Cimiano P., Pietsch C., & Wiljes C. (Eds.) (2021). Studies in Analytical Reproducibility: the Conquaire Project. Bielefeld. https://doi.org/10.4119/unibi/2942780

6 | Visualization of economic agent-based simulations: introducing the FLAViz toolbox

Sander van der Hoog[1], Philipp Cimiano[2]

1 – Faculty of Business Administration and Economics, Bielefeld University
2 – Semantic Computing Group, Faculty of Technology & Cognitive Interaction Technology Excellence Center (CITEC), Bielefeld University

Abstract

We describe the result of a collaboration between the Economic Theory and Computational Economics (ETACE) group at Bielefeld University and the Conquaire project. The Economic Theory and Computational Economics (ETACE) group, with a project led by Prof. van der Hoog, applies agent-based modeling approaches to study dynamic equilibrium models resulting from the interaction of heterogeneous rational agents. This allows insights into the application of different industrial policy measures in different regions, the existence of varying spatial frictions on goods and labour markets, the spatial dynamics of industrial activity, technical change and growth, the micro- and macro-prudential regulations and their effects on micro-fragility and macro-financial stability, as well as financialisation of the real sector and the need for productive credit for economic development. In this paper, we describe the implementation of the FLAViz library that realizes a data analytic processing pipeline supporting the computational analysis and visualization of simulation data generated in the FLAME environment. This library is a key step towards ensuring computational reproducibility of the analyses of the available simulation data.

Keywords

Computational Economics, Python, Pandas, simulation data, High-Performance Computing.

6.1 Introduction

The research group on Economic Theory and Computational Economics (ETACE) is concerned with the analysis of different aspects of economic dynamics and strategic interaction. It employs and extends both analytical methods, in particular dynamic optimization and dynamic game theory, and computational approaches, where the latter include numerical methods for the solution of (dynamic) equilibrium models as well as agent-based simulations.

Research at ETACE is based on the conviction that a thorough examination of (dynamic) economic phenomena should be based on a combination of (i) dynamic equilibrium analysis, providing benchmark results under full rationality (and foresight) of decision makers, and (ii) the explicit consideration of the economic dynamics unfolding under the interaction of rationally bounded heterogeneous agents. The aim of the work undertaken by the ETACE group is to extend the toolbox of economists and policy makers, and to apply these tools to relevant research questions, mainly in the areas of Industrial Economics, Labour Economics and Macroeconomic Dynamics.

Ongoing research at ETACE can be broadly categorized in the following research topics:

- Agent-based Modelling for Economic Policy Analysis
- Economics of Innovation and Industrial Dynamics
- Network Formation and Spatial Dynamics
- Labour Economics and Search Theory

Within the Conquaire project, we have addressed work in the first topic area, that is *Agent-based Modelling for Economic Policy Analysis*. In particular, we have identified the Eurace@Unibi Model, a specific agent-based simulation model, as a case study to test the notions of *analytical reproducibility* and *continuous integration of research data*, which are two key aspects of data management for the Conquaire project.

In recent years, it has been widely acknowledged by economic scholars that the explanatory power of standard representative agent models is in many cases limited. This has led to a surging interest in the empirical exploration of bounded rationality in economic decision making, mainly by means of laboratory experiments and attempts to incorporate heterogeneity in endowments or behavior into economic models. A particularly natural and promising approach to account for economic phenomena that result from the (bounded) rational interaction of heterogeneous economic agents is the use of agent-based computer simulation models. Phenomena of such types are abundant (the avalanche-like dynamics in the network of connected commercial banks inducing the current economic crisis is just one prominent example in that respect), and a large

amount of insightful agent-based research has addressed a wide range of relevant economic issues (see e.g. the Handbook of Computational Economics Volume II edited by Tesfatsion and Judd [1] for an overview, and the more recent Handbook of Computational Economics Volume IV edited by Hommes and LeBaron [2] for applications).

A main research topic at ETACE is the development of micro-founded macroeconomic heterogeneous agent-based models that can be used as an integrated framework for policy analysis in different economic policy areas. Based on work carried out in the EU-funded Eurace Project, the Eurace@Unibi model has been developed and used as a tool for the analysis of various economic policy questions related to issues of technological change and economic growth, labor market policies, social cohesion and convergence, and to study banking and credit market regulations. See [3] and [4] for a more detailed description of the model. The Eurace@Unibi model is among the most sophisticated and well-documented models in this domain of economic research. It has strong empirical micro-foundations and reproduces a large set of empirical stylized facts. Ongoing work focuses on the analysis of policy effects considering spatial factors and knowledge and information flows. In particular, the goal of the model is to allow to study the effects of:

- the application of different industrial policy measures in different regions,
- the existence of varying spatial frictions on goods and labour markets,
- the spatial dynamics of industrial activity, technical change and growth,
- microprudential and macroprudential regulations and their effects on microfragility and macro-financial stability, and the
- financialization of the real sector and the need for productive credit for economic development.

The Eurace@Unibi model is adapted and extended on a regular basis to address concrete research questions in economic policy. Finally, members of the ETACE group develop and apply statistical methods and concepts to systematically and rigorously analyse computational policy experiments using agent-based simulation models. The data being generated by such simulation models can be quite complex, not just in terms of data volumes but also in terms of its dimensions, heterogeneity, and variety. This is especially true when large-scale agent-based models with large agent populations are simulated. To analyse such high-dimensional data, new data visualization techniques must be developed, and this was one of the main tasks to be accomplished by the ETACE group in the context of the Conquaire project. Since the Eurace@Unibi model, which was selected as our use-case for the Conquaire project, has been implemented in the simulation environment FLAME, we give a brief description of this simulation platform below. In the following subsection 6.2, we describe how the

FLAME environment is used to generate the simulation executable and describe a library called FLAViz that has been developed in cooperation with Conquaire and supports the analysis of simulation data generated by a FLAME-generated model.

6.2 Methods

In this section, we describe the FLAME environment.

6.2.1 The FLAME Environment

The Flexible Large-scale Agent Modelling Environment (FLAME) is a generic agent-based modelling platform, which can be used to generate agent-based models in a wide range of applications, such as biology, crowd simulations, and economic analyses (see the FLAME website for examples and code).[1] The software components *XParser* and *Libmboard* can be downloaded from the GitHub repository of FLAME-HPC.[2]

In principle, FLAME is not a simulator, but a *simulator generator* since it creates a simulation executable that can be run on any hardware platform from laptops or servers, to HPC clusters. Currently, there exist different versions of FLAME for use with CPUs or GPUs, and efforts are underway to create a single, uniform environment that addresses all hardware architectures. The CPU version is called FLAME-HPC and is currently the most mature version (see [5, 6, 7, 8, 9, 10] for a more detailed description of FLAME).

Several features make FLAME particularly appealing as a framework to develop and analyse large-scale agent-based models since the framework has been specifically designed for use on high-performance computing clusters. It provides a very transparent and clean way to model information flows between agents using messages, both internal inside the conceptual model and outside of it through the use of a Message Passing Interface (MPI), provided by the Libmboard library. The only means to communicate private data between agents is through the exchange of messages, where the data an agent can transmit consists of a list of values of its own state variables (e.g. wealth, income, skills, profits, expectations about certain variables). Messages are added to a centralized message board and the sender determines which agents can read the message. Agents check the message boards in every iteration in order to collect all the information they are supposed to receive. An agent can use the collected information as input to its decision rules or as the basis for updating some of its own state variables.

Since high-performance computing clusters are involved, and computational resources on such clusters are still a scarce resource, the data generation and

[1] See the FLAME website <http://www.flame.ac.uk>.

[2] See the GitHub repository <https://github.com/FLAME-HPC>.

data analysis stages are a multi-stage process in which considerations of computational time and data storage play an important role. These two steps are separated in time, with the data first being generated and stored to disk, and afterwards the data is again loaded for analysis.

At the simulation design stage (before simulations are actually run), the model analyst can select to output either a complete snapshot of all variables of all agents (this is very data intensive), or select a subset of agents for which all variables will be stored. In addition, it is also possible to select a certain frequency at which the data is output, say every n iterations, or to select only a subset of variables (a much less data intensive mode of simulation).

6.2.2 Simulation Data

FLAME uses the XML format for data input and output files. In order to design a simulation model in FLAME, three types of XML files are typically required:

- **Model XML files**: This file follows a DTD (see FLAME User Manual, [5, pp.43-44]). It specifies the model's data structures and variable types, with XML tags for the environment, models, agents, messages, ADTs, and time units. The environment-tag contains static constants (model parameters) and file names for the C function files (user-created). The xagent-tag contains memory variables and functions. Messages and ADTs contain attributes, which are the variables contained in these data containers.

- **Data input XML files**: This file is an input argument to the simulator executable (see FLAME User Manual, [5, pp.30-31]). It contains all initial values for the model constants and agent variables. Usually the input file is called *0.xml*, and the default file size is now about 25 MB for our standard economic model.

- **Data output XML files**: These are the output files generated by the simulator executable (which itself is generated by FLAME by compiling the user-created and template C code). This type of file only contains the values for all the agents' variables. The environment constants have no output (except when the output file is a snapshot, see below), since the constants are static and are already contained in the input file.

In order to understand the structure and data content of the output XML files, a brief discussion about the notion of *agents* might be helpful. In research at the ETACE group, we deal with different economic *agent types*, such as *Eurostat, Bank, Firm, Household, Central bank*, etc. Each agent type has a different set of variables, since this depends on what activities the agent performs in the model. For example, the agent type *Bank* might contain variables such as cash, total credit, deposits, mean interest rate, etc. Another agent of type *Eurostat* might

contain variables like: *unemployment rate, total debt, monthly output, average wage*, etc.

Also, each agent type is an archetype, and many instances of each agent type may actually exist in the simulation. In this sense, the agent types are similar to an object class, and the individual agents are similar to object instances. Depending on the particular type of economic analysis, we have different requirements for the simulation output. For example, a particular simulation might contain only the agent type *Eurostat*, while for another analysis we might need more than one agent type, for example all *Eurostat, Firm and Bank* agents. Therefore, the agent types and their variable lists can be filtered before they are output to disk, saving on simulation time and storage requirements. This is one reason why the output XML files may vary in size. Some common file sizes (per iteration) are: 105 bytes (store only Eurostat, 1 variable), 2 MB (store multiple agent types, multiple instances of each type, and many variables per agent instance), 25 MB (store a population snapshot, containing all agents, and all variables per agent). If a certain analysis requires millions of runs for millions of iterations (for molecular dynamics for instance), it makes sense to filter out some of the data before it is output to disk.

The population snapshot file of 25 MB also contains the model constants/parameters in addition to the agent variables. These static constants are usually not part of the output file, as this would be redundant since they are already contained in the input XML file (0.xml). As the purpose of the snapshot file is to be used again as an input file to the simulator, the model constants must also be contained in this file.

The output XML files are named with the iteration numbers. Basically, a file named *1.xml* contains all the values at the end of the first iteration; similarly the file *2.xml* contains all the values at the end of the second iteration, and so on.

Visualizing Simulation Data

In order to generate the simulation data we have adopted the following ontology:

- **Sets**: a set reflects a model parameter setting. Each set differs from another set only in the parameter setting of the model.
- **Runs**: a run is a replication for a fixed parameter setting. Each run differs from other runs only by the random seed. The other initial conditions are kept exactly the same across runs.

Thus, parameter variations are captured in *settings* or *sets*. Each set reflects a different parametrization of the simulation model. In case the model contains random variables and stochasticity, the statistical properties of the model can be explored using different random seeds and a Random Number Generator (RNG). By default, we use the RNG from the open source *GNU Statistical Library*

(GSL), which is based on a Mersenne Twister (`mt19937`). For each data set, multiple runs are performed using different random seeds, producing different simulation output for each run. These runs can be called Monte Carlo replication runs since the random seeds are themselves varied in a random fashion. The seed is set randomly based on the system time at simulation launch time, and stored for later replication of the data, if required.

6.3 Analytical Reproducibility

In this section, we describe the implementation of the Flexible Large-scale Agent Visualization Library (FLAViz), which is a software library specifically designed for the analysis and visualization of data generated by Agent-Based Models (ABMs). Agent-based simulation models typically generate data across multiple dimensions, e.g. parameter sets, Monte Carlo replication runs, different agent types, multiple agent instances per type, many variables per agent, and time periods (iterations). This implies the data is structured as time series panel data sets. FLAViz has been developed in cooperation with the Conquaire project and has been specifically designed for FLAME-generated data, but in principle data from any ABM can be used, as long as the data adheres to the file specifications. FLAViz builds on the Python pandas library to deal with such high-dimensional time series panel data sets. The data is stored as structured data using multiple hierarchical levels in the HDF5 file format. This allows for proper data aggregation, filtering, selection, slicing, transformation, and visualization. The toolbox is setup in a modular way as a flexible set of tools that can be integrated into an automated work-flow for analysing the time series data generated by any computational model. The software code for the visualization library FLAViz is open-source and available for download from the GitHub repository.[3] The installation instructions and dependencies are documented in the readme file of the repository, as well as tutorials and example data.

6.3.1 Data Analysis Pipeline

FLAViz is an addition to the FLAME set of tools used for the simulation and analysis of large-scale agent-based models. FLAME natively outputs data in XML format. In FLAViz this gets processed using Python scripts and transformed into HDF5 files for final storage. Building on the pandas and matplotlib libraries, various plots can be specified, e.g., time series, box plots, scatter plots, histograms, and delay plots.

FLAViz version 0.1.0 (beta) is written in Python (ver- 3.6) and other package dependencies include:

- Pandas (ver-0.21.0)

[3] `<https://github.com/svdhoog/FLAViz>`

- YAML files for easy configuration management
- Matplotlib for data visualization
- HDF5, and
- PyTables

FLAViz uses two important inbuilt features of the pandas library, viz.:

- **Hierarchical indexing**: this allows a high dimensional data frame (the ndarray format)
- **Bygroup**: this allows to re-order the hierarchical index, to reshape the data dimensions

At the outset, the original simulation data are stored in XML files and are then converted to a more data-processing friendly format, viz. the HDF5 format. This is needed because the XML files that FLAME simulations generate are a fully tagged data format and is therefore very verbose. For large scale simulations this is prohibitive in terms of the sheer size of the data volumes generated. The storage and parsing of large data volumes generate a complex data structure. In order to reduce this storage footprint, yet retain the structured data format, the HDF5 standard was chosen for its hierarchical data storage structure. The Pandas library can easily read large *.h5 files and store the data internally into one of its native data formats (either *pandas.dataframe* or *numpy.ndarray*).

The data hierarchy is as follows:

1. **Agent types**: $a = 1, \ldots, A$ - Classes, groups of agent sub-populations
2. **Sets**: $s = 1, \ldots, S$ - Parameter settings (model calibrations)
3. **Runs**: $r = 1, \ldots, R$ - Monte Carlo replication runs (random seeds)
4. **Iterations**: $t = 1, .., T$ - Time periods
5. **Agents**: $i = 1, \ldots, n\ a$ - Individual agents (per type)
6. **Variables**: $j = 1, \ldots, m$ - Scalars, Arrays, Composites

Due to this large data heterogeneity, the file sizes may vary across simulations with the same model, even when using exactly the same input file, due to stochasticity. The data for each *agent-type* is stored in a single HDF5 file without any file-size limitations. The data is heterogeneous across several dimensions:

- **agent types**: there can be many different agent types (e.g., household agents, firm agents, bank agents, etc.)

- **agent instances**: there can be a different number of agent instances per agent type
- **agent memory variables**: there can be a different number of memory variables per agent type (but all agents of the same type have the same set of memory variables, specified a priori, in the *model.xml* file that fully specifies the model's structure)

HDF5 File Format

HDF5 has a simplified file structure that includes only two major types of objects:

1. **Datasets**: which are multidimensional arrays of a homogeneous type; and
2. **Groups**: which are container structures which can hold datasets and other groups.

The main restrictions of the HDF5 file standard are:

1. the HDF5 file format requires that the atomic data set at the lowest hierarchical level is a homogeneous data format (no ragged edges). This means that the choice of the 6 dimensions (Sets, Runs, Iterations, Agent types, Agent instances per type and Variables) requires us to choose those dimensions that remain invariant across all model simulations as the ones contained in this homogeneous data structure. These dimensions are: *Agent instances*, *Iterations* and *Variables*. These dimensions are invariant because we simulate the same model many times, and we do not change the model structure across simulations. Therefore the number of variables per agent remains the same, the number of agent instances per agent type is constant, and the total number of iterations also remains constant across simulation runs. Another reason for choosing those 3 dimensions is that the sets and runs form a unit of analysis, so it makes sense to choose those for the higher level in the hierarchy. Also, the simulation output for the sets and runs can be generated on a cluster in a massively distributed fashion, by distributing the compute load across many nodes. Logically, this implies storing the output in separate files according to the set/run combinations first, and only at the very end combining all these files according to the agent types.
2. The 3D Panel format in Python pandas has 3 axes (item, major and minor) and is specified as *row-major*. This means that the data structure requires the largest dimension to be on the major axis. In our case, the largest dimension is the number of iterations, typically 1000 or higher. The other dimensions are the number of agent instances (on the order of 100), and the number of variables (also on the order of 100).

Given the above constraints, we specify the 3D Panel data structure as follows:

1. **item axis**: agent instances
2. **major axis (table rows)**: iterations
3. **minor axis (table columns)**: variables

To deal with the remaining 3 dimensions (agent types, sets and runs), we proceed as follows. We generate a separate HDF5 file per agent type, using the naming convention `AgentType.h5`. To account for the two remaining data dimensions of sets and runs, we specify the data groups inside the HDF5 file using set/run combinations as follows: `set_s_run_r` ($s = 1, ..., S$ and $r = 1, ..., R$).

Summarizing, the simulation data is stored in a **HDF5** container file (*.h5, *.hdf5) using a hierarchical data format. Currently, these **HDF5** files are structured as follows:

- Each agent type is contained in a separate HDF5 file, with the same name as the agent type (e.g., *Firm.h5*, *Bank.h5*, etc.).
- Inside each HDF5 file there is a **Group** (similar to a folder structure) for each combination of *set* and *run*, using the naming convention `set_s_run_r`.
- Inside each **Group** there is a **Dataset** which contains a *pandas Panel*, which is a datastructure that consists of *items*, *major* and *minor* axes.
- the **Python pandas Panel** is written to the HDF5 file with the *PyTables* module of Python, which uses a write-once policy (no appending).

The HDF5 file structure described above can be created from SQLite database files that contain the results from set/run combinations by using the data processing scripts that are included in the FLAViz package. Alternatively, the HDF5 file could be created from the XML files directly, but a big disadvantage of this method is that the entire collection of XML files has to be available on disk in uncompressed form (very bulky), which could be prohibitive for large-scale applications. It is also not very resource-friendly, due to its lack in parallelism. Another option would be to stream the data into the final database file as it becomes available from the simulations. Unfortunately, however, streaming the data into an HDF5 file is not possible, due to the write-once feature of the *PyTables* module that we have chosen to adopt in the library to write to the HDF5 file. The reason for this choice is that appending data to an HDF5 file would require a different write method using the *h5py* module, which is less performant than doing it write-once.

6.3.2 Plotting with FLAViz

To adhere to the general principle that all results of a published paper should be computationally reproducible given the data from computational experiments, we should be able to reproduce the plots using various permutations and combinations of the data. FLAViz uses three configuration files, through which the necessary conditions can be set. The configuration files follow the hierarchical `yaml` format for clarity and functionality with specific indentation for the input to be interpreted correctly. For the general plot settings, the yaml file `config.yaml` contains settings for selecting the desired **sets** and **runs**, or to specify ranges for the **iterations** and **variables** along the **major** and **minor** axes, respectively. It is also possible to perform data transformations of agent variables, and to select data based on data slicing. For example, select all data at iteration $t = x$, or select all data for agent $ID = i$. Data filtering can also be performed, in which case the data is filtered based on agent conditions or variable conditions. For example, filter the selected data on the condition that the agent variable X has value v. For selecting the plotting styles, the yaml file `plot_config.yaml` contains settings to select what kind of features the plot should contain. Everything related to axes, legends, colours, etc, can be set in this file, which follows the basic features of `matplotlib`, which is the standard plotting library used by Python pandas. Currently, if the user specifies multiple plots, these are processed one by one. To speed up this process and parallelize the plotting routine, each plot could be run as a separate sub-process that retrieves data from the main data set once it has been read into main memory. This is left for future development of the FLAViz library.

Example config files As an example, the plot in Fig. 6.1 shows a visualization of data for the agent type "Firm", the variable "price", and is based on data for 4 sets (selected sets: 10, 13, 16, 17). Each set consists of 20 runs. The plotting style is specified as using a time series multiple-batch plot, showing the 20^{th} and 80^{th} percentiles. The construction and generation of this plot is specified in the following settings in the configuration files.

config.yaml:

```
plot1:
    timeseries:
        agent: Firm
        analysis: multiple_batch
        variables:
            var1: [price]
        set: [10,13,16,17]
        run: [range,[1,20]]
        major: [range,[6020,12500,20]]
        minor: [range,[1,80]]
```

```
        summary: custom_quantile
        quantile_values:
            lower_percentile : 0.20
            upper_percentile : 0.80
```

plot_config.yaml:

```
plot1:
    number_plots: one
    plot_legend: yes
    legend_location: best
    xaxis_label: Time
    yaxis_label: price
    linestyle: solid
    marker: None
    fill_between: yes
    fillcolor: darkgreen
```

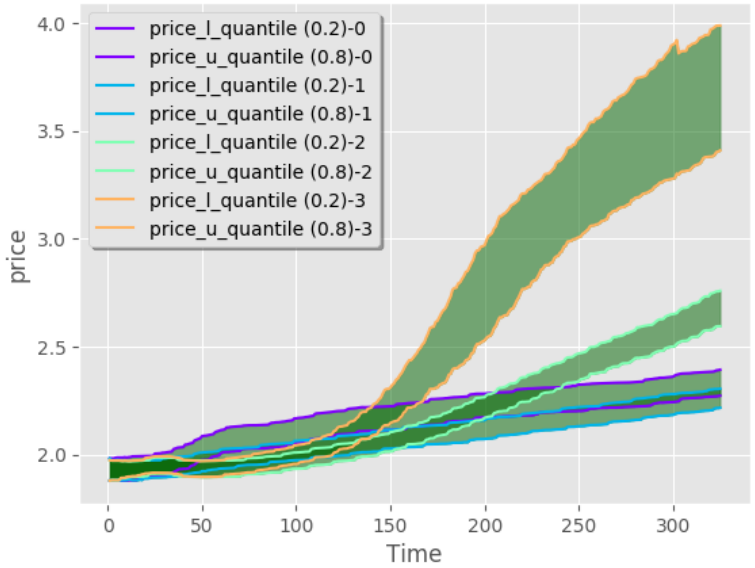

Figure 6.1: Plotting the price time series for data covering 4 sets, each consisting of 20 runs of 6.500 iterations, and for 80 Firm agents.

6.4 Summary and limitations

FLAME simulations with large-scale economic simulation models require high-performance super-computing (HPC) facilities and generate large datasets that typically represent a bottleneck from the perspective of both computational resources and storage requirements.

BigData storage using the HDF5 format (a hierarchical filesystem-like data format) works well for economic data that consists predominately of time series data (i.e., numerical, not text data). If needed, more complex storage APIs representing images and tables can be built using datasets, groups, attributes, types, dataspaces and property lists. Because the bulk of the data is transformed into straightforward arrays (the table objects) for processing, the data can be accessed in a much faster way than with more traditional row-based processing in an SQL database.

The time performance of the FLAViz Library could probably be reconfigured/refactored to work in a more distributed fashion by optimizing the order in which the data is post-processed. We can probably speed-up several operations that now are taking place sequentially. This is a matter of figuring out what the main loops on the various dimensions of the data (agents, variables, iterations, etc.) are, and then determine the optimal order in which these loops should be executed. The items in the loops can then be executed in different threads, and it should be investigated whether there are any information dependencies that need to be resolved between those threads.

Also the storage performance could be optimized. Not all data need to remain in memory at all times. Currently, we first read-in all the data into main memory, then filter it based on conditions, and then process it further.

In simulation science, we deal with very complex data objects with a wide variety of metadata that require a portable file format without any limits on the number or size of data objects in the collection. The HDF5 format is a versatile data model that makes it easier to manage extremely large and complex data collections with time and storage space optimizations. It also runs on a range of computational platforms. Some advantages of using the HDF5 format are:

- **HDF5 is a Self Describing Format**: Each file, group and dataset can have associated metadata that describes exactly what the data is, viz., data types, description, documentation of data ontologies, information about how the data in the dataset were collected, etc.

- **Compressed & Efficient subsetting**: The HDF5 format is a compressed format and data size optimization makes the overall file size smaller. The data slicing feature allows subsets of a dataset to be extracted for processing in order to avoid storing the whole dataset in main memory.

- **Heterogeneous Data Storage**: HDF5 files can store multiple types of data within the same file as sets of datasets containing heterogeneous data

types (e.g., both text and numeric data in one dataset)

- **Open Format**: HDF5 has technical support in many programming languages and tools, like 'R', 'Python' and 'Julia' due to its open format.

6.5 Conclusion

This paper has described a case study in the area of computational economics on the computational reproducibility of simulation results. In contrast to other chapters, we have not aimed at reproducing a particular result published by the ETACE group. Instead, Conquaire has cooperated with the ETACE group to implement a generic visualization library called FLAViz, to support the exploration and visualization of simulation data. The pipeline implemented in FLAViz makes use of the HDF5 format, which has turned out to be a very flexible and versatile data format.

FLAViz supports the analytical reproducibility of research data in two ways, both ex-ante and ex-post publication. Firstly, if researchers store their simulation data on an ongoing basis during a research project, and FLAViz configuration files are also available, then an automatic plot generation tool can be used in the sense of Continuous Integration of research data. This helps a lot in increasing the trustworthiness and credibility in the results, as well as giving us the ability to track how the results are changing over time as the research project progresses. Secondly, if pre-generated simulation data is available from a published paper from the original authors, then the FLAViz toolbox could be directly applied to this dataset to reproduce the plots of the published paper. These can then be used to check the validity of the claims made by the original authors in their paper. In these two important ways, toolboxes such as FLAViz can be regarded as helping us to ensure the analytical reproducibility of research data.

Acknowledgements

We would like to thank Krishna Devkota for implementing the FLAViz library and Fabian Hermann for documentation and bug fixing.

References

[1] Leigh Tesfatsion and Kenneth Judd, editors. *Handbook on Agent-Based Computational Economics*, volume 2. North-Holland: Elsevier, Amsterdam, 2006.

[2] Cars H. Hommes and Blake LeBaron, editors. *Handbook on Agent-Based Computational Economics*, volume 4. North-Holland: Elsevier, Amsterdam, 2018.

[3] Herbert Dawid, Simon Gemkow, Philipp Harting, Sander van der Hoog, and Michael Neugart. Agent-Based Macroeconomic Modeling and Policy Analysis: The Eurace@Unibi Model. In S-H Chen, Kaboudan M., and Y.-R. Du, editors, *The Oxford Handbook of Computational Economics and Finance*, chapter 17, pages 490–519. Oxford University Press, 2018.

[4] Herbert Dawid, Philipp Harting, Sander van der Hoog, and Michael Neugart. A Heterogeneous Agent Macroeconomic Model for Policy Evaluation: Improving Transparency and Reproducibility. *Journal of Evolutionary Economics*, 29:467–538, 2019.

[5] Mariam Kiran. *FLAME Flexible Large-sale Agent-based Modelling Environment User Manual*. University of Sheffield, 2010.

[6] Simon Coakley and Mariam Kiran. *FLAME User Manual*. University of Sheffield and Rutherford Appleton Laboratories, STFC, 2012.

[7] Simon Coakley, Marian Gheorghe, Mike Holcombe, Shawn-Lee Chin, David Worth, and Christopher Greenough. Exploitation of high performance computing in the FLAME agent-based simulation framework. In *Proceedings of the 14th International Conference on High Performance Computing and Communications*, pages 538–545, 2012.

[8] Paul Richmond. FLAME GPU Technical Report and User Guide. Technical Report CS-11-03, 2011.

[9] Simon Coakley, Paul Richmond, Marian Gheorghe, Shawn-Lee Chin, David Worth, Mike Holcombe, and Christopher Greenough. Large-Scale Simulations with FLAME. In Joanna Kołodziej, Luís Correia, and José Manuel Molina, editors, *Intelligent Agents in Data-intensive Computing*, Studies in Big Data Series, pages 123–142, 2016.

[10] Mariam Kiran. *X-Machines for Agent-Based Modeling: FLAME Perspectives*. Computer and Information Science Series. Chapman & Hall/CRC Press, Boca Raton, Fla., 2017.