ABCE: A Python Library for Economic Agent-Based Modeling

Davoud Taghawi-Nejad[1,2(✉)], Rudy H. Tanin[4,5], R. Maria Del Rio Chanona[1,2], Adrián Carro[1,2], J. Doyne Farmer[1,2,3], Torsten Heinrich[1,2], Juan Sabuco[1,2], and Mika J. Straka[1,6]

[1] Institute for New Economic Thinking at the Oxford Martin School, University of Oxford, Oxford OX2 6ED, UK
davoud@taghawi-nejad.de, {davoud.taghawinejad,rita.delriochanona, adrian.carro,doyne.farmer,torsten.heinrich}@maths.ox.ac.uk
[2] Mathematical Institute, University of Oxford, Oxford OX1 3LP, UK
[3] Santa-Fe Institute, Santa Fe, NM 87501, USA
[4] Department of Physics, MIT, Cambridge 02139, USA
rht@mit.edu
[5] Independent Scholar, Tallinn, Estonia
[6] IMT School for Advanced Studies Lucca, 55100 Lucca, Italy
mika.straka@imtlucca.it

Abstract. The rise of computational power makes agent-based modelling a viable option for models capturing the complex nature of an economy. However, the coding implementation can be tedious. Because of this, we introduce ABCE, the Agent-Based Computational Economics library. ABCE is an agent-based modeling library for Python that is specifically tailored for economic phenomena. With ABCE the modeler specifies the decision logic of the agents, the order of actions, the goods and their physical transformation (the production and the consumption functions). Then, ABCE automatically handles the actions, such as production and consumption, trade and agent interaction. The result is a program where the source code consists of only economically meaningful commands (e.g. decisions, buy, sell, produce, consume, contract, etc.). ABCE scales on multi-core computers, without the intervention of the modeler. The model can be packaged into a nice web application or run in a Jupyter notebook.

Keywords: Agent-based models · Agent-based macroeconomics · Python · Economic simulation · Computational economics · Computational techniques · Simulation modeling

1 Introduction

An economy is a complex system where the interaction of heterogeneous agents plays a crucial role. However, current economic models used by academics, the

D. Taghawi-Nejad—This author gratefully acknowledge the financial support from MS AMLIN plc, London.

G.L. Ciampaglia et al. (Eds.): SocInfo 2017, Part I, LNCS 10539, pp. 17–30, 2017.
DOI: 10.1007/978-3-319-67217-5_2

central banks and policy makers fail to acknowledge its complexity and are either based on empirical statistical models fitted to past data (econometrics) or assume a perfect world (DSGE) and by their very nature rule out crisis [4]. Agent-based models (ABM) can take into account of this complexity since they are expressive computer simulations of heterogeneous agents which interact across different spatial and temporal scales according to predefined rules. When modeling an economy, agents represent firms, households, banks and other important entities that define the systems dynamic.

In social sciences and in particular in economics where designing replicable experiments is usually impossible, agent-based models present an opportunity for policy testing. To give true insights on economic phenomena, a large number of heterogeneous agents and often stock-flow consistency[1] are crucial. Here we introduce the Agent-Based Computational Economics library (ABCE) a Python library with the computational efficiency needed for state of the art ABM's. ABCE has been continuously been updated and extended for 6 years after its conceptual framework was unveiled in Taghawi-Nejad (2013) [13].

While Taghawi-Nejad (2013) [13] discusses the theory that agent-based modeling is a language to express economic phenomena as dynamical processes, this paper gives a practical overview over how and why to use ABCE. It also incorporates the many updates that have been introduced in ABCE.

ABCE[2] is a Python-based modeling platform for economic simulations and part of the Economic Simulation Library[3]. For simulations of trade, production and consumption, ABCE comes with standard functions that implement these kinds of interactions and actions. The modeler only implements the decision logic of an agent, then ABCE takes care of all exchange of goods and production and consumption. Furthermore it can also handle contracts and even generate balance sheets.

One feature of ABCE is that goods have the physical properties of goods in reality. That means that, if agent A gives a good to agent B, then - unlike information - agent B receives the good and agent B does not have the good anymore. The library handles trade production and consumption of goods according to the decision of the agents.

The audience of ABCE are economists, computer scientists, mathematicians, scientists conducting interdisciplinary research, and people of similar areas that want to create agent-based models involving trade, production or contracts. Simulations can be similar to standard economic models such as general or partial equilibrium models, but allow the assumptions to be relaxed to the extent of removing equilibrium. Current development efforts will make it possible to automatically handle contracts, contractual obligations and balance-sheets. ABCE uses Python - a language that is especially beginner friendly, but also easy to learn for people who know object oriented programming languages such as Java,

[1] Meaning that goods and money are not created "out of thin air". ABCE is stock flow consistent, if the stock-flow consistency is not explicitly broken.

[2] https://abce.readthedocs.io.

[3] https://economicsl.github.io.

C++ or MATLAB. Syntactically, Python code looks like executable pseudo code [11]. Moreover, in science, code is meant to be read, verified, and reused by other scientists, so we decide to cut down the development time and the time required to comprehend our code rather than execution time. Where speed is relevant, ABCE's back-end is written in C via Cython.

2 Design

ABCE's main design goal is that code can be rapidly written to enable a modeler to quickly write down code and quickly explore different alternatives of a model. In Python, variables do not have to be declared, garbage does not have to be collected and classes have no boiler-plate code. Another advantage of Python that facilitates rapid coding is its rich environment of libraries. For example, Mesa enables representation of agents in a spatial world. Installation of packages is much simpler in Python than in Java, C, or C++.

Execution speed is a secondary concern to the goal of rapid development. Execution speed is increased by making use of multiple-cores/processors and using C (via Cython) for backend tasks. For the user, the library is a pure python library, with the C back-end completely hidden from the user. Although Python is slower than Java or C in terms of execution speed, this disadvantage can be largely overcome by using various optimized packages for numerical calculations, such as NumPy [15] and SciPy [5], which make use of back-end implementations in C. ABCE allows to parallelize the code and gain significant speed advantage over single-threaded code.

Thirdly the design is giving full modelling liberty to the programmer. This is done by not implementing any economic assumptions in the simulation engine of the library. Depending on the use case, a user may decide to not use the predefined agents in the library, and may instead define new agents from scratch without restrictions. While ABCE provides stock methods such as trade, consume and produce, it does not force specific economic assumptions.

There are predefined agents (Agent, Firm or Household) which are Python classes that can be inherited from. This will allow the modeler to concentrate on defining the behaviour of the agents and the specification of goods, production/consumption functions. ABCE automatically handles the communication, trade and consumption of goods.

ABCE agents are ordinary Python objects, but they can be run parallel on a multi-core/processor computer without further configuration or intervention. The parallel execution of agents is the only non-standard feature of ABCE, which bypasses Pythons global interpreter lock (GIL). The speed advantages of the parallelization are observed when running simulations involving 10000 agents or more.

A simulation run in ABCE is a sequence of rounds, with sub-rounds in which agents execute actions in parallel. Each agent executes these actions possibly using some of the built-in functions, such as messaging, trade, production and consumption of ABCE. As agents communicate only between sub-rounds, all

actions within one sub-round are executed as if they were executed in parallel. It is also possible to run a simulation as a discrete event simulation, where events are scheduled at particular times.

2.1 Physical Goods

Physical goods are at the heart of most economic models. The core feature and main difference to other ABM platforms is the implementation of physical goods. In contrast to information or messages, sharing a good means having less of it. If agent A transfers a good to agent B then agent A does not have this good anymore. One of the major strength of ABCE is that this is automatically handled.

In ABCE goods can be created, destroyed, traded, given or changed through production and consumption. All these functions are implemented in ABCE and can be inherited by an agent. These functions are automatically handled by ABCE upon decision from the modeler.

Every agent in ABCE must inherit from the abce.Agent class. This gives the agent a couple of stock methods: create, destroy, trade, and give. Create and destroy create or destroy a good immediately. Because trade and give involve a form of interaction between the agents, they are run over several sub-rounds. Selling of a good for example works as follow:

- subround 1: the first agent offers the goods
  post-subround ABCE handler: the good is automatically subtracted from the agents possessions, to avoid double selling
- subround 2: the counter agent receives the offer. The agent can
  - accept: the goods are added to the counter parts possessions. Money is subtracted
  - reject (or equivalently ignore): nothing happens in this sub-round
  - partially accept the offer: the partial amount of goods is added to the counter parts possessions. Money is subtracted
- subround 3: in case of
  - acceptance, the money is credited
  - rejection the original good is re-credited
  - partial acceptance the money is credited and the unsold part of the good is re-credited

Objects have a special stance in agent-based modeling:

- objects can be recovered (resources)
- exchanged (trade)
- transformed (production)
- consumed
- destroyed and depreciate over time

ABCE takes care of trade, production/transformation and consumption of goods automatically. Good categories can also be made to perish or yield another good. E.g. a field can grow crops and an oilfield oil. The modeler has only to decide on the when and how.

2.2 Services or Labor

We can model services and labor as goods that perish and that are replenished every round. This would amount to a worker that can sell one unit of labor every round, that disappears if not used.

Let us assume for the sake of the argument that we want to simulate actions that happen in real life in parallel, but we can only run agents sequentially in the simulation. Since these actions happen in parallel we know they (a) start with the same information set and (b) cannot affect each other. This has two implications: they must start with the same information set and the effects on the other agents must happen after they have all finished, i.e. the information set can only be updated after all agents actions have been executed.

Generally speaking, multithreading and parallel execution have to cope with the problem that actions do not necessarily finish at the same time, which has to be considered when updating the information set.

To address this issue, in ABCE time is discrete and actions happen in lock-step. Temporal events consist of rounds and sub-rounds. Rounds correspond to real-time intervals, e.g. days. In each subround all agents execute the same action. So for example in overnight-loans:

- subround 1: banks send a request for overnight loans
  post-subround ABCE handler: messages are delivered
- subround 2: banks receive requests and process them
  post-subround ABCE handler: messages and payments are delivered
- subround 3: money has arrived, banks can make new requests if they still need money
- ...

In conclusion, time in ABCE is discrete and actions that happen in the same subround do not causally affect each other. Communication happens only between the sub-rounds simulating simultaneity of the actions in one subround.

For example, a sales transaction is handled in the following way:

The transfer of goods between agents and its consistency is handled by ABCE. In other words, if a good is sent, it gets subtracted from one agent and added to the other. However, ABCE also ensures that objects cannot be transferred twice: if an agent gives an object to another agent it is immediately taken from his possessions and credited only in the next round. In case of a sale offer, or analogously a purchase offer, the good gets committed to the sale as soon as the offer is made.

The game tree is as follows

- subround 1: Agent A offers the good:
  self . sell ('receiver', receiver_id, 'good', quantity, price)
  post-subround ABCE handler: the quantity of the good is subtracted from his possessions immediately
- either of

- subround 2a: Agent B accepts self . accept (offer)
  post-subround ABCE handler: the money is subtracted immediately, then the good is added immediately
- subround 3a: (pre-subround ABCE handler) the money is added to Agent A's possessions
- ...

or

- subround 2b: Agent B rejects self . reject (offer)

- subround 3b: (pre-round ABCE handler) the good is refunded
- ...

2.3 Closed Economy

Imposing that goods can only be transformed, but neither created nor destroyed, amounts to modeling a physically closed economy, which is stock and flow consistent. In ABCE this can be achieved by calling the respective creation/destruction functions for goods, money, etc. only in the initialization phase.

3 Difference to Other Agent-Based Modeling Frameworks

We identified several survey articles as well as a quite complete overview of agent-based modeling software on Wikipedia [1,7,10,12,14,16]. The articles Tools of the Trade by Madey and Nikolai [7] and Survey of Agent Based Modelling and Simulation Tools by Allan [1] attempt to give a complete overview of agent-based modelling platforms/frameworks. The Madey and Nikolai paper categorizes the ABM-platforms according to several categories (Programming Language, Type of License, Operating System and Domain). According to this article, there is only one software platform which aims at the specific domain of economics: JASA. JASA aims specifically at auctions. Wikipedia [16] lists JAMEL as an economic platform JAMEL is closed source and an non-programming platform. The Survey of Agent Based Modelling and Simulation Tools by Allan [1] draws our attention to LSD, which follows a dynamical systems approach rather than an agent-based modeling platform.

While the formerly mentioned papers on modeling platforms aim to give a complete overview, Evaluation of free Java - libraries for social scientific agent based simulation by Tobias and Hoffman [14] chooses to concentrate on a smaller number of simulation packages. Tobias and Hoffman analyze: RePast, Swarm, Quicksilver, and VSEit. We will follow this approach and concentrate on a subset of ABM models. First as economics is a subset of social science we dismiss all platforms that are not explicitly targeted at social science. The list of social science platforms according to [7] Madey and Nikolai is: AgentSheets, LSD, FAMOJA, MAML, MAS-SOC, MIMOSE, NetLogo, Repast SimBioSys, StarLogo, StarLogoT, StarLogo TNG, Sugarscape, VSEit NetLogo and Moduleco.

We dismiss some of these frameworks/platforms from our analysis: AgentSheets, because it is closed source and not programmable. LSD, because it uses dynamical systems approach rather than an agent-based modeling environment. MAML, because it does not use a standard programming language, but it is its own. MAS-SOC, because we could not find it in the internet and its documentation according to [1] is sparse. MIMOSE, an interesting language, but we will not analyze as it is based on a completely different programming paradigm, i.e., functional programming, as opposed to object-oriented programming. SimBioSys, because it has, according to Allan [1] and our research, a sparse documentation. StarLogo, StarLogoT, StarLogo TNG, because they have been superseded by NetLogo Moduleco, because it has, according to Allan [1] and our research, a sparse documentation. Furthermore, it has not been updated since roughly 2001.

We will concentrate on the most widely used ABM frameworks/platforms: MASON, NetLogo, Repast.

4 General Differences to Other Agent-Based Modeling Platforms

First of all ABCE is specifically designed for economic problems. It provides the basic functions such as production, consumption, trade and communication as fully automated stock methods. Because any kind of agent interaction (communication and exchange of goods) is handled automatically by ABCE, it can run the agents (virtually) parallel and run simulations on multi-core/processor systems without any intervention by the modeler.

The second biggest difference between ABCE and other platforms is that ABCE introduces the physical good as an ontological object in the simulation. Goods can be exchanged and transformed. ABCE handles these processes automatically, so that for the model a physical good behaves like a physical good and not like a message.

Thirdly, ABCE is just a scheduler that schedules the actions of the agents and a Python class that enables the agent to produce, consume, trade and communicate. A model written in ABCE, is therefore standard Python code and the modeler can make use of the complete Python language and the Python language environment.

Fourthly, many frameworks such as FLAME, NetLogo, StarLogo, Ascape and SugarScape and, in a more limited sense, Repast are designed with spatial representation in mind. For ABCE, a spatial representation is possible, but not a design goal. Since agents in ABCE are ordinary Python objects, they can use Python modules such as Mesa and therefore gain a spatial representation much like NetLogo. This does not mean that ABCE could not be a good choice for a problem where the spatial position plays a role. Particularly if the model has different transport costs or other properties according to the geographical position of the agents, but the agents do not move or the movement does not have to be represented graphically, ABCE could still be a good choice.

5 Comparison to Other Agent-Based Modeling Platforms

5.1 MASON

MASON is a single-threaded discrete event platform that is intended for simulations of social, biological and economical systems. [6]. Mason is a platform that was explicitly designed with the goal of running it on large platforms. MASON distributes a large number of single threaded simulations over different computers or processors. ABCE on the other hand is multi-threaded it allows to run agents in parallel. A single run of a simulation in MASON is therefore not faster on a computing cluster than on a potent single-processor computer. ABCE on the other hand uses the full capacity of multi-core/processor systems for a single simulation run. The fast execution of a model in ABCE allows a different software development process, modelers can 'try their models while they are developing and adjust the code until it works as desired. The different nature of both platforms make it necessary to implement a different event scheduling system. MASON is a discrete event platform. Events can be scheduled by the agents. ABCE on the other hand is scheduled - it has global list of sub-rounds that establish the sequence of actions in every round. Each of these sub-rounds lets a number of agents execute the same actions in parallel. However ABCE also supports discrete event scheduling. MASON, like Repast Java is based on Java, while ABCE is based on Python.

5.2 NetLogo

Netlogo is a multi-agent special purpose programming language that integrates with Java. It can, if required, be supplemented with Java code and is run in a Java VM. First released in 1999, NetLogo dates back to the early days of agent-based modeling and is widely advertised for its ease of use and appeal to scholars without prior programming experience. This, however, comes at a price: As Netlogo is an interpreted language running on top of a Java implementation, performance and speed of NetLogo simulations are impeded. Further, NetLogo is heavily centered on visualization and spatial structure, even if the spatial structure does not have a useful interpretation for the use case at hand.

In contrast to Netlogo, ABCE is not a special purpose language but a library that can directly be included from programs written in the general purpose language Python. It also places much fewer restrictions on the modelers modelling decisions and will, for instance, not enforce the use of spatial (or any other) structures. That said, NetLogo-style spatial simulations can be implemented in an ABCE simulation by using Mesa.

5.3 Repast

Repast is a modeling environment for social science. It was originally conceived as a Java rewrite of SWARM. [3,8] Repast has API in several flavors: Java, .Net, and a Python-like language. Repast has been superseded by Repast Simphony

which maintains all functionality, but is limited to Java. Repast Simphony has a point and click interface for simple models. :raw-tex:citeNORTH2005a Repast supports static and dynamic scheduling. [3] Repast is vast, which contains 210 classes in 9 packages :raw-tex‘citeCollier’. ABCE, thanks to its limited scope and Python, has only 8 classes visible to the modeler in a single package.

5.4 JABM, JMAB and Special Purpose ABM Tools

Recent years have seen the emergence of a variety of other ABM platforms and tools. One of them is the Java Macro Agent-Based (JMAB) toolkit, which is explicitly designed for building stock-flow consistent Macroeconomic ABM [2]. As such, it limits the modelers possibilities by design, precluding for instance micro-level models that do not presume to fully represent the entirety of an economy, but also inter-disciplinary approaches that place their focus on a field other than economics. The JMAB toolkit is, in turn, based on the Java Agent-Based Modelling (JABM) toolkit, another ABM library. Both JMAB and JABM run on Java and concentrate significant parts of their design on dependency injection: Experiments can be defined by the modeler in a configuration file and run without having to touch any Java code. [9] This creates desirable properties in terms of controlling consistency across experiments but may impede the flexibility of the modeler.

5.5 Mesa

Mesa is a modular ABM framework, with a goal to be Python 3-based alternative to NetLogo, Repast, or MASON. Its specific strength is the spatial representation of agent positions. Which is the same as in Netlogo. ABCE, on the other hand, is specifically built for economic modelling, has core components written in Cython, and has been optimized for parallel execution.

6 How to Write an Agent-Based Model in ABCE

The first step to make an ABM is to define the agents. ABCE provides the programmer with a predetermined set of benchmark agents and functions. However the main advantage of ABCE is that gives total flexibility for the modeler to define its own agents. Therefore we explain how an agent class must be defined.

The agent classes must inherit the base agent (abce.Agent) and possibly the firm (abce.Firm) or the household class (abce.Household). The base class gives agents the ability to interact with other agents. That includes amongst others the ability to send messages and trade. The firm and household classes give agents the ability to produce and consume. For firm and household agents the modeler has to specify the functional form of the production or consumption function in the __init__ method. Below we show an example where a firm is created with a money budget of 10 and produces “GOOD” with a Cobb-Douglas production function.

```
import abce

class Firm(abce.Agent, abce.Firm):
    def init(self, simulation_params, agent_params):
        """
        1. Gets an initial amount of money
        2. create a cobb_douglas function:
           GOOD = 1 * labor ** 1.
        """
        self.create('money', 10)
        self.set_cobb_douglas("GOOD", 1, {"labor": 1})
```

The simulation_parameters and agent_parameters must be given when the simulation is called, as we will explain further down below.

The modeler must also specify the agents decision logic. For every method of the agent class the modeler has to decide under which conditions agents, interact, trade, produce, consume, etc. or do other things. Its is instructive to consider the following code snippet: look on code for this:

```
# Agent 1
def sell_cookies(self):
    self.sell(buyer, 2, 'cookies', price=1, quantity=2.5)


# Agent 2
def buy_cookies(self):
    offers = self.get_offers('cookies')
    for offer in offers:
        if offer.price < 0.5:
            try:
                self.accept(offer)
            except NotEnoughGoods:
                self.accept(offer,
                            (self.possession('money') /
                            offer.price))
```

Amongst others agent 1 sends an offer of cookies to agent 2. The agent receives all offers of cookies. She can iterate over all offers and decide, which offers to accept and which not. Note that the interesting thing here is that the modeler only implements the decision logic, but the transactions are handled by ABCE in the background.

Once the agents have been defined, the simulation must be run in a start file. Here the simulation class is instantiated, which then builds the agents from the agent classes already defined as explained above. It is in this part that the simulation parameters and agent parameters are defined as shown in the code snipet below. The order of execution of agents action has to be specified, as well as goods and their properties, such as services or perishable, respectively.

Finally, we need to define which observables should be measured throughout the simulation.

```
parameters = {'name': '2x2',
              'random_seed': None,
              'rounds': 10}

@gui(parameters)
def main(parameters):
    w = Simulation(rounds=parameters['rounds'])
    w.declare_round_endowment(resource='adult', units=1,
            product='labor')
    w.declare_perishable(good='labor')

    w.panel('household', possessions=['money', 'GOOD'],
            variables=['current_utiliy'])
    w.panel('firm', possessions=['money', 'GOOD'])

    firms = w.build_agents(Firm, 'firm', 1)
    households = w.build_agents(Household, 'household', 1)
    for r in w.next_round():
        households.do('sell_labor')
        firms.do('buy_labor')
        firms.do('production')
        firms.do('panel')
        firms.do('sell_goods')
        households.do('buy_goods')
        households.do('panel')
        households.do('consumption')

if __name__ == '__main__':
    main(parameters)
```

The fact that ABCE is designed to be a library means that the same start.py file can use other Python libraries. For example we can use a calibration tool and a statistical package that runs the simulation repeatedly with different parameters to achieve a certain fit, read from a relational database to calibrate the agents, frame the agents to move around and gather resources in a 2D/3D mesh or a graph.

7 GUI and Database

ABCE has several ways of collecting data. For example, agents can log data directly to the database or the simulation can monitor variables and possessions of agents and log them collectively to the database. The database is used to generate graphs and can be used for further processing.

All data that has been collected can be automatically displayed on an interactive webpage. What is more the whole simulation can be packaged into a web-app. In this case, a menu can be displayed in the browser, from which simulation parameters can be chosen. A user can set the parameters, run the simulation, and observe the results directly in the browser. Past simulations remain accessible in the menu. Examples of simulations written in ABCE with interactive web interface can be accessed in https://www.taghawi-nejad.de/portfolio (Figs. 1 and 2).

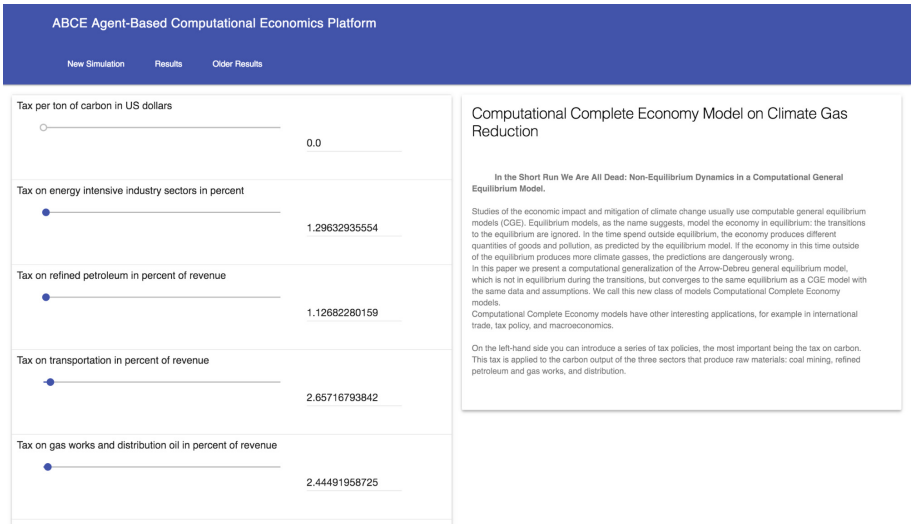

Fig. 1. An interactive web interface for the simulation

Fig. 2. A screenshot of an interactive exploration of the simulation as reflected in the graphs

8 Conclusion

ABCE is a modeling library that allows economic modelers to write agent-based models in a concise fashion. The modeler can concentrate on the decision logic of the model and does not have to deal with transactions between the agents or physical consistency. ABCE accelerates simulations from vanilla Python by providing core libraries in Cython and parallelizing computations on multi-core PCs. Moreover, ABCE collects statistics on observables defined by the user and makes them accessible in a nice web app interface, which provides an immediate representation of results. ABCE's documentation can be accessed in http://abce.readthedocs.io.

ABCE has been developed and used internally for the past 6 years. It is now our intention to open up the development to the community. For this purpose ABCE's source code is available at: https://github.com/DavoudTaghawiNejad/abce. ABCE is licensed under the Apache License, Version 2.0, and can thus be used permissively for academic and commercial purposes.

References

1. Allan, R.: Survey of agent based modelling and simulation tools. Technical report, October 2010. http://purl.org/net/epubs/work/50398
2. Caiani, A., Godin, A., Caverzasi, E., Gallegati, M., Kinsella, S., Stiglitz, J.E.: Agent based-stock flow consistent macroeconomics: towards a benchmark model. J. Econ. Dyn. Control **69**, 375–408 (2016). http://dx.doi.org/10.1016/j.jedc.2016.06.001
3. Collier, N.: RePast: an extensible framework for agent simulation (2003)
4. Farmer, J.D., Foley, D.: The economy needs agent-based modelling. Nature **460**(7256), 685–686 (2009). https://www.nature.com/nature/journal/v460/n7256/full/460685a.html
5. Jones, E., Oliphant, T., Peterson, P., et al.: SciPy: open source scientific tools for Python (2001). http://www.scipy.org/. Accessed 10 June 2017
6. Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K., Balan, G.: Mason: a multiagent simulation environment. Simulation **81**(7), 517–527 (2015). http://dx.doi.org/10.1177/0037549705058073
7. Nikolai, C., Madey, G.: Tools of the trade: a survey of various agent based modeling platforms. J. Artif. Soc. Soc. **12**(2), 2 (2009). http://jasss.soc.surrey.ac.uk/12/2/2.html
8. North, M., Howe, T., Collier, N., Vos, J.: The repast simphony runtime system. In: Proceedings of Agent 2005 Conference on Generative Social Processes, Models, and Mechanisms, p. 151 (2005)
9. Phelps, S.: Applying dependency injection to agent-based modeling: the JABM toolkit. Technical report WP056-12, Centre for Computational Finance and Economic Agents (CCFEA) (2012). http://www.bracil.net/ccfea/WorkingPapers/2012/CCFEA-Wpp.056-12-r2.pdf
10. Railsback, S.F., Lytinen, S.L., Jackson, S.K.: Agent-based simulation platforms: review and development recommendations. Simulation **82**(9), 609–623 (2006). http://dx.doi.org/10.1177/0037549706073695

11. Rossum, G.V.: Glue it all together with Python. In: Position Paper for OMG-DARPA-MCC Workshop on Compositional Software Architecture (1998). https://www.python.org/doc/essays/omg-darpa-mcc-position/
12. Serenko, A., Detlor, B.: Agent toolkits: a general overview of the market and an assessment of instructor satisfaction with utilizing toolkits in the classroom (2002). http://hdl.handle.net/11375/5601
13. Taghawi-Nejad, D.: Modelling the economy as an agent-based process: ABCE, a modelling platform and formal language for ACE. J. Artif. Soc. Soc. Simul. (2013). http://jasss.soc.surrey.ac.uk/16/3/1.html
14. Tobias, R., Hofmann, C.: Evaluation of free Java-libraries for social-scientific agent based simulation. J. Artif. Soc. Soc. Simul. **7**(1) (2004). http://jasss.soc.surrey.ac.uk/7/1/6.html
15. Walt, S.V.D., Colbert, S.C., Varoquaux, G.: The NumPy array: a structure for efficient numerical computation. Comput. Sci. Eng. **13**(2), 22–30 (2011)
16. Wikipedia: Comparison of agent-based modeling software – Wikipedia, the free encyclopedia (2017). https://en.wikipedia.org/w/index.php?title=Comparison_of_agent-based_modeling_software&oldid=783176104. Accessed 14 June 2017

http://www.springer.com/978-3-319-67216-8

Social Informatics
9th International Conference, SocInfo 2017, Oxford, UK, September 13-15, 2017, Proceedings, Part I
Ciampaglia, G.L.; Mashhadi, A.; Yasseri, T. (Eds.)
2017, XX, 645 p. 161 illus., Softcover
ISBN: 978-3-319-67216-8