic lattices, or constructing von Neumann neighbourhoods with radius greater than 1, for example. Researchers may be dissuaded from incorporating CA into their research if they are forced to work with unfamiliar languages and technologies, or are required to devote considerable effort to the implementation and testing of non-trivial algorithms. The availability of a user-friendly Python library for CA will likely encourage more researchers to consider these fascinating dynamical and computational systems. Moreover, having a standard implementation of CA in the Python environment helps to ensure that scientific results are reproducible. CellPyLib is a Python library aimed to meet this need, which supports the creation and analysis of models that exist on a regular array or uniform grid, such as elementary CA, and 2D CA with Moore or von Neumann neighbourhoods.

Researchers and students alike should find CellPyLib useful. Students and instructors can use CellPyLib in an educational context if they would like to learn about elementary CA and 2D CA on a uniform grid. Researchers in both the computer and physical sciences can use CellPyLib to answer serious questions about the computational and natural worlds. For example, the Abelian sandpile model included in the library can be used as part of a university course on complex systems to demonstrate the phenomenon of self-organized criticality. The same model may be used by professional physicists wishing to explore self-organized criticality more deeply.

While CellPyLib is expected to be of interest to students, educators, and researchers, there are certain scenarios in which alternate tools would be more appropriate. For example, if a researcher would like to evolve CA with a very large number of cells, or for very many iterations, in a timely fashion, then an implementation that is optimized for the specific model in question would be more appropriate. Also, if the model is not constrained to a uniform grid, then other solutions should be sought.

Example Usage

`CellPyLib` can be readily installed using `pip`:

```
$ pip install cellpylib
```

It has minimal dependencies, depending only on the commonly used libraries NumPy (Harris et al., 2020) and Matplotlib (Hunter, 2007).

The following example illustrates the evolution of the Rule 30 CA, described in `A New Kind of Science` (Wolfram, 2002), as implemented with `CellPyLib`:

```
import cellpylib as cpl

cellular_automaton = cpl.init_simple(200)

rule = lambda n, c, t: cpl.nks_rule(n, 30)

cellular_automaton = cpl.evolve(cellular_automaton, timesteps=100,
                                apply_rule=rule)
```

First, the initial conditions are instantiated using the function `init_simple`, which, in this example, creates a 200-dimensional vector consisting of zeroes, except for the component in the center of the vector, which is initialized with a value of 1. Next, the system is subjected to evolution by calling the `evolve` function. The system evolves under the rule specified through the `apply_rule` parameter. Any function that accepts the three arguments `n`, `c` and `t` can be supplied as a rule, but in this case the built-in function `nks_rule` is invoked to provide

Rule 30. The CA is evolved for 100 `timesteps`, or 100 applications of the rule to the initial and subsequent conditions.

During each timestep, the function supplied to `apply_rule` is invoked for each cell. The `n` argument refers to the neighbourhood of the current cell, and consists of an array (in the 1-dimensional CA case) of the activities (i.e. states) of the cells comprising the current cell's neighbourhood (an array with length 3, in the case of a 1-dimensional CA with radius of 1). The `c` argument refers to index of the cell under consideration. It serves as a label identifying the current cell. The `t` argument is an integer specifying the current timestep.

Finally, to visualize the results, the `plot` function can be utilized:

```
cpl.plot(cellular_automaton)
```

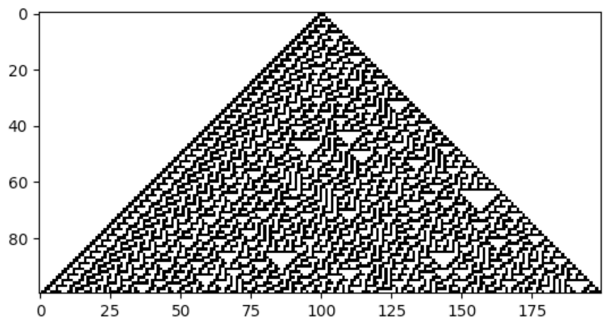

Figure 1: Rule 30, as rendered with CellPyLib.

The result is rendered, as depicted in Figure 1.

Scope

While `CellPyLib` is a general-purpose library that allows for the implementation of a wide variety of CA, it is important to note that CA constitute a very broad class of models. `CellPyLib` focuses on those that are constrained to a regular array or uniform grid, such as elementary CA, and 2D CA with Moore or von Neumann neighbourhoods.

References

Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M. H. van, Brett, M., Haldane, A., Río, J. F. del, Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, *9*(3), 90–95. https://doi.org/10.1109/MCSE.2007.55

Ilachinski, A. (2001). *Cellular Automata: A Discrete Universe*. World Scientific Publishing Company. https://doi.org/10.1142/4702

Langton, C. G. (1990). Computation at the Edge of Chaos: Phase Transitions and Emergent Computation. *Physica D: Nonlinear Phenomena*, *42*(1-3), 12–37. https://doi.org/10.1016/0167-2789(90)90064-v

Mordvintsev, A., Randazzo, E., Niklasson, E., & Levin, M. (2020). Growing neural cellular automata. *Distill*. https://doi.org/10.23915/distill.00023

Nichele, S., & Molund, A. (2017). Deep Learning with Cellular Automaton-based Reservoir Computing. *Complex Systems*, *26*(4), 319–240. https://doi.org/10.25088/complexsystems.26.4.319

Von Neumann, J. (1951). The General and Logical Theory of Automata, Cerebral Mechanisms in Behavior. The Hixon Symposium. *New York: John Wiley&Sons*.

Wolfram, S. (2002). *A New Kind of Science* (Vol. 5). Wolfram media Champaign, IL.