The Journal of Open Source Software

Melodie: Agent-based Modeling in Python

Songmin Yu [1*] and **Zhanyi Hou** [2*¶]

1 Fraunhofer Institute for Systems and Innovation Research, Germany **2** School of Reliability and Systems Engineering, Beihang University, China ¶ Corresponding author * These authors contributed equally.

DOI: 10.21105/joss.05100

Software

- Review
- Repository
- Archive

Editor: Mehmet Hakan Satman

Reviewers:

- @JoelForamitti
- @rusu24edward

Submitted: 21 January 2023
Published: 26 March 2023

Introduction

Agent-based models (ABMs) characterize physical, biological, and social economic systems as dynamic interactions among agents from a bottom-up perspective. The agents can be molecules, animals, or human beings. The interactions can be water molecules forming a vortex, ants searching for food, or people trading stocks in the market.

Agents' interactions can bring emergent properties to a system and turn it into a complex system. The core reason for using ABMs is usually to model such mechanisms. Besides, taking social economic systems as example, ABMs are also flexible to consider agents' (1) heterogeneity (e.g., wealth, risk attitude, preference, decision-making rule, etc.) based on micro-data; and (2) bounded rationality and adaptation behavior based on psychological and behavioral studies.

`Melodie` is a general framework for developing agent-based models (ABMs) in Python. It is published and maintained on the GitHub organization page of `ABM4ALL`, a developing community among agent-based modelers for sharing ideas and resources. Together with the code repository, we have also published the documentation of `Melodie`, including a tutorial explaining how a minimum example - an agent-based covid contagion model - can be developed with `Melodie` step by step.

Statement of need

Among numerous frameworks for agent-based modeling in different programming languages, `Mesa` (Kazil et al., 2020) and `AgentPy` (Foramitti, 2021) are the two open-source frameworks in Python. The object-oriented paradigm of Python seamlessly fits the "agent perspective" of ABM. Modelers can also benefit from the wealth of packages available for statistical analysis, data visualization, etc. Following the tradition of `NetLogo` (Wilensky, 1999), `Mesa` and `AgentPy` both support interactive simulation but with different focus and style.

In summary, `Melodie` is distinguished from `Mesa` and `AgentPy` by the following aspects.

First, `Melodie` separates an `environment` component from the `model` in `Mesa` and `AgentPy` for two dedicated tasks: (1) storing the macro-level variables; and (2) coordinating the agents' decision-making and interaction processes. With a separated `environment` component, the "storyline" of the model can be clearly summarized under a `run` function in the `model`. Compared to the use of `scheduler` and `step` functions in different layers in `Mesa` and the bundling of the behavior functions of agents to the `AgentList` in `AgentPy`, we think this makes it easier for users to understand the logic.

Second, `Melodie` enhances the `data_collector` component with higher configurability. Users can define functions for parsing specific data structure from the `agents` and the `environment`. For example, in a financial ABM, the transactions could be saved in the `environment` as `List[Transaction]`. Then, in the `data_collector`, the users could define a function `collect_transaction_data()` to first parse the list and to then save the results into the database.

Third, `Melodie` has wider infrastructure coverage and provides dedicated modules for scenario management.

- All input data are first registered and then loaded by a `data_loader` object into a `scenario` object. Then, as the input data container, `scenario` can be accessed by the `model` and its components, including `environment`, `data_collector`, and each `agent`.
- Melodie provides two standard classes - `DataFrameInfo` and `MatrixInfo` - with which the users can register the input dataframes and matrices, so they can be easily processed by the `data_loader` and the `scenario` objects.

In such a data flow, `Melodie` also checks if the registries are consistent with the input Excel files automatically. We think such design is helpful, especially when the scenario includes large and complicated input datasets. Having the channel through "Scenario" for delivering input data at different parts of the model is also conceptually clear. Finally, `Melodie` uses an `SQLite` database to save (1) a copy of the input data, and (2) the output data, i.e., model results. The interaction between model and database is facilitated by the `DB` module in `Melodie`. Users can easily save all the data in multiple long tables for post-processing or for sending the single `.sqlite` file to others.

Fourth, `Melodie` includes two modules that are not provided in `Mesa` and `AgentPy`: `Calibrator`, and `Trainer`. With these two modules, `Melodie` supports (1) automatic calibration of scenario parameters, and (2) evolutionary training of agents.

Fifth, `Melodie` uses the `Cython` package to accelerate its compatibility advantage over other packages like `numba`. The modules that are written in `Cython` are `agent`, `environment`, `agent_list`, and `grid`.

In the documentation, we also provide a detailed comparison between the three packages - Mesa, AgentPy, and Melodie - based on one ABM developed with the three packages. You can find the code in this repository.

Overview

The modules in the `Melodie` framework can be organized into four clusters: Model, Scenario, Modeling Manager, and Infrastructure.

Model

The modules in the Model Cluster focus on describing the target system. Developed with `Melodie`, a `model` object can contain following components:

- `agent` - makes decisions, interacts with others, and stores the micro-level variables.
- `agents` - contains a list of agents and provides relevant functions.
- `environment` - coordinates the agents' decision-making and interaction processes and stores the macro-level variables.
- `data_collector` - collects the micro- and macro-level variables from the agents and environment, and then saves them to the database.
- `grid` - constructed with `spot` objects, describes the grid (*if exists*) that the agents walk on, stores grid variables, and provides the relevant functions.
- `network` - constructed with `edge` objects, describes the network (*if exists*) that links the agents, and provides the relevant functions.

Scenario

The modules in the Scenario Cluster focus on formatting, importing, and delivering the input data to the model, including

- DataFrameInfo and MatrixInfo - used to create standard data objects for input tables.
- data_loader - loads all the input data into the model.
- scenario - contains all the input data that is needed to run the model, and can be accessed by the model and its components.

Modelling Manager

To combine everything and finally start running, the Modelling Manager Cluster includes three modules, which can be constructed and run for different objectives:

- Simulator - simulates the logic written in the model.
- Calibrator - calibrates the parameters of the scenario by minimizing the distance between model output and empirical evidence.
- Trainer - trains the agents to update their behavioral parameters for higher payoff.

Both the Calibrator and Trainer modules are based on a Genetic Algorithm (GA), and the Trainer framework is introduced in detail in Yu (2022).

Taking the Covid contagion model in the tutorial as an example, as shown below, the simulator is initialized with a config object (which includes a project name and a set of folder paths) and the class variables of the model, the scenario, and the data_loader.

```
from Melodie import Simulator
from config import config
from source.model import CovidModel
from source.scenario import CovidScenario
from source.data_loader import CovidDataLoader

simulator = Simulator(
     config = config,
     model_cls = CovidModel,
     scenario_cls = CovidScenario,
     data_loader_cls = CovidDataLoader
)
simulator.run()
```

Finally, by calling the simulator.run function, the simulation starts.

Infrastructure

The last Infrastructure Cluster includes the modules that provide support for the modules above.

- Visualizer - provides the APIs to interact with MelodieStudio for visualization.
- MelodieStudio - another library in parallel with Melodie, supports results visualization and interactive simulation in the browser.
- Config - provides the channel to define project information, e.g., project name, folder paths.
- DBConn - provides IO functions for the database.
- MelodieException - provides the pre-defined exceptions in Melodie to support debugging.

Resources

On our GitHub organization page `ABM4ALL`, apart from the `Melodie` package and its documentation, we also have published a series of example models showing how different modules can be used, including `Grid`, `Network`, `Calibrator`, `Trainer`, `Visualizer`, and `MelodieStudio`. These example models are also documented in the "Model Gallery" section in the `Melodie` documentation. Finally, for those who are familiar with `Mesa` or `AgentPy`, a comparison between `Melodie` and the two packages is provided in the documentation, based on the same Covid contagion model developed with all the three packages.

Acknowledgements

This work is not supported by any funding. Dr. Songmin Yu would like to thank the free and creative working atmosphere at Fraunhofer ISI, especially the inspiring talks and nice beer time with colleagues. Zhanyi Hou would like to thank his supervisor, Prof. Shunkun Yang, and his research partners from Beihang University for their support and guidance for programming.

References

Foramitti, J. (2021). AgentPy: A package for agent-based modeling in Python. *Journal of Open Source Software*, *6*(62), 3065. https://doi.org/10.21105/joss.03065

Kazil, J., Masad, D., & Crooks, A. (2020). Utilizing Python for agent-based modeling: The Mesa framework. In R. Thomson, H. Bisgin, C. Dancy, A. Hyder, & M. Hussain (Eds.), *Social, cultural, and behavioral modeling* (pp. 308–317). Springer International Publishing. ISBN: 978-3-030-61255-9

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

Yu, S. (2022). An agent-based framework for policy simulation: Modeling heterogeneous behaviors with modified Sigmoid function and evolutionary training. *IEEE Transactions on Computational Social Systems*, 1–13. https://doi.org/10.1109/TCSS.2022.3196737