Netlogo vs. Julia: Evaluating Different Options for the Simulation of Opinion Dynamics

Laura Burbach(✉), Poornima Belavadi, Patrick Halbach, Lilian Kojan, Nils Plettenberg, Johannes Nakayama, Martina Ziefle, and André Calero Valdez

Human-Computer Interaction Center, RWTH Aachen University, Campus-Boulevard 57, Aachen, Germany
{burbach,belavadi,halbach,kojan,plettenberg,nakayama, ziefle,calero-valdez}@comm.rwth-aachen.de

Abstract. Analysing complex phenomena, such as the world we live in, or complex interactions, also requires methods that are suitable for considering both the individual aspects of these phenomena and the resulting overall system. As a method well suited for the consideration of complex phenomena, we consider agent-based models in this study. Using two programming languages (Netlogo and Julia) we simulate a simple bounded-rationality opinion formation model with and without backfire effect. We analyzed, which of the languages is better for the creation of agent-based models and found, that both languages have some advantages for the creation of simulations. While Julia is much faster in simulating a model, Netlogo has a nice Interface and is more intuitive to use for non-computer scientists. Thus the choice of the programming language remains always a trade-off and in future more complex models should be considered using both programming languages.

Keywords: Agent-based modeling · Simulation · Julia · Netlogo · Programming languages

1 Introduction

Today, we live in a world, that is more complex than years ago. We are almost always and everywhere on the mobile Internet, using cloud storage or cloud computing and AI technologies such as deep learning. Also, when humans interact with each other or with digitized technology we speak of complex systems. The interaction of humans in such systems, for example in opinion-forming processes, leads to consequences that we cannot yet overlook or understand. An important component of socio-technical complex systems are single individuals that appear as human-in-the-loop [6]. To look at people, their interactions and the resulting overall behaviour, we need suitable methods, such as simulations.

V. G. Duffy (Ed.): HCII 2020, LNCS 12199, pp. 3–19, 2020.
https://doi.org/10.1007/978-3-030-49907-5_1

Simulations make it possible to observe the resulting overall system or the resulting behaviour by representing individual processes, procedures and behaviour. In addition, simulations make it possible to identify tipping points that lead to a different outcome of the overall system.

Agent-based models are a form of simulation. As the name implies, they always consist of agents. In addition to the agents, the environment in which the agents are located and with which they interact is also modelled. However, agents can be designed in different ways, depending on the context to be considered. For example, agents can be more than just people interacting with each other. If, for example, traffic jams are to be considered, cars are used as agents, if it is considered how possible forest fires can be avoided, the agents are trees. The agents differ not only in their form, but also in several other dimensions. For example, the agents can be completely or to a lesser extent autonomous. Their interests and character traits can also be different. For example, they can act selfishly or in favor of the totality of all agents. They can be outgoing or prefer to remain separate. Some agents are able to learn from their experiences or observations. Agents can also be of varying degrees of complexity [8]. Despite the potential complexity of agents and the possibility to model them in very different ways, most agent-based models tended to focus on simple, local rules [10]. Furthermore, there is a view that the simulations are mainly randomly implemented to run on a computer [14].

Various frameworks have been developed for creating agent-based models. The most established language or program of these is Netlogo [27]. But while Netlogo was authored by Uri Wilensky in 1999, the spread of the Internet also resulted in the evolving of different programming languages [6]. Thereby more languages can be used to create agent-based models. So far, it has not been considered which language is actually best suited for creating agent-based models. Therefore, in this study we investigate whether Netlogo or Julia is better suited for creating agent-based models.

2 Related Work

In this study, using agent-based modelling we consider opinion formation processes, thus we look at a complex system. We want to know, whether it is possible to create an agent-based model with the programming language Netlogo and the programming language Julia. We further consider, how the two languages differ, which are the strengths for creating agent-based models of each programming language and which are the disadvantages. Contentwise, we built a bounded rationality model to simulate opinion formation.

Therefore, we explain, which aspects lead to complexity, we introduce the method agent-based modelling and the two programming languages Netlogo and Julia. Besides, we eplain what is known in theory about opinion formation or the spread of information.

2.1 Complexity and How to Model It

When examining opinion-forming processes, we look at a complex system. Such complex systems can be divided into several ontological levels or interacting subsystems on a micro- or macro-level [9] We first have to look at how systems are structurally designed in order to deduce what leads to complexity [6]. Further, complex systems lead to emergent phenomena. These complex systems and emergent phenomena are difficult to understand, because while it is easy to observe the individual system components, the resulting overall system cannot be considered as the sum of its parts. Instead, understanding the system behavior requires more than understanding the individual parts of the system [6].

Complex vs. Complicated. When we look at complex systems, we do not necessarily mean complicated systems. A system consisting of components can initially be both complicated and complex. However, while the term complicated is always related to human understanding, the term complex is not necessarily so. If something is complicated, such as a mathematical differential equation, this means that it is difficult for us humans to understand. To be complex at the same time, the equation would have to contain many small parts. However, it is also possible that an equation consists of few parts and is therefore not complex, but is nevertheless complicated to understand. The two terms therefore both refer to a system consisting of components, but mean different aspects of the system and a system which is complicated does not necessarily have to be complex system and vice versa. A complex system consists of many subcomponents, whose interactions make it difficult to predict the behaviour of the system. The number of components as well as the complicated interactions of the parts are considered complex [4,24]. Another characteristic of complex systems, which is particularly important for our study, is that complex systems are always dynamic. If a system consists of many parts, but does not show dynamics but remains static, it is never complex. It is easy to investigate it comprehensively [6].

Emergence. Typically, we look at individual components of a system. From these subcomponents we then often infer the behavior of the overall system. However, as Aristotle said, the whole is more than the sum of its parts, and it is therefore not really correct to observe only the components and conclude on the overall behavior. However, it is problematic that we can usually observe and understand individual components or individual behavior, but the overall behavior is often more difficult to observe. If the interaction of the individual components results in a system that cannot be described by the sum of the individual components, we speak of emergence.

With agent-based models we can make emergent behavior visible. We can model the individual agents and design them according to individual rules that they follow at the micro level. When the agents interact with each other and with their environment, unpredictable social patterns, i.e. emergence, occur [3].

2.2 Agent-Based Modelling

To analyse complex systems we need a suitable approach, such as simulations, which enable to model the individual parts of a system and thus make the overall behavior visible. For the simulation of complex systems, agent-based models are very well suited [11].

Agent-based models always consist of the agents or individuals and the environment in which the agents reside [2]. They are neither a representation of reality, nor fully realistic or even complete. Instead, they show a simplified reality. Nevertheless, agent-based models show behaviour on an individual level close to reality. By mapping the individual behavior, the behavior of the overall system can then be qualitatively observed [20]. Agent-based models are well suited to replicate data and present the results to non-experts [17]. The use of a method always requires an evaluation of the method. Evaluating agent-based models is not easy. In order to evaluate them, independent replicating and comparing with other model as well as a validation are necessary [20].

The basis of agent-based modeling is the single agent or the individual. This agent is modelled programmatically as a template. In simulation, due agents make their own decisions based on how they perceive the environment in which they are situated. The perceptions of an agent usually determine the behavioral intent of the agent. If the agents are in a social network, as in our model, they influence their neighbours in the next iteration by their behavioural intention or the behaviour they show. To determine the probability of organizational acceptance, we analyze the results of several agent-based simulations.

A simple way to create agent-based models is to use software toolkits developed for the creation of simulations. These include the Netlogo toolkit considered in this study. With the use of such toolkits, it is easy to formulate the behavior of the individual agents. They also usually contain some useful interfaces. The interfaces allow to visualize the simulation states, interact with the simulation parameters and export the simulation results. In addition, they usually contain a batch mode. This is used to run a large number of simulations. Optimization strategies, such as genetic algorithms, help to find the most suitable parameters [7].

To create agent-based models, Netlogo [27] is the language most commonly used. Nevertheless, there are some other programming languages that are also suitable for creating agent-based models and that seem to be partly more intuitive, at least for people with programming experience. Therefore, in this study we compare two programming languages with respect to their suitability for creating agent-based models.

2.3 Opinion Formation and Bounded Rationality

In describing social phenomena, social scientists traditionally have tended to employ causal modeling techniques. That is, phenomena are explained by causally linking different variables. However, when describing phenomena like opinion formation in groups, repeated interactions between people appear to be

more influential than static variables [21,22]. Analytical models for the process of opinion formation therefore focus on group dynamics. They employ agents whose opinion develops over time as they interact with other agents whose opinion may be similar or different from their own. Computer simulations can be used to explore how varying different parameters, like the number of agents or the way agents interact with each other, will affect the distribution of opinions. Hegselmann and Krause [15] give an overview over how different models mathematically describe the process in varying complexities. One distinction between models is how opinion is represented. For continuous opinion dynamics, the assumption is that opinions are one-dimensional in that they can be described as a number. The smaller the difference between two numbers is, the closer are the opinions they represent. Another main distinction between the models is the way in which other agents' opinions influence one agent's own opinion, i. e., the weight which one agent puts on others' opinions. In the easiest case, this weight is modelled as constant, but it might also be modelled as differing, e.g., dependent on the susceptibility of each agent or as dependent on the disparity between two agents' opinions. This last case can be described by so-called bounded confidence models which have been proposed by both Hegselmann and Krause [15] and Nadal [18]. With a bounded confidence model, the agent will only interact with agents whose opinion is relatively close to their own. To put it another way, they will only put weight on similar opinions. The threshold for similarity is defined as the bounds of confidence epsilon which, assuming continous opinion dynamics, represents the maximum difference between the numbers ascribed to the opinions where the other's opinion will still be considered. An extension to this model of bounded confidence is something we call the backfire effect. As described by Jager [16], if an agent interacts with another agent whose opinion is very dissimilar, they will not just ignore that opinion. Instead, they will shift their opinion to be even further away from the other agents' dissimilar opinion. To summarize, for a bounded confidence model with backfire effect, an interaction between two agents has three possible outcomes: 1. If the difference between their opinions is smaller than or equal to a certain confidence interval epsilon, their opinions will converge. 2. If the difference between their opinions is bigger than or equal to a certain backfire threshold (which might be equal to epsilon), their opinions will diverge. 3. If epsilon and the backfire threshold are not equal and the difference between their opinions is between epsilon and the backfire threshold, their opinions will remain unchanged.

3 Method

Using two different programming languages (Netlogo and Julia language), we created two identical agent-based models that simulate opinion formation. Since our primary aim was to find out whether agent-based models could be implemented equally well in the two programming languages, we chose the most basic model of opinion-forming: bounded rationality.

We built the agent based models using the Atom editor of the Julia programming language and version 6.0.4 of the multi-agent programming language

Netlogo, which was developed by Wilensky [27]. For the following analysis of the results we used R Markdown.

3.1 First Steps in Agent-Based Models

While we have previously (see Sect. 2) explained what agent-based models are and what they are used for, we following describe how they are structured programmatically. We start with the most basic components.

An agent-based model usually contains a "setup" and a "go" procedure. The "setup" procedure defines a kind of basic state at the beginning of the simulation. The "go" procedure then specifies what happens in a single step of the simulation.

In Netlogo the "setup" procedure usually looks like in Fig. 1. In Netlogo, procedures always start with "to" and end with "end". Clear-all makes the world go back to its initial, empty state. For example, if colors were assigned to the spots where the agents are located, they will now turn black again. Create-turtles creates the specified number of turtles, here 100. The turtles usually start at the origin, i.e. in the middle of patch 0.0. The code in the square brackets after create-turtles here indicates that the turtles start at a random x and y coordinate. The square brackets could also be used to create other commands for the agents. Reset-ticks makes sure that the tick counter starts. Once this code is created, the simulation starts in the interface by clicking the "Setup" button. In Julia the setup includes an additional configuration.

Additionally, the agents and their environment are designed before the simulation starts. For example, properties are assigned to the agents and the agents' environment is designed to resemble the reality of what is being observed. In our case, the agents do not have specific properties and the environment is also in its default state.

```
to setup
  clear-all
  create-turtles 100 [ setxy random-xcor random-ycor ]
  reset-ticks
end
```

Fig. 1. Setup procedure in Netlogo

3.2 Bounded Rationality Model

Since our primary goal was to compare the two programming languages with each other, we designed the parameters of the Netlogo model and the Julia model the identical way. Thus, we increased the comparability of the results of both models and reduced the complexity as much as possible. In the beginning

of our bounded rationality model, we defined the maximum number of agents, the maximum steps of the simulation, the seed, an epsilon as well as whether a backfire effect takes place or not. The epsilon indicates how different the opinions of two people can be, so that they still include the other person's opinion in their opinion formation. We further defined from the beginning, that each agent has an (floating) opinion between 0 and 1. In each simulation step, every agent compares his opinion with the opinion of an other agent. For example, if Anna compares her opinion with Ralf and the distance between the opinion of Anna and Ralf is smaller than the defined epsilon, then the two converge in their opinions. Additionally we defined in the beginning, whether an backfire effect takes place or not. When the simulation includes the backfire-effect and Ralf's opinion deviates more than the epsilon indicates from Anna's opinion, then the opinion of Anna distances from the opinion of Ralf.

While in Netlogo the parameters for the simulation runs are determined in the Behavior Space (see Fig. 4), in Julia the initial settings are determined in the "main" procedure, what can be seen in Fig. 6.

As can be seen in Fig. 6 and 4, we set the number of agents 100 to 500 in increments of hundreds (100:100:500). We varied the epsilon between 0.1 and 1 in increments of 0.1 and varied between with backfire-effect and without (true/false). We set the maximum number of steps to 100.

Go Procedure. Here we compare the "go" procedures, so what happens in each step of the simulation, of Netlogo and Julia (see Fig. 2 and Fig. 3). Both codes look similar. In Netlogo (see Fig. 3), the procedure starts by addressing the agents (ask turtles). The next line of code says, that the addressed agent gets the opinion of one random other agent. The subsequent lines of code determine what happens to the (new) opinion of the agent. If the other agent's opinion differs less from his own opinion than the epsilon (see above), the agent assumes the average opinion of the two opinions. This means that the opinions of the two agents are added together and divided by two. However, if the opinion of the other agent is further away than the respective (may vary) epsilon indicates, it checks whether the backfire effect exists. If the simulation is set to show that the effect exists, the opinion of the agent is half the distance away from the opinion of the other agent. At the end, the code indicates that the color of the agents depends on the opinion. However, this is only for illustration in the interface. Before the procedure ends, one more "tick" is counted as one time unit.

The "go" procedure in Julia is very similar. One difference is that the procedure is passed a configuration (config) at the beginning. Furthermore, an agent list with the agents in random order is passed.

3.3 What Do We Compare

To find out whether both programming languages are equally suitable to simulate our bounded rationality model, we look at several measurable criteria. These criteria include the outcomes and performance of both models. They further include how many lines of code are necessary to program the simulation.

```julia
function go(config, rng, agent_list)
    for one_agent in shuffle(rng, agent_list)
        idx = rand(rng, 1:config.agent_count)
        other_agent = agent_list[idx]
        if abs(other_agent.opinion - one_agent.opinion) < config.epsilon
            one_agent.opinion = (other_agent.opinion + one_agent.opinion) / 2
        else
            if config.backfire
                if one_agent.opinion < other_agent.opinion
                    one_agent.opinion = one_agent.opinion -  abs(other_agent.opinion - one_agent.opinion)/2
                else
                    one_agent.opinion = one_agent.opinion + abs(other_agent.opinion - one_agent.opinion)/2
                end
                if one_agent.opinion < 0
                    one_agent.opinion = 0
                end
                if one_agent.opinion > 1
                    one_agent.opinion = 1
                end

            end
        end
    end
    agent_list
end
```

Fig. 2. Go procedure in Julia

```
to go
  ask turtles [
    ; get the opinion of one random other turtle
    let otheropinion [opinion] of one-of other turtles

    ; is opinion in range
    ifelse abs ( opinion - otheropinion ) < epsilon [
      ;then take average opinion
      set opinion ((opinion + otheropinion )/ 2)
    ]
    [ ; do we have backfire
      if backfire [
        ; shift away half the distance
        ifelse (opinion < otheropinion) [
          set opinion opinion - ( abs ( otheropinion - opinion ) / 2 )
        ] [
          set opinion opinion + ( abs ( otheropinion - opinion ) / 2 )
        ]

        if opinion < 0 [set opinion 0]
        if opinion > 1 [set opinion 1]
      ]
    ]
    ; set color to red range
    set color 11 + opinion * 8
  ]
  tick
end
```

Fig. 3. Go procedure in Netlogo

Another aspect, that we take into consideration, is, if learning Julia and Netlogo is equally difficult. For this aspect we consider both computer scientists who are familiar with other programming languages and a person who has no previous experience with programming languages. We further compare the explorability and scalability of both languages.

4 Results

Before we present the results of our bounded rationality model, we reflect on the extent to which the two languages Julia and Netlogo are suitable for developing agent-based models and how easy it is to get started with the two languages.

4.1 Getting Started with both Languages

Both Julia and Netlogo are languages that address both researchers and beginners. Netlogo is derived from Logo a language that is aimed at children to lern programming. The core aim of Netlogo is agent-based modeling and it has several primites for this purpose. Julia is aimed at scientists that require both performance and understandable code. The core aim of Julia is to make code fast, reusable and easy to understand. This quick introduction by no means covers the breadth of both of these languages, it aims to provide a high-level overview.

Netlogo. Netlogo as a modelling language for agent-based modeling is very well suited for beginners wanting to use agent-based modeling. It comes with a rich variety of example models that users can explore and provides a graphical user interface and a graphical user interface toolkit to create models that even non-experts can use. Thus, Netlogo is visually appealing and the interface enables users to create and test agent-based models and also simplifies the initial creation of a model. Figure 5 shows the Interface of our simulation. Netlogo also provides methods for inspecting the model (reporters and visualizations) and for exploring the impact of model parameters on system behavior (i.e. the *behavior space* feature, which allows the user to run any number (usually several hundred) of simulations). The latter allows turning of the GUI for faster simulations (see Fig. 4).

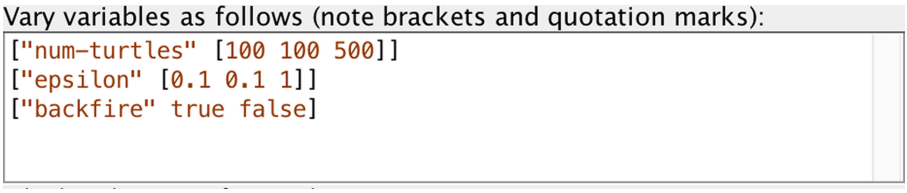

Fig. 4. Behavior space in Netlogo

Netlogo provides immediate visual feedback for the user of an agent-based model and has easy to understand primitives that allow modelling of agent behavior, agent interactions, and agent-environment interactions. It provides an API for extensions, to allow other researchers to complement the functionality of Netlogo.

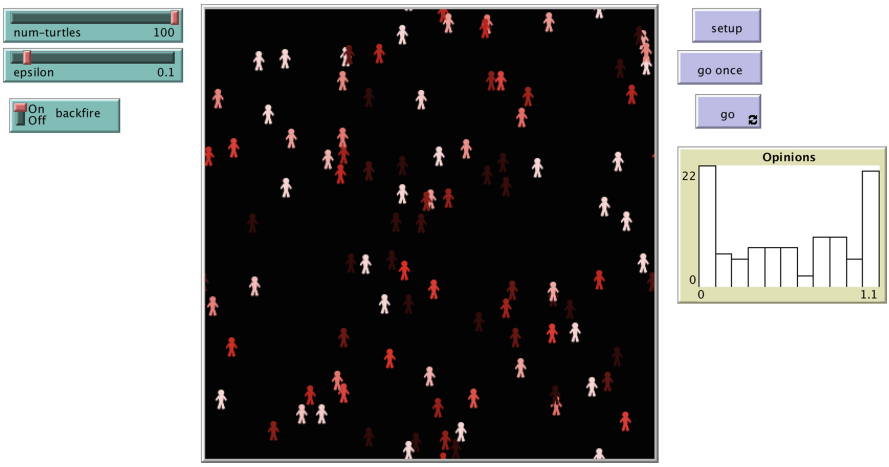

Fig. 5. Interface of our bounded rationality model in Netlogo

Overall, it is very easy to start using Netlogo. However, creating complex models requires understanding of usage contexts in the language. People coming to Netlogo with a computer science background may find some of the language concepts unintuitive and clunky. Several of the authors of this paper have found Netlogo syntax to be confusing and unnecessary simplistic.

Julia. Julia was initially introduced by a group of computer scientists and mathematicians at MIT under the direction of Alan Edelman. Compared to other programming languages Julia is considered fast, easy to learn and use and it is open source. Further advantages of Julia compared to other programming languages are that it supports parallelization or practical functional programming and can be easily combined with other programming languages and libraries. Finally, there is already a group of active users who develop packages (and thereby add functions to the base language; as of April 6, 2019, there are 1774 registered packages).

Julia is not a language specifically written for agent-based modeling. Julia is a general purpose programming language that uses a just-in-time compiler to generate low level machine code (using LLVM). This means there is no native support for typical agent-based modeling tasks. There is a library for agent-based modeling called `agents`. However, our intention here was to compare the

programming language itself without the use of a library. It is unclear whether the library is going to be maintained in the future, whereas Julia's support is not likely to expire soon.

This means the user has to design all tools for agent-based modeling themselves. However, this is not necessarily very hard. It depends on the complexity of the model. When this barrier has been overcome, writing a model becomes easier. The language is very similar to python.

```
function main()
    # create config objects
    agent_counts = 100:100:500
    epsilons = 0.1:0.1:1
    max_steps = [100]
    replications = 1:50
    my_config = generateBatchConfig(agent_counts,
                                    epsilons,
                                    max_steps,
                                    replications)
    startandsave(my_config, "results.csv")
```

Fig. 6. Main procedure in Julia

4.2 Comparison of Agent-Based Modeling Results of Julia and Netlogo

Following, we present some exemplarily results of our bounded rationality model. We also show, if the model created with Netlogo showed the same or different results as the model created with Julia. Based on these results, we compare the two considered programming languages and show their advantages and disadvantages.

Opinion Change of Agents. Following we consider, how the opinion of the agents changed during the simulation steps of the bounded rationality model with and without backfire effect. At this point we do not distinguish between the two programming languages used.

We use four examples (see Fig. 7) to illustrate how the agents change their opinion during the simulation and how different the opinions look at the end of the simulation. As can be seen at the top left of Fig. 7, one possible outcome is

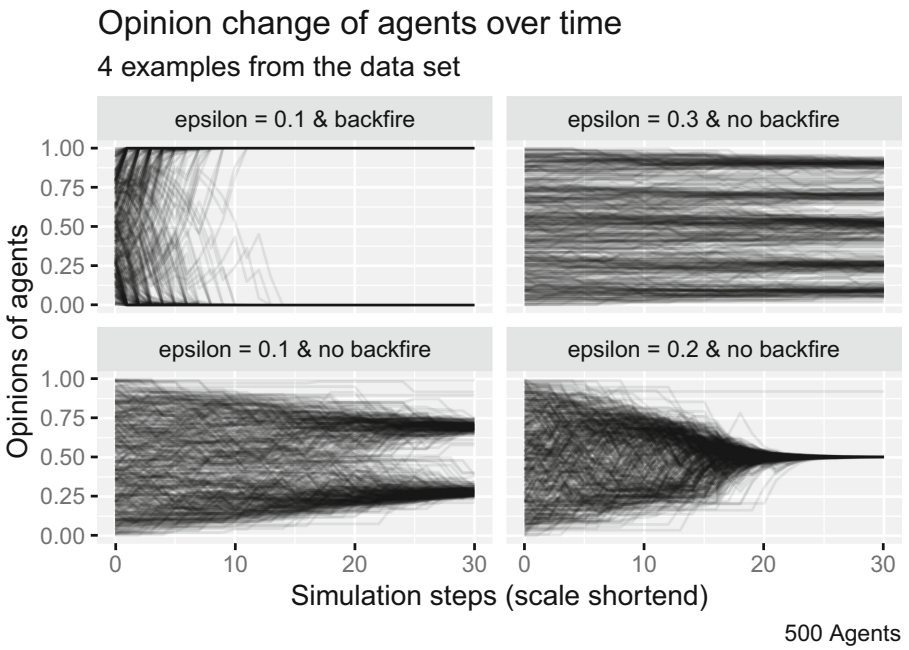

Fig. 7. Four exemplarily examples

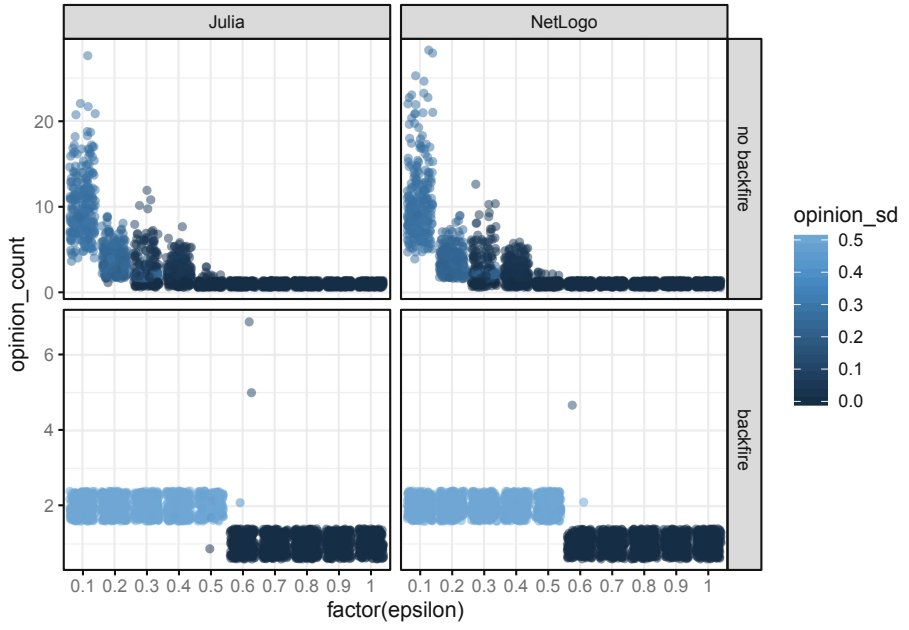

Fig. 8. How language, epsilon and backfire influence the opinion count

Table 1. Comparison of the ten most different settings for both languages

Epsilon	Backfire	agent_count	t-value	p-value	Degrees of freedom
0.3	FALSE	200	−2.245263	0.0269995	97.95147
0.1	TRUE	200	2.103480	0.0386304	78.40132
0.4	FALSE	200	1.989794	0.0494162	97.43888
0.3	FALSE	400	−1.606152	0.1115634	94.89243
0.1	FALSE	200	1.601283	0.1126316	95.08476
0.1	TRUE	100	−1.564258	0.1221526	71.82311
0.2	FALSE	200	1.412877	0.1609382	95.62983
0.5	FALSE	300	1.416342	0.1630015	49.00000
0.4	FALSE	100	1.392850	0.1669769	93.25748
0.1	TRUE	300	−1.301375	0.1965824	86.58427

that the opinions of the agents diverge completely and only two extreme opinions are formed. After less than 15 simulation steps, every agent has either opinion 0.00 or opinion 1.00. In this example, the epsilon is low and the backfire effect takes place.

In comparison to this example, in the third example (bottom left) no backfire effect takes place. In both examples, the epsilon is 0.1. Comparing the two examples, it becomes clear that the backfire effect increases the divergence of opinions. While in the first example two clear opinions quickly establish, in the third example there are more different opinions for a longer time. After 20 simulation steps, two groups of agents form whose opinions are similar to each other. Nevertheless, even after 30 simulation steps, these agents still have similar opinions, but not one uniform opinion.

In example 4 (bottom right), also no backfire takes place. Here, the different opinions converge to a consensus of opinion. After around 20 simulation steps each agent has the opinion 0.5.

In contrast, in example 2 (top right) no majority opinion develops, but several groups with the same opinions form. In this example, the epsilon is higher than in the other examples, which leads the agents to accept opinions that differ more from their own than in the other examples.

Influence of Programming Language, Epsilon and Backfire on Opinion Count. After we looked at the opinion formation of the agents itself, we now consider, whether the epsilon, if the backfire effect takes place or not and the programming language has an influence on the existence of different opinions. To look at the influence of the enumerated factors, we consider (see Fig. 8) how many different opinions exist (y-axis). We further consider the standard deviations of the opinions (color) to analyse how different the opinions are.

As Fig. 8 shows, if the epsilon is higher than 0.55, practically all agents have only one opinion (sd = 0.0), regardless of whether the backfire effect takes place

or not and which programming language is used. When the epsilon is lower than 0.55 and the backfire effect takes place, there are two opinions among the agents that diverge to the two extremes of opinion (sd = 0.5). In comparison, when the epsilon is lower than 0.55 and no backfire effect takes place, the agents have more different opinions, but the standard deviations of the opinions are lower (less bright) than in the simulations with backfire effect. The lower the epsilon is, the higher is the amount of opinions. Comparing the two programming languages, the amount of different opinions is a bit higher when NetLogo is used, but the difference is small. Overall, the two programming languages showed almost the same qualitative results. As Table 1 shows, the quantitative comparison of both languages showed, that except of three simulation runs, the t.test wasn't significant. Thus the languages showed the same results.

4.3 Comparison of Julia and Netlogo After Our Bounded Rationality Simulation

When comparing both programming languages to create an agent-based model that simulates the bounded rationality model, Julia proved to be a faster language. The whole simulation took only 82.23 s, whereas the Netlogo simulation took 36 min. While the model calculation in Julia is much faster, Netlogo required less than half the lines of code. To write the bounded rationality model in Julia 97 lines of code were necessary, in Netlogo only 44 lines of code were necessary.

When we consider how difficult it is to learn the two programming languages, we also have to take into account the previous knowledge of the users. Thus Netlogo proved to be a language that is easier to learn for people without programming skills. In contrast, people with previous programming skills reported, that it is easier to learn Julia, because it is more similar to other already used programming languages (for example Python).

An advantageous feature of Netlogo, is that the platform contains an easy to use, clear and attractive interface. These interface makes it easier to get started with and learn the language for people without previous programming experience. The interface offers the user direct feedback, as the simulation runs visibly if he has written the code correctly and also immediately reports back error messages if the code is wrong. In addition, the interface allows the user to try out and change various things in the process.

Also, the fact that there is already a large library of existing agent-based models in Netlogo, since the language is used exclusively for this method, makes it easier to use, since existing models can be built upon or users can orient themselves on them.

In addition to the interface, the Logo programming language, which Netlogo uses, is also easy to use because there is only a manageable number of structurally different commands and users can quickly get a feel for which procedures and functions always need to be set when creating agent-based models.

5 Discussion

In our study, no language turned out to be the perfect programming language for creating agent-based models, but the choice of language seems to be a trade-off between various advantages and disadvantages and also between different potential users and use cases.

For people who have never used a programming language before and are not supported by people with previous programming experience, the entrance to the Netlogo language is certainly easier than to the Julia language. Likewise, starting with Julia is easier for people with programming experience, because they already know, how the language is probably organized. It can be assumed that modelers who are already very familiar with the language they use also develop more complex simulations than simulation based on simple rules [10]. So less effort in learning a language can certainly increase the complexity of the models.

One other aspect, that could be taken into account, is the time, that is needed to run the simulation. Here Julia turned out to be much faster. But, in many research areas or for many research questions it does not really matter, whether the language is really fast. One aspect, that is probably more important is, that very big simulations in Netlogo require high computing power and that the computers sometimes crash, making it impossible to calculate the model In this case Julia makes it possible to calculate the simulation without any problems.

Of course, we have only focused on one very simple bounded rationality model, so that we would have to create further simulations with both languages to be able to make statements about the generality.

Historically, the basis for analytical opinion dynamics models is given by psychological research and philosophical theories about social influence (e.g., [13,23]). And simulations based on those models have frequently proven to reliably enough reproduce real-life phenomena [22]. However, as Flache et al. [12] argue, there is a lack of recent empirical studies reassessing and replicating the assumptions underlying those models, let alone studies examining the size of epsilon in real-life interactions. In future research, finding a way to link analytical opinion dynamics models with contemporary empirical psychological findings would be desirable.

6 Conclusion and Outlook

The results of our research have shown that, although Netlogo has been established for a longer time, both programming languages are well-suited to create agent-based models. Comparing the two languages, we could not find one perfect language, but each language is the better choice for creating an agent-based model in some aspects. The decision for a programming language depends on different trade-offs (previous experience vs. support; time to create the model vs. time used for simulation run; nice interface vs. higher functionality). In the end, however, it does not make sense to decide in favor of one language against

the other, but to take advantage of both languages and thus use Netlogo for prototyping and Julia for larger simulations based on these prototypes.

With this study we compared Julia and Netlogo to create a very simple bounded rationality model. In the future, we would like to extend this comparison by using both languages for more complex simulations. We further plan to pursue with studies, that combine both languages.

Acknowledgements. This research was supported by the Digital Society research program funded by the Ministry of Culture and Science of the German State of North Rhine-Westphalia.

We used the following packages to create this document: `knitr` [28], `tidyverse` [25], `rmdformats` [1], `scales` [26], `psych` [19], `rmdtemplates` [5].

References

1. Barnier, J.: rmdformats: HTML output formats and templates for 'rmarkdown' documents. R package version 0.3.6 (2019). https://CRAN.R-project.org/package=rmdformats
2. Bonabeau, E.: Agent-based modeling: methods and techniques for simulating human systems. Proc. Nat. Acad. Sci. U.S.A. **99**(Suppl 3), 7280–7287 (2002). https://doi.org/10.1073/pnas.082080899
3. Bruch, E., Atwell, J.: Agent-based models in empirical social research. Sociol. Methods Res. **44**(2), 186–221 (2015). https://doi.org/10.1177/0049124113506405
4. Byrne, D.: Complexity Theory and the Social Sciences. Routledge, London (1999). https://doi.org/10.4324/9780203003916
5. Valdez, A.C.: rmdtemplates: rmdtemplates - an opinionated collection of rmarkdown templates. R package version 0.4.0.0000 (2020). https://github.com/statisticsforsocialscience/rmd_templates
6. Calero Valdez, A., Ziefle, M.: Human factors in the age of algorithms. understanding the human-in-the-loop using agent-based modeling. In: Meiselwitz, G. (ed.) SCSM 2018. LNCS, vol. 10914, pp. 357–371. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-91485-5_27. ISBN 978-3-319-91485-5
7. Valdez, A.C., Ziefle, M.: Predicting acceptance of novel technology from social network data-an agent-based simulation-approach. In: Proceedings of the International Conference on Competitive Manufacturing (2019)
8. Conte, R., Paolucci, M.: On agent based modelling and computational social science. SSRN Electron. J. (2011). https://doi.org/10.2139/ssrn.1876517
9. Conte, R., et al.: Manifesto of computational social science. Eur. Phys. J.-Spec. Top. **214**, 325 (2012). https://doi.org/10.1140/epjst/e2012-01697-8
10. Epstein, J.: Generative social science: studies in agent-based computational models, January 2006
11. Epstein, J.M.: Generative Social Science: Studies in Agent-Based Computational Modeling. Princeton Studies in Complexity. Princeton University Press, Princeton (2007). ISBN 0-691-12547-3
12. Flache, A., et al.: Models of social influence: towards the next frontiers. JASSS **20**(4), 2 (2017). https://doi.org/10.18564/jasss.3521. ISSN 1460-7425
13. French, J.R.P.: A formal theory of social power. Psychol. Rev. **63**(3), 181–194 (1956). https://doi.org/10.1037/h0046123. ISSN 0033-295X

14. Gilbert, N., Troitzsch, K.G.: Simulation for The Social Scientist, 2nd edn. Open University Press, Buckingham (2005)
15. Hegselmann, R., Krause, U.: Opinion dynamics and bounded confidence: models, analysis and simulation. JASSS **5**(3), 3–33 (2002). ISSN 1460-7425
16. Jager, W.: Uniformity, bipolarization and pluriformity captured as generic stylized behavior with an agent-based simulation model of attitude change. Technical report, pp. 295–303 (2004)
17. Kiesling, E., et al.: Agent-based simulation of innovation diffusion: a review. CEJOR **20**, 183–230 (2012). https://doi.org/10.1007/s10100-011-0210-y
18. Nadal, J.-P.: Meet, discuss, and segregate!. Complexity **7**(3), 55–63 (2002)
19. Revelle, W.: psych: procedures for psychological, psychometric, and personality research. R package version 1.8.12 (2019). https://CRAN.R-project.org/package=psych
20. Rouchier, J., et al.: Progress in model-to-model analysis. J. Artif. Soc. Soc. Simul. **11**(2), 8 (2008)
21. Smith, E.R., Conrey, F.R.: Agent-based modeling: a new approach for theory building in social psychology. Pers. Soc. Psychol. Rev. **11**(1), 87–104 (2007). https://doi.org/10.1177/1088868306294789. ISSN 1088-8683. Official journal of the Society for Personality and Social Psychology, Inc. www.ncbi.nlm.nih.gov/pubmed/18453457
22. Vespignani, A.: Modelling dynamical processes in complex sociotechnical systems. Nat. Phys. **8**(1), 32–39 (2012). https://doi.org/10.1038/nphys2160. ISSN 1745-2473
23. Wagner, C.: Consensus through respect: a model of rational group decision-making. Philos. Stud. **34**(4), 335–349 (1978). https://doi.org/10.1007/BF00364701. ISSN 0031-8116
24. Waldrop, M.M., Gleick, J.: Complexity: The Emerging Science at the Edge of Order and Chaos. Viking Info, London (1992)
25. Wickham, H.: Tidyverse: easily install and load the 'Tidyverse'. R package version 1.3.0 (2019). https://CRAN.R-project.org/package=tidyverse
26. Wickham, H., Seidel, D.: Scales: scale functions for visualization. R package version 1.1.0 (2019). https://CRAN.R-project.org/package=scales
27. Wilensky, U.: Center for connected learning and computer based modeling. Northwestern University (1999). http://ccl.northwestern.edu/netlogo
28. Xie, Y.: knitr: a general-purpose package for dynamic report generation in R. R package version 1.28 (2020). https://CRAN.Rproject.org/package=knitr