Towards Rigorous Agent-Based Modelling
Linking, Extending, and Using Existing Software Platforms

Dissertation
zur Erlangung des
mathematisch-naturwissenschaftlichen Doktorgrades
"Doctor rerum naturalium"
der Georg-August-Universität Göttingen
im Promotionsprogramm Umweltinformatik (PEI)
der Georg-August University School of Science (GAUSS)

vorgelegt von
Jan Christoph Thiele
geboren in Braunschweig

Göttingen, 2014

Betreuungsausschuss

Prof. Dr. Winfried Kurth (Abteilung Ökoinformatik, Biometrie und Waldwachstum, Georg-August-Universität Göttingen)

Prof. Dr. Volker Grimm (Department Ökologische Systemanalyse, Helmholtz-Zentrum für Umweltforschung - UFZ, Leipzig und Institut für Biochemie und Biologie, Universität Potsdam und German Centre for Integrative Biodiversity Research - iDiv, Halle-Jena-Leipzig)

Mitglieder der Prüfungskommission

Referent: **Prof. Dr. Winfried Kurth** (Abteilung Ökoinformatik, Biometrie und Waldwachstum, Georg-August-Universität Göttingen)

Korreferent: **Prof. Dr. Volker Grimm** (Department Ökologische Systemanalyse, Helmholtz-Zentrum für Umweltforschung - UFZ, Leipzig und Institut für Biochemie und Biologie, Universität Potsdam und German Centre for Integrative Biodiversity Research - iDiv, Halle-Jena-Leipzig)

2. Korreferent: **Prof. Dr. Steven Lytinen** (School of Computing, DePaul University, Chicago)

Weitere Mitglieder der Prüfungskommission

Prof. Dr. Niko Balkenhol, (Abteilung Wildtierwissenschaften, Georg-August-Universität Göttingen)

Prof. Dr. Holger Kreft, (Free Floater Nachwuchsgruppe - Biodiversität, Makroökologie und Biogeographie , Georg-August-Universität Göttingen)

Prof. Dr. Stefan Schütz, (Abteilung Forstzoologie und Waldschutz, Georg-August-Universität Göttingen)

Prof. Dr. Kerstin Wiegand, (Abteilung Ökosystemmodellierung, Georg-August-Universität Göttingen)

Tag der mündlichen Prüfung: 08.12.2014

dedicated to *Lea & Lukas*
the most important outputs of the period of PhD

Acknowledgement

I would like to thank ...

- Prof. Dr. Winfried Kurth - for supervising me and giving me the freedom to follow my mind
- Prof. Dr. Volker Grimm - for being more than just a supervisor - becoming a teacher, inspirator, and friend
- Prof. Dr. Steven Lytinen - for kindly accepting to be co-referee
- Tanja - for her love and endurance
- Lea & Lukas - for showing me the meaning of life
- Mum and Dad - for their believe in my abilities and their continuous support
- my brother - for advices at the beginning of my programming experiences long time ago and for always having an open ear
- my grandfather - for never believing that modelling makes sense but believing in me
- Kent - for being my friend and brother in spirit
- Janka, Daniela, Thomas, and Marco - for being friends
- Michael, Niki, Robert, Sebastian and Tim - for being my colleagues and friends
- Tim for proofreading parts of the thesis
- Ilona - for doing all the secretary and organizational work and always having an open ear
- Prof. Dr. Dr. h.c. Branislav Sloboda and Prof. Dr. Joachim Saborowski - for giving me the chance to become a researcher
- Mr. Bellmund - for his help in darker days
- the colleagues from the Department of Ecoinformatics, Biometrics and Forest Growth and from the Department of Ecosystem Modelling for the joint lunches
- Uri Wilensky, Seth Tisue and the other developers - for NetLogo
- R Core Team - for R
- all people who crossed my way - for making me happy, my life easier, my dreams come true, or for just making me wiser and stronger.

Happy modelling!

Abstract

Agent- or individual-based modelling is a modelling approach where the heterogeneity of entities, i.e. agents, matters. In the past ten to twenty years agent-based models became increasingly popular and were applied in many different research areas, from computer science over sociology and economy to ecology. Built by rule systems instead of differential equations they cannot be solved analytically but have to be implemented as computer software and analysed by running simulations. As the field of agent-based modelling is relatively young compared to, for example, mathematical modelling with differential equations, established standards in developing, implementing, describing, and analysing are missing or currently being arising. Instead, many modellers build, implement and analyse their models from scratch and reinvent the wheel.

The work at hand aims to support the process of establishment of standards in agent-based modelling. After a short general introduction in the first chapter, the second chapter gives an introduction into the history of agent-based modelling in different research areas, discusses open issues in agent-based modelling, presents the most important toolkits/languages/Integrated Development Environments (IDE) for implementing agent-based models, and closes with a deeper look on the IDE/language NetLogo and some extensions developed here.

In the third chapter a framework for building and analysing agent-based models by linking two existing and well-known toolkits/languages, NetLogo and R, is described. Such a seamless integration of an agent-based modelling environment with a statistics software enables the modeller to design simulation experiments, store simulation results, and analyse simulation output in a more systematic way. It can therefore help close the gaps in agent-based modelling regarding standards of description and analysis.

The fourth chapter of this theses provides a "cookbook" of many important methods for calibration of agent-based models as well as for sensitivity analysis. Such a comprehensive overview of well-known and established techniques enables the modeller to become aware of existing methods, learn what they can deliver and where to apply them. Furthermore, the recipes contain application examples implemented and adaptable to other models implemented in NetLogo under the use of the framework introduced in the third chapter.

A key feature of science - the replication of experiments - is discussed in chapter five with focus on the field of agent-based modelling in ecology. It should encourage the community to replicate models and publish the replications. Replication of models fulfils different purposes: it uncovers implementation-dependent differences in model results, it shows lacks in documentation and/or documentation protocols as well as robustness tests, and it is a first step towards community-tested standard models or model components.

The work closes with an integrated discussion and outlook on open issues.

Contents

List of Tables

List of Figures

CHAPTER I

Introduction

"All models are wrong, some are useful."

*(George E. P. Box, * 1919 † 2013)*

Grimm and Railsback [2005, p. 3] describe modelling as the "attempts to capture the essence of a system well enough to address specific questions about the system". Thus, the core of modelling is a purpose-oriented abstraction and simplification. Imboden and Koch [2003] get it to the heart: a model is not a copy of the real system, it is the glasses seeing the real system applying some filter. This is in good accordance to other definitions of modelling from various research areas [e.g., Squazzoni, 2012, Töllner et al., 2010, Soetaert and Herman, 2009, Pretzsch, 2001, Lutz, 1998]. Stachowiak [1973], for example, characterizes a model by three features: mapping, reduction and pragmatism. The mapping feature describes that models are representations of real systems, which themselves could also be models. The reduction feature defines that a model does not cover all features of the represented system but only those that are relevant to the purpose. The last feature - pragmatism - means that the mapping between the model and the real system does not need to be unambiguous. It is purpose-oriented, e.g., for a limited time period. Such properties apply to models from various research areas and will be used here as a general characterization of a model.

Müller and Müller [2003] give a lot of examples of models where the mentioned properties are complied, ranging from model-cars used as a toy by small children, over archaeological reconstructions of, e.g., Pompeii to learn about life in the ancient world or a primeval man counting his hunted mammoths by pebbles. Following Weisberg [2013], these examples fit to the class of concrete models, i.e., real, physical objects.

A second class of models are mathematical ones [Weisberg, 2013]. Often, such models are described by differential equations. These models can be constructed with pen and paper and consist of a set of equations. They can sometimes be analysed, i.e., solved, with some algebra analytically. The most popular example from population ecology is the Lotka-Volterra predator-prey model [Lotka, 1925, Volterra, 1931].

Beside these two classes a third one exists: computational or simulation models. These

models can also be defined with pen and paper, however, results can only be processed reasonably by converting them into computer code, i.e., algorithms, and running simulations, i.e., executing the computer code. As Weisberg [2013] points out, computational models can also be mathematical models. For example, models defined by a large set of differential equations can make it desirable to be solved numerically by computer simulations as they are too complex for being solved analytically. However, there is another type of models which fits into the class of computational models but is not described by equations alone. Sometimes they are called complex or bottom-up models and are built by a set of rules. The first models of this class are called grid-based models and have been extensions of Cellular Automata models, processing rules on grid cells [Grimm, 2002b]. Later on, a related type, so-called agent-based models, has evolved. This is the kind of model this thesis focuses on.

I.1. Agent-Based Modelling

The specific nature of agent-based models (ABMs) is the representation of unique entities of a system as heterogeneous entities in the model. This is in contrast to equation-based mathematical models where the heterogeneous entities are averaged into a stock variable, i.e., treated as being homogeneous. However, North and Macal [2007] stated that the whole of many systems is greater than the simple sum of their constituent parts. Thus, ABMs instead consist of multiple individual agents, which can be humans, animals, organisms, institutions, vehicles, computers and so on, with explicitly represented traits and behaviours [Grimm and Railsback, 2005, Gilbert, 2007, Squazzoni, 2012]. A key characteristic of this modelling approach is the emergence of simulation results from the more or less complex interactions among the agents. Therefore, such models are useful when local interactions on the micro level are essential for the description of patterns on the macro level [Page, 2012]. Grimm [2008] suggests selecting an ABM approach when at least one of the three following agent-level aspects is considered important for explaining system-level behaviour: heterogeneity among individuals, local interactions, and adaptive behaviour based on decision making.

As described more detailed in Chapter II the origins of the ABM approach in computer science go back to the late 1970s [e.g., Hewitt, 1976] with the development of so-called multi-agent systems (MAS) as a part of the distributed artificial intelligence (DAI) research area [Green et al., 1997, Sycara, 1998]. Their wider use in computer science began only in the 1990s [Luck et al., 2003, Wooldridge, 2005, Weiss, 1999]. Definitions of the term MAS and what an agent is, can be found, for example, in Wooldridge [2005] and Jennings [2000]. Examples for the use of MAS with intelligent agents in the field of computer science include computer games, computer networks, robotics for manufacturing, and traffic-control systems [for examples, see Oliveira, 1999, Luck et al., 2003, Shen et al., 2006, Moonen, 2009].

With increasing importance of questions about coordination and cooperation within the MAS the connections to social sciences arose [Conte et al., 1998] and the field of agent-based social simulation (ABSS), that is, an agent-based modelling approach as part of computational sociology became a "counter-concept" to the classical top-down system dynamics and microsimulation approaches [Gilbert, 1999, Squazzoni, 2010]. ABSS is mainly used for theory testing and development [Macy and Willer, 2002, Conte, 2006] and applied to simulations of differentiation, diffusion, and emergence of social order in social systems [for examples, see listings in Macy and Willer, 2002, Squazzoni, 2010] as well as to ques-

tions about demographic behaviour [Billari and Prskawetz, 2003]. The most famous models in social sciences are Schelling's segregation model [Schelling, 1969] and the Sugarscape model of Epstein and Axtell [1996].

Strongly related to the development of ABMs in social sciences is the establishment of the ABM approach in economics, which is called agent-based computational economics (ACE) and related to the field of cognitive and evolutionary economics. The aims of ACE can be divided into four categories: empirical understanding, normative understanding, qualitative insight as well as theory generation and methodological advancement [for details, see Tesfatsion, 2006]. ACE was applied, for example, to the reproduction of the classical cobweb theorem [e.g., Arifovic, 1994], to model financial/stock markets [see LeBaron, 2000, for a review] as well as to the simulation of industry and labour dynamics [e.g., Leombruni and Richiardi, 2004].

In contrast to ABSS and ACE, the agent-based modelling approach has a slightly longer tradition in ecology [Grimm and Railsback, 2005]. The development of so called individual-based models is less closely related to the developments of MAS, because ecologists early became aware of the restrictions in classical population models (differential equation models) and looked for alternatives. Over the last three to four decades hundreds of IBMs were developed in ecology [DeAngelis and Mooij, 2005]. For reviews see, for example, Grimm [1999] and DeAngelis and Mooij [2005].

Besides these four main research areas, there are many other disciplines in which ABMs are increasingly used, often within an interdisciplinary context. Examples include ecological economics [e.g., Heckbert et al., 2010], marketing/socio-psychology [e.g., North et al., 2010], archaeology/anthropology [e.g., Griffin and Stanish, 2007], microbiology [e.g., Ferrer et al., 2008], biomedicine/epidemiology [e.g., Carpenter and Sattenspiel, 2009], criminology [strongly related to ABSS, e.g., Malleson et al., 2010] and land-use management [e.g., Meyer et al., 2012, Matthews et al., 2007].

I.2. Rigorous Agent-Based Modelling

As shown, the ABM approach is relatively new compared to, for example, classical mathematical modelling. Therefore, it is missing an established theoretical framework as known from mathematics with its notations, well-known formulas, and methods. Standards or best-practices for designing, implementing, analysing and communicating ABMs are still missing or under establishment [Galán et al., 2009, Janssen et al., 2008, Grimm, 2008, Grimm and Railsback, 2005]. On the conceptional level, recently several attempts to develop such standards in ABM have been made. The most important ones are described in the following.

Modelling Strategy For designing, fitting and validating ABMs, for example, the *Pattern-Oriented Modelling* (POM) approach was proposed by Grimm et al. [2005]. The idea of POM is that complex systems usually can be characterized by multiple patterns that can be observed at different hierarchical levels (i.e., individual and system) and scales (i.e., spatial and temporal). These patterns reflect the internal organization of a system. POM then means to decode this information and thereby reveal the internal organization. Examples of patterns at the population level include size-distributions of animals, frequency-area distributions of wildfires, and sex ratios [Grimm and Railsback, 2012]. At the individual level, patterns can be, for example, distribution of body sizes at certain ages or of life spans. A

central idea of POM is to design models so that in principle they can reproduce the entire set of multiple patterns simultaneously. Focusing only on the problem, i.e., a single pattern that should be addressed with a model, often results in too simple models whereas designing a model on the basis of available data make models often more complex than necessary and useful. POM make models fall into the "Medaware zone", the zone with intermediate complexity where pay-off of a model regarding answering a research question is maximized. POM can be used for the selection of adequate submodels that are able to represent the formerly selected processes. For this, different submodel implementations of varying complexity are treated as alternative hypotheses for the represented processes. The submodel with the lowest complexity but ability to reproduce the multiple patterns simultaneously is selected. In a similar manner, POM can be applied to parameter fitting. Here, not submodels are selected but values of entire sets of parameters that are uncertain but essential. For this, a model is run with large numbers of parameter sets, sometimes more than a billion, and those sets are selected that make the model reproduce the multiple patterns (see also Chapter IV). Patterns are also critical for validation. With the POM strategy, new and independent patterns which are observed in the model output but have not been used or known during model development are taken to validate the model. POM is a widely accepted strategy in ABM, often applied intuitively but increasingly used systematically. A mini-review of model studies following the POM approach can be found in Grimm and Railsback [2012].

Model Communication One initiative to establish a standard protocol for documenting and communicating ABMs was started by Grimm et al. [2006]. With the *Overview-Design Concepts-Details* (ODD) protocol model descriptions should become more structured and complete to support understandability and reproducibility. It is a hierarchical model documentation protocol with increasing levels of detail starting with very general information from a meta-perspective to a very detailed description at the end of the documentation. By using the questions catalogue of Grimm et al. [2010], it can not only be used for the description of models but can also serve as a model development guide. Therefore, ODD also supports designing ABMs and can be embedded into the model structure development process in POM [Grimm and Railsback, 2012]. In the *Overview* section the modeller provides a short description of the model's purpose, entities, state variables and scales as well as an overview of the processes and their schedule. In the *Design Concepts* section the underlying concepts of the model are shortly described. Eleven categories are available: basic principles, emergence, adaptation, objectives, learning, prediction, sensing, interaction, stochasticity, collectives, and observation. In the *Details* section information about the initialization of the model and external input data as model drivers are given. Furthermore, this section contains a complete and detailed description of the submodels. For the protocol update Grimm et al. [2010] listed 54 publications where the ODD protocol was used. Furthermore, it has been successfully tested for the application to land-use management models [Polhill et al., 2008], used as a basis for the *Dahlem ABM documentation guidelines* [Wolf et al., 2013], extended to the *ODD+D protocol* for human decisions [Müller et al., 2013], and is recommended by the *openABM Consortium* [openABM Consortium, 2012]. The ODD protocol is currently on a promising way to become an established and accepted standard [Grimm et al., 2013].

Beside the ODD protocol, several other less successful attempts have been undertaken to develop and establish standards for ABM descriptions, like *AGENT UML* [Bauer et al.,

2001]. *AGENT UML* is an extension of the Unified Modelling Language for the specific purposes of ABMs and can be used not only for the documentation of models but also for the development based on the model-driven architecture (MDA) approach known from Computer Science. The same way was gone with the *Agent Modelling Language* (AML) proposed by Cervenka and Trencansky [2007], which is not only a protocol like ODD but a modelling language.

Modelling Guidelines A new guideline for planning, performing, and documenting simulation models along the modelling cycle was presented recently by Grimm et al. [2014] and Schmolke et al. [2010]. The *Transparent and Comprehensive Ecological Modelling* (TRACE) document structure serves as a best-practice guideline developed in the context of agent-based models for environmental decision support. The transparency requirements for such models are very high because decision making requires traceable information. Nevertheless, scientific requirements for replicability also need a high level of transparency. Therefore, such a best-practice guidance is required for all kinds of agent-based models. It ensures "that a model was thoughtfully designed, correctly implemented, thoroughly tested, well understood, and appropriately used for its intended purpose" [Grimm et al., 2014]. The TRACE structure comprises eight sections: problem formulation, model description, data evaluation, conceptual model evaluation, implementation verification, model output verification, model analysis, and model output corroboration.

The first section starts with the problem formulation, i.e., gives a definition of the questions the model should answer and the target audience. In the next section, the above mentioned ODD protocol can be placed for the model description. This is followed by the data evaluation section where the modeller should document which data have been used for the design and parameterization of the model. Furthermore, the reliability of the data used should be discussed and the parameters that have been calibrated should be named. In the next section, conceptual model evaluation, a list of the most important conceptual design decisions and a discussion of their choice to show why they have been selected, should be provided. Next, the section implementation verification refers to tests of the implementation to assess that the model implementation is doing what it is intended to do. The sixth section, model output verification, shows how well a model matches real system patterns by defining features and quantitative criteria, which refers to the aforementioned POM strategy. The modeller should also show here how much of this match results from calibration, how the fittings have been performed, and where extrapolations have been needed. In the model analysis section, the modeller shows that model mechanisms have been understood by running and explaining simulation experiments. Moreover, the importance, uncertainty, and functioning of parameters and submodels is assessed and explained by performing and documenting a sensitivity analysis in this section. In the last section, model output corroboration, the modeller compares model output with independent empirical data that have not been used for model design and development.

As the TRACE documentation format is new, it is currently in an early establishment phase and not an accepted standard. However, it addresses all parts needed for transparent and replicable modelling studies and could be used not only as a documentation schema but also as a modelling notebook guiding the modeller through the model development and application process.

Rand and Rust [2011] proposed another guideline for the rigorous use of ABM in the

context of marketing research. Their two main topics are verification and validation. Three major steps are summarized under rigorous verification. The first is documentation meaning documenting both the conceptual design and the implementation. The second is programmatic testing of the model code containing unit tests, code and debugging walkthroughs, and formal logical testing. The last step in verification is test cases and scenarios. This contains extreme value tests, checks of specific scenarios, and tests of known input-output relationships. Rigorous validation contains four major steps. First one is micro-face validation meaning checks on individual level that processes and properties correspond to the real world. The same is done on system level called macro-face validation. The third step, empirical input validation, refers to model input data corresponce to real world data. The last step is the empirical output validation where model outputs are compared to real world data and is therefore the key element of validation. Depending on the model purpose and data availability three different methods can be used for this task, namely stylized facts, real-world data, or cross-validation. As this guideline does not include a standard way of documentation and does not necessarily need to be cited in studies where it is used it is not possible to measure its acceptance and usage. However, as shown in Chapter IV it seems that it is not yet a widely accepted standard in Ecology and Social Sciences.

A subset of the former mentioned guidelines is addressed with the *Visual Debugging* method proposed by Grimm [2002a]. The idea is to use Graphical User Interfaces that integrate elements of classical debugging and graphical representations of the model's state variables for testing. Grimm [2002a] listed eleven features a Visual Model Debugger, i.e., a model implementation, must provide: a trace mode with a step-by-step model run, an automatic mode running the model for a longer time without user interaction, a batch mode without GUI for fast runs, input screens for changing all model and control parameters, input screens to select all model variants, input screens to manipulate low-level state variables, controls for random processes, graphical representations of state variables, file output of raw data, and file output of all simulation results that are represented graphically. Again, the acceptance of such a standard way of implementation and testing is hard to measure. However, all modern agent-based modelling platforms for implementing models provide all these features.

Process Representation Rigorous agent-based modelling can also be supported by standardization of the model representation of real-world processes. One example is the abstraction of competition processes in ecology among plants. Several approaches have been developed to describe the competition process of plants and have been used by others again [Berger et al., 2008]. A very prominent example is the *Zone-of-Influence* (ZOI) approach where a circular zone of influence around the centre of each plant is drawn [see, e.g., Wyszomirski, 1983, Wyszomirski et al., 1999, Weiner, 1982]. The size of the circle is derived from plant properties such as age or diameter. Overlapping circles imply competition between the plants. This competition process abstraction has been used many times in different models and became one standard, but not the only one, in plant modelling [see, e.g., review in Berger et al., 2002].

Berger et al. [2002] proposed the *Field-of-Neighbourhood* (FON) approach as a new standard for plant competition modelling. The FON approach is an extension of ZOI where the circle of influence has a decreasing competition value with increasing distance to the plant's position using a scalar field. Thus, it merges the widespread ZOI approach with the idea of

the more realistic *Ecological Field* (EF) approach, which has been too hard to parametrize and computer power demanding for being successfully applied to large areas [Berger et al., 2002]. Therefore, FON is a nice example of incremental submodel development, however, it was not that successful in becoming standard than expected by the authors, may be due to missing comprehensive documentation and reference implementations.

Consortium An indispensable element of standardization efforts is the establishment of an community-accepted consortium managing, developing and publishing standard definitions. A step in this direction was the foundation of the *openABM Consortium* [Janssen et al., 2008]. A very promising activity was the provision of a model archive with review process and citeable Digital Object Identifier (DOI) assignment to foster incremental model development. However, the consortium seems not yet having the necessary influence to declare standards that are either directly accepted by the modellers or indirectly by journal editors.

As ABMs are computational models, those standards on the conceptional level need to be applied on the technical level. As algorithms are needed for the definition as well as implementation of ABMs, the borders can become fluid between conceptional and technical solutions. Therefore, it is desirable to have a common language for implementing and documenting models. Furthermore, it is meaningful to reuse as much well-tested program code as possible. This will reduce the time for implementing models, the time for changing models and the risk of programming errors. This is realized best by using specified programming languages for ABM implementation providing pre-defined commands and included into high-level modelling platforms [Railsback, 2001]. Such a standard language further simplifies not only the implementation and communication but also increases the chance of reuse, testing and extension by other modellers.

There are several programming environments available tailored to the implementation of ABMs. Following Railsback et al. [2006], the most popular ones under an open-source license are Swarm [Minar et al., 1996], MASON [Luke et al., 2005], Repast [Collier et al., 2003], and NetLogo [Wilensky, 1999]. The ABM programming environments themselves, however, provide only limited support for advanced model analysis [Bakshy and Wilensky, 2007]. Therefore, analysis of ABMs is often weak and ad hoc [Schmolke et al., 2010, Janssen and Ostrom, 2006] although it is one of the most important tasks in the modelling cycle [Railsback and Grimm, 2012]. This thesis presents attempts to provide tools to professionalize model implementation, parametrization and analysis to make ABM more rigorous and support above mentioned conceptional standardization by linking, extending, and using existing software platforms. It is aimed that modellers need to invest less time in technical parts of modelling and can invest more time in analysing and interpreting models.

I.3. NetLogo

As Railsback and Grimm [2012] conclude, NetLogo stands apart among the various platforms for ABMs especially for beginners. It provides a simple programming language and a development and simulation environment to build and observe ABMs very fast. Nevertheless, it is also flexible enough to implement also fairly complex scientific models in NetLogo, see, for example, the BEEHAVE model with nearly 6000 lines of code [Becher, 2014]. Furthermore, it has a complete set of documentation and tutorial materials and an active

user community. The importance of NetLogo is underlined by the development of ReLogo, a NetLogo language implementation in the Repast environment [Ozik, 2013, Lytinen and Railsback, 2012].

NetLogo is developed and maintained since 1999 by the Center for Connected Learning and Computer-based Modeling at the Northwestern University, Illinois. Since 2011 it is released under an open-source license and programmed in Java and Scala. It provides many predefined methods (so-called primitives and reporters) for behavioural rules of the agents. Because it has a Logo-like syntax and standard agent types (turtles, patches, links), in combination with a built-in GUI, it is very easy to learn. The specialized language tailored to ABM development can be, in contrast to classical object-oriented languages, perfectly used not only for implementing but also for documenting parts of a model in, for example, the details section of the ODD-Protocol. Therefore, the work of this thesis focuses on developments for the NetLogo platform as the most promising language for becoming standard in ABMs.

I.4. Structure of This Thesis

This thesis consists of different research papers. A general introduction into the foundations, history and techniques of agent-based modelling is given in Chapter II based on Thiele et al. [2011]: *Agent- and Individual-Based Modelling with NetLogo: Introduction and New NetLogo Extensions*.

Chapter III describes techniques for linking NetLogo and R and hosts four different papers. The first paper, Thiele et al. [2012b]: *Agent-Based Modelling: Tools for Linking NetLogo and R*, gives an overview of the three tools R-Extension, Rserve-Extension, and RNetLogo and describes their purpose as well as similarities and differences. The second paper Thiele and Grimm [2010]: *NetLogo Meets R: Linking Agent-Based Models With a Toolbox for Their Analysis* describes the R-Extension in more detail and provides some usage examples from ecology. The third paper Thiele et al. [2012a]: *RNetLogo: An R Package for Running and Exploring Individual-Based Models Implemented in NetLogo* is structurally similar to the former one but describes the RNetLogo package in the context of ecology. The last paper in this chapter, Thiele [2014]: *R Marries NetLogo: Introduction to the RNetLogo Package*, provides an in-depth presentation of the functioning and possibilities of the RNetLogo package in an application neutral context.

Chapter IV provides a cookbook for parameter fitting and sensitivity analysis published as Thiele et al. [2014]: *Facilitating Parameter Estimation and Sensitivity Analysis of Agent-Based Models: A Cookbook Using NetLogo and R*. This paper aims to make agent-based modellers aware of existing methods and tools for parameter estimation and sensitivity analysis and to provide accessible tools for using these methods based on NetLogo and R using the RNetLogo package. The long-term target is the establishment of an advanced culture of relating agent-based models to data and patterns observed in real systems and to foster rigorous and structured analysis of agent-based models.

Chapter V wraps back to the beginning of Chapter II and leaves the technical level. The manuscript Thiele and Grimm [minor revisions]: *Modellers in Ecology: Replicate!*, highlights the importance of replication and robustness analysis in agent-based modelling and present this fundamental scientific practice as a key of increasing trustability in ABM. Furthermore, it shows that replication is essential for theory-building and advancement.

Chapter VI closes this thesis with a review of the impact of the developed tools and gives an outlook to further tools and concepts supporting rigorous agent-based modelling.

I.5. References

J. Arifovic. Genetic Algorithm Learning and the Cobweb Model. *Journal of Economic Dynamics and Control*, 18(1):3–28, 1994.

E. Bakshy and U. Wilensky. Turtle Histories and Alternate Universes: Exploratory Modeling with NetLogo and Mathematica. In M.J. North, c.M. Macal, and D.L. Sallach, editors, *Proceedings of the Agent 2007 Conference on Complex Interaction and Social Emergence*, pages 147–158. IL: Argonne National Laboratory and Northwestern University, 2007.

B. Bauer, J.P. Müller, and J. Odell. Agent UML: A Formalism for Specifying Multiagent Software Systems. In Paolo Ciancarini and MichaelJ. Wooldridge, editors, *Agent-Oriented Software Engineering*, volume 1957 of *Lecture Notes in Computer Science*, pages 91–103. Springer Berlin Heidelberg, 2001. ISBN 978-3-540-41594-7. doi: 10.1007/3-540-44564-1_6. URL http://dx.doi.org/10.1007/3-540-44564-1_6.

M. Becher. BEEHAVE - The Model, 2014. URL http://beehave-model.net/download/. (last accessed 2014/06/03).

U. Berger, H. Hildenbrandt, and V. Grimm. Towards a Standard for the Individual-Based Modeling of Plant Populations: Self-Thinning and the Field-Of-Neighborhood Approach. *Natural Resource Modeling*, 15(1):39–54, 2002.

U. Berger, C. Piou, K. Schiffers, and V. Grimm. Competition Among Plants: Concepts, Individual-Based Modelling Approaches, and a Proposal for a Future Research Strategy. *Perspectives in Plant Ecology, Evolution and Systematics*, 9:121–135, 2008.

F.C. Billari and A. Prskawetz. *Agent-Based Computational Demography: Using Simulation to Improve Our Understanding of Demographic Behaviour*. Contributions to Economics. Physica, 2003.

C. Carpenter and L. Sattenspiel. The Design and Use of an Agent-Basedased Model to Simulate the 1918 Influenza Epidemic at Norway House, Manitoba. *American Journal of Human Biology*, 21(3):290–300, 2009.

R. Cervenka and I. Trencansky. *The Agent Modeling Language - AML*. Birkhäuser, 2007.

N. Collier, T. Howe, and M.J. North. Onward and Upward: The Transition to Repast 2.0. In *First Annual North American Association for Computational Social and Organizational Science Conference*, Pittsburgh, PA USA, 2003. North American Association for Computational Social and Organizational Science.

R. Conte. From Simulation to Theory (and Backward). In F. Squazzoni, editor, *Epistemological Aspects of Computer Simulation in the Social Sciences, Second International Workshop, EPOS 2006, Brescia, Italy, October 5-6*, volume 5466 of *Lecture Notes in Computer Science*, pages 29–47. Springer, 2006.

R. Conte, N. Gilbert, and J.S. Sichman. MAS and Social Simulation: A Suitable Commitment. In J.S. Sichman, R. Conte, and N. Gilbert, editors, *Multi-Agent Systems and Agent-Based*

Simulation, First International Workshop, MABS '98, Paris, France, July 4-6, volume 1534 of *Lecture Notes in Computer Science*, pages 1–9. Springer, 1998.

D.L. DeAngelis and W.M. Mooij. Individual-Based Modeling of Ecological and Evolutionary Processes. *Annual Review of Ecology, Evolution, and Systematics*, 36:147–168, 2005.

J.M. Epstein and R. Axtell. *Growing Artificial Societies: Social Science from the Bottom Up*. The Brookings Institution, Washington, DC, 1996.

J. Ferrer, C. Prats, and D. López. Individual-Based Modelling: An Essential Tool for Microbiology. *Journal of Biological Physics*, 34(1-2):19–37, 2008.

J.M. Galán, L.R. Izquierdo, S.S. Izquierdo, J.I. Santos, R. Del Olmo, A. López-Paredes, and B. Edmonds. Errors and Artefacts in Agent-Based Modelling. *Journal of Artificial Societies and Social Simulation*, 12(1), 2009. URL http://jasss.soc.surrey.ac.uk/12/1/1.html. (last accessed 2014/01/06).

N. Gilbert. Simulation: A New Way of Doing Social Science. *American Behavioral Scientist*, 40:1485–1487, 1999.

N. Gilbert. *Agent-Based Models*. Quantitative Applications in the Social Sciences. Sage, Los Angeles, CA, 2007.

S. Green, L. Hurst, B. Nangle, P. Cunningham, F. Somers, and R. Evans. Software Agents: A Review. Technical Report TCD-CS-1997-06, Trinity College Dublin, Department of Computer Science, 1997.

A.F. Griffin and C. Stanish. An Agent-Based Model of Prehistoric Settlement Patterns and Political Consolidation in the Lake Titicaca Basin of Peru and Bolivia. *Structure and Dynamics*, 2:1–46, 2007.

V. Grimm. Ten Years of Individual-Based Modelling in Ecology: What Have We Learned and What Could We Learn in the Future? *Ecological Modelling*, 115:129–148, 1999.

V. Grimm. Visual Debugging: A Way of Analyzing, Understanding, and Communicating Bottom-Up Simulation Models in Ecology. *Natural Resource Modeling*, 15:23–38, 2002a.

V. Grimm. Bottom-Up Simulation Modelling in Ecology: Strategies and Examples, 2002b. Habilitation Thesis, Univ. Potsdam.

V. Grimm. Individual-Based Models. In S.E. Jørgensen, editor, *Ecological Models*, pages 1959–1968. Elsevier, Oxford, 2008.

V. Grimm and S.F. Railsback. *Individual-Based Modeling and Ecology*. Princeton University Press, Princeton, N.J., 2005.

V. Grimm and S.F. Railsback. Pattern-Oriented Modelling: A 'Multi-scope' for Predictive Systems Ecology. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 367 (1586):298–310, 2012.

V. Grimm, E. Revilla, U. Berger, F. Jeltsch, W.M. Mooij, S.F. Railsback, H.-H. Thulke, J. Weiner, T. Wiegand, and D.L. DeAngelis. Pattern-Oriented Modeling of Agent-Based Complex Systems: Lessons from Ecology. *Science*, 310(5750):987–991, 2005.

V. Grimm, U. Berger, F. Bastiansen, S. Eliassen, V. Ginot, J. Giske, J. Goss-Custard, T. Grand, S.K. Heinz, G. Huse, A. Huth, J.U. Jepsen C. Jørgensen, W.M. Mooij, B. Müller, G. Pe'er,

C. Piou, S.F. Railsback, A.M. Robbins, M.M. Robbins, E. Rossmanith, N. Rüger, E. Strand, S. Soissi, R.A. Stillman, R. Vabø, U. Visser, and D.L. DeAngelis. A Standard Protocol for Describing Individual-Based and Agent-Based Models. *Ecological Modelling*, 198:115–126, 2006.

V. Grimm, U. Berger, D.L. DeAngelis, J.G. Polhill, J. Giske, and S.F. Railsback. The ODD Protocol: A Review and First Update. *Ecological Modelling*, 221:2760–2768, 2010.

V. Grimm, G.P. Polhill, and J. Touza. Documenting Social Simulation Models: The ODD Protocol as a Standard. In B. Edmonds and R. Meyer, editors, *Simulating Social Complexity – A Handbook*, pages 117–133. Springer, 2013.

V. Grimm, J. Augusiak, A. Focks, B.M. Frank, F. Gabsi, A.S.A. Johnston, C. Liu, B.T. Martin, M. Meli, V. Radchuk, P. Thorbek, and S.F. Railsback. Towards Better Modelling and Decision Support: Documenting Model Development, Testing, and Analysis Using TRACE. *Ecological Modelling*, 280:129–139, 2014.

S. Heckbert, T. Baynes, and A. Reeson. Agent-Based Modeling in Ecological Economics. *Annals of the New York Academy of Sciences*, 1185:39–53, 2010.

C. Hewitt. *Viewing Control Structures as Patterns of Passing Messages*. A.I.Memo 410. MIT Press, 1976.

D. Imboden and S. Koch. *Systemanalyse. Eine Einführung in die mathematische Modellierung natürlicher Systeme*. Springer, Berlin, Heidelberg, 2003.

M.A. Janssen and E. Ostrom. Empirically Based, Agent-Based Models. *Ecology and Society*, 11:37, 2006. URL http://www.ecologyandsociety.org/vol11/iss2/art37/. (last accessed 2014/05/21).

M.A. Janssen, L.N. Alessa, M. Barton, S. Bergin, and A. Lee. Towards a Community Framework for Agent-Based Modelling. *Journal of Artificial Societies and Social Simulation*, 11 (2) 6, 2008. URL http://jasss.soc.surrey.ac.uk/11/2/6.html. (last accessed 2014/01/06).

N.R. Jennings. On Agent-Based Software Engineering. *Artificial Intelligence*, 117:277–296, 2000.

B. LeBaron. Agent-Based Computational Finance: Suggested Readings and Early Research. *Journal of Economic Dynamics and Control*, 24(5-7):679–702, 2000.

R. Leombruni and M. Richiardi. *Industry and Labor Dynamics: The Agent-based Computational Economics Approach: Proceedings of the Wild@ace2003 Workshop, Torino, Italy, 3-4 October 2003*. World Scientific, 2004.

A.J. Lotka. *Elements of Physical Biology*. Williams and Wilkins, Baltimore, MD, 1925.

M. Luck, P. McBurney, and C. Preist. *Agent Technology: Enabling Next Generation Computing (A Roadmap for Agent Based Computing)*. AgentLink, Southampton: University of Southampton, 2003.

S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, and G. Balan. MASON: A Multi-Agent Simulation Environment. *Simulation*, 82:517–527, 2005.

M. Lutz. *Operations Research Verfahren - verstehen und anwenden*. Fortis Verlag FH, Wien, Mainz, 1998.

S.L. Lytinen and S.F. Railsback. The Evolution of Agent-Based Simulation Platforms: A Review of NetLogo 5.0 and Relogo. In *Proceedings of the Fourth International Symposium on Agent-Based Modeling and Simulation*, 2012. URL http://condor.depaul.edu/slytinen/abm/Lytinen-Railsback-EMCSR_2012-02-17.pdf. (last accessed 2014/05/19).

M.W. Macy and R. Willer. From Factors to Actors: Computational Sociology and Agent-Based Modeling. *Annual Review of Sociology*, 28:143–166, 2002.

N. Malleson, A. Heppenstall, and L. See. Crime Reduction Through Simulation: An Agent-Based Model of Burglary. *Computers, Environment and Urban Systems*, 34(3):236–250, 2010.

R.B. Matthews, N.G. Gilbert, A. Roach, J.G. Polhill, and N.M. Gotts. Agent-Based Land-Use Models: A Review of Applications. *Landscape Ecology*, 22(10):1447–1459, 2007.

K.M. Meyer, M. Vos, W.M. Mooij, W.H.G. Hol, A.J. Termorshuizen, and W.H. van der Putten. Testing the Paradox of Enrichment along a Land Use Gradient in a Multitrophic Aboveground and Belowground Community. *PLoS ONE*, 7(11):e49034, 2012.

N. Minar, R. Burkhart, C. Langton, and M. Askenazi. The Swarm Simulation System: A Toolkit for Building Multi-Agent Simulations. Working Paper 96-06-042, Santa Fe Institute, Santa Fe, 1996.

B Müller, F Bohn, G. Dreßler, J. Groeneveld, C. Klassert, R. Martin, M. Schlüter, J. Schulze, H. Weise, and N. Schwarz. Describing Human Decisions in Agent-Based Models – ODD + D, an Extension of the ODD Protocol. *Environmental Modelling & Software*, 48:37–48, 2013.

T. Müller and H. Müller. *Modelling in Natural Sciences. Design, Validation and Case Studies*. Springer, Berlin, Heidelberg, 2003.

J.M. Moonen. *Multi-Agent Systems for Transportation Planning and Coordination*. ERIM Ph.D. Series Research in Management. Erasmus Research Institute of Management (ERIM), Erasmus University Rotterdam, 2009.

M.J. North and C.M. Macal. *Managing Business Complexity: Discovering Strategic Solutions with Agent-Based Modeling and Simulation*. Oxford University Press, Oxford etc., 2007.

M.J. North, C.M. Macal, J.S. Aubin, P. Thimmapuram, M.J. Bragen, J. Hahn, J. Karr, N. Brigham, M.E. Lacy, and D. Hampton. Multiscale Agent-Based Consumer Market Modeling. *Complexity*, 15(5):37–47, 2010.

E. Oliveira. Applications of Intelligent Agent-Based Systems. In *Proceedings of SBAI - Simpósium Brasileiro de Automação Inteligente. 1999: São Paulo*, pages 51–58, 1999.

openABM Consortium. Standards, 2012. URL http://www.openabm.org/page/standards. (last accessed 2014/06/30).

J. Ozik. ReLogo Getting Started Guide, July 2013. URL http://repast.sourceforge.net/docs/ReLogoGettingStarted.pdf. (last accessed 2014/05/19).

S.E. Page. Aggregation in Agent-Based Models of Economics. *The Knowledge Engineering Review*, 27:151–162, 2012.

J.G. Polhill, D. Parker, D. Brown, and V. Grimm. Using the ODD Protocol for Describing

Three Agent-Based Social Simulation Models of Land-Use Change. *Journal of Artificial Societies and Social Simulation*, 11, 2008. URL http://jasss.soc.surrey.ac.uk/11/2/3.html. (last accessed 2014/05/19).

H. Pretzsch. *Modellierung des Waldwachstums*. Parey, Berlin, Wien, 2001.

S.F. Railsback. Concepts From Complex Adaptive Systems as a Framework For Individual-Based Modelling. *Ecological Modelling*, 13:47–62, 2001.

S.F. Railsback and V. Grimm. *Agent-Based and Individual-Based Modeling: A Practical Introduction*. Princeton University Press, 2012.

S.F. Railsback, S.L. Lytinen, and S.K. Jackson. Agent-Based Simulation Platforms: Review and Development Recommendations. *Simulation*, 82:609–623, 2006.

W. Rand and R.T. Rust. Agent-Based Modeling in Marketing: Guidelines for Rigor. *International Journal of Research in Marketing*, 28:181–193, 2011.

T.C. Schelling. Models of Segregation. *The American Economic Review*, 59(2):488–493, 1969.

A. Schmolke, P. Thorbek, D.L. DeAngelis, and V. Grimm. Ecological Modelling Supporting Environmental Decision Making: A Strategy for the Future. *Trends in Ecology and Evolution*, 25:479–486, 2010.

W. Shen, Q. Hao, H.J. Yoon, and D.H. Norrie. Applications of Agent-Based Systems in Intelligent Manufacturing: An Updated Review. *Advanced Engineering Informatics*, 20(4): 415–431, 2006.

K. Soetaert and P.M.J. Herman. *A Practical Guide to Ecological Modelling: Using R as a Simulation Platform*. Springer, Dordrecht, 2009.

F. Squazzoni. The Impact of Agent-Based Models in the Social Sciences After 15 Years of Incursions. *History of Economic Ideas*, XVIII/2010/2:197–233, 2010.

F. Squazzoni. *Agent-Based Computational Sociology*. John Wiley & Sons, 2012.

H. Stachowiak. *Allgemeine Systemtheorie*. Springer, Wien a.o., 1973.

K.P. Sycara. Multiagent Systems. *AI Magazine*, 19(2):79–92, 1998.

L. Tesfatsion. Agent-Based Computational Economics: A Constructive Approach to Economic Theory. In L. Tesfatsion and K.L. Judd, editors, *Handbook of Computational Economics*, volume 2, chapter 16, pages 831–880. Elsevier, 2006.

J.C. Thiele. R Marries NetLogo: Introduction to the RNetLogo Package. *Journal of Statistical Software*, 58(2):1–41, 2014.

J.C. Thiele and V. Grimm. NetLogo Meets R: Linking Agent-Based Models with a Toolbox for Their Analysis. *Environmental Modelling & Software*, 25(8):972–974, 2010.

J.C. Thiele and V. Grimm. Modellers in Ecology: Replicate! *Oikos*, minor revisions.

J.C. Thiele, W. Kurth, and V. Grimm. Agent- and Individual-Based Modeling with NetLogo: Introduction and New NetLogo Extensions. In K. Römisch, A. Nothdurft, and U. Wunn, editors, *22. Tagung der Sektion Forstliche Biometrie und Informatik des Deutschen Verbandes Forstlicher Forschungsanstalten und der Arbeitsgemeinschaft Ökologie und Umwelt der*

Internationalen Biometrischen Gesellschaft - Deutsche Region, 20-21th September 2010 in Göttingen (Germany), Die Grüne Reihe, pages 68–101, 2011.

J.C. Thiele, W. Kurth, and V. Grimm. RNetLogo: An R Package for Running and Exploring Individual-based Models Implemented in NetLogo. *Methods in Ecology and Evolution*, 3: 480–483, 2012a.

J.C. Thiele, W. Kurth, and V. Grimm. Agent-Based Modelling: Tools for Linking NetLogo and R. *Journal of Artificial Societies and Social Simulation*, 15 (3) 8, 2012b. URL http://jasss.soc.surrey.ac.uk/15/3/8.html. (last accessed 2014/01/06).

J.C. Thiele, W. Kurth, and V. Grimm. Facilitating Parameter Estimation and Sensitivity Analysis of Agent-Based Models: A Cookbook Using NetLogo and R. *Journal of Artificial Societies and Social Simulation*, 17 (3) 11, 2014. URL http://jasss.soc.surrey.ac.uk/17/3/11.html. (last accessed 2014/07/02).

A. Töllner, T. Jungmann, M. Bücker, and T. Brutscheck. Modelle und Modellierung. In G. Bandow and H.H. Holzmüller, editors, *Das ist gar kein Modell! Unterschiedliche Modelle und Modellierungen in Betriebswirtschaftslehre und Ingenieurwissenschaften*, pages 3–22. Gabler, Wiesbaden, 2010.

V. Volterra. Variations and Fluctuations of the Number of Individuals in Animal Species Living Together. *Animal Ecology*, pages 409–448, 1931. (Reprint).

J. Weiner. A Neighborhood Model of Annual-Plant Interference. *Ecology*, 63:1237–1241, 1982.

M. Weisberg. *Simulation and Similarity*. Oxford University Press, New York, 2013.

G. Weiss. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, Cambridge, MA, 1999.

U. Wilensky. NetLogo. Center for Connected Learning and Computer-Based Modeling, 1999. URL http://ccl.northwestern.edu/netlogo. (last accessed 2014/01/06).

S. Wolf, J.-P. Bouchaud, F. Cecconi, S. Cincotti, H. Dawid, H. Gintis, S. Hoog, C.C. Jaeger, D. Kovalevsky, A. Mandel, and L. Paroussos. Describing Economic Agent-Based Models - Dahlem ABM Documentation Guidelines. *Complexity Economics*, 2(1):63–74, 2013.

M.J. Wooldridge. *An Introduction to Multiagent Systems*. John Wiley & Sons, New York, NY, 2005.

T. Wyszomirski. A Simulation Model of the Growth of Competing Individuals of a Plant Population. *Ekologia Polska*, 31:73–92, 1983.

T. Wyszomirski, I. Wyszomirska, and I. Jarzyna. Simple Mechanisms of Size distribution Dynamics in Crowded and Uncrowded Virtual Monocultures. *Ecological Modelling*, 115: 253–273, 1999.

CHAPTER II

Agent- and Individual-Based Modelling with NetLogo: Introduction and New NetLogo Extensions

This manuscript is published as: JC Thiele, W Kurth, and V Grimm [2011]. Agent- and Individual-Based Modelling with NetLogo: Introduction and New NetLogo Extensions. In: K Römisch, A Nothdurft, and U Wunn (eds.): Die Grüne Reihe. 22. Tagung der Sektion Forstliche Biometrie und Informatik des Deutschen Verbandes Forstlicher Forschungsanstalten und der Arbeitsgemeinschaft Ökologie und Umwelt der Internationalen Biometrischen Gesellschaft - Deutsche Region, 20-21th September 2010 in Göttingen (Germany), pages 68-101, ISSN 1860-4064.

Authorship

- Winfried Kurth wrote the paragraph about Functional-Structural Plant Modelling and supported the writing of the rest of the manuscript.
- Volker Grimm supported the writing of the manuscript.

II.1. Abstract

Agent-based models (ABM) or individual-based models (IBM), as they are called in ecology and biology, are a widely used modelling approach when local interactions on the micro level are essential for the description of patterns on the macro level. This chapter is divided into four sections. In the first section, the history and definitions of ABMs in various research disciplines, namely computer science, social science, economics and ecology, are reviewed. This section closes with a discussion of similarities and differences in the different research fields and a discussion of current challenges in agent-based modelling. One of these difficulties is the lack of accepted standards for communication and programming. The second section refers to this point by a presentation of some widely used ABM libraries, namely Swarm, Mason, Repast and NetLogo and is followed by a more detailed description of NetLogo as a potential standard tool in ABM communication. In the last section extensions to NetLogo, developed by the authors of this chapter, are presented. This includes the MultiView-Extension, the R-Extension and the NetLogo Plug-In for the Pygments syntax highlighter. The chapter closes with an outlook to further tools for NetLogo which aim at making NetLogo even more relevant as a standard tool in ABM.

II.2. Agent-/Individual-Based Modelling

II.2.1. Introduction

Building and using models is part of everybody's life. For example, if we wait for the train, we decide where to stay so that we can get into the train fast and get a good seat. We include our former experience about where it is best to find a seat: in the middle, the front or the back of the train. Furthermore we check the other passengers waiting on the track. People carrying heavy luggage are slower than others and so on. But it will be impossible to include all information and details. Therefore, simplification and aggregation of the real system (abstraction) is the key in modelling. Starfield et al. [1990] called this "purposeful representation", which means that the real system is represented only by those elements which are important for answering the question the model is designed for. Thus, the problem to solve should stand at the beginning of the model building process. The second step is the definition of the system and its boundaries which leads to a verbal, conceptual model [Bossel, 1994] followed by the selection of the formal model structure with scales, state variables, processes and parameters. If such a model is too complex to calculate its outcome analytically, it has to be implemented as a computer model, called computer simulation model, before the model can be analysed and developed [see Modelling Cycle in Grimm and Railsback, 2005].

There are different reasons to perform a computer simulation. They are useful to test theories and to try understanding system behaviour, but they are also used to make forecasts and to design experiments which could not be done in real life (e.g., for ethical or technical reasons).

With increasing computer power and growing criticism of conventional modelling methods, a new approach to simulation models, agent-based modelling (ABM), established in different kinds of scientific fields. Since the development of ABM took place more or less independently in different scientific fields, they differ in definitions, history and context. Therefore, we will give a short overview over the most important sectors: computer science,

sociology, economics and ecology. But it should be mentioned that ABM is now increasingly applied interdisciplinary, which makes the following separate descriptions slightly obsolete.

II.2.2. ABM in Computer Science

The basis for agent-based modelling in many other fields has been the development of so-called multi-agent systems (MAS) in computer science as a part of the distributed artificial intelligence (DAI) research area [Green et al., 1997, Sycara, 1998]. The development of MAS started around the late 1970s [e.g., Hewitt, 1976] but their wider use began only in the 1990s, mainly to distribute large computational problems over multiple computers [Luck et al., 2003, Wooldridge, 2005, Weiss, 1999]. Later influenced and adapted by different disciplines, it is not surprising that many different definitions of the term MAS and what an agent is exist [e.g., Jennings et al., 1998, Jennings, 2000]. In a very general case a MAS is defined as a system which is composed of multiple (semi-) autonomous components to reach a common goal [Jennings et al., 1998]. Many modular software systems could be summarized under this weak definition. A narrower definition is given by Wooldridge and Jennings [1995]. They added the requirement of using intelligent agents to their definition of MAS. Their (computational) agents have been characterized as autonomous, social, reactive and pro-active entities. In this concept, an agent takes input from the environment and produces actions as outputs that affect the environment but with incomplete information at the agent level and without a global control mechanism [Wooldridge, 2005]. Some classical examples for the use of MAS with intelligent agents can be found in computer games, computer networks, robotics for manufacturing, or traffic-control systems [for examples, see Oliveira, 1999, Luck et al., 2003, Shen et al., 2006, Moonen, 2009]. But the more autonomous the local agents became, the more important the question about coordination and cooperation within the system was. These questions had strong similarities to the research questions in sociology [Conte et al., 1998].

II.2.3. ABM in Social Sciences

Since some researches from the DAI area adapted ideas from social sciences, sociologists became aware of the MAS techniques in the 1990s, encouraged by the advent of personal computers and the development of object-oriented programming languages [Epstein and Axtell, 1996, Gilbert, 1999, Squazzoni, 2010]. As a part of computational sociology, the approach is called agent-based social simulation (ABSS) or just agent-based modelling (ABM) [Conte et al., 1998, Gilbert, 2007, Macy and Willer, 2002]. In such ABMs, agents typically represent social actors which can be individual persons, organizations (e.g., companies), or countries [Gilbert, 2007] which are simulated as an artificial society [Epstein and Axtell, 1996]. Gilbert [2007] defined an ABM as a "computational method that enables a researcher to create, analyse and experiment with models composed of agents that interact within an environment", which is very similar to the definition given by [Wooldridge, 2005] for MAS.

The ABM approach differs substantially from former microsimulation and system dynamics approaches in social computations. The latter typically consists of sets of differential equations which describe the change of stock variables and is known as "top-down" approach. In this way, individual social actors are averaged and their behaviour is aggregated. A famous example for that kind of models is the world model of Forrester [1971], which was used by Meadows [1974] for predicting the ecological limitation of economic and de-

mographic growth. Such models have been mainly used for quantitative forecasts, are often highly sensitive to the (frequently unknown) model parameters, and ignore the heterogeneity of individuals. Therefore, they are rarely appropriate for social sciences, which are mainly interested in understanding and explanation [Gilbert and Troitzsch, 1999].

The classical microsimulation approach incorporates individual heterogeneity but not interactions between individuals [Macy and Willer, 2002]. This approach is based on transition probabilities for each individual, which are typically derived from empirical data, and delivers quantitative forecasts [Gilbert and Troitzsch, 1999]. Such data-driven microsimulations have been applied mainly to forecast the effects of policy changes and are highly related to macroeconomic analysis but also well established in non-social sciences like traffic/transportation analysis and economic research. Example applications in social sciences include tax-benefit analysis, analysis of demographic developments, social security and labour analysis. Reviews can be found in Spielauer [2007], Zaidi and Rake [2001] or Algers et al. [1997].

With the introduction of ABM, interactions between complex, adaptive individuals [Macy and Willer, 2002] and the opportunity to take spatial effects into account [Gilbert, 2007] were added to social computer simulations. The main idea is to analyse the way in which macro-level system properties emerge from interaction of the agents on microlevel to describe social systems [Davidsson, 2002]. Therefore, it is mainly used for theory testing and development [Macy and Willer, 2002, Conte, 2006]. Furthermore, the development of an ABM constrains the social scientist to clearly define his assumptions of local behaviour which helps to clearly communicate basic ideas [Gilbert, 2004].

In general, ABMs are a good choice for studying social processes that do not include central coordination [Macy and Willer, 2002]. ABMs in social science have been applied to the simulation of differentiation, diffusion and emergence of social order in social systems [for examples, see listing in Macy and Willer, 2002 and references in Li et al., 2008 and Squazzoni, 2010 as well as to questions about demographic behaviour Billari and Prskawetz, 2003]. Most famous models in social sciences are Schelling's [1969, 1971] segregation model and the Sugarscape model of Epstein and Axtell [1996], which have often been cited and extended. However, as stated by Macy and Willer [2002] and recently by Hamill [2010], ABMs are still considered very sceptically by many sociologists and have been used rather rarely compared to other disciplines, although Squazzoni [2010] reported about a constant increase of ABM awareness in recent years. This could be the result of the increasing use of ABM techniques in interdisciplinary projects with socio-ecological and socio-economical contexts.

II.2.4. ABM in Economics

The neoclassical Walrasian general equilibrium (GE) model as well as the (New) Keynesian framework are still the most used fundamental paradigms in (Macro-) economics [Tesfatsion, 2006, Oeffner, 2008]. They simplify the economic system by representing (averaged) agents with perfect rationality, information and foresight. Furthermore, interactions between agents (e.g., firms and households) take place only indirect, in case of Walrasian GE model, for example, by pricing using the concept of "Walrasian auctioneer" [Oeffner, 2008].

Such "top-down" approaches were developed despite Smith [1776] and others stated that economic processes are the result of parallel, local interactions between large numbers of individuals. Therefore, Gun [2004] criticized macroeconomic approaches as he wrote: "How

can any reasonable person admit, that, for example, the evolution of the US aggregates' results from decisions made by a single individual who owns all factories and who decides how much to produce, how much labour to use, how production will be distributed between consumption and investment, and so on?". Consequently, to overcome these weaknesses the ABM approach was adapted to economic modelling by using heterogeneous, interacting (and learning) agents [Farmer and Foley, 2009].

The ABM approach in economics is called agent-based computational economics (ACE) and is related to the field of cognitive and evolutionary economics. Arthur [2006] described this as follows: "Standard neoclassical economics asks what agent's actions, strategies, or expectations are in equilibrium with (consistent with) the outcome or pattern these behaviours aggregatively create. Agent-based computational economics enables us to ask a wider question: how agent's actions, strategies or expectations might react to - might endogenously change with - the pattern they create. In other words, it enables us to examine how the economy behaves out of equilibrium, when it is not at a steady state." The beginnings of ACE are going back to the work of the Santa Fe Institute which started in the late 1980s with its work on describing and analysing the economy as an evolving and complex system encouraged by the development of personal computers and object- and agent-oriented programming languages [Richiardi, 2007].

Tesfatsion [2006] has defined ACE as "the computational study of economic processes modelled as dynamic systems of interacting agents" and an agent as "bundled data and behaviour methods" which could represent individuals, social grouping and institutions, biological or physical entities. This definition is nearly identical to the definition of social ABMs given by Gilbert [2007], which is not surprising since there is no strict boundary between both disciplines. Even the aims of ACE have similarities to those of social ABMs. They can be divided into four main categories: empirical understanding, normative understanding, qualitative insight as well as theory generation and methodological advancement [for details, see Tesfatsion, 2006]. Therefore, ACE can be used complementary to mathematical theorizing as well as a substitute for it [Phan, 2004, Axtell, 2000]. ACE have been used, for example, for the reproduction of classical cobweb theorem [e.g., Arifovic, 1994], for modelling financial/stock markets [for a review, see LeBaron, 2000] as well as for simulating industry and labour dynamics [e.g., Leombruni and Richiardi, 2004]. Nevertheless, many economists still prefer the conventional mathematical models by tradition [Buchanan, 2009].

II.2.5. ABM in Ecology

In contrast, in the field of ecology the agent-based approach has a slightly longer tradition and is well established nowadays [Grimm and Railsback, 2005], although there are critics as well [Caswell, 2001]. The development of so called individual-based models (IBM) is less closely related to the developments of MAS as it is the case in social sciences, because ecologists early became aware of the restrictions in classical population models (differential equation models) and looked for alternatives. The most obvious approach for the simulative reproduction of observed population-level effects was to take the heterogeneous properties of the individuals into account.

Since sufficient computational power became available in the early 1970s, first ecologists started to develop such IBMs [e.g., Myers, 1976, DeAngelis et al., 1980]. One of these pioneer works is the forest succession model JABOWA of Botkin et al. [1972]. In his review,

however, Grimm [1999] distinguishes this work, which often was pragmatically motivated, from the later paradigmatic motivations of the IBM approach, in which IBMs are not used because of the limitation of more aggregate models but because of the motivation to overcome the limitations of the population-level paradigm of theoretical ecology. Grimm [1999] refers to Kaiser [1974] and Lomnicki [1978, 1988] as the pioneers of the paradigmatic motivation, but noticed that still only a minority of ecological IBMs directly addressed theoretical issues. The main driver of the success of IBMs in ecology was and still is the pragmatic motivation.

Milestones in the establishment of the IBM approach in ecology were the review of Huston et al. [1988], followed by the substantial conference proceedings of DeAngelis and Gross [1992] and the influential articles of Hogeweg and Hesper [1990], Judson [1994], Uchmanski and Grimm [1996], Grimm [1999], Lomnicki [1999], DeAngelis and Mooij [2005] as well as the monograph of Grimm and Railsback [2005]. As Grimm and Railsback [2005] noticed, up to mid-1990s, a clear definition of IBMs was still missing and the use of the term became fuzzy.

Therefore, Uchmanski and Grimm [1996] defined criteria for the classification of ecological models which allow separating IBMs from classical state-variable approaches [for details see Uchmanski and Grimm, 1996 or Grimm and Railsback, 2005]: IBMs include (potentially) heterogeneous, discrete entities which represent real individuals. These individuals can be adaptive in behaviour and life history. Moreover, IBMs include feedbacks between dynamic (food) resources and individuals. In contrast, age- or state-structured population models and models defining resources via constant carrying capacities do not fulfil these conditions. Grimm and Railsback [2005] defined such models in the middle between unstructured population models and IBMs.

Over the last three to four decades hundreds of IBMs were developed in ecology [DeAngelis and Mooij, 2005]. For reviews see, for example, Grimm [1999], Grimm and Railsback [2005], Hogeweg and Hesper [1990], DeAngelis et al. [1990], DeAngelis and Gross [1992], DeAngelis et al. [1994] and DeAngelis and Mooij [2005]. See also Bunsing and Mailly [2004] and Liu and Ashton [1995] on IBMs of forest dynamics. In case of modelling forests, these IBMs are sometimes called individual-tree models [e.g., by Pretzsch, 2009 and Coates et al., 2003].

A family of models in biology which has some common features with IBMs is that of functional-structural plant models (FSPM) [Godin and H.Sinoquet, 2005, Vos et al., 2007]. In an FSPM, a plant individual is decomposed into morphological units like internodes, leaves, root segments etc., which all have their own functions and state variables and which interact with each other. The three-dimensional architecture of a plant and its functioning are both represented in the same model. Since the morphological units are modelled as entities with certain autonomy, this approach can be considered as an extension of IBMs to a spatial scale level below that of the individual. However, FSPMs can also be used to model interactions between several individuals and thus the behaviour of whole plant stands [Hemmerling et al., 2008, Cournède et al., 2010].

II.2.6. Synopsis

As we have seen, there are strong similarities in the definitions of ABMs in the different fields but there are also some differences. The narrowest definition gave Uchmanski and Grimm [1996] in the field of ecology while no strict line can be drawn between ACE and ABSS. In the last two fields, however, the influence of MAS from computer sciences was

much stronger than in ecology. Whilst ABMs in social science and economy are often more paradigmatic and abstract, they are usually pragmatic and specific in ecology. Nevertheless, they have in common that they present an alternative to aggregated state-variable (equilibrium) models; they do not impose system-level results but are dedicated to answer the question how system-level patterns emerge from adaptive behaviour and local interactions.

Although we here focused on the research fields computer science, sociology, economy and ecology, there are many other disciplines in which ABMs are increasingly used, often within an interdisciplinary context. Examples are ecological economics [e.g., Heckbert et al., 2010, Drechsler et al., 2007], marketing/socio-psychology [e.g., North et al., 2010, Ben Said et al., 2002], archaeology/anthropology [e.g., Griffin and Stanish, 2007, Premo, 2007, Premo and Kuhn, 2010], microbiology [e.g., Ferrer et al., 2008, Ginovart et al., 2002], biomedicine/epidemiology [e.g., An, 2009, Carpenter and Sattenspiel, 2009], criminology [strongly related to ABSS, e.g., Malleson et al., 2010], land-use management [e.g., Polhill, 2009, Matthews et al., 2007] and forest management planning [e.g., Simon and Etienne, 2009].

The wide use of the ABM approach in all these disciplines has in common that it was strongly related to the increasing availability and power of personal computers and the development of object- and agent-oriented programming languages.

II.2.7. Current Challenges in ABMs

As with every method there is no light without shadow. There are several challenges as well as stereotypes related to the ABM approach. Some of them are shortly discussed in the following.

ABMs are often seen as very data hungry [e.g., Reed et al., 2002]. It is true that data are needed when building ABMs, but it is wrong to believe that an ABM should not be built without complete information [Starfield, 1997]. As Grimm and Railsback [2005] and Squazzoni [2010] noticed, ABMs can be used to test hypothesis about unknown parts of a system and can help empiricists to identify which are the important things to measure. Furthermore, it can be easier to collect data at the level of the individual than to collect them at an (abstract) macro-level [Hogeweg and Hesper, 1990, Huston et al., 1988], and the model might react less sensibly on parameter variations at the level of the individual [Breckling, 2002]. The main problem with the measurements is that results of laboratory experiments with sometimes isolated individuals/humans could be invalid for natural conditions. In general, Grimm [1999] reported about a tendency towards an overuse of empirical knowledge in IBMs in the sense of putting all available data into a model. He advised to find the appropriate level of aggregation through the modelling process and not through availability of data or the aim to let the model (not the outcome) look more "realistic".

This leads directly to the next point, the difficulties in analysing, validating and understanding the outcomes of an ABM due to its complexity. ABMs should be kept as simple as possible otherwise they could contain unnecessary details to answer the research question [Macy and Willer, 2002, Hiebeler, 1994]. Nevertheless, even in simple ABMs outcomes can be hard to understand. Different guidelines about how to analyse ABMs are available [e.g., Richiardi et al., 2006, Gilbert, 2007, Windrum et al., 2007, Galán et al., 2009]; Grimm and Railsback [2005] dedicated a whole chapter to this topic. But an accepted standard way of analysing ABMs is still missing.

The communication of ABMs is also very difficult, since ABMs are simulation models and

cannot be described completely by means of mathematics. Therefore, several communication protocols have been proposed. Some are based on an extension of the Unified Modelling Language (UML) but such approaches are usually too technical for non-computer scientists. The ODD protocol of Grimm et al. [2006] is much simpler and allows providing a precise overview over the general properties of the model. It has the potential to become a standard protocol for ABMs [Grimm et al., 2010, Polhill, 2010, Janssen et al., 2008].

However, since the ODD protocol only provides a structure for verbal model descriptions but does not specify how to document the details of the model it will be just one half of an overall model documentation. It does not yet provide a "lingua franca" by itself, which via some kind of pseudo code would allow describing all submodels unambiguously. In the next two sections we will discuss how the software platform NetLogo [Wilensky, 1999] might fill this gap.

The last point to be discussed here is the high risk of programming "bugs" in ABM implementations [Lorek and Sonnenschein, 1999, Gilbert, 2007]. A bug means that the computer is doing something different to what the programmer/modeller intended. Such bugs can lead to misinterpretations of the model outcomes [Axelrod, 1997].

Simulation platforms for agent-based models can help reducing the risk of programming bugs and can increase the understandability of the source code. Furthermore, such platforms will make the implementation process much easier [Hamill, 2010]. Heath et al. [2009] surveyed 279 ABM articles of which just 175 offered details about their programming language/tool. He identified 68 different tools/languages. Heath et al. [2009] conclude that there will never be one standard programming language.

We think, however, that the fact that modellers usually are not computer scientists and the challenges of ABMs described so far should encourage us to try and find a common language and standard procedures or template (sub) models. There is actually no point in implementing an ABM from scratch, using a general purpose programming language like Java or C++ because ABM libraries exist which provide standard software designs and code. Standardized model descriptions and implementations would facilitate to understand the underlying concepts, structure and algorithms of ABMs. Without such standards, it is difficult or even impossible to reproduce the results of published models, which undermines the scientific credibility of the ABM approach [Janssen et al., 2008].

Therefore, the next part of this chapter focuses on software platforms and languages for ABMs and is followed by a more detailed description of the NetLogo language and platform.

II.3. Software Libraries, Environments and Languages for ABMs

As discussed above, to decrease implementation time and the risk of making errors and to increase re-usability, traceability and communicability, it is helpful to use established software libraries or languages that were especially designed for implementing ABMs. One of the first programming languages which provided features for object-oriented modelling and simulation was SIMULA [Dahl et al., 1967]. It was indeed used for ABMs in ecology [e.g., Reuter, 1998], but required still considerable skills in mastering a general-purpose language, and its supportive features for ABMs were rather limited. Since that time, many general and special purpose libraries and platforms for the development of ABMs have been developed. Wikipedia [2010] lists 69 different agent-based modelling software libraries/-platforms and several reviews can be found in the literature [e.g., Allan, 2009, Nikolai and Madey, 2009, Berryman, 2008, Gilbert, 2007, Railsback et al., 2006, Tobias and Hofmann,

2004]. The most popular, non proprietary general-purpose ones are Swarm [Minar et al., 1996], MASON [Luke et al., 2005], Repast [Collier et al., 2003], and NetLogo [Wilensky, 1999], which are briefly presented in the following. Besides we would like to mention that any review of ABM software platforms is due to be outdated within a year or two because some platforms develop rapidly while others are no longer maintained. Therefore, in the following we do not list software features in detail but describe the history and the main concepts underlying the platforms.

Swarm can be seen as the mother of many other ABM libraries. The project was initiated in 1994 at the Santa Fe Institute, New Mexico [Hiebeler, 1994] and is maintained since 1999 by the Swarm Development Group [Swarm, 2010a]. The aim of Swarm is to provide a vocabulary and a set of standard computer tools for the development of multi-agent simulation models [Swarm, 2010b]. It comes as an open source library collection for Objective-C and has a port for using it in Java, which both follow the object-oriented programming (OOP) paradigm. Applications using Swarm often contain a hierarchical structure, with an `observerSwarm` object responsible for the creation of screen displays and a `modelSwarm` object, which manages the individual agents and schedules their activities in discrete time intervals as well as delivers information to the observer [Swarm, 2010b]. Agents are instances of user-written classes derived/extended from the class `SwarmObject`. Furthermore, the Swarm libraries provide classes for the creation of Graphical User Interfaces (GUI) for controlling simulations and for visualization [Swarm, 2010b]. Since Swarm consists just of libraries, the models are written in the language of the chosen library (Objective-C or Java). This requires strong programming skills but gives a maximum of flexibility and extendibility to the (experienced) user.

MASON is considerably younger as the project was initiated in 2003 at the George Mason University, USA [Balan et al., 2003]. It is written as an open-source library in Java and is conceptually strongly inspired by Swarm. The main development goal was a good computational performance to allow simulations with many runs, a large number of agents and a good support for 3D-simulations and visualizations [Luke et al., 2004]. It is, like Swarm, a pure but very concise library. The implementation of a simulation model in MASON is encapsulated completely from the code for visualization. The highest level of a simulation model is the `SimState` super-class which holds an instance of the `Schedule` class for managing the time and sub-schedules in the model. Agents can be added to the simulation/schedule by instancing user-written classes, which have to implement the interface `Steppable` with the agent schedule method step. Space can be represented by using the grid-based or continuous space classes whereas networks are created by using the classes `Network` and `Edge` [Mason, 2010]. The same as for Swarm holds about flexibility and required programming skills.

Repast (Recursive Porous Agent Simulation Toolkit) is slightly older than MASON as the first release was available in 2000 offered by a group of researches of the University of Chicago [Collier et al., 2003]. Started as a pure Java library, it was available for the programming languages Java (Repast J), Microsoft .Net (Repast .Net) and Python (Repast Py) in the meantime since Version 3.0, which was released in 2004 [Road, 2010a]. In 2008 the Repast development team, more precisely the Repast Organization for Architecture and Development (ROAD), introduced Repast Simphony (called Repast S), a customized version of the integrated development environment (IDE) Eclipse, which enables the user to build models using graphical editing tools via drag and drop. At the end of 2010, a beta version of Repast S 2.0 was released and opened different ways to build models: the point-and-click

flowcharts, Java, Groovy and ReLogo as a Logo dialect and an import and translation routine for NetLogo models [Road, 2010b]. In December 2010 as well, a first beta version of Repast High Performance Computing (HCP) was published, designed to run simulations in parallel on large-scale distributed computer platforms such as clusters and supercomputers. Models for Repast HCP can be implemented in standard or Logo-style C++ [Road, 2010c]. Although the Repast toolkit has its origins in social sciences, it is usable for the implementation of most agent-based models. The basis of a classical Repast model is a `Context` which is built by the `ContextBuilder` class. A `Context` manages a set of agents, like a population. Agents are user-defined classes for each type of agents (e.g., one for wolves and another for sheep) which implements behaviours by methods and state variables by class members. If spatial aspects are required, a `Projection` can be added to a `Context`, which places the agents into a `Grid` or `ContinousSpace`. To create relations between agents, the `NetworkBuilder` class is used [Collier and North, 2010]. The ReLogo part is conceptually equivalent to that of NetLogo, what is not surprising, since ReLogo is adapted from/inspired by NetLogo. Therefore, details about the ReLogo concept can be left out here as NetLogo is described in detail later. The Java/Groovy and ReLogo part of Repast deliver an expendable standard window for simulation visualization, a way for creating stand-alone programs and an option to run them in batch mode as well as a couple of interfaces to helpful programs like GNU R, GRASS GIS or MATLAB.

By now, the presented modelling libraries are conceptually very similar, except for ReLogo. Models are written in a common high level programming language by implementing interfaces or extending classes of the used libraries. There is always a top-level class coordinating the simulation and bundling the simulation entities. The class or interface which are extended or implemented by agent classes are unspecific, i.e., do not differentiate between different types of agents. Space is added explicitly to agent classes. Although some helping methods for implementing the behavioural rules for the agents exist, a lot of code has to be written by hand.

Because **NetLogo** is not a library but a complete language especially designed for the implementation of ABM, it provides many pre-defined methods for the implementation of the behavioural rules of the agents. Furthermore, its Logo-like syntax (see next section) and different standard agent types (`Turtles`, `Patches`, `Links`) in combination with a point-and-click GUI-building mechanism make it possible to learn the language very fast without programming experiences. No less important are the excellent and comprehensive documentation of the NetLogo language and simulation environment and the different tutorials. Moreover, there is a very large library of sample models and code examples as well as a very active mailing list. Furthermore, textbooks with practical introductions to agent- and individual-based modelling using NetLogo are available [e.g., Railsback and Grimm, 2012, Wilensky and Rand, 2014].

Because of these features, NetLogo is very popular and widely used. It has the potential to become a standard language for describing ABMs (as pseudo language) and is ideal for prototyping since in no other language ABMs can be written and changed as fast as in NetLogo. This hypothesis is underlined by the ReLogo project, which adapted the NetLogo language as described above and opens the possibility to translate NetLogo models to Repast, which allows to run also models which are computational very expansive. Therefore, in the following NetLogo is in focus as it has the potential to help to overcome the weakness of missing standards in ABMs in respect to the description of model details.

II.4. NetLogo: Modelling Language and Simulation Platform

II.4.1. History of NetLogo

The development of NetLogo started in 1999 [Wilensky, 1999] and the first beta version was released in 2000 by Uri Wilensky [Tisue and Wilensky, 2004a]. It is now maintained by the Center for Connected Learning and Computer-based Modeling at the Northwestern University, Illinois [Wilensky, 1999]. NetLogo is the successor of the StarLogo simulation environment family. Although StarLogo was mainly developed for the use in education it was used more and more by researches. This led to the development of NetLogo, which was designed for educational and research purposes [Tisue and Wilensky, 2004b]. Milestones in the development of NetLogo were

- the release of the first stable version 1.0 in 2002,
- the introduction of the HubNet functionality in 2003,
- the Extension and Controlling API as well as a headless mode for GUI-free command line runs in 2004,
- the System Dynamics Modeler for models including ordinary differential equations and a 3D view in 2005,
- an improved compiler that speeded up many models and the introduction of links as own agent type for network models in 2007,
- a GIS-Extension published in 2008 and
- BehaviorSpace, a tool for running NetLogo repeated with different parameter values, became open source and supports running multiple simulations with the BehaviorSpace in parallels in 2009 [NetLogo, 2010a].

The current version, released in December 2010, is 4.1.2.

A drawback of NetLogo is that although it is available free of charge, its source code is not available (with exception for BehaviorSpace and the extensions). This is often criticized in reviews as it means that the proper functioning of built-in NetLogo procedures, called "primitives", can only be validated by testing and not by inspecting and directly testing the underlying source. On the other hand, this is more a theoretical than a practical discussion: NetLogo includes a compiler [Sondahl et al., 2006] whose implementation is presumably beyond the programming skills of most modellers, who are often not computer scientists. Nevertheless, it is not good practice to make use of open source libraries, like the MersenneTwisterFast from the Repast project or the MovieEncoder from the MASON project, without opening the own project [NetLogo, 2010b]. But on the FAQ page of the NetLogo project the NetLogo team wrote, that "We are working on eventually releasing the source under an open source license." [NetLogo, 2010c].

II.4.2. NetLogo Programming Language

The NetLogo language is especially designed for the implementation of ABMs. As the name suggests, it is an extension of the Logo language. Logo was designed as a functional programming language for educational use in the 1960s and is a dialect of Lisp [Logo Foundation, 2010]. Logo is often known for its turtle, which is used to create graphics on the

monitor by giving movement commands to the turtle. NetLogo adapted this turtle approach and combined it with the concept of multiple turtles/agents and concurrency from *Lisp ("StarLisp"). The design goal of the Logo language was "low threshold and no ceiling", which was borrowed for the development of NetLogo [Tisue and Wilensky, 2004a]. Low threshold stands for easy to learn also by people with less or without modelling and programming experiences. No ceiling means that it should not be restrictive for advanced users, who need high flexibility.

The basic entity in NetLogo is the agent. There are different pre-defined types of agents: the observer, patches, turtles, and links. The observer is the global instance which provides global variables and manages and has access to the other agents. There is only one observer. Patches are immobile agents, i.e., spatial units with a location in space. All patches together form the grid of the world and define the extent of the world. Patches have pre-defined variables, such as x- and y-coordinates, a colour and a label, but the user can add further variables to the patches. Patches are, like all other agents, programmable. Turtles are, contrary to the patches, mobile agents. They can move on the patches in continuous space within the world defined by the patches. Turtles have, like patches, pre-defined variables, like their position, shape etc. but can get user-defined variables. It is possible to declare different types of turtles, called breeds; breeds inherit all variables of the turtles, but can have additional own variables. The last agent type is the link, which is a connection between two turtles. All agents can communicate and interact with each other.

The available data types in NetLogo are numbers (double and integer), boolean, string, colour, agent, agentset and list. The agentset data type is a collection of agents and the list data type is a vector to store multiple values. Blocks of code are defined by embedding them into squared brackets and comments are available just as line comments beginning with a semicolon.

Variable declarations and assignments can be done by using the operator `set` or by using `let` for local variables, as shown in Listing II.1. Advanced programmers need to get used to it, but for the novice this is a very natural way for assignments, as it is written like a spoken command.

Listing II.1: Declaration (and assignment) of a new local variable in the first line and re-assignment in the second line. The assignment of global variables is the same as the re-assignment of local variables. This code fragment could be embedded into a user defined procedure.

```
let my-local-var 10 ; declaration and first assignment
set my-local-var 20 ; re-assignment
```

The NetLogo language distinguishes between commands and reporters. Commands are instructions to an agent whereas reporters calculate and return a value. There are around 400 built-in commands and reporters, called primitives. Some of them have to be executed in a specific agent type context, for example, `move-to` is just for turtle context since it does not make sense to ask a patch for moving away as they are immobile. User-defined commands and reporters are called procedures and can consist of sequences of commands and reporters. They are defined by the opening keyword `to` or `to-report` followed by the name given to the procedure, which includes the things to do, and are closed by the keyword `end`. An example is given in Listing II.2.

Listing II.2: A user defined procedure with the name `setup` which resets everything and creates two turtles.

```
to setup
  clear-all
  create-turtles 2
end
```

If necessary, input variables for procedures can be defined. Commands and reporters are executed by using their name. If inputs should be passed to the command or reporter, the user has to write them directly after the name of the command or reporter in a whitespace separated list. Because reporters return values on the right side of a reporter call, there has to be an output command or an assignment. Typical for Logo languages is that commands and reporters, although comparable with functions or methods in other languages, do not have parentheses after the name containing the input variables. There is no terminal character, like the semicolon in Java or C++. Everything is separated just by whitespace. In some cases, when a primitive gets optional or repeatable inputs, the primitive and its inputs have to be declared as belonging together by using parentheses.

A very important command is `ask` which iterates over the given agentset and executes the commands and reporters given in a block with the context of the current agent of the iterated agentset. This `ask` command is very powerful in combination with the `with` reporter, which creates a new agentset containing only those agents that satisfy a given condition. A scheduler for a simulation model could then look like the procedure `go` in Listing II.3 and would be executed in observer context.

Listing II.3: A (fragmented) example of a model source code with a user-defined turtle type (beech breed), a `setup` procedure to initialize the simulation, a `go` procedure as a scheduler for one simulation step which iterates over all beech-turtles and some other procedures called from the `go` procedure.

```
breed [beeches beech] ; create a new breed (turtle subtype)
   named beech

to setup ; a user-defined procedure setup, to initialize
   simulation
  ...
end

to go ; procedure, executed in observer context
  tick ; increment internal simulation time
  ask patches [ ; do for all patches
    update-soil-water ; call procedure update-soil-water in
       patch context
  ]
  ask beeches [ ; do for all beeches
    grow
    mortality
  ]
end
```

```
to update-soil-water
  ...
end

to grow
  ...
end

to mortality
  ...
end
```

II.4.3. NetLogo Integrated Simulation Environment

As mentioned before, NetLogo comes with an integrated, interactive simulation environment. There are three basic tabs, one for the model source code (Procedures Tab), one for the model description (Information Tab) and one for the Graphical User Interface (Interface Tab), as shown in Figure II.1.

When leaving the Procedures Tab or when dropping the Check Button, NetLogo runs the spell checker, reports errors, if found (see Figure II.2), and tokenize as well as compiles parts of the code for execution/interpretation (for details on the combined compiler-interpreter architecture with the byte code inlining technique see Sondahl et al. [2006]).

Graphical elements can be placed somewhere on the Interface Tab by drag-and-drop. The user can add Buttons, Sliders, Switches, Choosers, Inputs, Monitors, Plots (Scatter, Line, and Bar plots), Outputs and Notes. Some of them take additional NetLogo commands or reporters. For example, a button, when pushed, executes the command the user gave to it, like the execution of a procedure.

One graphical element which is always there is the View, which visualizes the patches and the turtles. If one wants to implement a non-spatial model, it is possible to hide the turtles and reduce the world to just one remaining patch, which could be leaved unused and hidden behind a plot. If a spatial model is implemented, users can choose a wrapping world, choose the number and size of the patches and set the location of the origin of the coordinates. Within the View, patch, link and turtle variables can be inspected by clicking on the desired agent with the mouse. The variables of this agent are shown in a new window and can be watched and changed during and after the simulation. If required, NetLogo offers a 3D view of the world for perspective visualizations.

Furthermore, users can control the speed of a simulation via a slider which opens the possibility to slow down the simulation to observe the turtle's movement in detail.

With the Command Center, NetLogo delivers an interpreter. It is possible to execute any command or reporter during the simulation within the context of the observer, patch, turtle or link.

II.4.4. NetLogo Extensions and Controlling API

Due to NetLogo's design philosophy "low threshold, no ceiling", the developers included an interface for everyone to extend the NetLogo language. The Extension API offers a way to

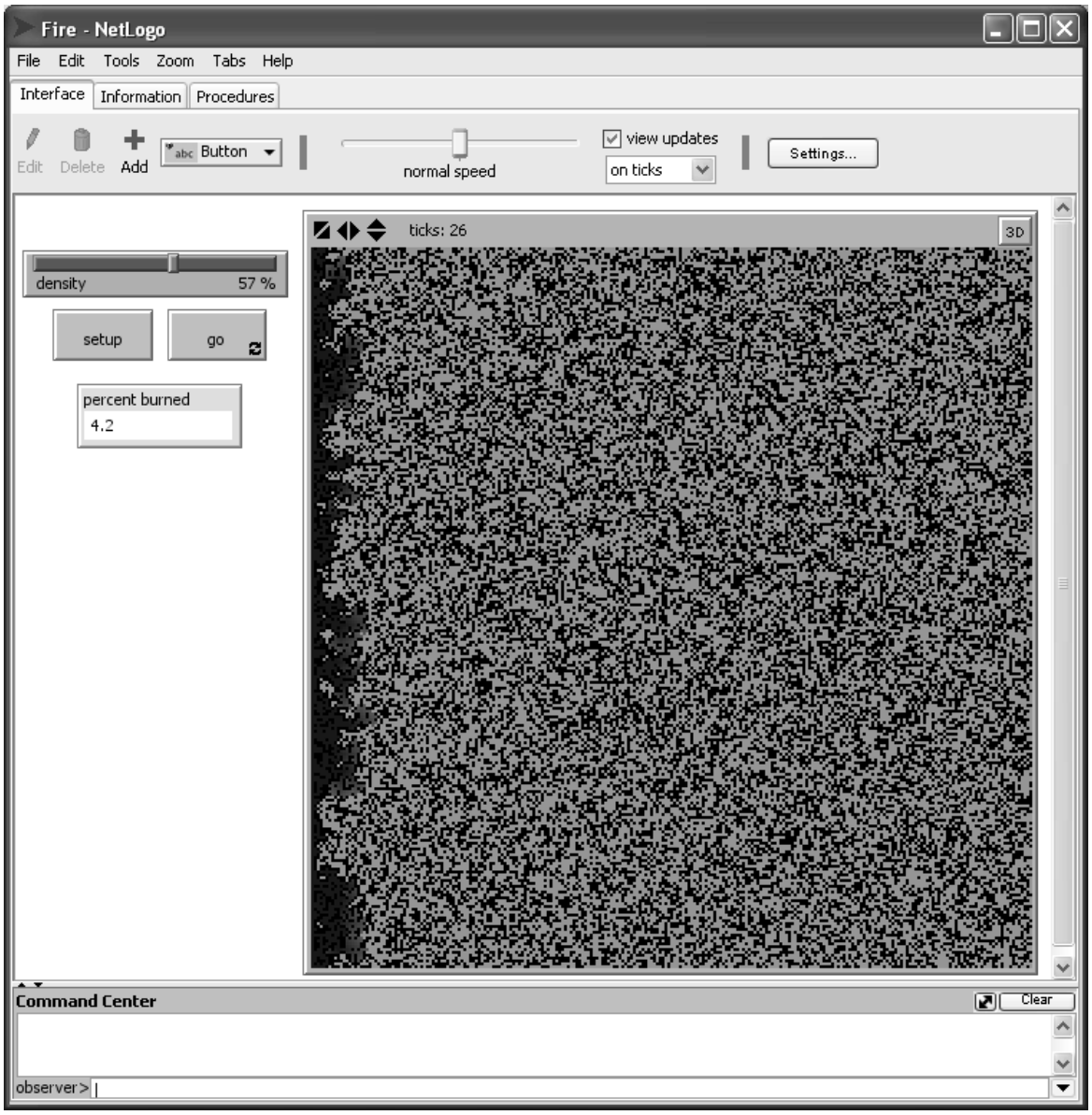


Figure II.1.: NetLogo GUI, Interface Tab with the View/World on the right, some control widgets on the left and Command Center at the bottom.

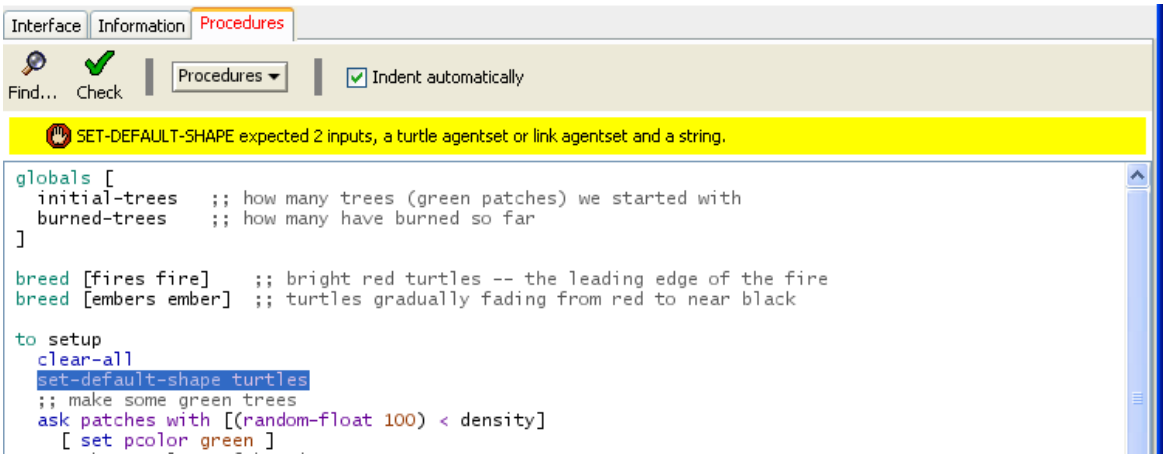


Figure II.2.: NetLogo GUI, Procedures Tab with Syntax Checker indicating an error.

extend the language by adding user-defined primitives [Stonedahl et al., 2008]. Extensions can be written in Java or Scala and can access NetLogo objects, like turtles or lists.

NetLogo comes with a bundle of standard extensions, like the GIS-Extension, which adds functionality to import raster and vector datasets into a NetLogo model or the array, table and matrix extensions, which add multi-dimensional data storages. The Controlling API allows running the NetLogo application by remote control. It is possible to open models and execute NetLogo commands and reporters from other Java and Scala programs.

II.5. Extensions to NetLogo

This last section of this chapter gives an overview over new extensions to and tools for NetLogo written by the authors of this chapter. The extensions were developed to extend the functionality of NetLogo for making this software more relevant as a standard tool of ABM and, in case of the R-Extension, to combine ABMs with standardized methods from statistics and to avoid hand-written solutions.

II.5.1. MultiView

As described above, NetLogo visualizes the patches in the world widget. For this, the built-in patch variable `pcolor` is used by default. The value of `pcolor` or any other patch variable, if defined, determines the colour of each patch. But there is no way to create more than one world widget to visualize more than one patch variable at a time. For example, if we take a model like BEFORE [Rademacher et al., 2004] this could be a restriction. BEFORE simulates the dynamic of natural beech forests. It distinguishes four horizontal layers which are characterized by their coverage percentage. The forest itself is divided into quadratic patches, which are represented in NetLogo by the patches of the NetLogo world. The coverage percentage of each horizontal layer is stored in a user-defined patch variable. There is no useful way to visualize the four patch variables simultaneously.

Another example is a model where we want to compare different temporal stages at a time. For example, if one defines a patch variable which saves the value of the variable of

interest of the last simulation step there is no way to see the current spatial pattern of the patch variables and the pattern of the last simulation step simultaneously.

Therefore, we developed an extension which adds new windows to the simulation showing the patches of the world. The windows contain a copy of the view, i.e., uses the settings of the view regarding patch size and the number of patches. The user can define the patch variable which is to be used for colourization of the patches in the new view window. There is also the right mouse click functionality within the view window available to inspect patches and to export the view into an image file (see Figure II.3 and Listing II.4). The user can add as many additional view windows as desired.

Listing II.4: An example for the usage of the MultiView-Extension which creates two additional view windows, one as a copy of the original NetLogo world view and a second with a colourization using variable `pcolor2`.

```
; load the extension
extensions [multiview]

; a patch variable to save the 2nd colour (beside pcolor) for
   the 2nd view window
patches-own [ pcolor2 ]

globals [
  view1 ; a variable to save the reference to a view window
  view2 ; a 2nd variable to save the reference to a 2nd view
     window
]

to setup
  ; clear-all, closes the view windows as well, view1 & view2
     has value 0
  ca
  ; create two turtles
  crt 2 [ set xcor random xcor set ycor random ycor]
  ; set the colours of the patches, pcolor and pcolor2 will be
     the same initially
  ask patches [
    set pcolor random 300
    set pcolor2 pcolor
  ]
  ; open the 1st view window, using pcolor (as in the NetLogo
     world view) to
  ; colourize the patches
  ; first parameter "pcolor" is the title of the window
  ; second parameter "pcolor2" is the patch variable used for
     colouring the patches
  ; save the reference to the view window in the variable
     "view1"
  set view1 multiview:newView "pcolor" "pcolor"
  ; open the second view window, using pcolor2
```

```
  set view2 multiview:newView "pcolor2" "pcolor2"
end

to go
  tick
  ; change the colour values of the patches
  ask patches [
    set pcolor random 300
    set pcolor2 random 100
  ]
  ; update the colours of the view windows (using the
     predefined patch variables,
  ; see above: multiview:newView)
  multiview:repaint view1
  multiview:repaint view2
end
```

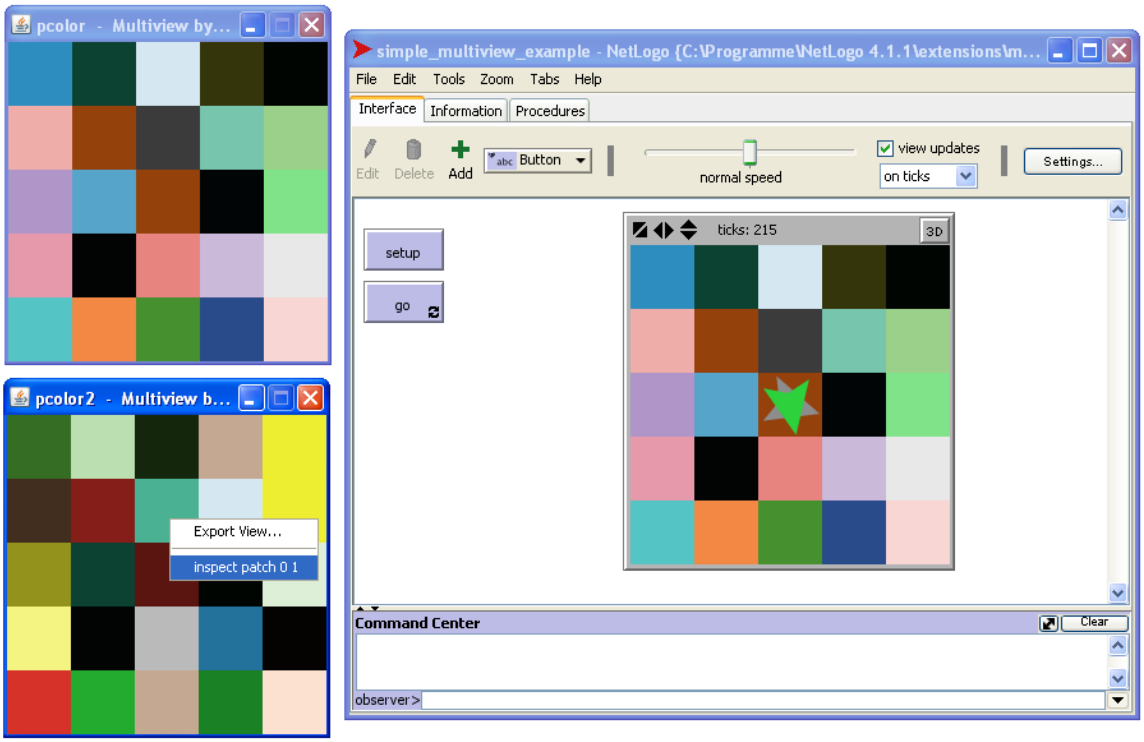


Figure II.3.: NetLogo GUI on the right with two additional view windows on the left created by the MultiView- Extension. The additional view window on the top uses the `pcolor` variable for colourization and is therefore a duplication of the original view, the bottom view uses the `pcolor2` variable for colourization as shown in Listing II.4. On the bottom view, the right click functionality is shown. The user can export the current view into an image file and can inspect the selected patch.

The extension adds just four new primitives to the NetLogo language: One for the cre-

ation of a new window, one for repainting/updating the view, another for renaming the window and a last for (tidy) closing the window. The extension is available for download at the official NetLogo web page: http://ccl.northwestern.edu/netlogo/resources.html. It is released under the GNU GPL v2 open source license and comes with a short documentation (see also Appendix A) and the source code.

II.5.2. R-Extension

NetLogo already provides some primitives for data analysis, like mean, median, mode, variance and standard-deviation but it lacks advanced methods. Therefore, we developed an extension to link NetLogo directly to the statistical software GNU R [Thiele and Grimm, 2010]. There are two typical use cases of this extension that we had in mind while developing it. First, the integration of advanced statistical methods within the model itself and second the integrated and immediate stepwise analysis of simulation results.

The first goal includes methods like regression analysis or just random numbers from special random distributions. Imagine, for instance, that the amount of food intake of an agent is dependent on the expectations for the future which is based on the experiences of the past and current circumstances. Here, non-linear regression models could be fitted to the past values and used to make the forecast. In such a case, the stepwise fitting of the regression model and the forecast could be computed using GNU R and used by the NetLogo simulation. This would require an interactive use of both software packages.

An example of the second intended use of our R-Extension of NetLogo is the analysis of the spatial distribution of agents using spatial point pattern statistics like Ripley's K. Another example is the calculation of diversity indices. Furthermore, as R delivers advanced plotting functionalities, the R-Extension can be used to extend the limited plotting capabilities of NetLogo. Moreover, connections to all common databases could be established via GNU R.

By using the R-Extension, the modeller can save a lot of work and time by using R functions instead of programming statistical analysis from scratch. This guarantees to use reliable functions implemented and tested by the R programmers/contributors and users. Because R is a standard tool in statistical analysis, listing the R functions used describes the used methods comprehensively. Therefore, using the R-Extension keeps the NetLogo code short, clear, traceable and reduces the chance of doing bugs.

The extension makes use of the rJava package for GNU R [Urbanek, 2013] and the Extension API of NetLogo. Via the rJava package, or more precisely the JRI library within the rJava package, it is possible to create R data types from Java via the Java Native Interface (JNI) and C. The NetLogo data types are accessed via the Extension API and converted into R data types within the R-Extension. R commands are evaluated within the R-Extension and return values are converted into NetLogo data types. Even NetLogo turtles, links, patches and lists are supported as well as R vectors, lists and data frames.

The R-Extension adds nine new primitives to the NetLogo language for the interaction between NetLogo and R and six additional primitives for debugging purposes. Table II.1 lists the different basic primitives.

Table II.1.: Primitives added by the R-Extension with a short description (without the additional debugging primitives).

Primitive	Function
`r:clear`	Clears the R workspace, deletes all variables.
`r:eval`	Evaluates the submitted string in R, used for R function without a return value.
`r:get`	Gets a value from R; either submits a string including an R function with return value, or gets the value of a variable; return values can be strings, numbers, booleans, lists, or a list of lists.
`r:put`	Creates a new variable in R with value(s) from NetLogo; submitted values/variables can be strings, numbers, booleans, or lists (NetLogo lists become R vectors, to create R lists see putlist).
`r:putagent`	Creates a list in R from variables of a set of NetLogo agents; collections of agent-variables can be stored in a single named list.
`r:putagentdf`	Same as putagent, but creates an R data frame instead of a list because many R functions requires data frames as input.
`r:putlist`	Creates a new R list based on the submitted NetLogo variables and values.
`r:putnamedlist`	Same as putlist, but creates a named R list, i.e., the columns of the list can be called via their names.
`r:putdataframe`	Same as putnamedlist, but creates an R data frame instead of a list because many R functions require data frames as input.

A nice, but often not recognized, feature is the `r:interactiveShell` primitive. If NetLogo has been started via the shell/command line/MS-DOS prompt and uses the R-Extension the user can redirect the connection to the underlying R session to the shell. Then, the user can work directly in the R session, can access the variables created with the R-Extension in NetLogo, can execute R commands and access newly created R variables within NetLogo using the R-Extension (see Figure II.4).

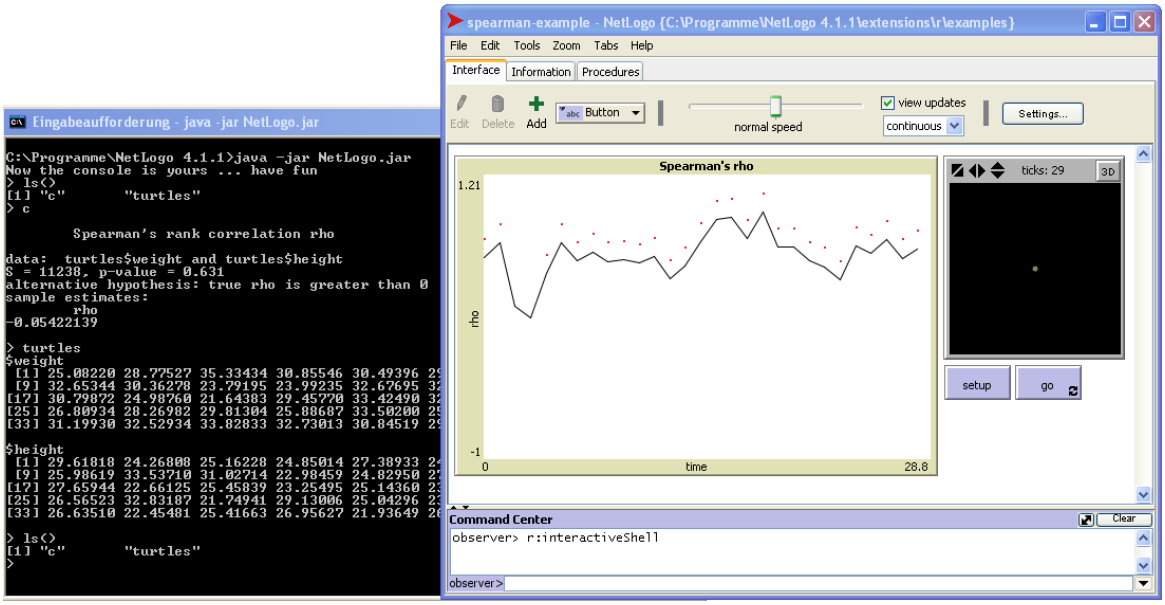


Figure II.4.: NetLogo started from an MS-DOS prompt with calculation of Spearman's Rho over time using the R-Extension (on the right hand side); using the R-Extension's interactiveShell primitive to get access to the R session in the MS-DOS prompt (on the left hand side) and checking the content of the turtles variable created by the R-Extension in NetLogo as well as listing the content of the R session.

Figure II.5 shows an example for the usage of the R-Extension in spatial statistics. The point pattern of the agent positions is used to calculate the L-function (based on Ripley's K) with Monte-Carlo-Simulations for the null hypotheses test (complete spatial randomness). For this, the agent positions are sent to an R data frame via the primitive `r:putagentdf`. By using the R package spatstat [Baddeley and Turner, 2013] this agent positions data frame is transformed into a planar point pattern object (ppp) and used as input for the envelope function, which performs 99 replicated simulations (Monte-Carlo-Simulation) for the hypotheses test. The point wise critical envelopes for L(r) are then plotted using the standard plot function of R, which creates a grey band for the envelope with a dotted line for the theoretical value under complete spatial randomness and a solid line for the observed pattern. The code for this operation in NetLogo is shown in Listing II.5. In Figure II.5 you can see that it is also possible to send the result of the envelope calculation back to NetLogo and make a simple plot there, but without the nice grey bands and dotted lines. The NetLogo code for this step, which requires some data transformations for the plot routine, can be found in the examples which are delivered with the R-Extension.

Listing II.5: An example for the usage of the R-Extension. In the `setup` procedure some turtles are created with a clustered spatial distribution. Furthermore, the R package spatstat for spatial statistics is loaded. In the `go_with_Rplot` procedure the turtles walk around randomly first. Then, the positions of the turtles are sent to R into a data frame. This data frame is used to create a point pattern object which is used to calculate the L-function with critical bands. The result of the L-function calculation/simulation is plotted using R.

```
extensions [r]

to cluster_setup
  ; clear all
  ca
  r:clear
  ask patches [ set pcolor 9.9]
  random-seed 1234567
  ; load R package spatstat for spatial statistics
  r:eval "library(spatstat)"
  ask n-of 10 patches [
    ask neighbors [
      sprout 3 [
        right random 360
        forward random 2
      ]
    ]
  ]
end

to go_with_Rplot
  tick
  ; let the turtles walk around randomly
  ask turtles [
    right random 360
    forward random 4
  ]
  ; send agent variables into an R data frame
  (r:putagentdf "agentset" turtles "who" "xcor" "ycor")
  ; create point pattern with vectors of x- and y-coordinates
     of turtles and the
  ; dimension of the world
  let revalstring (word "agppp <- ppp(agentset$xcor,
     agentset$ycor, c("min-pxcor","max-pxcor"),
     c("min-pycor","max-pycor"))")
  r:eval revalstring
  ; calculate L-function with critical bands from 99
     Monte-Carlo-Simulations
  r:eval "Lsim <- envelope(agppp, Lest)"
  ; plot the L-function
  r:eval "plot(Lsim, main=\"L function (based on Ripley's K)\")"
```

```
end
```

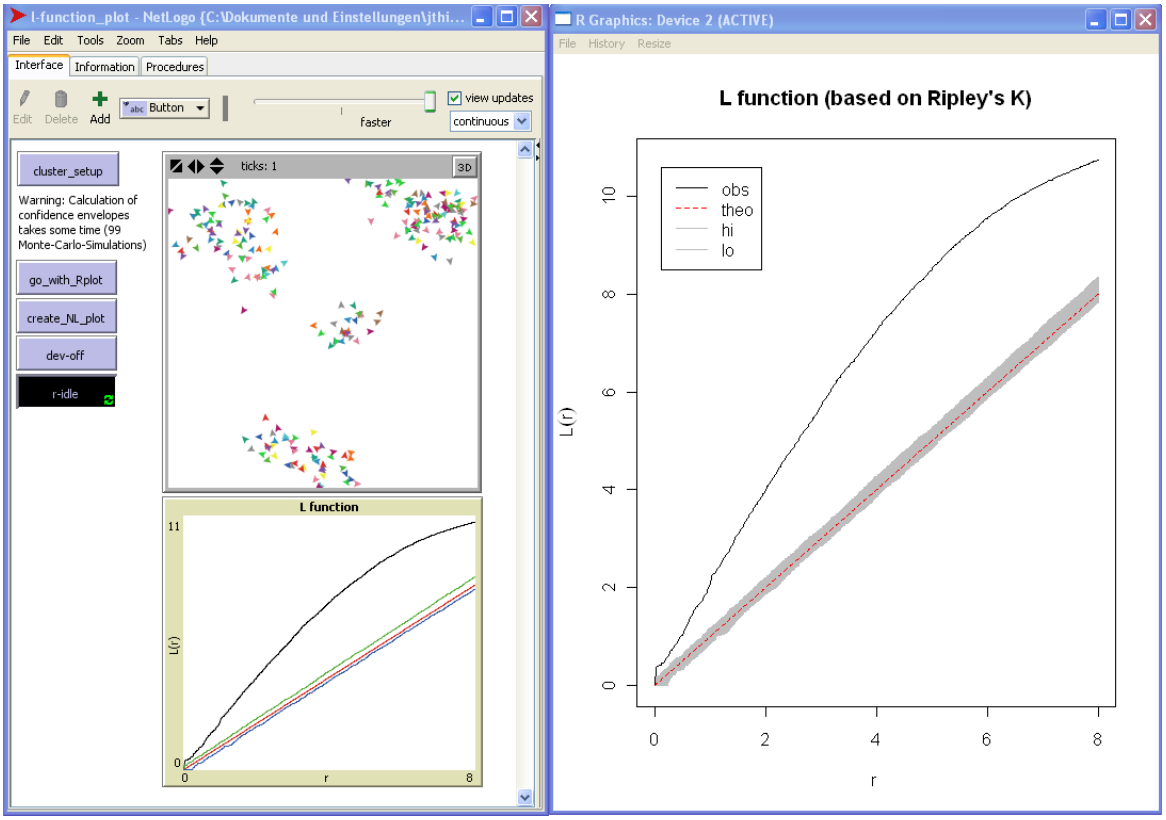


Figure II.5.: NetLogo GUI on the left, GNU R plot window called from NetLogo R-Extension on the right; the results of the L-function calculation are sent back to NetLogo and are visualized in a plot on the bottom of the NetLogo GUI as well.

The R-Extension is specific to the version number of GNU R/r-Java and NetLogo. It comes with documentation (see Appendix C) and different examples as well as the source code under the GNU GPL v2 open source license. It is available for download at the BerliOS repository: http://netlogo-r-ext.berlios.de/. The extension is tested on Windows XP, Windows Vista, Windows 7, Linux (Ubuntu, SuSe) and Apple Macintosh. It runs also on 64-bit systems. Since the release at the beginning of 2010 until December of 2010, it was downloaded more than 300 times. The most frequent problem which has been reported was the installation/configuration process. To create the connection to R, the user has to set two environment variables, which turned out to be too difficult for some users.

II.5.3. Pygments Parser

Not a real extension but a small supporting tool is the NetLogo language definition for Pygments [Pocco, 2010]. Pygments is an open source syntax highlighting engine written in Python which takes source code and produces output in different formats that contain syntax highlighting markup. Output formats include HTML, LaTeX, RTF, GIF, PNG, JPEG and others. It can be used as a library or as a command-line tool.

The NetLogo language definition makes it possible to generate automatically NetLogo model source code in different formats, as mentioned above, directly from the original source file. The output looks like in the Procedures Tab of NetLogo with respect to the colourization as well as the indentation and is therefore easily available and editable for publications in text processing software or for using it on websites.

It is written as a Plug-In for Pygments with a lexer based on regular expressions and keyword lists as well as a style definition for the colourization. It works for the primitives of the bundled extensions as well as for the MultiView- and R-Extension.

The Plug-In includes a setup script which automatically adds the Plug-In to Pygments. A command line call for creating an HTML output could look as shown in Listing II.6. The user can choose between an embedded css-style (Cascading Style Sheet) definition within the HTML file or without. It is possible to create a separate css-file based on the style definition.

Listing II.6: An example for the usage of the Pygments NetLogo Plug-In to create an HTML file (test1.html) from a NetLogo model file (test1.nlogo) with embedded css-style (a) and separated css-file (b).

```
# a. with embedded css-style:
pygmentize -l NetLogo -O full,style=NetLogo -f html -o
   test1.html test1.nlogo

# b. with extra css-file:
# b.I. Create the html file:
pygmentize -l NetLogo -f html -o test1.html test1.nlogo

# b.II. Export the style to css file:
pygmentize -f html -S NetLogo -a .syntax > netlogosyle.css
```

The Pygments Plug-In for NetLogo language is available at http://www.uni-goettingen.de/de/72779.html (see also Appendix B).

II.6. Outlook

To further extend the functionality of NetLogo, which will help to strengthen its potential as a standard tool, further tools for NetLogo are currently in preparation. One of these tools is RNetLogo, a package for GNU R to include NetLogo simulations within R. This is the reverse connection of the R-Extension with its own strength. It will overcome the difficulties in the setup process of the R-Extension with the creation of environment variables and will have the functionalities of the Mathematica Link for NetLogo described in Bakshy and Wilensky [2007]. It could be used to establish a standard protocol for calibrating and analysing ABMs. GNU R with its huge amount of packages is the ideal basis for designing simulation experiments and analysing their results.

Another important functionality, which is currently missing in NetLogo, is a stepwise debugger as mentioned by Railsback et al. [2006]. Such a tool is currently under development and will fill this gap.

II.7. Acknowledgements

We would like to thank Michael Henke for some helpful comments on the manuscript.

II.8. References

S. Algers, E. Bernauer, M. Boero, L. Breheret, C. Di Taranto, M. Dougherty, K. Fox, and J.-F. Gabard. A Review of Micro-Simulation Models. Technical Report Project Report SMARTEST/D3, Institute of Transport Studies, University of Leeds, 1997.

R. Allan. Survey of Agent Based Modelling and Simulation Tools. Technical report, STFC Daresbury Laboratory, 2009. URL http://epubs.cclrc.ac.uk/bitstream/3637/ABMS.pdf. (last accessed 2014/01/06).

G. An. Dynamic Knowledge Representation Using Agent-based Modeling: Ontology Instantiation and Verification of Conceptual Models. *Methods in Molecular Biology*, 500:445–68, 2009.

J. Arifovic. Genetic Algorithm Learning and the Cobweb Model. *Journal of Economic Dynamics and Control*, 18(1):3–28, 1994.

W.B. Arthur. Out-of-Equilibrium Economics and Agent-Based Modeling. In L. Tesfatsion and K.L. Judd, editors, *Handbook of Computational Economics Vol. 2: Agentbased Computational Economics*, pages 1551–1564. Elsevier, Amsterdam etc., 2006.

R. Axelrod. *The Complexity of Cooperation: Agent-Based Models of Competition and Collaboration*. Princeton Univ. Press, Princeton, 1997.

R. Axtell. Why Agents? On the Varied Motivations for Agent Computing in the Social Sciences. Technical Report 17, The Brookings Institution, Washington, DC, 2000.

A. Baddeley and R. Turner. *Package 'spatstat' Manual. R package version 1.35-0*, 2013. URL http://cran.r-project.org/web/packages/spatstat/. (last accessed 2014/01/06).

E. Bakshy and U. Wilensky. Turtle Histories and Alternate Universes: Exploratory Modeling with NetLogo and Mathematica. In M.J. North, c.M. Macal, and D.L. Sallach, editors, *Proceedings of the Agent 2007 Conference on Complex Interaction and Social Emergence*, pages 147–158. IL: Argonne National Laboratory and Northwestern University, 2007.

G.C. Balan, C. Cioffi-Revilla, S. Luke, L. Panait, and S. Paus. MASON: A Java Multi-Agent Simulation Library. In *Proceedings of the Agent 2003 Conference*, 2003. URL http://cs.gmu.edu/∼eclab/projects/mason/publications/Agent2003.pdf. (last accessed 2014/01/06).

L. Ben Said, T. Bouron, and A. Drogoul. Agent-Based Interaction Analysis of Consumer Behavior. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 1*, AAMAS '02, pages 184–190, New York, NY, USA, 2002. ACM.

M. Berryman. Review of Software Platforms for Agent Based Models. Technical Report DSTO-GD-0532, Australian Government - Department of Defence, Defence Science and Technology Organisation, 2008. URL http://dspace.dsto.defence.gov.au/dspace/bitstream/1947/9306/1/DSTO-GD-0532%20PR.pdf. (last accessed 2014/01/06).

F.C. Billari and A. Prskawetz. *Agent-Based Computational Demography: Using Simulation*

to Improve Our Understanding of Demographic Behaviour. Contributions to Economics. Physica, 2003.

H. Bossel. *Modeling and Simulation*. A.K. Peters, Wellesley, 1994.

D.B. Botkin, J.F. Janak, and J.R. Wallis. Some Ecological Consequences of a Computer Model of Forest Growth. *Journal of Ecology*, 60(3):849–872, 1972.

B. Breckling. Individual-Based Modelling - Potentials and Limitations. *TheScientificWorld-JOURNAL*, 2:1044–1062, 2002.

M. Buchanan. Economics: Meltdown Modelling. *Nature*, 460:680–682, 2009.

R.T. Bunsing and D. Mailly. Advances in Spatial, Individual-Based Modelling of Forest Dynamics. *Journal of Vegetation Science*, 15:831–842, 2004.

C. Carpenter and L. Sattenspiel. The Design and Use of an Agent-Basedased Model to Simulate the 1918 Influenza Epidemic at Norway House, Manitoba. *American Journal of Human Biology*, 21(3):290–300, 2009.

H. Caswell. *Matrix Population Models: Construction, Analysis, and Interpretation*. Sinauer Associates, 2nd edition, 2001.

K.D. Coates, C.D. Canham, M. Beaudet, D.L. Sachs, and C. Messier. Use of a Spatially Explicit Individual-Tree Model (SORTIE/BC) to Explore the Implications of Patchiness in Structurally Complex Forests. *Forest Ecology and Management*, 186:297–310, 2003.

N. Collier and M. North. *Repast Java Getting Started*, 2010. URL http://repast.sourceforge.net/docs/RepastJavaGettingStarted.pdf. (last accessed 2014/01/06).

N. Collier, T. Howe, and M.J. North. Onward and Upward: The Transition to Repast 2.0. In *First Annual North American Association for Computational Social and Organizational Science Conference*, Pittsburgh, PA USA, 2003. North American Association for Computational Social and Organizational Science.

R. Conte. From Simulation to Theory (and Backward). In F. Squazzoni, editor, *Epistemological Aspects of Computer Simulation in the Social Sciences, Second International Workshop, EPOS 2006, Brescia, Italy, October 5-6*, volume 5466 of *Lecture Notes in Computer Science*, pages 29–47. Springer, 2006.

R. Conte, N. Gilbert, and J.S. Sichman. MAS and Social Simulation: A Suitable Commitment. In J.S. Sichman, R. Conte, and N. Gilbert, editors, *Multi-Agent Systems and Agent-Based Simulation, First International Workshop, MABS '98, Paris, France, July 4-6*, volume 1534 of *Lecture Notes in Computer Science*, pages 1–9. Springer, 1998.

P.-H. Cournède, T. Guyard, B. Bayol, S. Griffon, F. De Coligny, P. Borianne, M. Jaeger, and P. De Reffye. A Forest Growth Simulator Based on Functional-Structural Modelling of Individual Trees. In B. Li, M. Jaeger, and Y. Guo, editors, *Plant Growth Modeling, Simulation, Visualization and Applications, Proceedings PMA09*, pages 34–41, Los Alamitos, 2010. IEEE Computer Society.

O.-J. Dahl, B. Myhrhaug, and K. Nygaard. *SIMULA 67, Common Base Language*. Norwegian Computing Center, Oslo, 1967.

P. Davidsson. Agent Based Social Simulation: A Computer Science View. *Journal of Artificial*

Societies and Social Simulation, 5 (1), 2002. URL http://jasss.soc.surrey.ac.uk/5/1/7.html. (last accessed 2014/01/06).

D.L. DeAngelis and L.J. Gross. *Individual-based Models and Approaches in Ecology: Populations, Communities, and Ecosystems*. Chapman & Hall, 1992.

D.L. DeAngelis and W.M. Mooij. Individual-Based Modeling of Ecological and Evolutionary Processes. *Annual Review of Ecology, Evolution, and Systematics*, 36:147–168, 2005.

D.L. DeAngelis, D.K. Cox, and C.C. Coutant. Cannibalism and Size Dispersal in Young-of-the-year Largemouth Bass: Experiment and Model. *Ecological Modelling*, 8:133–148, 1980.

D.L. DeAngelis, L.W. Barnthouse, W. Van Winkle, and R.G. Otto. A Critical Appraisal of Population Approaches in Assessing Fish Community Health. *Journal of Great Lakes Research*, 16(4):576–590, 1990.

D.L. DeAngelis, K.A. Rose, and M.A. Huston. Individual-Oriented Approaches to Modeling Ecological Population and Communities. In S.A. Levin, editor, *Frontiers in Mathematical Biology*, pages 390–410. Springer, 1994.

M. Drechsler, V. Grimm, J. Mysiak, and F. Wätzold. Differences and Similarities Between Ecological and Economic Models for Biodiversity Conservation. *Ecological Economics*, 62 (2):232–241, 2007.

J.M. Epstein and R. Axtell. *Growing Artificial Societies: Social Science from the Bottom Up*. The Brookings Institution, Washington, DC, 1996.

J.D. Farmer and D. Foley. The Economy Needs Agent-Based Modeling. *Nature*, 460:685–686, 2009.

J. Ferrer, C. Prats, and D. López. Individual-Based Modelling: An Essential Tool for Microbiology. *Journal of Biological Physics*, 34(1-2):19–37, 2008.

J.W. Forrester. *World Dynamics*. MIT Press, Cambridge, 1971.

J.M. Galán, L.R. Izquierdo, S.S. Izquierdo, J.I. Santos, R. Del Olmo, A. López-Paredes, and B. Edmonds. Errors and Artefacts in Agent-Based Modelling. *Journal of Artificial Societies and Social Simulation*, 12(1), 2009. URL http://jasss.soc.surrey.ac.uk/12/1/1.html. (last accessed 2014/01/06).

N. Gilbert. Simulation: A New Way of Doing Social Science. *American Behavioral Scientist*, 40:1485–1487, 1999.

N. Gilbert. Agent-Based Social Simulation: Dealing with Complexity, 2004. URL http://cress.soc.surrey.ac.uk/resources/ABSS%20-%20dealing%20with%20complexity-1-1.pdf. (last accessed 2014/01/06).

N. Gilbert. *Agent-Based Models*. Quantitative Applications in the Social Sciences. Sage, Los Angeles, CA, 2007.

N. Gilbert and K.G. Troitzsch. *Simulation for the Social Scientist*. Open Univ. Press, Buckingham etc., 1999.

M. Ginovart, D. López, and J. Valls. INDISIM, an Individual-Based Discrete Simulation Model to Study Bacterial Cultures. *Journal of Theoretical Biology*, 214(2):305–319, 2002.

C. Godin and H.Sinoquet. Functional-Structural Plant Modelling. *New Phytologist*, 166:

705–708, 2005.

S. Green, L. Hurst, B. Nangle, P. Cunningham, F. Somers, and R. Evans. Software Agents: A Review. Technical Report TCD-CS-1997-06, Trinity College Dublin, Department of Computer Science, 1997.

A.F. Griffin and C. Stanish. An Agent-Based Model of Prehistoric Settlement Patterns and Political Consolidation in the Lake Titicaca Basin of Peru and Bolivia. *Structure and Dynamics*, 2:1–46, 2007.

V. Grimm. Ten Years of Individual-Based Modelling in Ecology: What Have We Learned and What Could We Learn in the Future? *Ecological Modelling*, 115:129–148, 1999.

V. Grimm and S.F. Railsback. *Individual-Based Modeling and Ecology*. Princeton University Press, Princeton, N.J., 2005.

V. Grimm, U. Berger, F. Bastiansen, S. Eliassen, V. Ginot, J. Giske, J. Goss-Custard, T. Grand, S.K. Heinz, G. Huse, A. Huth, J.U. Jepsen C. Jørgensen, W.M. Mooij, B. Müller, G. Pe'er, C. Piou, S.F. Railsback, A.M. Robbins, M.M. Robbins, E. Rossmanith, N. Rüger, E. Strand, S. Soissi, R.A. Stillman, R. Vabø, U. Visser, and D.L. DeAngelis. A Standard Protocol for Describing Individual-Based and Agent-Based Models. *Ecological Modelling*, 198:115–126, 2006.

V. Grimm, U. Berger, D.L. DeAngelis, J.G. Polhill, J. Giske, and S.F. Railsback. The ODD Protocol: A Review and First Update. *Ecological Modelling*, 221:2760–2768, 2010.

O. Gun. Why Do We Have Separate Courses in 'Micro' and 'Macro' Economics. In E. Fullbrook, editor, *A Guide to What's Wrong with Economics*. Anthem Press, London, 2004.

L. Hamill. Agent-Based Modelling: The Next 15 Years. *Journal of Artificial Societies and Social Simulation*, 13 (4) 7, 2010. URL http://jasss.soc.surrey.ac.uk/13/4/7.html. (last accessed 2014/01/06).

N Heath, R. Hill, and F. Ciarallo. A Survey of Agent-Based Modeling Practices (January 1998 to July 2008). *Journal of Artificial Societies and Social Simulation*, 12 (4) 9, 2009. URL http://jasss.soc.surrey.ac.uk/12/4/9.html. (last accessed 2014/01/06).

S. Heckbert, T. Baynes, and A. Reeson. Agent-Based Modeling in Ecological Economics. *Annals of the New York Academy of Sciences*, 1185:39–53, 2010.

R. Hemmerling, O. Kniemeyer, D. Lanwert, and W. KurW. Kurth. Buck-Sorlin. The Rule-Based Language XL and the Modelling Environment GroIMP Illustrated with Simulated Tree Competition. *Functional Plant Biology*, 35:739–750, 2008.

C. Hewitt. *Viewing Control Structures as Patterns of Passing Messages*. A.I.Memo 410. MIT Press, 1976.

D. Hiebeler. The Swarm Simulation System and Individual-Based Modeling. In J.M. Power, M. Strome, and T.C. Daniel, editors, *Decision Support 2001. 17th Annual Geographic Information Seminar and the Resource Technology '94 Symposium*, pages 474–494. American Society for Photogrammetry and Remote Sensing, 1994.

P. Hogeweg and B. Hesper. Individual-Oriented Modelling in Ecology. *Mathematical and Computer Modelling*, 13(6):83–90, 1990.

M. Huston, D. DeAngelis, and W. Post. New Computer Models Unify Ecological Theory. *BioScience*, 38:682–691, 1988.

M.A. Janssen, L.N. Alessa, M. Barton, S. Bergin, and A. Lee. Towards a Community Framework for Agent-Based Modelling. *Journal of Artificial Societies and Social Simulation*, 11 (2) 6, 2008. URL http://jasss.soc.surrey.ac.uk/11/2/6.html. (last accessed 2014/01/06).

N.R. Jennings. On Agent-Based Software Engineering. *Artificial Intelligence*, 117:277–296, 2000.

N.R. Jennings, K. Sycara, and M.J. Wooldridge. A Roadmap of Agent Research and Development. *Autonomous Agents and Multi-Agent Systems*, 1(1):7–38, 1998.

O.P. Judson. The Rise of Individual-Based Model in Ecology. *Trends in Ecology and Evolution*, 9:9–14, 1994.

H. Kaiser. Populationsdynamik und Eigenschaften einzelner Individuen. *Verhandlungen der Gesellschaft für Ökologie*, 4:25–38, 1974.

B. LeBaron. Agent-Based Computational Finance: Suggested Readings and Early Research. *Journal of Economic Dynamics and Control*, 24(5-7):679–702, 2000.

R. Leombruni and M. Richiardi. *Industry and Labor Dynamics: The Agent-based Computational Economics Approach: Proceedings of the Wild@ace2003 Workshop, Torino, Italy, 3-4 October 2003*. World Scientific, 2004.

X. Li, W. Mao, D. Zeng, and F.-Y. Wang. Agent-Based Social Simulation and Modeling in Social Computing. In C.C. Yang, H. Chen, M. Chau, K. Chang, S.-D. Lang, P.S. Chen, R. Hsieh, D. Zeng, F.-Y. Wang, K.M. Carley, W. Mao, and J. Zhan, editors, *Intelligence and Security Informatics, IEEE ISI 2008 International Workshops: PAISI, PACCF, and SOCO 2008, Taipei, Taiwan, June 17, 2008. Proceedings*, Lecture Notes in Computer Science, pages 401–412. Springer, 2008.

J. Liu and P.S. Ashton. Individual-Based Simulation Models for Forest Succession and Management. *Forest Ecology and Management*, 73:157–175, 1995.

Logo Foundation. The Logo Programming Language, 2010. URL http://el.media.mit.edu/logofoundation/logo/programming.html. (last accessed 2014/01/06).

A. Lomnicki. Individual Differences Between Animals and the Natural Regulation of Their Numbers. *Journal of Animal Ecology*, 47:461–475, 1978.

A. Lomnicki. *Population Ecology of Individuals*. Princeton University Press, Princeton, 1988.

A. Lomnicki. Individual-Based Models and the Individual-Based Approach to Population Ecology. *Ecological Modelling*, 115:191–198, 1999.

H. Lorek and M. Sonnenschein. Modelling and Simulation Software to Support Individual-Based Ecological Modelling. *Ecological Modelling*, 115:199–216, 1999.

M. Luck, P. McBurney, and C. Preist. *Agent Technology: Enabling Next Generation Computing (A Roadmap for Agent Based Computing)*. AgentLink, Southampton: University of Southampton, 2003.

S. Luke, C. Ciofii-Revilla, L. Panait, and K. Sullivan. MASON: A New Multiagent Simulation Toolkit. In *Proceedings of the 2004 SwarmFest Workshop*, 2004. URL http://cs.gmu.edu/

~eclab/projects/mason/publications/SwarmFest04.pdf. (last accessed 2014/01/06).

S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, and G. Balan. MASON: A Multi-Agent Simulation Environment. *Simulation*, 82:517–527, 2005.

M.W. Macy and R. Willer. From Factors to Actors: Computational Sociology and Agent-Based Modeling. *Annual Review of Sociology*, 28:143–166, 2002.

N. Malleson, A. Heppenstall, and L. See. Crime Reduction Through Simulation: An Agent-Based Model of Burglary. *Computers, Environment and Urban Systems*, 34(3):236–250, 2010.

Mason. MASON Class Overview, 2010. URL http://cs.gmu.edu/~eclab/projects/mason/docs/overview.html. (last accessed 2014/01/06).

R.B. Matthews, N.G. Gilbert, A. Roach, J.G. Polhill, and N.M. Gotts. Agent-Based Land-Use Models: A Review of Applications. *Landscape Ecology*, 22(10):1447–1459, 2007.

D. Meadows. *The Limits to Growth: A Report for the Club of Rome's Project on the Predicament of Mankind*. Universe Books, New York, 2nd edition, 1974.

N. Minar, R. Burkhart, C. Langton, and M. Askenazi. The Swarm Simulation System: A Toolkit for Building Multi-Agent Simulations. Working Paper 96-06-042, Santa Fe Institute, Santa Fe, 1996.

J.M. Moonen. *Multi-Agent Systems for Transportation Planning and Coordination*. ERIM Ph.D. Series Research in Management. Erasmus Research Institute of Management (ERIM), Erasmus University Rotterdam, 2009.

J.H. Myers. Distribution and Dispersal in Populations Capable of Resource Depletion: A Simulation Model. *Oecologia*, 23(4):255–269, 1976.

NetLogo. What's New?, 2010a. URL http://ccl.northwestern.edu/netlogo/docs/versions.html. (last accessed 2014/01/06).

NetLogo. Copyright and License Information, 2010b. URL http://ccl.northwestern.edu/netlogo/docs/copyright.html. (last accessed 2014/01/06).

NetLogo. FAQ (Frequently Asked Questions), 2010c. URL http://ccl.northwestern.edu/netlogo/faq.html. (last accessed 2014/01/06).

C. Nikolai and G. Madey. Tools of the Trade: A Survey of Various Agent Based Modeling Platforms. *Journal of Artificial Societies and Social Simulation*, 12 (2) 2, 2009. URL http://jasss.soc.surrey.ac.uk/12/2/2.html. (last accessed 2014/01/06).

M.J. North, C.M. Macal, J.S. Aubin, P. Thimmapuram, M.J. Bragen, J. Hahn, J. Karr, N. Brigham, M.E. Lacy, and D. Hampton. Multiscale Agent-Based Consumer Market Modeling. *Complexity*, 15(5):37–47, 2010.

M. Oeffner. *Agent-Based Keynesian Macroeconomics - An Evolutionary Model Embedded in an Agent-Based Computer Simulation*. PhD thesis, Universität Wüzburg, 2008.

E. Oliveira. Applications of Intelligent Agent-Based Systems. In *Proceedings of SBAI - Simpósium Brasileiro de Automação Inteligente. 1999: São Paulo*, pages 51–58, 1999.

D. Phan. From Agent-Based Computational Economics towards Cognitive Economics. In P. Bourgine and J.P. Nadal, editors, *Cognitive Economics: An Interdisciplinary Approach*,

pages 371–398. Springer, 2004.

Pocco. Pygments Syntax Highlighter, 2010. URL http://www.pocoo.org/projects/pygments/#pygments. (last accessed 2014/01/06).

J.G. Polhill. Agent-Based Modeling of Socio-Economic Processes Related to the Environment: Example of Land-Use Change. In P.C. Baveye, M. Laba, and J. Mysiak, editors, *Uncertainties in Environmental Modelling and Consequences for Policy Making*, NATO Science for Peace and Security Series C: Environmental Security, pages 61–76. Springer, 2009.

J.G. Polhill. ODD Updated. *Journal of Artificial Societies and Social Simulation*, 11 (2) 3, 2010. URL http://jasss.soc.surrey.ac.uk/13/4/9.html. (last accessed 2014/01/06).

L.S. Premo. Exploratory Agent-based Models: Towards an Experimental Ethnoarchaeology. In J.T. Clark and E.M. Hagemeister, editors, *Digital Discovery: Exploring New Frontiers in Human Heritage: CAA 2006: Computer Applications and Quantitative Methods in Archaeology*, pages 29–36. Archaeolingua, 2007.

L.S. Premo and S.L. Kuhn. Modeling Effects of Local Extinctions on Culture Change and Diversity in the Paleolithic. *PLoS ONE*, 5(12):1–10, 12 2010.

H. Pretzsch. *Forest Dynamics, Growth and Yield: From Measurement to Model*. Springer, Berlin etc., 2009.

C. Rademacher, C. Neuert, V. Grundmann, C. Wissel, and V. Grimm. Reconstructing Spatiotemporal Dynamics of Central European Beech Forests: The Rulebased Model BEFORE. *Forest Ecology and Management*, 194:349–368, 2004.

S.F. Railsback and V. Grimm. *Agent-Based and Individual-Based Modeling: A Practical Introduction*. Princeton University Press, 2012.

S.F. Railsback, S.L. Lytinen, and S.K. Jackson. Agent-Based Simulation Platforms: Review and Development Recommendations. *Simulation*, 82:609–623, 2006.

M.J. Reed, L.S. Mills, J.B. Dunning, E.S. Menges, K.S. McKelvey, R. Freye, S.R. Beissinger, M.C. Anstett, and P. Miller. Emerging Issues in Population Viability Analysis. *Conservation Biology*, 16:7–19, 2002.

H. Reuter. Multidimensional Biotic Community Interaction of Small Rodents: Assessment Through Object Oriented Modelling. *ASU Newsletter*, 24 Suppl.:27–40, 1998.

M. Richiardi. Agent-Based Computational Economics. A Short Introduction. LABORatorio R. Revelli Working Papers Series 70, LABORatorio R. Revelli, Centre for Employment Studies, 2007.

M. Richiardi, R. Leombruni, N.J. Saam, and M. Sonnessa. A Common Protocol for Agent-Based Social Simulation. *Journal of Artificial Societies and Social Simulation*, 9 (1) 15, 2006. URL http://jasss.soc.surrey.ac.uk/9/1/15.html. (last accessed 2014/01/06).

Road. Repast Files, 2010a. URL http://sourceforge.net/projects/repast/files/Repast/. (last accessed 2014/01/06).

Road. Repast Simphony, 2010b. URL http://repast.sourceforge.net/repast_simphony.html. (last accessed 2014/01/06).

Road. Repast for High Performance Computing, 2010c. URL http://repast.sourceforge.net/

repast_hpc.html. (last accessed 2014/01/06).

T. Schelling. Dynamic Models of Segregation. *Journal of Mathematical Sociology*, 1:143–186, 1971.

T.C. Schelling. Models of Segregation. *The American Economic Review*, 59(2):488–493, 1969.

W. Shen, Q. Hao, H.J. Yoon, and D.H. Norrie. Applications of Agent-Based Systems in Intelligent Manufacturing: An Updated Review. *Advanced Engineering Informatics*, 20(4): 415–431, 2006.

C. Simon and M. Etienne. A Companion Modelling Approach Applied to forest Management Planning with the Société Civile des Terres du Larzac. *Environmental Modelling and Software*, 25:1371–1384, 2009.

A. Smith. *An Inquiry into the Nature and Causes of the Wealth of Nations*. Printed For W. Strahan And T. Cadell, London, 1776.

F. Sondahl, S. Tisue, and U. Wilensky. Breeding Faster Turtles: Progress Towards a NetLogo Compiler. In *Paper Presented at Agent 2006*, Chicago, IL., 2006.

M. Spielauer. Dynamic Microsimulation of Health Care Demand, Health Care Finance and the Economic Impact of Health Behaviours: Survey and Review. *International Journal of Microsimulation*, 1:35–53, 2007.

F. Squazzoni. The Impact of Agent-Based Models in the Social Sciences After 15 Years of Incursions. *History of Economic Ideas*, XVIII/2010/2:197–233, 2010.

A.M. Starfield. A Pragmatic Approach to Modeling for Wildlife Management. *Journal of Wildlife Management*, 61:261–270, 1997.

A.M. Starfield, K.A. Smith, and A.L. Bleloch. *How to Model It: Problem Solving for the Computer Age*. McGraw-Hill, New York, 1990.

F. Stonedahl, D. Kornhauser, E. Russell, C. Brozefsky, E. Verreau, S. Tisue, and U. Wilensky. Tinkering With Turtles: An Overview of NetLogo's Extensions API. In *Paper Presented at the Annual Meeting of the Swarm Development Group*, Chicago, IL, 2008.

Swarm. Swarm Development Group, 2010a. URL https://web.archive.org/web/20121304003700/http://www.swarm.org/index.php/Talk:Swarm_Development_Group. (last accessed 2014/01/06).

Swarm. Documentation Set for Swarm 2.2, 2010b. URL https://web.archive.org/web/20120519043907/http://www.swarm.org/swarmdocs-2.2/set/set.html. (last accessed 2014/01/06).

K.P. Sycara. Multiagent Systems. *AI Magazine*, 19(2):79–92, 1998.

L. Tesfatsion. Agent-Based Computational Economics: A Constructive Approach to Economic Theory. In L. Tesfatsion and K.L. Judd, editors, *Handbook of Computational Economics*, volume 2, chapter 16, pages 831–880. Elsevier, 2006.

J.C. Thiele and V. Grimm. NetLogo Meets R: Linking Agent-Based Models with a Toolbox for Their Analysis. *Environmental Modelling & Software*, 25(8):972–974, 2010.

S. Tisue and U. Wilensky. NetLogo: Design and Implementation of a Multi-Agent Modeling

Environment. In *Presented at SwarmFest, May 9-11. 2004*, Ann Arbor, 2004a.

S. Tisue and U. Wilensky. NetLogo: A Simple Environment for Modeling Complexity. In *Paper Presented at the International Conference on Complex Systems, May 16 -21. 2004*, Boston, 2004b.

R. Tobias and C. Hofmann. Evaluation of Free Java-Libraries for Social-Scientific Agent Based Simulation. *Journal of Artificial Societies and Social Simulation*, 7 (1) 6, 2004. URL http://jasss.soc.surrey.ac.uk/7/1/6.html. (last accessed 2014/01/06).

J. Uchmanski and V. Grimm. Individual-Based Modelling in Ecology: What Makes the Difference? *Trends in Ecology and Evolution*, 11(10):437–441, 1996.

S. Urbanek. *Package 'rJava' Manual. R package version 0.9-6*, 2013. URL http://cran.r-project.org/web/packages/rJava/. (last accessed 2014/01/06).

J. Vos, L.F.M. Marcelis, and J.B. Evers. Functional-Structural Plant Modelling in Crop Production - Adding a Dimension. In J. Vos, L.F.M. Marcelis, P.H.B. de Visser, P.C. Struik, and J.B. Evers, editors, *Functional-Structural Plant Modelling in Crop Production*, pages 1–12. Springer, Dordrecht, 2007.

G. Weiss. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, Cambridge, MA, 1999.

Wikipedia. Comparison of Agent-Based Modeling Software, 2010. URL http://en.wikipedia.org/wiki/Comparison_of_agent-based_modeling_software. (last accessed 2014/01/06).

U. Wilensky. NetLogo. Center for Connected Learning and Computer-Based Modeling, 1999. URL http://ccl.northwestern.edu/netlogo. (last accessed 2014/01/06).

U. Wilensky and W. Rand. *An Introduction to Agent-Based Modeling: Modeling Natural, Social and Engineered Complex Systems with NetLogo*. MIT Press, 2014. Forthcomming.

P. Windrum, G. Fagiolo, and A. Moneta. Empirical Validation of Agent-Based Models: Alternatives and Prospects. *Journal of Artificial Societies and Social Simulation*, 10 (2) 8, 2007. URL http://jasss.soc.surrey.ac.uk/10/2/8.html. (last accessed 2014/01/06).

M. Wooldridge and N.R. Jennings. Intelligent Agents: Theory and Practice. *Knowledge Engineering Review*, 10:115–152, 1995.

M.J. Wooldridge. *An Introduction to Multiagent Systems*. John Wiley & Sons, New York, NY, 2005.

A. Zaidi and K. Rake. Dynamic Microsimulation Models: A Review and Some Lessons from SAGE. SAGE Discussion Paper 2, ESRC SAGE Research Group - The London School of Economics, 2001.

CHAPTER III

Linking NetLogo and R

III.1. Agent-Based Modelling: Tools for Linking NetLogo and R

This manuscript is published as: JC Thiele, W Kurth, and V Grimm [2012]. Agent-Based Modelling: Tools for Linking NetLogo and R. Journal of Artificial Societies and Social Simulation 15 (3) 8 <http://jasss.soc.surrey.ac.uk/15/3/8.html>.

Authorship

- Winfried Kurth supported the writing of the manuscript.
- Volker Grimm supported the writing of the manuscript.

III.1.1. Abstract

A seamless integration of software platforms for implementing agent-based models and for analysing their output would facilitate comprehensive model analyses and thereby make agent-based modelling more useful. Here we report on recently developed tools for linking two widely used software platforms: NetLogo for implementing agent-based models, and R for the statistical analysis and design of experiments. Embedding R into NetLogo allows the use of advanced statistical analyses, specific statistical distributions, and advanced tools for visualization from within NetLogo programs. Embedding NetLogo into R makes it possible to design simulation experiments and all settings for analysing model output from the outset, using R, and then embed NetLogo programs in this virtual laboratory. Our linking tools have the potential to significantly advance research based on agent-based modelling.

III.1.2. Introduction

Agent-based models (ABMs) are simulation models that explicitly represent individual agents, which can be humans, institutions, or organisms with their traits and behaviour [Grimm and Railsback, 2005, Gilbert, 2007, Squazzoni, 2012]. They are an established and increasingly used tool in a wide range of research fields including social sciences, economics, ecology and evolution [Thiele et al., 2011].

To learn as much as possible from ABMs, it would be desirable if they were routinely analysed in a comprehensive and structured way. However, in the ABM literature such analyses are the exception rather than the rule. One reason seems to be that software platforms for implementing ABMs and for statistical analysis are separated, so that thorough model analysis requires the cumbersome transfer of data via file output and input. Thorough model analysis could be simplified if agent-based simulation platforms were embedded into statistical analysis tools. A seamless integration of software for implementation and analysis would support, for example, the design of simulation experiments, the systematic storage of simulation results, and the use of advanced statistics for analysing model outputs. Furthermore, it is sometimes useful to use the functions provided by statistical software directly within an ABM implementation, for example, specific random distributions, or advanced graphical output.

Such a seamless link already exists between NetLogo [Wilensky, 1999] and Mathematica [Wolfram Research Inc., 2013]: the NetLogo-Mathematica link [Bakshy and Wilensky, 2007]. It would, however, also be desirable to link NetLogo with open source statistical software. NetLogo, which was designed for implementing ABMs, has become an established and widely used free software platform and language. It has a flat learning curve, includes powerful software concepts, and is on the way to becoming a standard tool in ABM development and prototyping [Railsback and Grimm, 2012]. Page et al. [2012] show that the use of NetLogo has increased dramatically in recent years and it was the most frequently used ABM platform in 2009.

We scanned agent-based modelling studies published between January 2010 and January 2012 in JASSS regarding the language or software platform used for implementation (Figure III.1). In nearly one-third of the ABM studies where information about the language/toolkit was given, NetLogo was used. Moreover, at the end of 2011, the NetLogo source code was opened to the public under GPL license [Wilensky, 2011], increasing its relevance as a scientific tool [Greve, 2003].

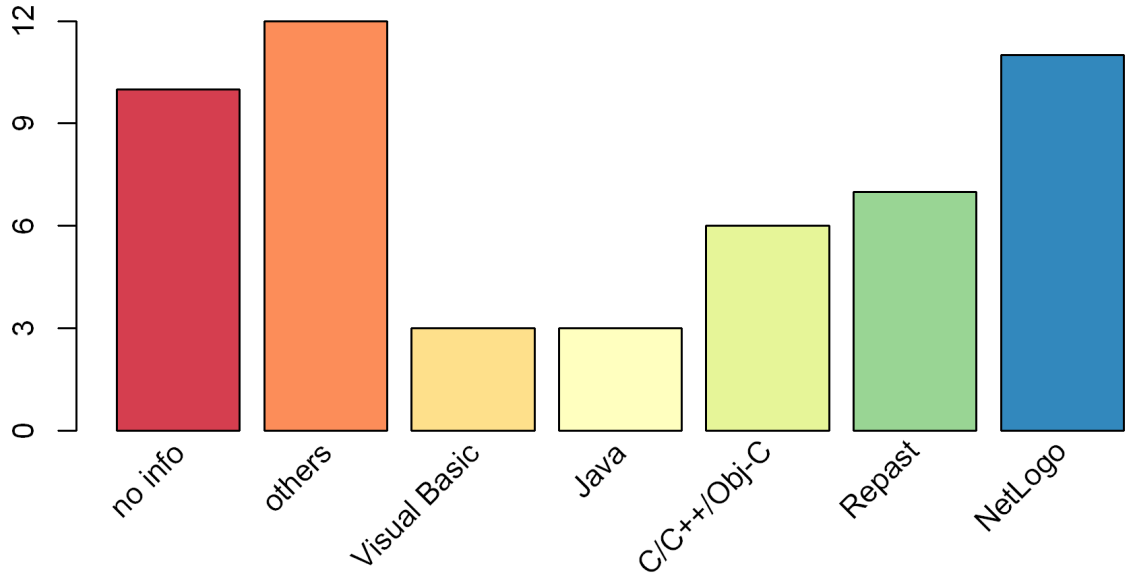


Figure III.1.: Review of agent-based simulation studies published in JASSS between January 2010 and January 2012 regarding the language/software platform used for the model implementation. Languages/platforms used less than three times are summarized into "others", which are namely Delphi, Python, LISP, AnyLogic, Fortran, JAS, PS-I, LEADSTO/TTL, Blanche and Cormas. "No info" means that no information about the language/platform used was given in the article.

On the statistics side, R [R Core Team, 2013a] is already the standard open-source software for scientific statistical analysis as indicated, for example, by the large number of textbooks. There are currently more than 30 textbooks available on R [R Core Team, 2013b], e.g., Crawley [2005]; Dalgaard [2008]; Zuur et al. [2009]. Furthermore, the R language ranked 19th in the TIOBE Programming Community Index for the year 2011 and competes with general purpose languages [Smith, 2012]. In doing so, it outpaced SAS, S, S-Plus, and Matlab [Smith, 2011]. Due to its extensibility a huge number of packages exist which extend the basic functionality of R or connect R to other software. Examples are the gam package [Hastie, 2013] for fitting generalized additive models, the sna package [Butts, 2010] for analysing social networks, or the survival package [Therneau, 2013] for survival analysis. There are several so-called CRAN Task Views, where lists of available packages addressing specific topics can be found, like "Statistics for the Social Sciences" [Fox, 2013] or "Computational Econometrics" [Zeileis, 2014]. Two other Task Views of potential interest in the context of social simulation are "Psychometric Models and Methods" [Mair and Hatzinger, 2013] and "Empirical Finance" [Eddelbuettel, 2013].

The purpose of this communication is to make agent-based modellers in the social sciences aware of recently developed tools that allow them to link NetLogo with R. Two of these have been described in more detail elsewhere [Thiele and Grimm, 2010, Thiele et al., 2012], but the Rserve-Extension is new.

III.1.3. Embedding R in NetLogo

The R-Extension [Thiele and Grimm, 2010] and the Rserve-Extension of NetLogo have been developed to make the functionality of R available in NetLogo. Both extensions make it possible to send NetLogo variables to R and to get results from R back to NetLogo. They include functions (called primitives/reporters in NetLogo language) for sending variable values of agents to R, which are then transformed to appropriate R data structures. Potential uses of these extensions of NetLogo are advanced plots provided by R, the calculation of home ranges in ecological models, spatial statistics, network analysis, and the usage of specific random distributions.

The difference between the R- and the Rserve-Extension is the underlying technique for communicating with R. As shown in Figure III.2, the R-Extension uses a direct path via the R package rJava [Urbanek, 2013a] whereas the Rserve-Extension communicates via a network connection with an Rserve server (for details about Rserve see Urbanek, 2013b). Both extensions share the same syntax, but the Rserve-Extension does not offer the possibility to attach the underlying R session in an interactive R editor/console to NetLogo (called interactiveShell). The Rserve-Extension makes it possible to connect not only to local servers but also to remote servers. This means that multiple users can share the same R installation via a network connection and, for example, some basic data and custom functions can be supplied via a central Rserve remote server. This functionality may be of interest for class rooms or summer schools where the R-Extension is not to be configured on every computer or where a team is working with large, centrally maintained datasets in R. Furthermore, Rserve can be used simultaneously with the RNetLogo package (described in the next section).

After a successful installation, as described in the documentation, the R- and Rserve-Extension can be included into a NetLogo model by pasting `extensions [r]` and `extensions [rserve]`, respectively, at the top of the Procedures Tab. For the Rserve-Extension one has to connect to an Rserve server with the rserve:init primitive, first.

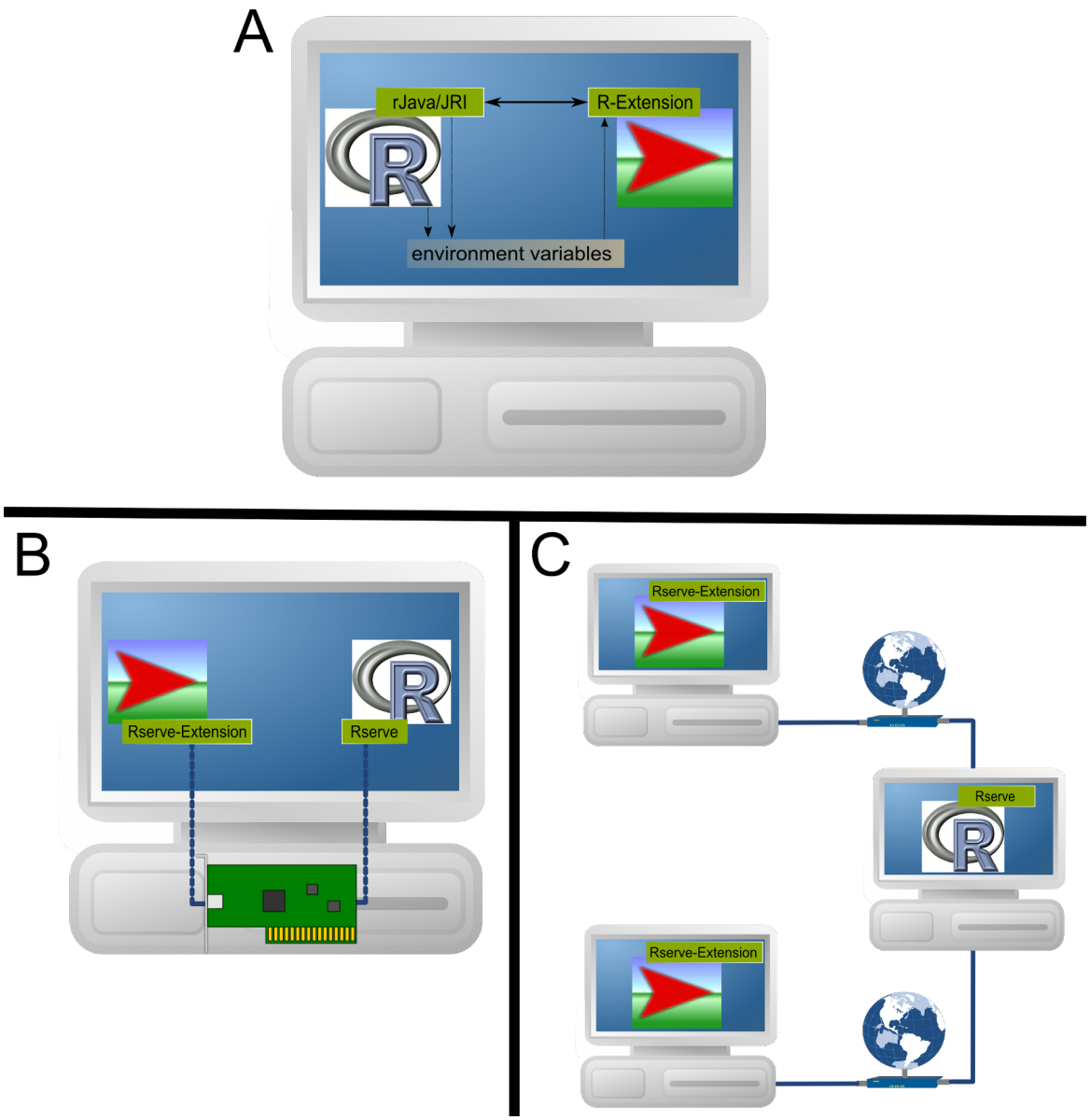


Figure III.2.: Communication of NetLogo's R- and Rserve-Extension with R. For the R-Extension, NetLogo uses the extension to communicate directly with R's rJava package (A). The R-Extension locates R and the rJava/JRI package by using environment variables of the operating system. For the Rserve-Extension, NetLogo uses a local network connection to communicate to an Rserve server running locally on the same machine (B) or one or multiple NetLogo clients use the Rserve-Extension to connect to a remote Rserve server via a network connection (C).

After this, it is possible to send NetLogo variables to R using primitives like `r:put`, `r:putdataframe` or `r:putagent` (for the Rserve-Extension just replace the `r:` part in the primitives by `rserve:`). Assuming a NetLogo model contains two lists, `mylist1` and `mylist2`, with the same number of entries, a call of `(r:putdataframe "df1" "v1" mylist1 "v2" mylist2)` would create an R data.frame with the name df1 and two columns `v1` and `v2`. The values of the columns would come from the values of the two NetLogo lists. The same would be possible with agent variables. Assuming the NetLogo turtles have two turtle-own variables `v1` and `v2`, one could create a data.frame with the same structure as above by executing `(r:putagentdf "df1" turtles "v1" "v2")`.

To execute an R function there is the `r:eval` primitive available. To get a visual impression of the above created data.frame one could create a boxplot in R by executing `r:eval "boxplot(df1)"`. Furthermore, calculating a Spearman's correlation coefficient on the data.fame is possible by executing `r:eval "cor <- corr.test(df1$v1, df1$v2, method=\"spearman\")"` in NetLogo.

To get values/variables from R into NetLogo there is the `r:get` reporter available. For example, to use the result of the correlation analysis in NetLogo just execute `r:get "cor$estimate"` to receive the correlation coefficient and `r:get "cor$p.value"` to get the corresponding p-value.

It is also possible to get the result of an R function directly into NetLogo. For example, a NetLogo list with ten random values following a Weibull distribution can be processed by simply executing `r:get "rweibull(10, shape=1)"`. This calls the rweibull function of R and sends the result to NetLogo.

Some examples of use are included in the examples folder of the extensions and one example is visualized in Figure III.3. The extensions are available for download at sourceforge: http://r-ext.sourceforge.net and http://rserve-ext.sourceforge.net, respectively.

III.1.4. Embedding NetLogo in R

As experienced modellers know, it is much more time consuming and complicated to analyse ABMs than to formulate and implement them. It therefore makes sense to use model analysis software as the primary working basis for simulations. The RNetLogo package [Thiele et al., 2012] for R makes it possible to control and analyse NetLogo simulations from R. NetLogo can be started in the so-called GUI mode or in the headless mode. The first option opens the NetLogo Graphical User Interface (GUI). In this mode, the modeller can control simulations from R as well as from the NetLogo GUI. In the headless mode, NetLogo runs in the background without a GUI. In this case, it is possible to start several NetLogo sessions in one R session.

The RNetLogo package has functions for loading models, sending commands from R to NetLogo and reporting NetLogo variables to R. While using R, parameter values of the NetLogo program can be set, primitives, procedures and reporters can be executed, and values of agents can be assigned to R variables. Agents can be created and even NetLogo model source code can be extended and changed from within R.

Potential uses include the exploration of models (Figure III.4), the comparison of analytical models with ABM implementations, simultaneous visualizations of different state variables, and self-documentation and reporting of simulation experiments using tools like Sweave [Leisch, 2002], odfWeave [Kuhn et al., 2012] or SWord [Baier, 2009]. Furthermore, with the RNetLogo package and tools like RExcel [Heidberger and Neuwirth, 2009], NetL-

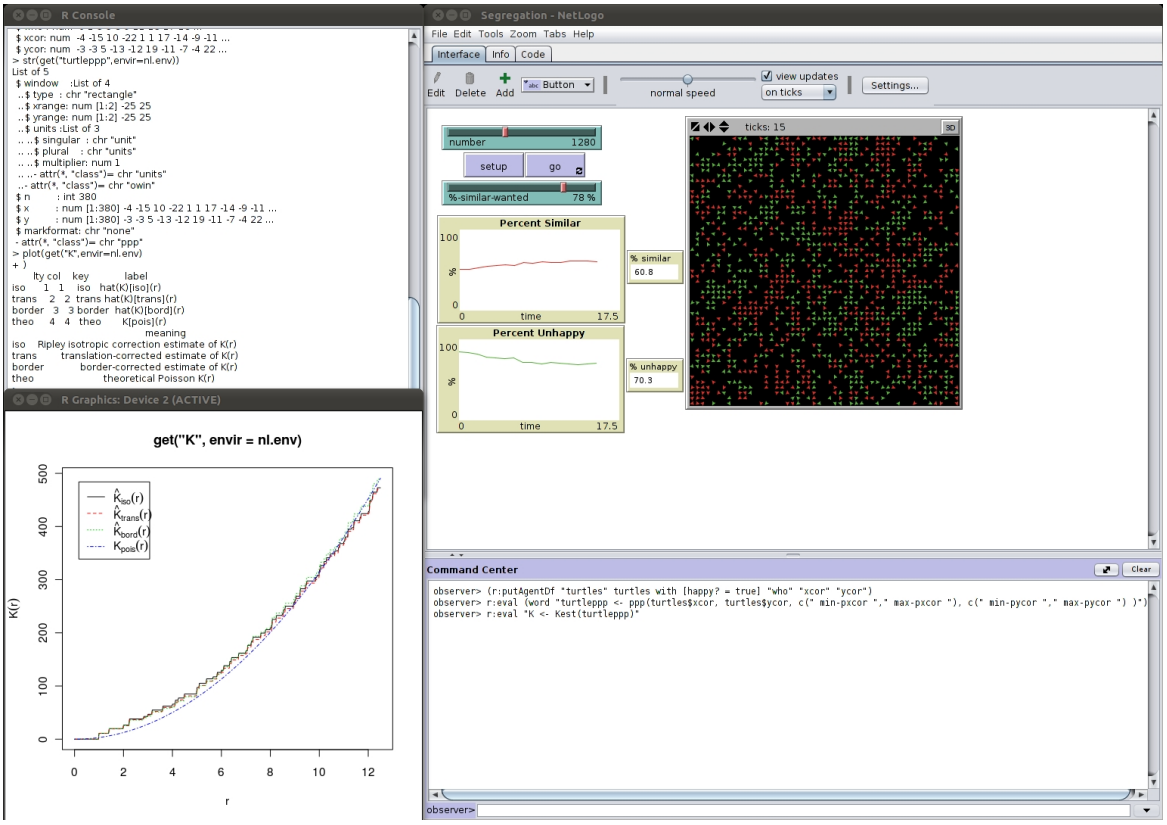


Figure III.3.: Example application of the R-Extension to calculate Ripley's L (from package spatstat; Baddeley and Turner, 2013) for analysing the spatial distribution of happy people based on the Segregation model [Wilensky, 1997b] from NetLogo's Model Library. The upper left window shows the interactiveShell editor (an R console for using/accessing the underlying R session opened directly from NetLogo; available since R-Extension version 1.0beta). The lower left window is the R plot showing Ripley's L-function, and on the right window is the NetLogo interface.

ogo simulations can be embedded into spreadsheets. RNetLogo is available for download at RForge and CRAN.

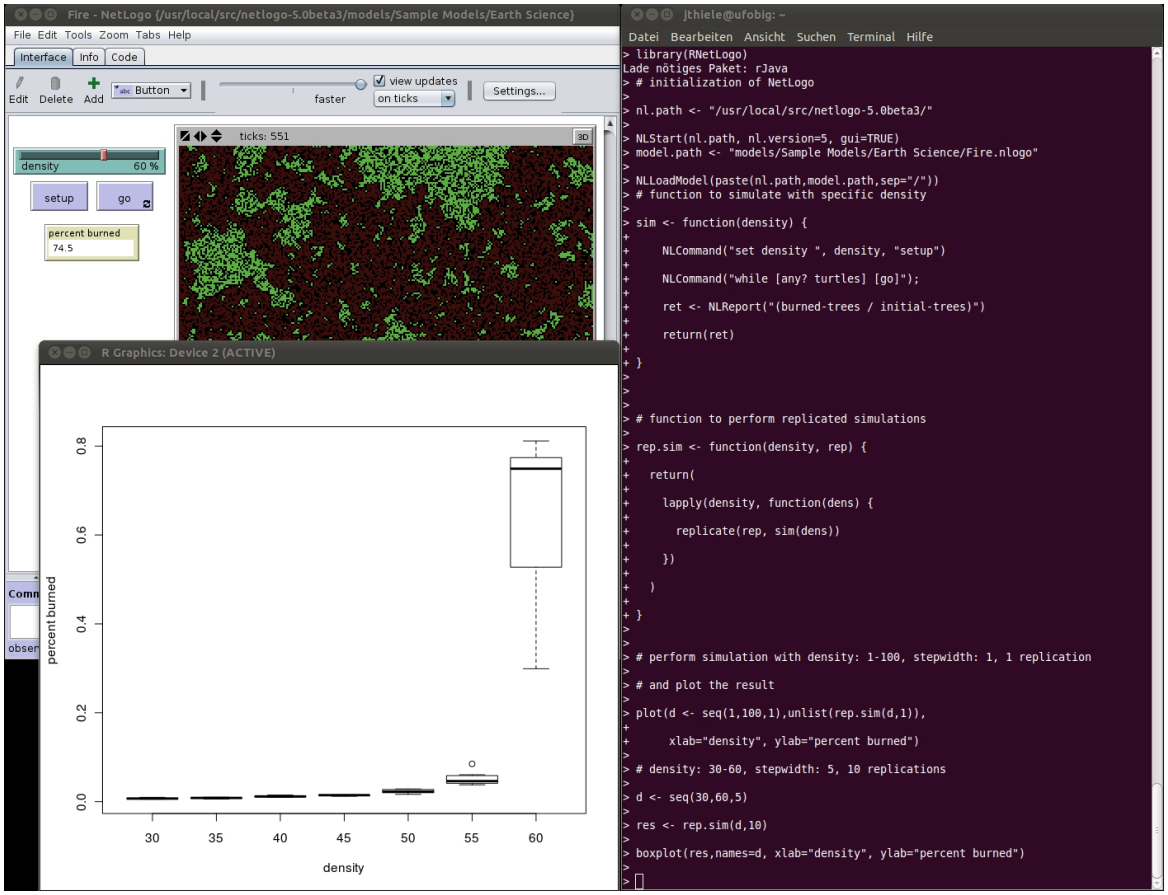


Figure III.4.: Example application of the RNetLogo package for model exploration, here of the Fire model [Wilensky, 1997a] from NetLogo's Model Library (full example can be found in the tutorial of the RNetLogo package). On the right hand side is the basic R shell, on the upper left the NetLogo instance controlled by the R shell, and on the lower left the R plot window with the aggregated output of multiple runs for model exploration.

III.1.5. Conclusions

Agent-based models usually include a large number of entities, processes, variables, and parameters quantifying relations between state variables. Therefore, in contrast to simple mathematical models, simulation experiments are required to test the model's implementation, to compare model output to data, patterns, and stylized facts [Meyer, 2011], and to understand how model behaviour emerges. Such experiments fully correspond to real experiments in empirical research: they need to be carefully designed and controlled, and their output needs to be analysed thoroughly [Lorscheid et al., 2012].

Agent-based modelling has not yet adopted the professional attitude of experimenters. Often, simulation experiments are designed ad hoc, are not comprehensive, and are not

well communicated [Schmolke et al., 2010]. A change in this situation will be an indicator of the maturation of agent-based modelling as a scientific tool. To foster this development, the next generation of modellers will need to be better trained in model analysis [Railsback and Grimm, 2012, Squazzoni, 2012]. And, we need software tools that allow agent-based modellers to make direct use of the vast amount of software available for model analysis. This is the purpose of the tools we presented here.

III.1.6. Acknowledgements

We thank an anonymous reviewer for valuable comments on an earlier version of the paper.

III.1.7. References

A. Baddeley and R. Turner. *Package 'spatstat' Manual. R package version 1.35-0*, 2013. URL http://cran.r-project.org/web/packages/spatstat/. (last accessed 2014/01/06).

T Baier. *SWordInstaller: SWord: Use R in Microsoft Word (Installer). R package version 1.0-2*, 2009. URL http://cran.r-project.org/src/contrib/Archive/SWordInstaller/. (last accessed 2014/01/06).

E. Bakshy and U. Wilensky. Turtle Histories and Alternate Universes: Exploratory Modeling with NetLogo and Mathematica. In M.J. North, c.M. Macal, and D.L. Sallach, editors, *Proceedings of the Agent 2007 Conference on Complex Interaction and Social Emergence*, pages 147–158. IL: Argonne National Laboratory and Northwestern University, 2007.

C.T. Butts. *Package 'sna' Manual. R package version 2.3-1*, 2010. URL http://cran.r-project.org/web/packages/sna/. (last accessed 2014/01/06).

M.J. Crawley. *Statistics: An Introduction Using R*. John Wiley & Sons, 2005.

P. Dalgaard. *Introductory Statistics with R*. Springer, New York, 2nd edition, 2008.

D. Eddelbuettel. *CRAN Task View: Empirical Finance. Version 2013-12-20*, 2013. URL http://cran.fyxm.net/web/views/Finance.html. (last accessed 2014/01/06).

J. Fox. *CRAN Task View: Statistics for the Social Sciences. Version 2013-12-10*, 2013. URL http://cran.r-project.org/web/views/SocialSciences.html. (last accessed 2014/01/06).

N. Gilbert. *Agent-Based Models*. Quantitative Applications in the Social Sciences. Sage, Los Angeles, CA, 2007.

G.C.F. Greve. Brave GNU World. *Linux Magazine*, 12:89–91, 2003. URL http://www.linux-magazine.com/w3/issue/37/Brave_GNU_World.pdf. (last accessed 2014/08/07).

V. Grimm and S.F. Railsback. *Individual-Based Modeling and Ecology*. Princeton University Press, Princeton, N.J., 2005.

T. Hastie. *Package 'gam' Manual. R package version 1.09*, 2013. URL http://cran.r-project.org/web/packages/gam/. (last accessed 2014/01/06).

R.M. Heidberger and E. Neuwirth. *R Through Excel: A Spreadsheet Interface for Statistics, Data Analysis, and Graphics*. Springer, New York, 2009.

M. Kuhn, S. Weston, N. Coulter, P. Lenon, and Z. Otles. *odfWeave: Sweave Processing of Open*

Document Format (ODF) Files. R package version 0.8.2, 2012. URL http://CRAN.R-project.org/package=odfWeave. (last accessed 2014/01/06).

F. Leisch. Sweave: Dynamic Generation of Statistical Reports Using Literate Data Analysis. In W. Härdle and B. Rönz, editors, *Compstat 2002 - Proceedings in Computational Statistics*, pages 575–580. Physica, 2002.

I. Lorscheid, B.-O. Heine, and M. Meyer. Opening the 'Black Box' of Simulations: Increased Transparency and Effective Communication Through the Systematic Design of Experiments. *Computational & Mathematical Organization Theory*, 18(1):22–62, 2012.

P. Mair and R. Hatzinger. *CRAN Task View: Psychometric Models and Methods. Version 2013-12-01*, 2013. URL http://cran.r-project.org/web/views/Psychometrics.html. (last accessed 2014/01/06).

M. Meyer. Bibliometrics, Stylized Facts and the Way Ahead: How to Build Good Social Simulation Models of Science? *Journal of Artificial Societies and Social Simulation*, 14(4) 4, 2011. URL http://jasss.soc.surrey.ac.uk/14/4/4.html. (last accessed 2014/01/06).

C. Le Page, N. Becu, P. Bommel, and F. Bousquet. Participatory Agent-Based Simulation for Renewable Resource Management: The Role of the Cormas Simulation Platform to Nurture a Community of Practice. *Journal of Artificial Societies and Social Simulation*, 15(1) 10, 2012. URL http://jasss.soc.surrey.ac.uk/15/1/10.html. (last accessed 2014/01/06).

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, 2013a. URL http://www.r-project.org/. (last accessed 2014/01/06).

R Core Team. Book Related To R, 2013b. URL http://www.r-project.org/doc/bib/R-books.html. (last accessed 2014/01/06).

S.F. Railsback and V. Grimm. *Agent-Based and Individual-Based Modeling: A Practical Introduction*. Princeton University Press, 2012.

A. Schmolke, P. Thorbek, D.L. DeAngelis, and V. Grimm. Ecological Modelling Supporting Environmental Decision Making: A Strategy for the Future. *Trends in Ecology and Evolution*, 25:479–486, 2010.

D. Smith. R Overtakes SAS and Matlab in Programming Language Popularity. Revolutions Blog, 2011. URL http://blog.revolutionanalytics.com/2011/02/r-overtakes-sas-and-matlab-in-programming-language-popularity.html. (last accessed 2014/01/06).

D. Smith. R Jumps from 25 to 19 in Annual TIOBE Rankings of Programming Language Popularity. Revolutions Blog, 2012. URL http://blog.revolutionanalytics.com/2012/01/r-jumps-from-25-to-19-in-tiobe-rankings.html. (last accessed 2014/01/06).

F. Squazzoni. *Agent-Based Computational Sociology*. John Wiley & Sons, 2012.

T. Therneau. *Package 'survival' Manual. R package version 2.37-4*, 2013. URL http://cran.r-project.org/web/packages/survival/. (last accessed 2014/01/06).

J.C. Thiele and V. Grimm. NetLogo Meets R: Linking Agent-Based Models with a Toolbox for Their Analysis. *Environmental Modelling & Software*, 25(8):972–974, 2010.

J.C. Thiele, W. Kurth, and V. Grimm. Agent- and Individual-Based Modeling with NetLogo: Introduction and New NetLogo Extensions. In K. Römisch, A. Nothdurft, and U. Wunn, editors, *22. Tagung der Sektion Forstliche Biometrie und Informatik des Deutschen Verbandes Forstlicher Forschungsanstalten und der Arbeitsgemeinschaft Ökologie und Umwelt der Internationalen Biometrischen Gesellschaft - Deutsche Region, 20-21th September 2010 in Göttingen (Germany)*, Die Grüne Reihe, pages 68–101, 2011.

J.C. Thiele, W. Kurth, and V. Grimm. RNetLogo: An R Package for Running and Exploring Individual-based Models Implemented in NetLogo. *Methods in Ecology and Evolution*, 3: 480–483, 2012.

S. Urbanek. *Package 'rJava' Manual. R package version 0.9-6*, 2013a. URL http://cran.r-project.org/web/packages/rJava/. (last accessed 2014/01/06).

S. Urbanek. *Package 'Rserve' Manual. R package version 1.7-3*, 2013b. URL http://cran.r-project.org/web/packages/Rserve/. (last accessed 2014/01/06).

U. Wilensky. NetLogo Fire Model, 1997a. URL http://ccl.northwestern.edu/netlogo/models/Fire. (last accessed 2014/01/06).

U. Wilensky. NetLogo Segregation Model, 1997b. URL http://ccl.northwestern.edu/netlogo/models/Segregation. (last accessed 2014/01/06).

U. Wilensky. NetLogo. Center for Connected Learning and Computer-Based Modeling, 1999. URL http://ccl.northwestern.edu/netlogo. (last accessed 2014/01/06).

U. Wilensky. Open Source, Message on the NetLogo Mailing List at October 27th, 2011, 2011. URL http://groups.yahoo.com/group/netlogo-users/message/13238. (last accessed 2014/01/06).

Wolfram Research Inc. *Mathematica, Version 9.0*, 2013. URL http://www.wolfram.com/mathematica/. (last accessed 2014/01/06).

A. Zeileis. *CRAN Task View: Computational Econometrics. Version 2014-01-01*, 2014. URL http://cran.r-project.org/web/views/Econometrics.html. (last accessed 2014/01/06).

A.F. Zuur, E.N. Ieno, and E. Meesters. *A Beginner's Guide to R*. Use R. Springer, 2009.

III.2. NetLogo Meets R: Linking Agent-Based Models With a Toolbox for Their Analysis

This manuscript is published as: JC Thiele and V Grimm [2010]. NetLogo Meets R: Linking Agent-Based Models With a Toolbox for Their Analysis. Environmental Modelling & Software (25): 972-974.

Authorship

- Volker Grimm wrote the abstract, introduction and discussion of the manuscript and supported the writing of the other parts.

III.2.1. Abstract

NetLogo is a software platform for agent-based modelling that is increasingly used in ecological and environmental modelling. So far, for comprehensive analyses of agent-based models (ABMs) implemented in NetLogo, results needed to be written to files and evaluated by using external software, for example R. Ideally, however, it would be possible to call any R function from within a NetLogo program. This would allow sophisticated interactive statistical analysis of model structure and dynamics, using R functions and packages for generating certain statistical distributions and experimental design, and for implementing complex descriptive submodels within ABMs. Here we present an R extension of NetLogo. It consists of only nine new NetLogo primitives for sending data between NetLogo and R and for calling R functions (six additional primitives for debugging). We demonstrate the usage of the R extension with three short examples.

III.2.2. Introduction

Agent-based models (ABMs) have become an established tool in ecological and environmental modelling [Huse et al., 2002, Porté and Bartelink, 2002, Parker et al., 2003, Bousquet and Page, 2004, Grimm and Railsback, 2005]. In these models, individual agents, which can be organisms, humans, or institutions, and their behaviour are represented explicitly. ABMs are used when one or more of the following individual-level aspects are considered important for explaining system-level behaviour: heterogeneity among individuals [Uchmanski, 2000], local interactions, and adaptive behaviour which is based on decision making [Grimm, 2008]. Implementing and analysing ABMs can be a challenge because even simple ABMs can generate complex behaviours. For implementing ABMs, various software platforms have been developed [Railsback et al., 2006] which provide specific libraries or programming languages. One of these platforms, NetLogo [Wilensky, 1999], has become very popular in recent years. Originally being more designed for teaching, it is increasingly used for research. It is easy to learn, provides powerful concepts for implementing ABMs, and it has continuously been supported by its developers and a large and growing user community for more than ten years. NetLogo also provides tools for analysing ABMs. For interactive work, a suite of elements for a graphical user interface exists that allow changes of parameters and settings, visual inspection [Grimm, 2002] of the model world's structure and dynamics, and summary outputs. For batch simulations, the "BehaviorSpace" tool allows simulations to be run repeatedly for different parameter combinations and also allows individual- and system-level output to be written to files. These files are then further analysed with other software, for example, Excel or statistics software packages such as R [R Core Team, 2013]. Usually, ABM developers using NetLogo would develop and test their model and program using the interactive mode of analysis and only later, once the model is considered good enough to be analysed more thoroughly, would they run simulation experiments via batch mode and use, e.g., R to analyse the results of these experiments. However, only being able to fully use the statistics and analysis toolbox provided by toolboxes like R in the batch mode can limit model development, testing, and understanding. It would be desirable also to have access to the full toolbox in the interactive mode, so that the direct interaction with the model would include the option of calculating complex summary statistics or other tools for experimental design and analysis provided by R. For example, if we want to understand how model rules and parameters affect the spatial distribution of agents, it would be good to see immediately, without the detour via output files and using external programs, how point pattern statistics such as Ripleyś K respond. Here we present an extension of NetLogo that allows any R function (except functions with multi-line string return values) to be called directly from NetLogo programs (see Figure III.5 and Listing III.1). This extension is based on NetLogo's interface (Extension API) for user defined extensions programmed in Java.

Listing III.1: Sending coordinates of NetLogo agents (turtles) to R and plotting them.

```
extensions [r]
...
;; send turtles variables "who", "xcor" and "ycor" to a new R
   list with the name "agentlist"
(r:putagent "agentlist" turtles "who" "xcor" "ycor")
;; open the plot window of R
r:eval "Windows()"
```

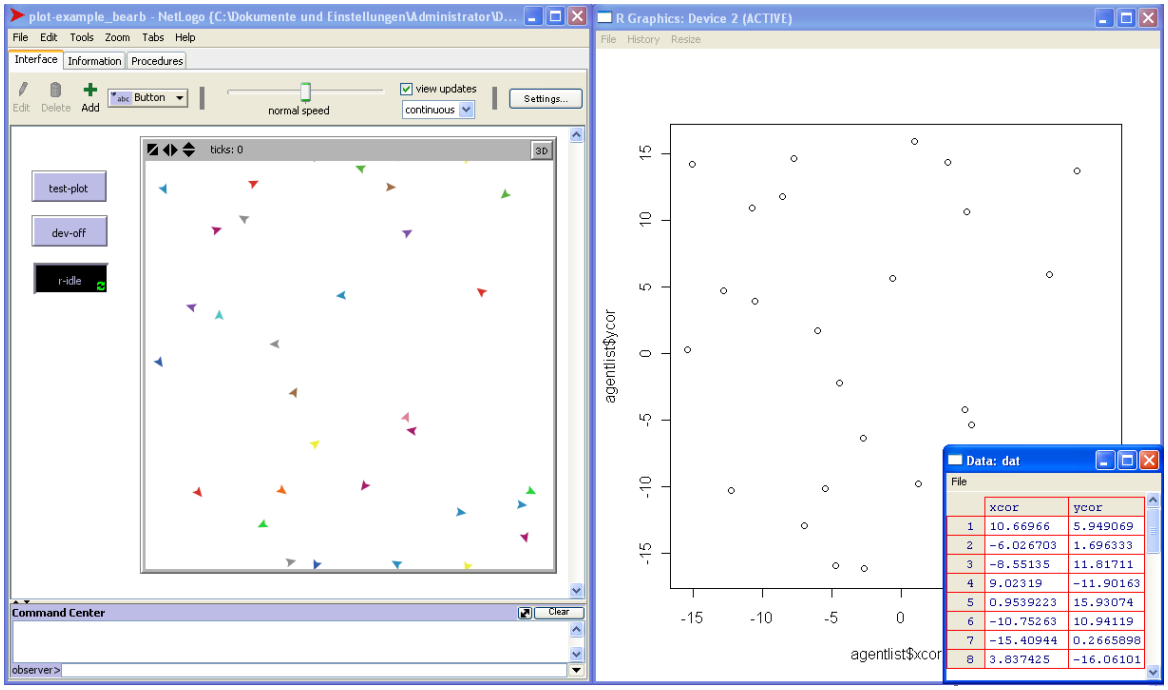


Figure III.5.: An example of the interaction between NetLogo and R. Coordinates of NetLogo agents (on the left hand side) were submitted to R and plotted there (on the right) using the new primitive "r:putagent".

```
;; plot the x- and y-positions of the turtles in the R window
r:eval "plot(agentlist$xcor, agentlist$ycor)"
...
```

III.2.3. New Primitives

NetLogo's programming language consists of a large number of commands, or "primitives". Our R extension adds only nine primitives (see documentation in Appendix C and Table III.1) and six additional primitives for debugging (see Table III.2). The new primitives provide means for sending data from NetLogo to R and vice versa, for evaluating any R command (with the exception mentioned above) and for observing the processes.

Table III.1.: Primitives (=NetLogo commands) of the R-Extension of NetLogo.

Primitive	Description	Example
`clear`	Clears the R workspace, deletes all variables.	`r:clear`

`eval`	Evaluates the submitted string in R, used for R function without a return value.	`;; create new vector x in R with a sequence from 1 to 10:`
		`r:eval "x <- seq(1,10)"`
		`;; load an R package:`
		`r:eval "library(spatstat)"`

`get`	Gets a value from R; either submits a string including an R function with return value, or gets the value of a variable; return values can be strings, numbers, booleans, lists, or a list of lists.	`;; get value of variable x:`
		`r:eval "x <- seq(1,10)"`
		`print r:get "x"`
		`;; save ten random values from binomial distribution in new local variable b:`
		`let b r:get "rbinom(10,1,0.5)"`
		`print b`

`put`	Creates a new variable in R with value(s) from NetLogo; submitted values/variables can be strings, numbers, booleans, or lists (NetLogo lists become R vectors, to create R lists see putlist).	`;; create an R variable r_var containing the value of NetLogo variable b:`
		`let b 12.95`
		`r:put "r_var" b`
		`;; create an R list r_turt containing the size of all turtles:`
		`r:put "r_turt" [size] of turtles`

`putagent`	Creates a list in R from variables of a set of NetLogo agents; collections of agent-variables can be stored in a single named list.	`;; create an R list r_patch containing the pxcor of patches:`
		`r:putagent "r_patch" patches "pxcor"`
		`;; create an R list r_turtles containing the values of the variables who, size and colour of the turtles (agentset) with a value of who less than five:`
		`(r:putagent "r_turtles" turtles with [who < 5] "who" "size" "color")`
		`print r:get "r_turtles"`
		`print r:get "r_turtles$who"`

`putagentdf`	Same as putagent, but creates an R data-frame instead of a list because many R functions requires data frames as input.	`;; create an R data-frame r_patch containing the pxcor and pycor of patches and show that r_patch is a data-frame:`
		`(r:putagentdf "r_patch" patches "pxcor" "pycor")`
		`print r:get "class(r_patch)"`

`putlist`	Creates a new R list based on the submitted NetLogo variables and values.	`;; create an R list r_list containing the values of NetLogo list b:`
		`let b [12.95 10 11.3]`
		`r:putlist "r_list" b`
		`;; create an R list r_list with three columns:`
		`(r:putlist "r_list" b [3 5] 4.5)`
		`print r:get "r_list[1]"`

`putnamedlist`	Same as putlist, but creates a named R list, i.e., the columns of the list can be called via their names.	`;; create a named R list r_list with three columns:`
		`(r:putnamedlist "r_list" "col1" b "col2" [3 5 2] "col3" 4.5)`
		`print r:get "r_list$col1"`

putdataframe	Same as putnamedlist, but creates an R data-frame instead of a list because many R functions require data frames as input.	`;; create an R data-frame r_df with 3 columns:`
		`(r:putdataframe "r_df" "col1" [4 2 5]`
		`"col2" [3 5 2] "col3" 4.5)`
		`print r:get "class(r_df)"`

Table III.2.: Additional primitives (=NetLogo commands) of the R-Extension of NetLogo for debugging.

Primitive	Description	Example
`interactiveShell`	Opens the underlying R shell, if NetLogo was started from a console/shell.	`;; start the R shell:` `r:interactiveShell`
`messageWindow`	Opens a message window to display debugging messages (see startDebug and startJRIDebug).	`;; open a message window:` `r:messageWindow`
`startDebug`	Starts the debugging mode of the R extension. The results of eval and get commands will be displayed in the Console, interactiveShell or Message Window.	`;; start the debugging mode:` `r:startDebug`
`startJRIDebug`	Starts the debugging mode of the underlying JRI library (of the rJava package). Displays the messages in the Console, interactiveShell or Message Window.	`;; start the debugging mode of the JRI library:` `r:startJRIDebug`
`stopDebug`	Stops showing debugging messages of the R extension in the Console, interactiveShell or Message Window.	`;; stop showing debugging messages:` `r:endDebug`
`stopJRIDebug`	Stops showing debugging messages of the underlying JRI library in the Console, interactiveShell or Message Window.	`;; stop showing debugging messages of the JRI library:` `r:endJRIDebug`

III.2.4. Examples

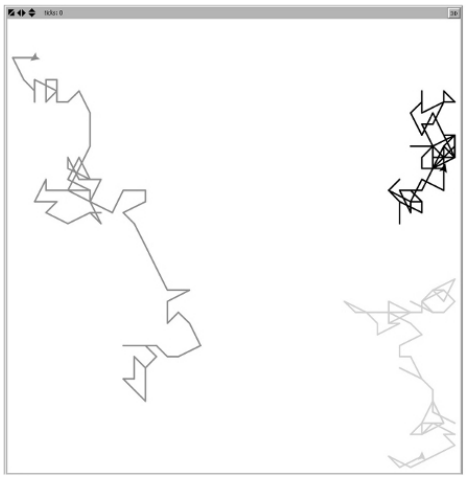
Three examples illustrate how our R-Extension of NetLogo can be used. The NetLogo program code in the listings contains only the parts where the R-Extension is used. The complete programs are provided in the examples folder of our R-Extension of NetLogo.

In the first example, (Listing III.2) the R-Extension is used in the `setup` procedure to get random values from a Beta and a Weibull distribution, which are not available in NetLogo. In the `go` procedure the correlation coefficient (Spearman's rho) between the turtle's weight and height variables is calculated and plotted.

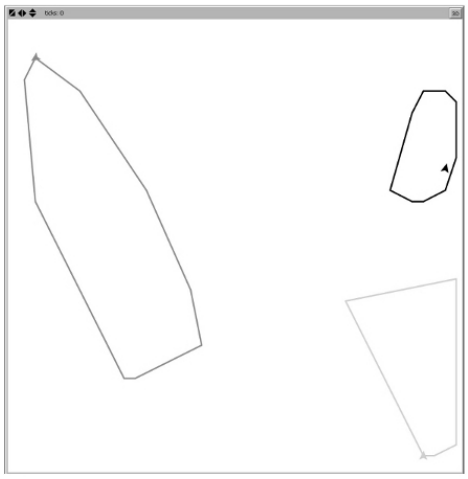
Listing III.2: Example 1: Random values and Spearman's rho (correlation).

```
extensions [r]

turtles-own [weight height]

to setup
  ...
  ;; create turtles with weight and height randomly chosen from
     Weibull and Beta distribution
  crt 40
  [
    set weight r:get "rweibull(1,1)"
    set height r:get "rbeta(1,1,1)"
  ]
end

to go
  tick
  ...
  ;; create R list from turtles
  (r:putagent "turtles" turtles "weight" "height")

  ;; calculate correlation between weight and height
  r:eval "c <- cor.test(turtles$weight,turtles$height, method =
     'spearm', alternative = 'g'"
  let rho r:get "c$p.value"
  let p r:get "c$estimate"
  ...
end
```

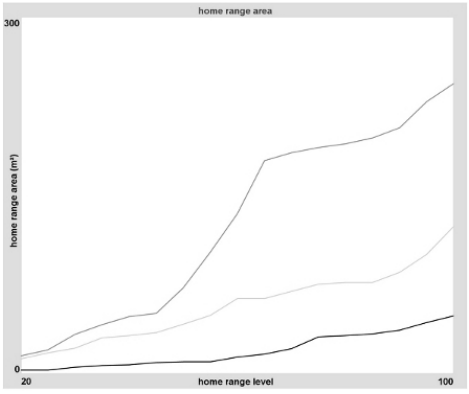
The second example makes use of the R package adehabitat [Calenge, 2006] (Listing III.3). Three animals move for 100 time steps and the points visited are stored in a list. These lists are transformed into an R data-frame, which is used for the home range analysis. The points of vertices of the home range polygons are sent back to NetLogo and plotted for visualization. In a second step, the adehabitat package is used to calculate the size of the home range area depending on different home range levels, i.e., the percentage of removed locations for the estimation of the home range polygon. The higher the level, the lower the removal percentage (see documentation of adehabitat package for details). The results are visualized in a NetLogo plot (Figure III.6).



(a)



(b)



(c)

Figure III.6.: Using R function `mcp` from package adehabitat to calculate home ranges from animal movement data generated by a NetLogo model. (a): movement path of three animals; (b): home ranges as determined by the R package adehabitat for home range level=95, i.e., 5% of the visited locations have been removed for the home range estimation; (c): area of the estimated home range polygons at different home range levels (the higher the level, the lower the percentage of removed locations farthest away from the barycenter of the home range; see documentation of adehabitat package for details).

Listing III.3: Example 2: Home Range Analysis with NetLogo and R.

```
extensions [r]
...
to calc-homerange
  ;; load R package adehabitat
  r:eval "library(adehabitat)"

  ;; create an empty data-frame
  r:eval "turtles <- data.frame()"

  ;; merge the Name-, X- and Y-lists of all animals to one
     data-frame
  ask animals
  [
    (r:putdataframe "turtle" "X" X "Y" Y)
    r:eval (word "turtle <- data.frame(turtle, Name = '" Name
       "')")
    r:eval "turtles <- rbind(turtles, turtle)"
  ]

  ;; split the data-frame into coordinates and factor variable
  r:eval "xy <- turtles[,c('X','Y')]"
  r:eval "id <- turtles$Name"

  ;; calculate home range
  r:eval "homerange <- mcp(xy, id)"
  ...
end

to mark-homeranges
  ...
  ask animals
  [
    pen-up

    ;; get the points of the home range polygon for the current
       animal
    r:eval (word "temp <- subset(homerange, ID=='"Name"')")
    let tempX r:get "temp$X"
    let tempY r:get "temp$Y"
    let tempXY (map [list ?1 ?2] tempX tempY)
    ...
  ]
end

to plot-area
  ...
  let precstart 20
```

```
  let precincre 5

  ;; calculate the size of the home range depending on home
     range level
  r:eval (word "area <- mcp.area(xy, id, unout='m2', percent =
     seq(" precstart ",100, by = " precincre "), plotit=FALSE)")
  ...
end
```

In the third example we used the R package spatstat [Baddeley and Turner, 2013] to analyse spatial point patterns. It is possible to calculate the L-function (based on Ripley's K) for the spatial distribution of the turtles for every time step of the simulation together with the theoretical Poisson function and confidence bands of Complete Spatial Randomness (CSR) from Monte-Carlo-Simulations (Listing III.4). In the `setup` procedure, turtles are created with random positions. In each simulation step (`go` procedure) turtles move around randomly. The new positions are sent to R into a data-frame and are transformed into a point pattern. The L-function is then calculated for this point pattern and the results are sent back to NetLogo, where they are transformed and plotted. For illustration purposes we created three extreme scenarios directly in the `setup` procedure (Figure III.7).

Listing III.4: Example 3: Point pattern analysis with NetLogo agents.

```
extensions [r]

to setup
  ...
  ;; load R package spatstat for spatial statistics
  r:eval "library(spatstat)"
  ...
end

to go
  ...
  ;; send agent variables into an R data-frame
  (r:putagentdf "agentset" turtles "who" "xcor" "ycor")

  ;; create point pattern with vectors of x- and y-coordinates
     of turtles and the dimensions of the window/world
  let revalstring (word "agppp <- ppp(agentset$xcor,
     agentset$ycor, c(" min-pxcor "," max-pxcor "), c("
     min-pycor "," max-pycor "))")
  r:eval revalstring

  ;; calculate L-function with simulation of goodness-of-fit
     envelope
  r:eval "Lsim <- envelope(agppp, Lest)"

  ;; get results from R
  let r r:get "Lsim$r"
  let obs r:get "Lsim$obs"
```

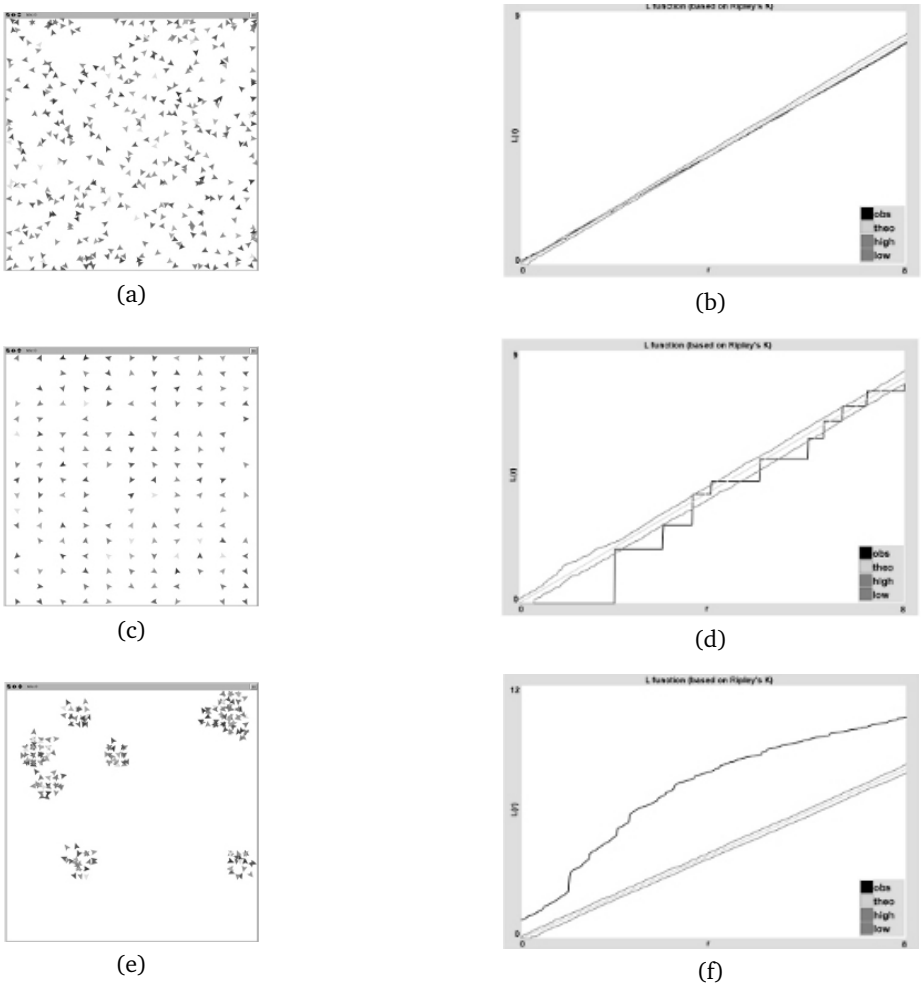


Figure III.7.: Using R functions `Lest` and `envelope` from the package spatstat to calculate the L-function (based on Ripley's K) with confidence envelopes of three different point patterns (spatial distributions of turtles in a NetLogo model). On the left the analysed point patterns are shown, on the right side the plots of the corresponding L-functions (values of L(r) vs. different radii r). The L-function was calculated using R and plotted with NetLogo. The area between the dark grey lines marks the 98% confidence envelope of the L-function under complete spatial randomness (CSR) calculated from 99 Monte-Carlo-Simulations; the light gray line shows the theoretical L-function under CSR; the black line shows the L-function for the observed pattern. (a): turtles are distributed randomly in space using uniformly distributed random values; (b): the values of the L-function for the spatial distribution of the turtles do not deviate from the envelope of the CSR at almost all distances (r), i.e., there is no significant deviation from CSR ; (c): nearly regular distribution of turtles; (d): there are significant differences to CSR with a tendency to regularity (obs < theo) especially at short distances, whereby the steps of the function values correspond to the horizontal, vertical and diagonal distances between the turtles; (e): turtles are clustered; (f): the black line (obs) shows a significant deviation from the CSR envelope with a tendency to aggregation (obs > theo) over all radii.

```
    let theo r:get "Lsim$theo"
    let high r:get "Lsim$hi"
    let low r:get "Lsim$lo"
    ...
  end
```

III.2.5. Concluding Remarks

Both NetLogo and R are powerful tools with growing user communities. In the fields of agent-based modelling and statistics, respectively, they are increasingly considered as standard software platforms. Combining these tools to tackle environmental and ecological problems provides many benefits. NetLogo users can utilize the power of R without needing to communicate via data files. This offers new and fascinating opportunities to analyse agent-based models interactively and to implement submodels of, for example, delineating home ranges that are then sensed by the model animals. R users, on the other hand, may be motivated to use NetLogo in cases where R is too limited to implement full-fledged ABMs [Petzoldt and Rinke, 2007].

Our R extension requires users to be familiar with both NetLogo and R, but does not add further complexity. The interface created by our extension consists of only additional nine NetLogo primitives (six additional for debugging), dealing with the communication between NetLogo and R via data and with calling R functions. Our extension will require future updates since the implementation and overall rationale of R and, in particular, NetLogo are continually changing. We will of course attempt to perform these updates ourselves, but since our extension is based on open-source Java code, this process does not depend on us.

III.2.6. References

A. Baddeley and R. Turner. *Package 'spatstat' Manual. R package version 1.35-0*, 2013. URL http://cran.r-project.org/web/packages/spatstat/. (last accessed 2014/01/06).

F. Bousquet and C. Le Page. Multi-Agent Simulations and Ecosystem Management: A Review. *Ecological Modelling*, 176:313–332, 2004.

C. Calenge. The Package 'adehabitat' for the R Software: A Tool for the Analysis of Space and Habitat Use by Animals. *Ecological Modelling*, 197:516–519, 2006.

V. Grimm. Visual Debugging: A Way of Analyzing, Understanding, and Communicating Bottom-Up Simulation Models in Ecology. *Natural Resource Modeling*, 15:23–38, 2002.

V. Grimm. Individual-Based Models. In S.E. Jørgensen, editor, *Ecological Models*, pages 1959–1968. Elsevier, Oxford, 2008.

V. Grimm and S.F. Railsback. *Individual-Based Modeling and Ecology*. Princeton University Press, Princeton, N.J., 2005.

G. Huse, J. Giske, and A.G.V. Salvanes. Individual-Based Modelling. In P.J.B. Hart and J. Reynolds, editors, *Handbook of Fish and Fisheries*, pages 228–248. Blackwell, Oxford, 2002.

D.C. Parker, S.M. Manson, M.A. Janssen, M.J. Hoffmann, and P. Deadman. Multiagent

Systems for the Simulation of Land-Use and Land-Cover Change: A Review. *Annals of the Association of American Geographers*, 93:314–337, 2003.

T. Petzoldt and K. Rinke. Simecol: An Object-Oriented Framework For Ecological Modeling in R. *Journal of Statistical Software*, 22 (9):1–31, 2007. URL http://www.jstatsoft.org/v22/i09. (last accessed 2014/01/06).

A. Porté and H.H. Bartelink. Modelling Mixed Forest Growth: A Review of Models for Forest Management. *Ecological Modelling*, 150:141–188, 2002.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013. URL http://www.rproject.org/. (last accessed 2014/01/06).

S.F. Railsback, S.L. Lytinen, and S.K. Jackson. Agent-Based Simulation Platforms: Review and Development Recommendations. *Simulation*, 82:609–623, 2006.

J. Uchmanski. Individual Variability and Population Regulation: An Individual-Based Model. *Oikos*, 90:539–548, 2000.

U. Wilensky. NetLogo. Center for Connected Learning and Computer-Based Modeling, 1999. URL http://ccl.northwestern.edu/netlogo. (last accessed 2014/01/06).

III.3. RNetLogo: An R Package for Running and Exploring Individual-Based Models Implemented in NetLogo

This manuscript is published as: JC Thiele, W Kurth, and V Grimm [2012]. RNetLogo: An R Package for Running and Exploring Individual-Based Models Implemented in NetLogo. Methods in Ecology and Evolution (3): 480-483.

Authorship

- Winfried Kurth supported the writing of the manuscript.
- Volker Grimm supported the writing of the manuscript.

III.3.1. Abstract

1. NetLogo is a free software platform for implementing individual-based and agent-based models. However, NetLogo's support of systematic design, performance and analysis of simulation experiments is limited. The statistics software R includes such support.
2. RNetLogo is an R package that links R and NetLogo: any NetLogo program can be controlled and run from R and model results can be transferred back to R for statistical analyses. RNetLogo includes 16 functions, which are explained and demonstrated in the user manual and tutorial. The design of RNetLogo was inspired by a similar link between Mathematica and NetLogo.
3. RNetLogo is a powerful tool for making individual-based modelling more efficient and less ad hoc. It links two fast growing user communities and constitutes a new interface for integrating descriptive statistical analyses and individual-based modelling.

III.3.2. Introduction

Individual-based models (IBMs) are simulation models that explicitly represent individual organisms and how they interact with each other and their environment [Grimm and Railsback, 2005, Thiele et al., 2011]. IBMs are an established and widely used tool in ecology and evolution [DeAngelis and Mooij, 2005]. A remaining challenge, however, is that analyses of many IBMs are still more or less superficial [Grimm, 1999, Grimm and Railsback, 2005, Lorscheid et al., 2011]. Much more could be learned from IBMs if they would be embedded in a rigorous framework for designing simulation experiments [Oh et al., 2009], storing simulation results in a systematic way and using statistical toolboxes for analysing these results.

RNetLogo is designed for this purpose. It is a package for the free statistics software R [R Core Team, 2013] which allows running and analysing IBMs that are implemented in NetLogo [Wilensky, 1999], a free software platform for implementing individual-based or agent-based models. Both R and NetLogo are increasingly used in their fields, slowly but surely turning into standard software platforms which are also the basis for training the next generation of researchers [see, e.g., Bolker, 2008, Railsback and Grimm, 2012].

Linking NetLogo with R is therefore desirable. One such link already exists: the R-Extension of NetLogo [Thiele and Grimm, 2010]. It allows calling any R command from a NetLogo program. It is mainly designed for using R commands to support the implementation of IBMs, or their sub models. For example, if a population model of a territorial animal requires, while the model is running, to calculate home range sizes based on the animals' movement, existing R packages for calculating home range sizes can be used [e.g., Calenge, 2006]. Or, if random numbers are needed from probability distributions which are not provided by NetLogo, they can easily be imported from R. There are, however, good reasons for linking R and NetLogo also the other way round, i.e. to call NetLogo programs and commands from R: the R user community is much larger than the NetLogo user community so it makes sense to provide a tool that starts with R; the R-Extension can, for technical reasons, be cumbersome to install whereas RNetLogo is as easy to load as any other R package; RNetLogo can be used to create self-documented simulation experiments and reports using Sweave [Leisch, 2002], SWord [Baier, 2009] or odfWeave [Kuhn et al., 2012]; RNetLogo opens a way to integrate NetLogo simulation into spreadsheets using RExcel [Heidberger and Neuwirth, 2009], ROOo [Drexel, 2011] or R4Calc [Gryc, 2008].

RNetLogo is not designed for using R within NetLogo programs but for running and exploring simulation experiments of a given NetLogo program. R is already widely used to analyse file output of simulation models, including those implemented in NetLogo. However, a seamless integration of both tools would facilitate the combined use of R and NetLogo. Such a seamless integration was already the reason for linking Mathematica [Wolfram Research Inc., 2013] and NetLogo [Bakshy and Wilensky, 2007], which was designed to make use of the Mathematica tools for "advanced import capabilities, statistical functions, data visualization and document creation. With the NetLogo-Mathematica link, you can run all of these tools side-by-side with NetLogo" [Bakshy and Wilensky, 2007], RNetLogo's scope is virtually the same: all these tools support systematic and comprehensive analyses of model behaviour.

NetLogo itself includes a flexible tool for performing experiments on models, BehaviorSpace [Shargel and Wilensky, 2002], which is routinely used by NetLogo users. However, links to Mathematica or R provide direct access to a wide array of additional ready-to-use powerful tools which go beyond BehaviorSpace's scope.

For an overview of RNetLogo, its main functions are listed in Table III.3. Basic usage examples of the different functions are shown in the examples section of the manual pages and are included in the *examples/code_samples* folder of the package. In the following we briefly describe typical example applications. These and further examples are included in the tutorial and the *examples/applications* folder of the package.

Table III.3.: Most important functions of RNetLogo. RNetLogo includes six further functions. For details, see the user manual and tutorial.

RNetLogo function	The function's scope
`NLStart`	Initializes the connection to NetLogo, creates an object storing the instance
`NLLoadModel`	Loads a NetLogo model into an NetLogo instance
`NLCommand`	Executes the submitted string in NetLogo
`NLDoCommand`, `NLDoCommandWhile`	Repeated execution of the submitted string. In the first case with a fixed number of repetitions, in the second case while a condition remains true within NetLogo.
`NLReport`	Send the result of a NetLogo reporter back to R. Results can be String, Number, Boolean or NetLogo lists which are transformed to appropriate R data types.
`NLDoReport`, `NLDoReportWhile`	Repeated execution of a NetLogo command combined with a reporter. The result of the report is sent back to R after every execution of the NetLogo command. In the first case with a fixed number of repetitions, in the second case while a condition remains true within NetLogo.
`NLGetAgentSet`	An easy-to-use way to access variables of an agent or an agentset. An agent is a NetLogo turtle, breed, patch or link. An agentset is a collection of agents. Results can be transformed to R lists or R data.frames.

III.3.3. Examples

Exploring models Simulation experiments can be defined, run and evaluated using RNetLogo for any existing NetLogo program (Figure III.8). For this, first model and simulation parameters are set and then the central `go` procedure, which contains the schedule of an IBM, is run for a given number of time steps or repetitions. Simulation output can be conveniently stored in R data.frames, lists or matrices and then processed for visualization and statistical analyses. Running simulation experiments via R has the advantage that modellers more easily and directly adopt the perspective of experimentalists, which will facilitate more thorough model analyses than with homespun designs programmed in NetLogo. R contains a large number of powerful packages that can be used for analysing simulation models (see, for example, the R function and package listings of Groemping [2013] for Design of Experiments, Simpson [2013] for the analysis of ecological and environmental data, Montana [2013] for statistical genetics, Bivand [2013] for the analysis of spatial data, Allignol and Latouche [2011] for the analysis of survival data or Hyndman and Zeileis [2013] for the analysis of time series data).

Linking to analytical models For many ecological and evolutionary questions, it can be helpful to compare output from simulation models to analytical approximations. By using R packages for solving analytical models, for example, Ryacas [Goedman et al., 2012], this can be performed directly via RNetLogo.

Visualization R comes with all kinds of graphics packages which facilitate visual analyses of model output. In particular, it can be used to overcome the limitation of NetLogo that only one instance of the model world can be displayed. If a model's grid cells and individuals have several state variables, which usually is the case, it is helpful to visualize them simultaneously in separate panels, plus any summary statistics of interest. If in addition such visualizations for a given time are stored for entire simulations, modellers can "slide" forth and back through the output of a simulation run, thereby developing a better understanding of spatio-temporal dynamics and how they are related to aggregated output variables. The RNetLogo tutorial includes such an example.

Database For complex models, exploratory simulations can create so much output data that it can be difficult to store them in a logical, consistent and easily accessible way. Especially the small, file-based SQLite [Hipp, 2011] database enables the modeller to save model results together with model metadata (like inputs such as parameter sets) in a single file. R comes with all kinds of database management packages, which can be used by the individual-based modeller via RNetLogo. This later allows for the use of database query language (SQL) to extract the right data needed for further analyses.

III.3.4. Conclusions

RNetLogo links two "worlds" and related software platforms. R is a free and open source software and has turned into a standard tool, which is clearly indicated by the growing number of R textbooks [Crawley, 2007, Bolker, 2008, Dalgaard, 2008]. In RNetLogo, R represents the world of statistics, including design of experiments, visualizations and all kinds of statistical inference. R can also be used for implementing simple simulations [e.g.,

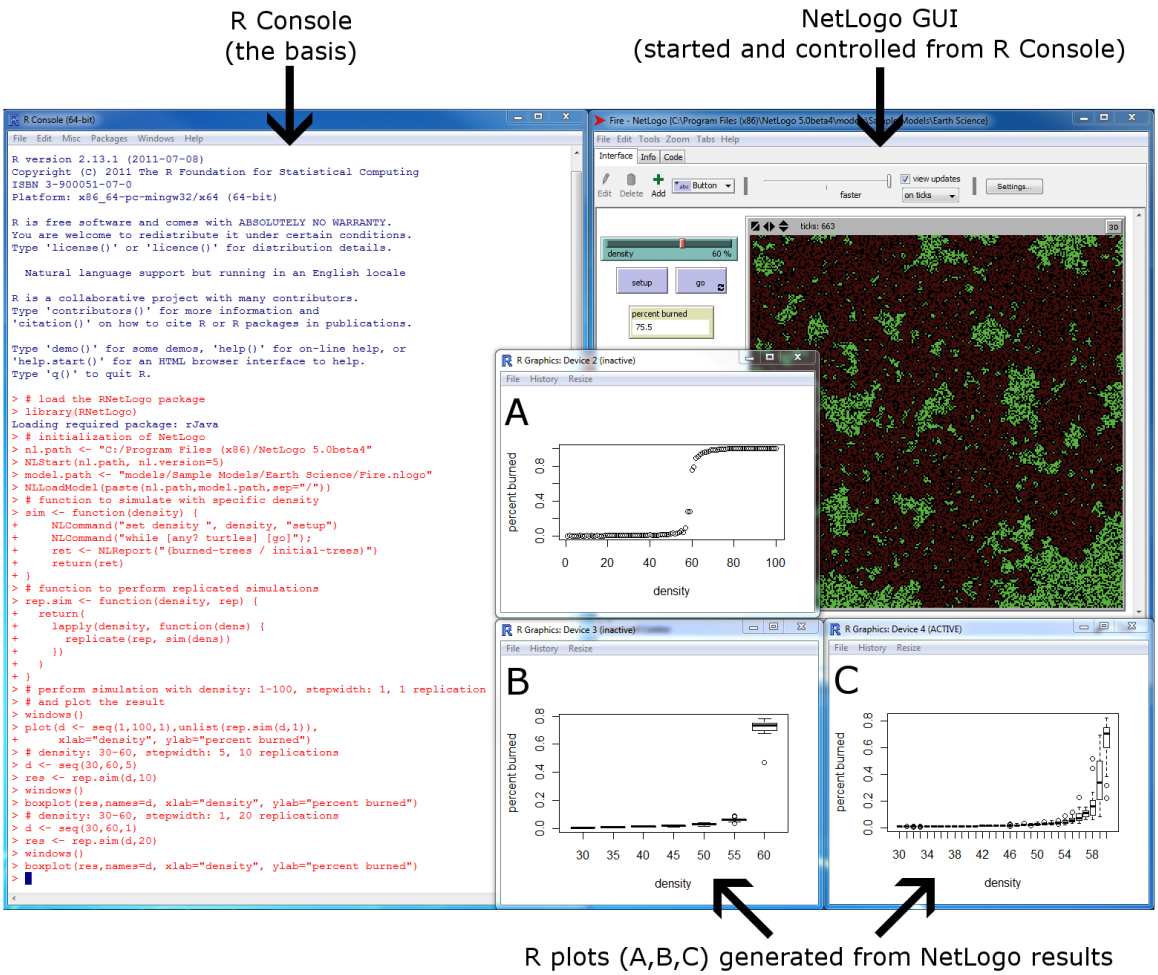


Figure III.8.: R Console (on the left) with loaded RNetLogo package and a NetLogo (on the upper right) instance started in interactive mode with Graphical User Interface. The Fire model [Wilensky, 1997] was loaded from within R and the model output (percentage of burned forest patches) has been evaluated over different initial forest density values from 1% to 100% (R plot window A). The critical range between 30% and 60% of forest density is then evaluated with repeated simulations to take into account stochasticity in the model, with a step width of 5% and 10 replications (R plot window B) and with a step width of 1% and 20 replications (R plot window C). The full code for this analysis is shown in red in the RConsole (29 lines of code).

Petzoldt and Rinke, 2007] but does not provide specific support for making model development and simulation efficient. NetLogo was originally developed as a teaching tool, but is increasingly used for research. In RNetLogo, it represents the world of individual-based and agent-based modelling, which has considerably matured over the last 10 years but still has not yet established a culture of systematic design and analysis of simulation experiments.

RNetLogo is easy to install and use and thereby opens R and NetLogo users' access to each others' world and software platform. This, we hope, will lead to more rigorous model analyses and, thereby, to making better use of individual- and agent-based models for answering theoretical and applied questions in ecology and evolution.

The RNetLogo package is available on CRAN (http://cran.r-project.org/web/packages/RNetLogo) and R-Forge (http://rnetlogo.r-forge.r-project.org). This package includes the aforementioned user manual and tutorial. See Appendix E for installation and quick start usage instructions.

III.3.5. Acknowledgements

The authors would like to thank three anonymous reviewers for their useful comments on an earlier version of the manuscript.

III.3.6. References

A. Allignol and A. Latouche. *CRAN Task View: Survival Analysis, Version: 2011-08-04*, 2011. URL http://cran.r-project.org/web/views/Survival.html. (last accessed 2014/01/06).

T Baier. *SWordInstaller: SWord: Use R in Microsoft Word (Installer). R package version 1.0-2*, 2009. URL http://cran.r-project.org/src/contrib/Archive/SWordInstaller/. (last accessed 2014/01/06).

E. Bakshy and U. Wilensky. *NetLogo-Mathematica Link*. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL, 2007. URL http://ccl.northwestern.edu/netlogo/mathematica.html. (last accessed 2014/01/06).

R. Bivand. *CRAN Task View: Analysis of Spatial Data, Version 2013-12-16*, 2013. URL http://cran.r-project.org/web/views/Spatial.html. (last accessed 2014/01/06).

B. Bolker. *Ecological Models and Data in R*. Princeton University Press, Princeton N.J., 2008.

C. Calenge. The Package 'adehabitat' for the R Software: A Tool for the Analysis of Space and Habitat Use by Animals. *Ecological Modelling*, 197:516–519, 2006.

M.J. Crawley. *The R Book*. Wiley, Chichester, 2007.

P. Dalgaard. *Introductory Statistics with R*. Springer, New York, 2nd edition, 2008.

D.L. DeAngelis and W.M. Mooij. Individual-Based Modeling of Ecological and Evolutionary Processes. *Annual Review of Ecology, Evolution, and Systematics*, 36:147–168, 2005.

R. Drexel. *ROOo v 0.751*, 2011. URL http://rcom.univie.ac.at/download.html#ROOo. (last accessed 2014/01/06).

R. Goedman, G. Grothendieck, S. Højsgaard, and A. Pinkus. *Ryacas: R Interface to the Yacas Computer Algebra System. R package version 0.2-11*, 2012. URL http://CRAN.R-project.org/package=Ryacas. (last accessed 2014/01/06).

V. Grimm. Ten Years of Individual-Based Modelling in Ecology: What Have We Learned and What Could We Learn in the Future? *Ecological Modelling*, 115:129–148, 1999.

V. Grimm and S.F. Railsback. *Individual-Based Modeling and Ecology*. Princeton University Press, Princeton, N.J., 2005.

U. Groemping. *CRAN Task View: Design of Experiments (DoE) & Analysis of Experimental Data, Version: 2013-03-20*, 2013. URL http://cran.r-project.org/web/views/ExperimentalDesign.html. (last accessed 2014/01/06).

W. Gryc. *R and Calc v 0.1.12*, 2008. URL http://wiki.services.openoffice.org/wiki/R_and_Calc. (last accessed 2014/01/06).

R.M. Heidberger and E. Neuwirth. *R Through Excel: A Spreadsheet Interface for Statistics, Data Analysis, and Graphics*. Springer, New York, 2009.

D.R. Hipp. About SQLite, 2011. URL http://www.sqlite.org/about.html. (last accessed 2014/01/06).

R.J. Hyndman and A. Zeileis. *CRAN Task View: Time Series Analysis, Version: 2013-12-22*, 2013. URL http://cran.r-project.org/web/views/TimeSeries.html. (last accessed 2014/01/06).

M. Kuhn, S. Weston, N. Coulter, P. Lenon, and Z. Otles. *odfWeave: Sweave Processing of Open Document Format (ODF) Files. R package version 0.8.2*, 2012. URL http://CRAN.R-project.org/package=odfWeave. (last accessed 2014/01/06).

F. Leisch. Sweave: Dynamic Generation of Statistical Reports Using Literate Data Analysis. In W. Härdle and B. Rönz, editors, *Compstat 2002 - Proceedings in Computational Statistics*, pages 575–580. Physica, 2002.

I. Lorscheid, B.-O. Heine, and M. Meyer. Opening the 'Black Box' of Simulations: Transparency of Simulation Models and Effective Results Reports Through the Systematic Design of Experiments. Technical Report Research Paper No. 001, Hamburg University of Technology (TUHH), 2011. URL http://ssrn.com/abstract=1860904. (last accessed 2014/01/06).

G. Montana. *CRAN Task View: Statistical Genetics, Version: 2013-12-10*, 2013. URL http://cran.r-project.org/web/views/Genetics.html. (last accessed 2014/01/06).

R. Oh, S. Sanchez, T. Lucas, H. Wan, and M. Nissen. Efficient Experimental Design Tools for Exploring Large Simulation Models. *Computational & Mathematical Organization Theory*, 15:237–257, 2009.

T. Petzoldt and K. Rinke. Simecol: An Object-Oriented Framework For Ecological Modeling in R. *Journal of Statistical Software*, 22 (9):1–31, 2007. URL http://www.jstatsoft.org/v22/i09. (last accessed 2014/01/06).

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, 2013. URL http://www.r-project.org/. (last accessed 2014/01/06).

S.F. Railsback and V. Grimm. *Agent-Based and Individual-Based Modeling: A Practical Introduction*. Princeton University Press, 2012.

B. Shargel and U. Wilensky. *BehaviorSpace*. Northwestern University, Evanston, IL, 2002. URL http://ccl.northwestern.edu/netlogo/docs/behaviorspace.html. (last accessed 2014/01/06).

G. Simpson. *CRAN Task View: Analysis of Ecological and Environmental Data, Version: 2013-12-12*, 2013. URL http://cran.r-project.org/web/views/Environmetrics.html. (last accessed 2014/01/06).

J.C. Thiele and V. Grimm. NetLogo Meets R: Linking Agent-Based Models with a Toolbox for Their Analysis. *Environmental Modelling & Software*, 25(8):972–974, 2010.

J.C. Thiele, W. Kurth, and V. Grimm. Agent- and Individual-Based Modeling with NetLogo: Introduction and New NetLogo Extensions. In K. Römisch, A. Nothdurft, and U. Wunn, editors, *22. Tagung der Sektion Forstliche Biometrie und Informatik des Deutschen Verbandes Forstlicher Forschungsanstalten und der Arbeitsgemeinschaft Ökologie und Umwelt der Internationalen Biometrischen Gesellschaft - Deutsche Region, 20-21th September 2010 in Göttingen (Germany)*, Die Grüne Reihe, pages 68–101, 2011.

U. Wilensky. NetLogo Fire Model, 1997. URL http://ccl.northwestern.edu/netlogo/models/Fire. (last accessed 2014/01/06).

U. Wilensky. NetLogo. Center for Connected Learning and Computer-Based Modeling, 1999. URL http://ccl.northwestern.edu/netlogo. (last accessed 2014/01/06).

Wolfram Research Inc. *Mathematica, Version 9.0*, 2013. URL http://www.wolfram.com/mathematica/. (last accessed 2014/01/06).

III.4. R Marries NetLogo: Introduction to the RNetLogo Package

This manuscript is published as: JC Thiele [2014]. R Marries NetLogo: Introduction to the RNetLogo Package. Journal of Statistical Software 58 (2): 1-41.

III.4.1. Abstract

The RNetLogo package delivers an interface to embed the agent-based modelling platform NetLogo into the R environment with headless (no graphical user interface) or interactive GUI mode. It provides functions to load models, execute commands, push values, and to get values from NetLogo reporters. Such a seamless integration of a widely used agent-based modelling platform with a well-known statistical computing and graphics environment opens up various possibilities. For example, it enables the modeller to design simulation experiments, store simulation results, and analyse simulation output in a more systematic way. It can therefore help close the gaps in agent-based modelling regarding standards of description and analysis. After a short overview of the agent-based modelling approach and the software used here, the paper delivers a step-by-step introduction to the usage of the RNetLogo package by examples.

III.4.2. Introduction

Agent- and individual-based modelling

Agent-based models (ABMs) or individual-based models (IBMs), as they are called in ecology and biology, are simulation models that explicitly represent individual agents, which can be, for example, humans, institutions, or organisms with their traits and behaviour [Grimm and Railsback, 2005, Gilbert, 2007, Thiele et al., 2011]. A key characteristic of this modelling approach is that simulation results emerge from the more or less complex interactions among the agents. Therefore, such models are useful when local interactions on the micro level are essential for the description of patterns on the macro level.

The origins of the ABM approach go back to the late 1970s [e.g., Hewitt, 1976] with the development of so-called multi-agent systems (MASs) in computer science as a part of the distributed artificial intelligence (DAI) research area [Green et al., 1997, Sycara, 1998]. Their wider use in computer science began only in the 1990s [Luck et al., 2003, Wooldridge, 2005, Weiss, 1999]. Definitions of the term MAS and what an agent is, can be found for example in Wooldridge [2005] and Jennings [2000]. Examples for the use of MASs with intelligent agents in the field of computer science include computer games, computer networks, robotics for manufacturing, and traffic-control systems [for examples, see Oliveira, 1999, Luck et al., 2003, Shen et al., 2006, Moonen, 2009].

With increasing importance of questions about coordination and cooperation within the MASs the connections to social sciences arose [Conte et al., 1998] and the field of agent-based social simulation (ABSS), that is, an agent-based modelling approach as part of computational sociology became a "counter-concept" to the classical top-down system dynamics and microsimulation approaches [Gilbert, 1999, Squazzoni, 2010]. ABSS is mainly used for theory testing and development [Macy and Willer, 2002, Conte, 2006] and applied to simulations of differentiation, diffusion, and emergence of social order in social systems [for examples, see listings in Macy and Willer, 2002, Squazzoni, 2010] as well as to questions about demographic behaviour [Billari and Prskawetz, 2003]. The most famous models in social sciences are Schelling's segregation model [Schelling, 1969] and the Sugarscape model of Epstein and Axtell [1996].

Strongly related to the development of ABMs in social sciences is the establishment of the ABM approach in economics, which is called agent-based computational economics (ACE) and related to the field of cognitive and evolutionary economics. The aims of ACE can be

divided into four categories: empirical understanding, normative understanding, qualitative insight as well as theory generation and methodological advancement [for details, see Tesfatsion, 2006]. It was applied, for example, to the reproduction of the classical cobweb theorem [e.g., Arifovic, 1994], to model financial/stock markets [see LeBaron, 2000, for a review] as well as to the simulation of industry and labour dynamics [e.g., Leombruni and Richiardi, 2004].

In contrast to ABSS and ACE, the agent-based modelling approach has a slightly longer tradition in ecology [Grimm and Railsback, 2005]. The development of so called individual-based models is less closely related to the developments of MASs, because ecologists early became aware of the restrictions in classical population models (differential equation models) and looked for alternatives. Over the last three to four decades hundreds of IBMs were developed in ecology [DeAngelis and Mooij, 2005]. For reviews see, for example, Grimm [1999] and DeAngelis and Mooij [2005].

Besides these four main research areas, there are many other disciplines in which ABMs are increasingly used, often within an interdisciplinary context. Examples include ecological economics [e.g., Heckbert et al., 2010], marketing/socio-psychology [e.g., North et al., 2010], archaeology/anthropology [e.g., Griffin and Stanish, 2007], microbiology [e.g., Ferrer et al., 2008], biomedicine/epidemiology [e.g., Carpenter and Sattenspiel, 2009], criminology [strongly related to ABSS, e.g., Malleson et al., 2010] and land-use management [e.g., Matthews et al., 2007].

Links to statistics

Links to statistics can be found in agent-based modelling along nearly all stages of the modelling cycle. Often, models are developed on the basis of empirical/field data. This gives the first link to statistics as data are analysed with statistical methods to derive patterns, fit regression models and so on to construct and parametrize the rules and to prepare input as well as validation data.

Often, agent-based model rules depend on statistical methods applied during a simulation run. In very easy cases, for example, animal reproduction could depend on the sum of the food intake in a certain period but it is also possible for agent behaviours to be based on correlation, regression, network, point pattern analysis etc.

The third link comes into play when the model is formulated and implemented and some parameters of the model are unknown. Then, methods of inverse modelling with different sampling schemes, Bayesian calibration, genetic algorithms and so on can be used to obtain feasible parameter values.

In the next stage, the model application, methods like uncertainty and sensitivity analysis provide important tools to gain an understanding of the systems' behaviour and functioning, i.e., to open the black box of complexity.

The last link to statistics is the further analysis of the model output using descriptive as well as inferential statistics. Depending on the type of model, this can include correlation analysis, hypothesis testing, network analysis, spatial statistics, time series analysis, survival analysis etc.

The focus in this article is on those parts where statistical methods are applied in combination with the model runs.

NetLogo

Wilensky's NetLogo [Wilensky, 1999] is an agent-based modelling tool developed and maintained since 1999 by the Center for Connected Learning and Computer-Based Modeling at Northwestern University, Illinois. It is an open-source software platform programmed in Java and Scala and especially designed for the development of agent-based simulation models. It comes with an integrated development and simulation environment. It provides many predefined methods (so-called primitives and reporters) for behavioural rules of the agents. Because it has a Logo-like syntax and standard agent types (turtles, patches, links), in combination with a built-in GUI, it is very easy to learn. Due to its simplicity and relatively large user community, it is becoming the standard platform for communicating and implementing ABMs that previously has been lacking.

For an introduction to NetLogo see its documentation [Wilensky, 2013]. An introduction into agent-based modelling using NetLogo can be found, for example, in Railsback and Grimm [2012] or Wilensky and Rand [2014].

R

R [R Core Team, 2014a] is a well-known and established language and open source environment for statistical computing and graphics with many user-contributed packages.

For NetLogo users not yet familiar with R: R is very well documented; see, for example, the R language definition [R Core Team, 2014b]. Furthermore, many tutorials can be found in the web, for example, Maindonald [2008], Venables et al. [2014], Kabacoff [2013], Owen [2010]; and many books are available, for example, Zuur et al. [2009], Crawley [2005], Kabacoff [2010], Venables and Ripley [2002].

Note on this article

This work is a mixture of scientific article and tutorial for a scientific tool; writing styles differ between these two elements, but section headings indicate what element each section contains.

III.4.3. Introducing RNetLogo

RNetLogo [Thiele, 2014] is an R package that links R and NetLogo; i.e., any NetLogo model can be run and controlled from R and simulation results can be transferred back to R for statistical analyses. This is desirable as NetLogo's support of systematic design, performance, and analysis of simulation experiments is limited. In general, much more could be learned from ABMs if they were embedded in a rigorous framework for designing simulation experiments [Oh et al., 2009], storing simulation results in a systematic way, and using statistical toolboxes for analysing these results. RNetLogo can be used to bridge this gap since R (together with the enormous number of packages) delivers such tools. Such a seamless integration was already the scope of the NetLogo-Mathematica Link [Bakshy and Wilensky, 2007b], which was designed to make use of Mathematica's functionality for "advanced import capabilities, statistical functions, data visualization, and document creation. With NetLogo-Mathematica Link, you can run all of these tools side-by-side with NetLogo" [Bakshy and Wilensky, 2007a]. RNetLogo offers such a framework for two freely available open

source programs with fast-growing communities. RNetLogo itself is open-source software published under the GNU GPL license.

RNetLogo consists of two parts: R code and Java code (Figure III.9). The R code is responsible for offering the R functions, for connecting to Java, and for doing data transformations, while the Java code communicates with NetLogo.

To connect the R part of RNetLogo to the Java part the rJava package for R [Urbanek, 2013b] is used. The rJava package offers the ability to create objects, call methods and access class members of Java objects through the Java Native Interface [JNI, Oracle, 2013] from C. The Java part of the RNetLogo package connects to the Java Controlling API of NetLogo. This API allows controlling NetLogo from Java (and Scala) code [for details, see Tisue, 2012].

When NetLogo code is given to an RNetLogo function, i.e., to the R part of RNetLogo, it is submitted through rJava to the Java part of RNetLogo, and from there to NetLogo's Controlling API and thence to NetLogo. In case of reporters, i.e., primitives with return values, the return value is collected by the Java part of RNetLogo, transformed from Java to R by rJava and sent through the R part of RNetLogo to R.

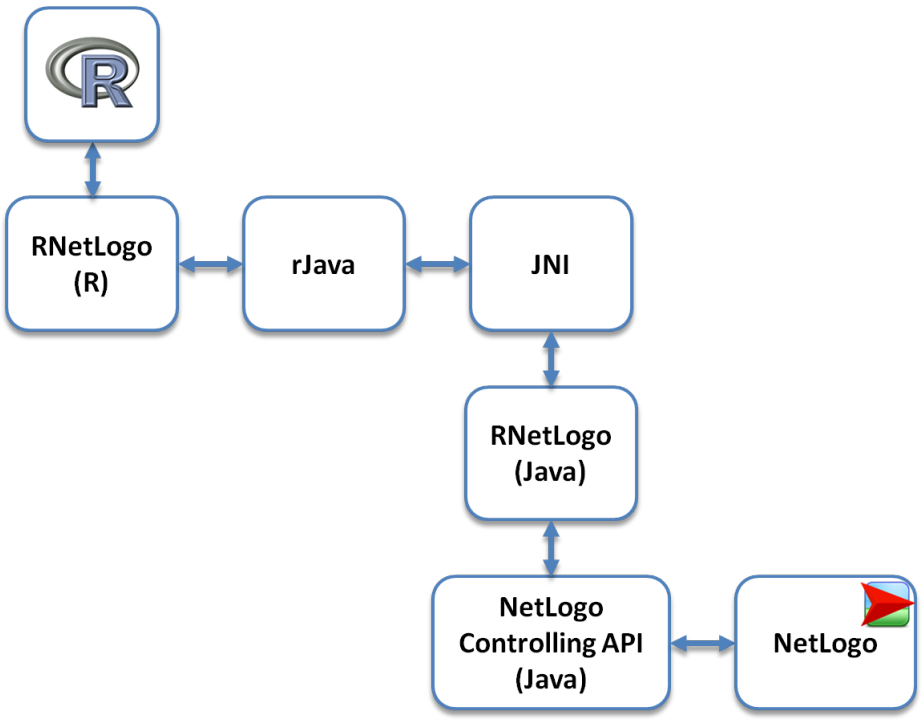


Figure III.9.: RNetLogo consists of two parts: an R and a Java part. The R part adds the RNetLogo functions to R and uses rJava to connect the Java part. The Java part connects to NetLogo via the Controlling API of NetLogo.

Currently RNetLogo provides 17 functions (Table III.4).

The functions that handle NetLogo code, like `NLCommand` or `NLReport`, expect it as a string. Some other functions, e.g., `NLGetAgentSet`, construct such strings internally from the different function arguments in the R part of RNetLogo. This string is then sent to the Java part of RNetLogo and from there it is evaluated through NetLogo's Controlling API.

When the submitted NetLogo code is not valid NetLogo throws an exception of type "`LogoException`" or "`CompilerException`" containing the corresponding error message. This exception is further thrown by the Java part of RNetLogo, handled by rJava, and requested finally by the R part of RNetLogo and printed to R's command line. Runtime errors in NetLogo, like "`java.lang.OutOfMemoryError`", are reported in the same manner. A message in R's command line is printed. But errors where the JVM crashes can cause crashes in rJava, which can affect the R session as well.

Some functions of RNetLogo, like `NLDoCommand` or `NLDoReportWhile`, require further control flow handling, i.e., loops and condition checkings, which are done by the Java part of RNetLogo. The methods `command` and `report` of class `org.nlogo.workspace.Controllable` of NetLogo's Controlling API are used as interfaces to NetLogo. All other things are done by the R and the Java part of RNetLogo.

Table III.4.: Functions provided by RNetLogo. All functions take an additional (optional) argument `nl.obj` which is not listed in the table. It is a string identifying a NetLogo instance created with `NLStart`. Where functions take wildcard arguments (...) a short description is given in squared brackets. Optional arguments are marked with an asterisk. Details to the functions can be found in the manual pages of RNetLogo.

Function	Scope	Arguments	Return value
`NLStart`	Creates an instance of NetLogo.	`nl.path` `gui`* `nl.version`* `is3d`*	-
`NLLoadModel`	Loads a model into the NetLogo instance.	`model.path`	-
`NLQuit`	Quits a NetLogo instance.	`all`*	-
`NLCommand`	Executes a command in the referenced NetLogo instance.	`...` [strings containing NetLogo commands]	-
`NLDoCommand`	Repeats execution of a command in the referenced NetLogo instance a defined number of times.	`iterations` `...` [strings containing NetLogo commands]	-
`NLDoCommandWhile`	Repeats a command in the referenced NetLogo instance while a NetLogo reporter returns `TRUE`.	`condition` `...` [strings containing NetLogo commands] `max.minutes`*	-
`NLReport`	Reports a value or list of values.	`reporter`	Result of the reporter.

`NLDoReport`	Repeats a command and a reporter in the referenced NetLogo instance a defined number of times.	`iterations`
		`command`
		`reporter`
		`as.data.frame`*
		`df.col.names`*
			Concatenated result of repeated reporter calls.

`NLDoReportWhile`	Repeats execution of a command and a reporter in the referenced NetLogo instance while a conditional reporter returns `TRUE`.	`condition`
		`command`
		`reporter`
		`as.data.frame`*
		`df.col.names`*
		`max.minutes`*
			Concatenated result of repeated reporter calls.

`NLGetAgentSet`	Reports variable values of one or more agents as a data frame (optionally as a list or vector).	`agent.var`
		`agentset`
		`as.data.frame`*
		`agents.by.row`*
		`as.vector`*
			Values of all requested agent variables of all requested agents.

`NLGetPatches`	Reports the values of patch variables as a data frame (optionally as a list, matrix or simple vector).	`patch.var`
		`patchset`
		`as.matrix`*
		`as.data.frame`*
		`patches.by.row`*
		`as.vector`*
			Values of all requested patch variables of all requested patches.

`NLGetGraph`	Captures a network of links.	`link.agentset`*	**igraph** graph object of link agents

`NLSetAgentSet`	Sets a variable of one or more agents to values in a data frame or vector.	`agentset`
		`input`
		`var.name`
			-

`NLSetPatches`	Sets a variable of all patches in the NetLogo World to the values in a matrix.	`patch.var` `in.matrix`	-
`NLSetPatchSet`	Sets the variable value of one or more patches to values in a data frame.	`input` `patch.var`	-
`NLDfToList`	Transforms a data frame into a NetLogo list or multiple NetLogo lists (one for each column of the data frame).	`in.data.frame`	-

`NLSourceFromString` Appends a string to the NetLogo model's code.

`...` [strings containing model source code]

`append.model*`

What else?

If only the integration of R calculations into NetLogo (i.e., the other way around) is of interest, a look at the R-Extension to NetLogo at http://r-ext.sourceforge.net/ [see also Thiele and Grimm, 2010] can be useful.

If we want to use the R-Extension within a NetLogo model controlled by RNetLogo, we should use the Rserve-Extension instead (available at http://rserve-ext.sourceforge.net), because loading the R-Extension will crash as it is not possible to load the JRI library when rJava is active.

III.4.4. Using RNetLogo - Hands-on

Installation

To install and use RNetLogo we must have R (available from the Comprehensive R Archive Network at http://CRAN.R-project.org/) and NetLogo (http://ccl.northwestern.edu/netlogo/download.shtml) installed. The RNetLogo package is available from CRAN (http://CRAN.R-project.org/package=RNetLogo/) and is installed like any other R package; see chapter 6 of R's installation and administration manual [R Core Team, 2014c] for information on how to install a package. However, RNetLogo requires the rJava package [Urbanek, 2013b], available from CRAN. It can happen that we have to reconfigure Java/R after installing rJava on Unix machines. This topic has been discussed several times; see, for example, RWiki [2006]. The following sections provide an introduction to the usage of RNetLogo, however, there are some pitfalls described in section III.4.6 one should be aware before starting own projects.

Loading NetLogo

To use the RNetLogo package the first time in an R session we have to load the package, like any other packages, with

```
R> library("RNetLogo")
```

When loading RNetLogo it will automatically try to load rJava. If this runs without any error we are ready to start NetLogo (if not, see section III.4.4). To do so, we have to know where NetLogo is installed. What we need is the path to the folder that contains the *NetLogo.jar* file. On Windows machines this could be `C:/Program Files/NetLogo 5.0.5/`. Here, we assume that the R working directory is set (see function `setwd(<path>)`) to the path where NetLogo is installed.

Now, we have to decide whether we want to run NetLogo in the background without seeing the graphical user interface (GUI) and control NetLogo completely from R or if we want to see and use the NetLogo GUI. In the latter case, we can use NetLogo as it was started independently, i.e., can load models, change the source code, click on buttons, see the NetLogo View, inspect agents, and so on, but also have control over NetLogo from R. The disadvantage of starting NetLogo with GUI is that we cannot run multiple instances of NetLogo in one R session. This is only possible in the so called headless mode, i.e., running NetLogo without GUI (see section III.4.4 for details). Linux and Mac users should read the details section of the `NLStart` manual page (by typing `help(NLStart)`).

Due to the NetLogo's Controlling API changes with the NetLogo version, we have to use an extra parameter `nl.version` to start RNetLogo for NetLogo version 4 (`nl.version=4` for NetLogo 4.1.x, `nl.version=40` for NetLogo 4.0.x). The default value of `nl.version` is 5, which means that we do not have to submit this parameter when using NetLogo 5.0.x. Since NetLogo 5.0.x operates much faster on lists than older versions it is highly recommended to use it here (see also the RNetLogo package vignette "Performance Notes and Tests", Appendix G).

To keep it simple and comprehensible we start NetLogo with GUI by typing:

```
R> nl.path <- getwd()
R> NLStart(nl.path)
```

If everything goes right, a NetLogo Window will be opened. We can use the NetLogo window as if it had been started independently, with the exception that we cannot close the window through clicking. On Windows, NetLogo appears in the same program group at the taskbar as R. If possible, arrange the R and NetLogo windows so that we have them side by side (Figure III.10), and can see what is happening in NetLogo when we submit the following examples.

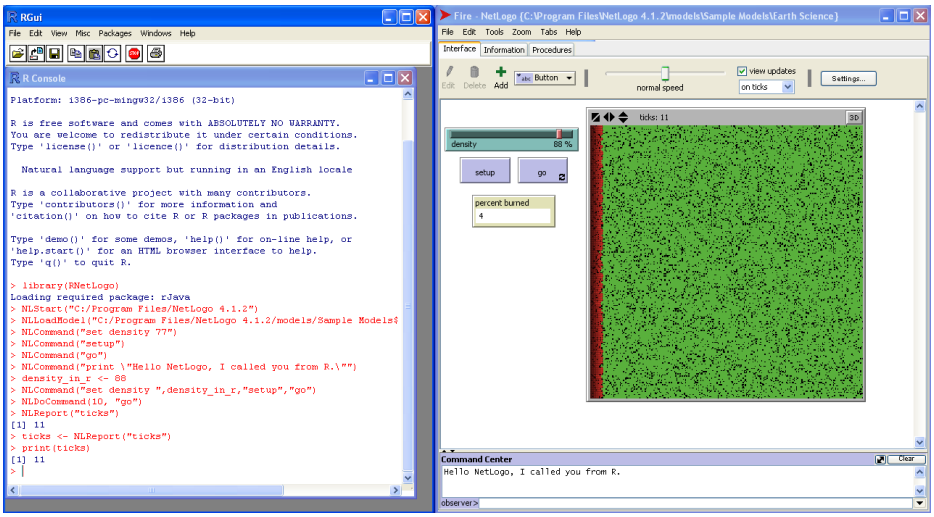


Figure III.10.: NetLogo (on the right) started and controlled from R (on the left).

Loading a model

We can now open a NetLogo model by just clicking on *File -> Open...* or choosing one of the sample models by clicking on *File -> Models Library*. But to learn to control NetLogo from R as when starting NetLogo in headless mode, we type in R:

```
R> model.path <- file.path("models"', "Sample Models", "Earth
   Science", "Fire.nlogo")
R> NLLoadModel(file.path(nl.path, model.path))
```

The Forest Fire model [Wilensky, 1997b] should be loaded. This model simulates a fire spreading through a forest. The expansion of the fire depends on the density of the forest. The forest is defined as a tree density value of the patches, while the fire is represented by turtles. If we want, we can now change the initial tree density by using the slider on the interface tab and run the simulation by clicking on the setup button first and then on the go button. In the next section, we will do the same by controlling NetLogo from R.

Principles of controlling a model

In a first step, we will change the density value, i.e., the position of the density slider, by submitting the following statement in R:

```
R> NLCommand("set density 77")
```

The slider goes immediately to the position of 77 percent. We can now execute the `setup` procedure to initialize the simulation. We just submit in R:

```
R> NLCommand("setup")
```

And again, the command is executed immediately. The tick counter is reset to 0, the View is green and first fire turtles are found on the left side of the View. Please notice that the `NLCommand` function does not press the `setup` button, but calls the `setup` procedure. In the Forest Fire example it makes no difference as the setup button also just calls the `setup` procedure, but it is possible to add more code to a button than just calling a procedure. But we can copy and paste such code into the `NLCommand` function as well.

We now want to run one tick by executing the `go` procedure. This is nothing new; we just submit in R:

```
R> NLCommand("go")
```

We see that the tick counter was incremented by one and the red line of the fire turtles on the left of the View extended to the next patch.

As we have seen, the `NLCommand` function can be used to execute any command which could be typed into NetLogo's command center. We can, for example, print a message into NetLogo's command center with the following statement:

```
R> NLCommand("print \"Hello NetLogo, I called you from R.\"")
```

The backslashes in front of the quotation marks are used to "mask" the quotation marks; otherwise R would think that the command string ends after the `print` and would be confused. Furthermore, it is possible to submit more than one command at once and in combination with R variables. We can change the density slider and execute `setup` and `go` with one `NLCommand` call like this:

```
R> density.in.r <- 88
R> NLCommand("set density ", density.in.r, "setup", "go")
```

In most cases, we do not want to execute a `go` procedure only a single time but for, say, ten times (ticks). With the RNetLogo package we can do this with:

```
R> NLDoCommand(10, "go")
```

Now we have run the simulation eleven ticks and maybe want to have this information in R. Therefore, we execute:

```
R> NLReport("ticks")
```

Output:

```
[1] 11
```

As you might expect, we can save this value in an R variable by typing:

```
R> ticks <- NLReport("ticks")
R> print(ticks)
```

Output:

```
[1] 11
```

This was already the basic functionality of the RNetLogo package. In the following section we mostly modify and/or extend this basic functionality.

NetLogo users should note that there is no "forever button". To run a simulation for several ticks we can use one of the loop functions (`NLDoCommand`, `NLDoCommandWhile`, `NLDoReport`, `NLDoReportWhile`) or write a custom procedure in NetLogo that runs the `go` procedure the desired number of times when called once by R.

To quit a NetLogo session, i.e., to close a NetLogo instance, we have to use the `NLQuit` function. If we used the standard GUI mode without assigning the NetLogo instance to an R variable, we can write:

```
R> NLQuit()
```

Otherwise, we have to specify which NetLogo instance we want to close by specifying the R variable storing it. Please note that there is currently no way to close the GUI mode completely. That is why we cannot run `NLStart` again in the same R session when NetLogo was started with its GUI.

Advanced controlling functions

In subsection III.4.4, we used the `NLDoCommand` function to run the simulation for ten ticks. Here, we will run the model for ten ticks as well, but we will collect the percentage of burned trees after every tick automatically:

```
R> NLCommand("setup")
R> burned <- NLDoReport(10, "go", "(burned-trees /
   initial-trees) * 100")
R> print(unlist(burned))
```

Output:

```
 [1] 0.4192073 0.7821574 1.1287747 1.4790215 1.8238240 2.1649971
 [7] 2.5116144 2.8836382 3.2629210 3.6349448
```

This code ran the simulation for ten ticks and wrote the result of the given reporter (the result of the calculation of the percentage of burned trees) after every tick into the R list `burned`.

If we want to run the simulation until no trees are left and know the percentage of burned trees in every tick, we can execute:

```
R> NLCommand("setup")
R> burned <- NLDoReportWhile("any? turtles",
+ "go",
+ c("ticks",
+ "(burned-trees / initial-trees) * 100"),
+ as.data.frame = TRUE,
+ df.col.names = c("tick", "percent burned"))
R> plot(burned, type = "s")
```

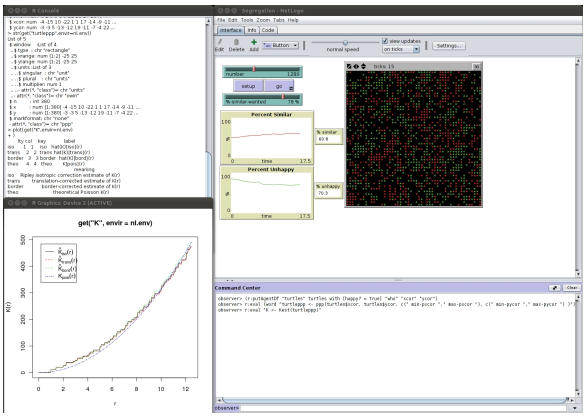


Figure III.11.: The percentage of burned trees over time as the result of `NLDoReportWhile`, which runs as long as there are turtles (`any? turtles`).

The first argument of the function takes a NetLogo reporter. Here, the `go` procedure will be executed while there are turtles in the simulation, i.e., `any? turtles` reports `true`. Moreover, we have used not just one reporter (third argument) but a vector of two reporters; one returning the current simulation time (`tick`) and a second with the percentage of burned trees. Furthermore, we have defined that our output should be saved as a data frame instead of a list and we have given the names of the columns of the data frame by using a vector of strings in correspondence with the reporters. At the end, the R variable `burned` is of type `data.frame` and contains two columns; one with the `tick` number and a second with the corresponding percentage of burned trees. By using the standard plot function, we graph the percentage of burned trees over time (Figure III.11).

To demonstrate the `NLGetAgentSet` function, we use a different model. Therefore, we load the Tumor model from NetLogo's Models Library, set it up and run it for 20 ticks, as follows:

```
R> model.path <- file.path("models", "Sample Models",
   "Biology", "Tumor.nlogo"
R> NLLoadModel(file.path(nl.path, model.path))
R> NLCommand("setup")
R> NLDoCommand(20, "go")
```

After we have run 20 ticks, we load the x and y positions of all tumour cells (which are turtles) into a data frame and show them in a plot. But before we call the plot function, we will get the spatial extent of the NetLogo World to use in the plot window (Figure III.12):

```
R> cells <- NLGetAgentSet(c("xcor", "ycor"), "turtles")
R> x.minmax <- NLReport("(list min-pxcor max-pxcor)")
R> y.minmax <- NLReport("(list min-pycor max-pycor)")
R> plot(cells, xlim = x.minmax, ylim = y.minmax, xlab = "x",
   ylab = "y")
```

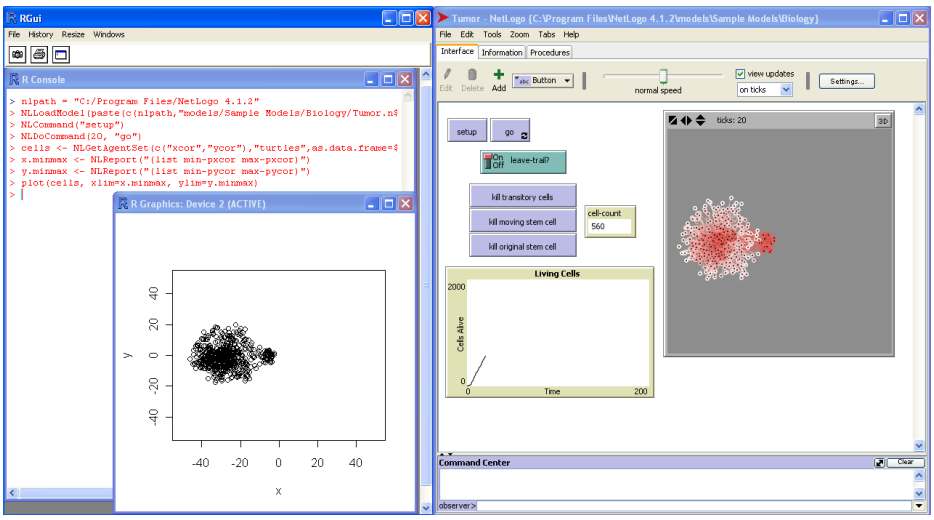


Figure III.12.: A visualization of turtle locations obtained via `NLGetAgentSet`. Turtle locations are displayed in the original NetLogo simulation (right) and in the R GUI of Windows (left).

In a second step, we get only the metastatic cells and plot them again (Figure III.13):

```
R> cells.metastatic <- NLGetAgentSet(c("xcor", "ycor"),
+ "turtles with [metastatic? = True]")
R> plot(cells.metastatic, xlim = x.minmax, ylim = y.minmax,
+ xlab = "x", ylab = "y")
```

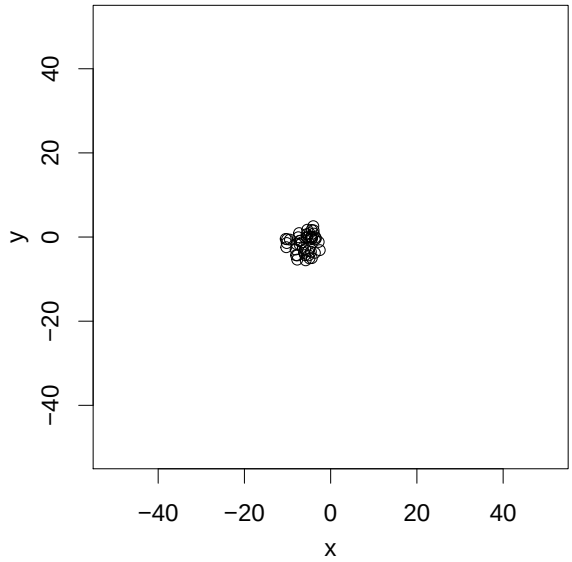


Figure III.13.: Same as in Figure III.12 but only with a subset of turtles that fulfil a condition (are metastatic cells).

We can use the `NLGetAgentSet` function to get patches and links as well. But there is a special function for patches, called `NLGetPatches`, which makes life easier by returning the patch values as a matrix. We test this function by using the Fur model about patterns on animals' skin self-organization and plot the result in a simple raster image (Figure III.14). We load the model, set it up and get the patches as a matrix

```
R> model.path <-file.path("models", "Sample Models",
   "Biology", "Fur.nlogo")
R> NLLoadModel(file.path(nl.path, model.path))
R> NLCommand("setup")
R> NLDoCommand(5, "go")
R> patches.matrix <- NLGetPatches("pcolor", "patches",
   as.matrix = TRUE)
```

Now, we reorganize the matrix to make it fit for the image function and define the image colours:

```
R> patches.matrix.rot <- t(patches.matrix)
R> patches.matrix.rot <- as.data.frame(patches.matrix.rot)
R> patches.matrix.rot <- rev(patches.matrix.rot)
R> patches.matrix.rot <- as.matrix(patches.matrix.rot)
R> col <- c("black", "white")
```

Afterwards, we get the x and y limits (of the World) to use them for the image and draw the matrix as an image:

```
R> x.minmax <- NLReport("(list min-pxcor max-pxcor)")
R> y.minmax <- NLReport("(list min-pycor max-pycor)")
R> image(x.minmax[1]:x.minmax[2], y.minmax[1]:y.minmax[2],
+ patches.matrix.rot, col = col, xlab = "x", ylab = "y")
```

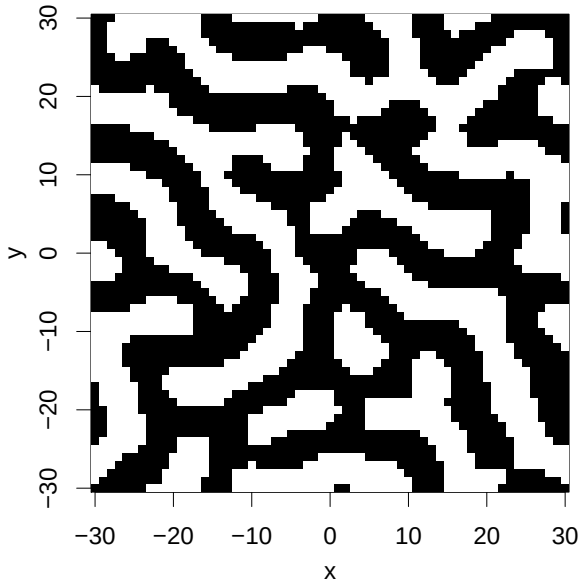


Figure III.14.: A simple visualization of the result of `NLGetPatches` as an image.

The code produced a simple raster image from the patches. It is also possible to create a spatial object from the result of `NLGetPatches` as we see in the next example, where packages gstat [Pebesma, 2004] and sp [Pebesma and Bivand, 2005] are used.

We start by loading the required packages and get the patches or, more precisely, the colours and coordinates of the patches:

```
R> library("sp", "gstat")
R> patches <- NLGetPatches(c("pxcor", "pycor", "pcolor"),
   "patches")
```

Next, we convert the patches `data.frame` to a `"SpatialPointsDataFrame"` and then use this `"SpatialPointsDataFrame"` to create a `"SpatialPixelsDataFrame"`:

```
R> coordinates(patches) <- ~ pxcor + pycor
R> gridded(patches) <- TRUE
```

Now, we convert `pcolor` to a factor, define the colours for the plot and create it (not shown here, similar to Figure III.14):

```
R> patches$pcolor <- factor(patches$pcolor)
R> col <- c("black", "white")
R> spplot(patches, "pcolor", col.regions = col, xlab = "x",
   ylab = "y")
```

We see that it is possible to get the whole NetLogo View. As we can see in its manual page, we can save the result of `NLGetPatches` into a list, matrix or, like here, into a data frame. Furthermore, we can reduce the patches to a subset, e.g., all patches that fulfil a condition, as we have done in the `NLGetAgentSet` example.

There are two other functions that operate the other way around. With `NLSetPatches` and `NLSetPatchSet` we can push an R matrix/data frame into the NetLogo patches. `NLSetPatches` function works only if we fill all patches, i.e., if we use a matrix which has the dimension of the NetLogo World. For filling just a subset of patches we can use the `NLSetPatchSet` function.

The following example shows the usage of the `NLSetPatches` function.

We reuse the `patches.matrix` variable from `NLGetPatches`, change the values from 0 (black) to 15 (red) and use this new matrix as input for the NetLogo patch variable `pcolor` (Figure III.15):

```
R> my.matrix <- replace(patches.matrix,
+ patches.matrix == 0,
+ 15)
R> NLSetPatches("pcolor", my.matrix)
```

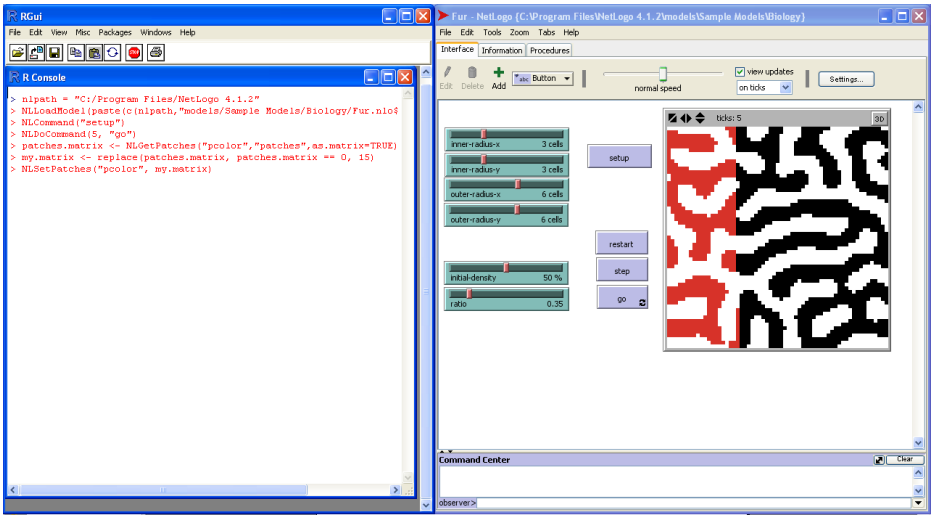


Figure III.15.: A screenshot while `NLSetPatches` is executing. The colour of the NetLogo patches on the right hand side is changed gradually from black to red.

Another function, `NLGetGraph`, makes it possible to get a NetLogo network built by NetLogo links into an igraph network. This function requires the R package igraph [Csárdi and Nepusz, 2006]. As an example, we can use the Small World model from NetLogo's Models Library. We build the NetLogo link network and transform it into an igraph network and finally plot it.

We start by loading as well as setting up the model and get the graph from NetLogo:

```
R> model.path <- file.path("models", "Sample Models",
   "Networks", "Small Worlds.nlogo")
```

```
R> NLLoadModel(file.path(nl.path, model.path))
R> NLCommand("setup", "rewire-all")
R> my.network <- NLGetGraph()
```

Now, the directed network graph plot (Figure III.16) can be obtained with:

```
R> plot(my.network, layout = layout.circle,
+ vertex.label = V(my.network)$name,
+ vertex.label.cex = 0.7,
+ asp = FALSE)
```

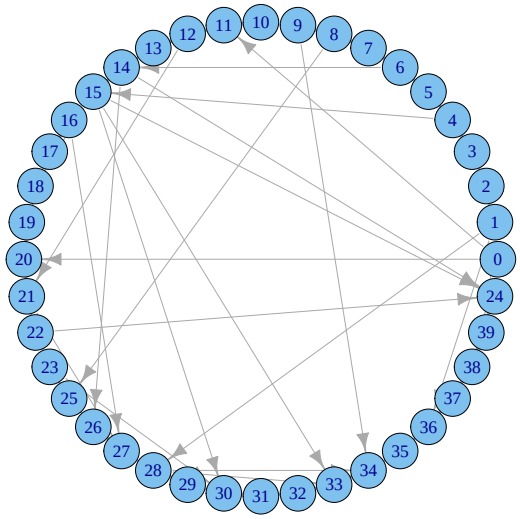


Figure III.16.: A graph generated by NetLogo links, sent to R via `NLGetGraph`, and plotted using the igraph package [Csárdi and Nepusz, 2006].

There are two further functions, which are not presented here in detail. The first one is the `NLSourceFromString` function, which enables us to create or append model source code from strings in R. A usage example is given in the code sample folder (No. 16) of the RNetLogo package. Another helper function to send a data frame into NetLogo lists is `NLDfToList`. The column names of the data frame have to be equivalent to the names of the lists in the NetLogo model. The code sample folder (No. 9) includes a usage example.

Headless mode/Multiple NetLogo instances

As mentioned above, it is possible to start NetLogo in background (headless mode) without a GUI. For this, we have to execute the `NLStart` function with a second argument. This will fail if we do not open a new R session (after using RNetLogo in GUI mode) because, as mentioned above, we cannot start several NetLogo sessions if we have already started one in GUI mode.

The `NLStart` function will save the NetLogo object reference in an internal variable in the local environment `.rnetlogo`. If we want to work with more than one NetLogo

model/instance at once, we can specify an identifier (as a string) for the NetLogo instance in the third argument of `NLStart`.

We start with the creation of three NetLogo instances (maybe beside the one with the default identifier which is `_nl.intern_`):

```
R> my.netlogo1 <- "my.netlogo1"
R> NLStart(nl.path, gui = FALSE, nl.obj = my.netlogo1)
R> my.netlogo2 <- "my.netlogo2"
R> NLStart(nl.path, gui = FALSE, nl.obj = my.netlogo2)
R> my.netlogo3 <- "my.netlogo3"
R> NLStart(nl.path, gui = FALSE, nl.obj = my.netlogo3)
```

All functions presented until now take as last (optional) argument (`nl.obj`) a string which identifies a specific NetLogo instance created with `NLStart`. Therefore, we can specify which instance we want to use. When working in headless mode, the first thing to do is always to load a model. Executing a command or reporter without loading a model in headless mode will result in an error. Therefore, we load a model into all instances:

```
R> model.path <- file.path("models", "Sample Models", "Earth
   Science", "Fire.nlogo")
R> NLLoadModel(file.path(nl.path, model.path)), nl.obj =
   my.netlogo1)
R> NLLoadModel(file.path(nl.path, model.path), nl.obj =
   my.netlogo2)
R> NLLoadModel(file.path(nl.path, model.path), nl.obj =
   my.netlogo3)
```

Now, we will set up and run the models over different simulation times.
We run the first instance (`my.netlogo1`) for 25 ticks:

```
R> NLCommand("setup", nl.obj = my.netlogo1)
R> NLDoCommand(25, "go", nl.obj = my.netlogo1)
```

Then, we run the second instance (`my.netlogo2`) for 15 ticks:

```
R> NLCommand("setup", nl.obj = my.netlogo2)
R> NLDoCommand(15, "go", nl.obj = my.netlogo2)
```

and we simulate 5 ticks with the third instance:

```
R> NLCommand("setup", nl.obj = my.netlogo3)
R> NLDoCommand(5, "go", nl.obj = my.netlogo3)
```

To check if the above worked well, we compare the number of burned trees in the different instances, which should be different:

```
R> NLReport("burned-trees", nl.obj = my.netlogo1)
```

Output:

```
[1] 1289
```

```
R> NLReport("burned-trees", nl.obj = my.netlogo2)
```

Output:

```
[1] 1067
```

```
R> NLReport("burned-trees", nl.obj = my.netlogo3)
```

Output:

```
[1] 413
```

At the end, we quit the NetLogo sessions (the standard session with internal identifier `_nl.intern_` as well, if open):

```
R> NLQuit(nl.obj = my.netlogo3)
R> NLQuit(nl.obj = my.netlogo2)
R> NLQuit(nl.obj = my.netlogo1)
R> NLQuit()
```

III.4.5. Application examples

The following examples are (partly) inspired by the examples presented for NetLogo-Mathematica Link [see Bakshy and Wilensky, 2007a]. These are all one-directional examples (from NetLogo to R), but the package opens the possibility of letting NetLogo and R interact and send back results from R (e.g., statistical analysis) to NetLogo and let the model react to them. Even manipulation of the model source by using the `NLSourceFromString` function is possible. This opens the possibility to generate NetLogo code from R dynamically.

Exploratory analysis

A simple parameter sensitivity experiment illustrates exploratory analysis with RNetLogo, even though NetLogo has a very powerful built-in tool, BehaviorSpace [Shargel and Wilensky, 2002], for this simple kind of experiment. Here, we will use the Forest Fire model [Wilensky, 1997b] from NetLogo's Models Library and explore the effect of the density of trees in the forest on the percentage of burned trees as described in Bakshy and Wilensky [2007a].

We start, as always, by loading and initializing the package (if not already done) and model:

```
R> library("RNetLogo")
R> nl.path <- file.path(getwd(), "NetLogo 5.0.5")
R> NLStart(nl.path, gui = FALSE)
R> model.path <- file.path("models", "Sample Models", "Earth
   Science", "Fire.nlogo")
R> NLLoadModel(file.path(nl.path, model.path))
```

Next, we define a function which sets the density of trees, executes the simulation until no turtles are left, and reports back the percentage of burned trees:

```
R> sim <- function(density) {
+ NLCommand("set density ", density, "setup")
+ NLDoCommandWhile("any? turtles", "go");
+ ret <- NLReport("(burned-trees / initial-trees) * 100")
+ return(ret)
+ }
```

We run the simulation for density values between 1 and 100 with a step size of 1, to identify the phase transition (Figure III.17):

```
R> d <- seq(1, 100, 1)
R> pb <- sapply(d, function(dens) sim(dens))
R> plot(d, pb, xlab = "density", ylab = "percent burned")
```

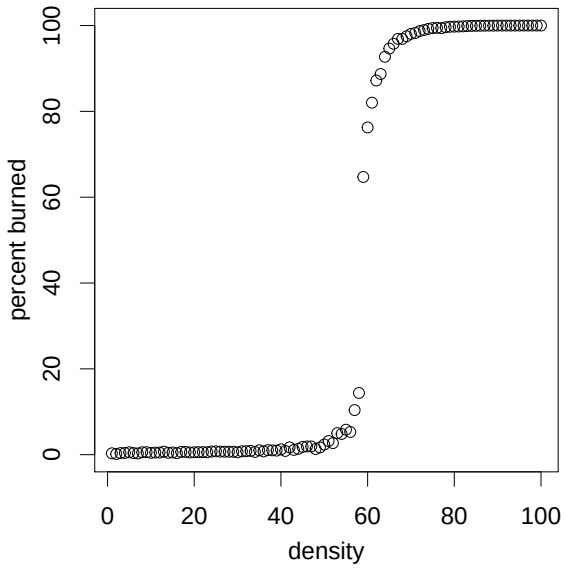


Figure III.17.: Results of the Forest Fire model varying the density of trees. The y-axis is the percentage of burned trees after no burning patches (i.e., no turtles) were left in the simulation.

As we know the region of phase transition (between a density of 45 and 70 percent), we can explore this region more precisely. As the Forest Fire model uses random numbers, it is interesting to find out how much stochastic variation occurs in this region. Therefore, we define a function to repeat the simulations with one density several times:

```
R> rep.sim <- function(density, rep)
+ lapply(density, function(dens) replicate(rep, sim(dens)))
```

To get a rough overview we use this new function for densities between 45 and 70 percent with a step size of 5, and 10 replications each (Figure III.18):

```
R> d <- seq(45, 70, 5); res <- rep.sim(d, 10)
R> boxplot(res, names = d, xlab = "density", ylab = "percent
   burned")
```

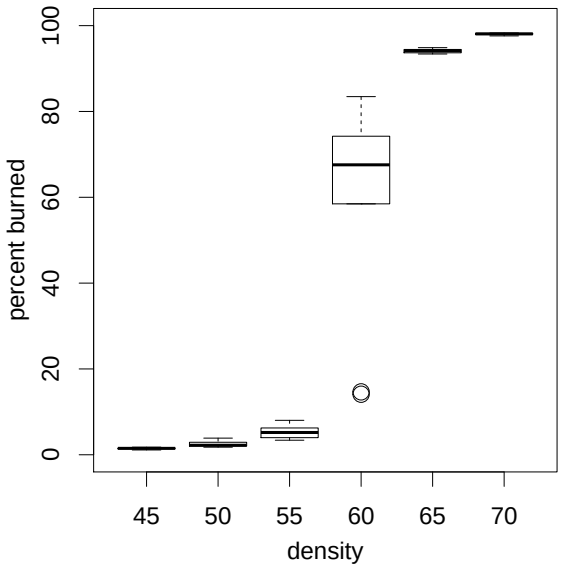


Figure III.18.: Boxplots of repeated simulations (10 replications) with the Forest Fire model with varying density (45-70 percent) of trees and the percentage of burned trees after no turtles were left in the simulation.

Now, we have seen that the variation of burned trees at densities below 55 and higher than 65 is low. As a result, we can skip these values and have a deeper look into the region of density values between 55 and 65. Therefore, we perform a simulation experiment for this value range with a smaller step size of 1 percent and a higher amount of replication of 20 per density value (Figure III.19):

```
R> d <- seq(55, 65, 1)
R> res <- rep.sim(d, 20)
R> boxplot(res,names = d, xlab = "density", ylab = "percent
   burned")
```

Database connection

There are R packages available to connect R to all common database management systems, e.g., RMySQL [James and DebRoy, 2012], RPostgreSQL [Conway et al., 2012], ROracle [Mukhin et al., 2013], RJDBC [Urbanek, 2013a], RSQLite [James, 2013] or RODBC [Ripley, 2013]. Thus the RNetLogo package opens the possibility to store the simulation results into a database.

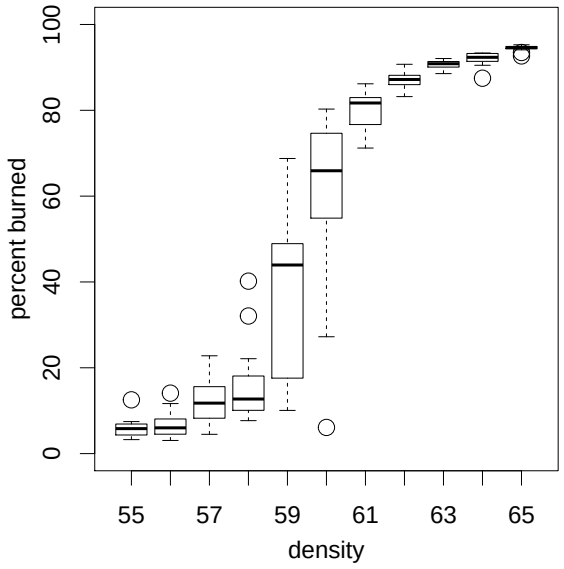


Figure III.19.: Boxplots of repeated simulations (20 replications) with the Forest Fire model with varying density (55-65 percent) of trees and the percentage of burned trees after no turtles were left in the simulation.

In the following example we use the RSQLite package [James, 2013], which provides a connection to SQLite databases [Hipp, 2012], because this is a very easy-to-use database in a single file. It does not need a separate database server and is, therefore, ideal for agent-based modelling studies, where no large database management systems (DBMS) are used. The database can store the results of different simulation experiments in different tables together with metadata in one file. This makes it very easy to share simulation results. There are small and easy-to-use GUI-programs available to browse and edit SQLite databases; see, for example, the SQLite Database Browser [Piacentini, 2013].

In a first step we have to set up the connections to NetLogo (if not already done) and load as well as initialize the example model:

```
R> library("RNetLogo")
R> nl.path <- file.path(getwd(), "NetLogo 5.0.5")
R> NLStart(nl.path, gui = FALSE)
R> model.path <- file.path("models", "Sample Models", "Earth
   Science", "Fire.nlogo")
R> NLLoadModel(file.path(nl.path, model.path))
R> NLCommand("setup")
```

Then, we load the required RSQLite package and database driver as well as create a connection to the database. If the database does not exist, this creates a file `test_netlogo.db`:

```
R> library("RSQLite")
R> m <- dbDriver("SQLite")
R> database.path = "test_netlogo.db"
R> con <- dbConnect(m, dbname = database.path)
```

Next, we run the model for ten ticks and save the results (ticks and burned-trees) in the table `Burned1` of the database:

```
R> dbWriteTable(con, "Burned1",
+ NLDoReport(10, "go", c("ticks", "burned-trees"),
+ as.data.frame = TRUE, df.col.names = c("tick", "burned")),
+ row.names = FALSE, append = FALSE)
```

Afterwards, we can send a first query: how many lines has the new table?

```
R> dbGetQuery(con, "select count(*) from Burned1")[[1]]
```

Output:

```
[1] 10
```

In the second query, we select all rows from table `Burned10` where tick is greater than 5:

```
R> rs <- dbSendQuery(con, "select * from Burned1 where tick >
   5")
```

Then, we ask for the result of the query and print it:

```
R> data <- fetch(rs, n = -1)
R> str(data)
```

Output:

```
'data.frame': 5 obs. of 2 variables:
 $ tick : num 6 7 8 9 10
 $ burned: num 547 606 665 716 757
```

Next, we delete/clear the query:

```
R> dbClearResult(rs)
```

Afterwards, we append further results to the existing table:

```
R> dbWriteTable(con, "Burned1",
+ NLDoReport(10, "go", c("ticks", "burned-trees"),
+ as.data.frame = TRUE, df.col.names = c("tick", "burned")),
+ row.names = FALSE, append = TRUE)
```

and take a look at the table:

```
R> select.all <- dbGetQuery(con, "select * from Burned1")
R> str(select.all)
```

Output:

```
'data.frame': 20 obs. of 2 variables:
 $ tick : num 1 2 3 4 5 6 7 8 9 10 ...
 $ burned: num 141 227 319 398 471 547 606 665 716 757 ...
```

Now, we create a second table and save the results of ten repeated simulations of 20 ticks each:

```
R> for (x in 1:10)
+ {
+ NLCommand("setup")
+ dbWriteTable(con, "Burned2",
+ NLDoReport(20, "go", c("ticks", "burned-trees"),
+ as.data.frame = TRUE, df.col.names = c("tick", "burned")),
+ row.names = FALSE, append = TRUE)
+ }
```

and calculate the mean number of burned trees (out of the 10 repetitions) for each tick, get the result of the query and show it:

```
R> rs <- dbSendQuery(con, "select avg(burned) as mean_burned
+ from Burned2 group by tick")
R> data <- fetch(rs, n = -1)
R> str(data)
```

Output:

```
'data.frame': 20 obs. of 1 variable:
 $ mean_burned: num 146 228 309 381 447 ...
```

Finally, we delete/clear the query and close the connection to the database:

```
R> dbClearResult(rs)
R> dbDisconnect(con)
```

Note that there is also an extension to connect databases directly to NetLogo (see http://code.google.com/p/netlogo-sql/).

Analytical comparison

The example application of Bakshy and Wilensky [2007a] compares results of an agent-based model of gas particles to velocity distributions found by analytical treatments of ideal gases. To reproduce this, we use the Free Gas model [Wilensky, 1997a] of the GasLab model family from NetLogo's Models Library. In this model, gas particles move and collide with each other without external constraints. Bakshy and Wilensky [2007a] compared this model's results to the classical Maxwell-Boltzmann distribution. R itself is not symbolic mathematical software but there are packages available which let us integrate such software. Here, we use the Ryacas package [Goedman et al., 2012] which is an interface to the open-source Yacas Computer Algebra System [Pinkus et al., 2007].

We start with the agent-based model simulation. Because this model is based on random numbers we run repeated simulations.

We start with loading and initializing the RNetLogo package (if not already done) and the model:

```
R> library("RNetLogo")
R> nl.path <- file.path(getwd(), "NetLogo 5.0.5")
R> NLStart(nl.path, gui = FALSE)
R> model.path1 <- file.path("models", "Sample Models",
   "Chemistry & Physics", "GasLab")
R> model.path2 <- "GasLab Free Gas.nlogo"
R> NLLoadModel(file.path(nl.path, model.path1, model.path2))
R> NLCommand("set number-of-particles 500", "no-display",
   "setup")
```

Next, we run the simulation for 40 times of 50 ticks (= 2000 ticks), save the speed of the particles after every 50 ticks, and flatten the list of lists (one list for each of the 40 runs) to one big vector:

```
R> particles.speed <- NLDoReport(40, "repeat 50 [go]",
+ "[speed] of particles")
R> particles.speed.vector <- unlist(particles.speed)
```

To calculate the analytical distribution, we have to solve the following equations:

$$B(v) = v \cdot e^{(-m \cdot v)^2 \cdot (2 \cdot k \cdot T)^{-1}} \tag{III.1}$$

$$normalizer = \int_0^\infty B(v)\, dv \tag{III.2}$$

$$B(v)_{normalized} = \frac{B[v]}{normalizer} \quad for\ v = [0, \max(speed)] \tag{III.3}$$

Now, Yacas/Ryacas will be used. For this, we define Equation III.1 with the mean energy derived from the NetLogo simulation. We then define the normalizer integral and solve it numerically.

We start by loading the Ryacas package:

```
R> library("Ryacas")
```

We can install Yacas, if currently not installed (only for Windows - see Ryacas/Yacas documentation for other systems) with:

```
R> yacasInstall()
```

Next, we get the mean energy from the NetLogo simulation and define the function `B` and register it in Yacas:

```
R> energy.mean <- NLReport("mean [energy] of particles")
R> B <- function(v, m = 1, k = 1)
+ v * exp((-m * v^2) / (2 * k * energy.mean))
R> yacas(B)
```

Then, we define the integral of function `B` from 0 to infinity and register the integral expression in Yacas:

```
R> B.integr <- expression(integrate(B, 0, Infinity))
R> yacas(B.integr)
```

Now, we calculate a numerical approximation using Yacas's function `N()` and get the result from Yacas in R (the result is in the list element `value`):

```
R> normalizer.yacas <- yacas(N(B.integr))
R> normalizer <- Eval(normalizer.yacas)
R> print(normalizer$value)
```

Output:

```
[1] 50
```

In a further step, we calculate the theoretical probability values of particle speeds using Equation III.1. We do this from 0 to the maximum speed observed in the NetLogo simulation.

First, we get the maximum speed from the NetLogo simulation:

```
R> maxspeed <- max(particles.speed.vector)
```

Next, we create a sequence vector from 0 to `maxspeed`, by `stepsize`, and calculate the theoretical values at the points of the sequence vector:

```
R> stepsize <- 0.25
R> v.vec <- seq(0, maxspeed, stepsize)
R> theoretical <- B(v.vec) / normalizer$value
```

At the end, we plot the empirical/simulation distribution together with the theoretical distribution of particle speeds (Figure III.20):

```
R> hist(particles.speed.vector, breaks =
   max(particles.speed.vector) * 5,
+ freq = FALSE, xlim = c(0, as.integer(maxspeed) + 5),
+ ylab = "density", xlab = "speed of particles", main = "")
R> lines(v.vec, theoretical, lwd = 2, col = "blue")
```

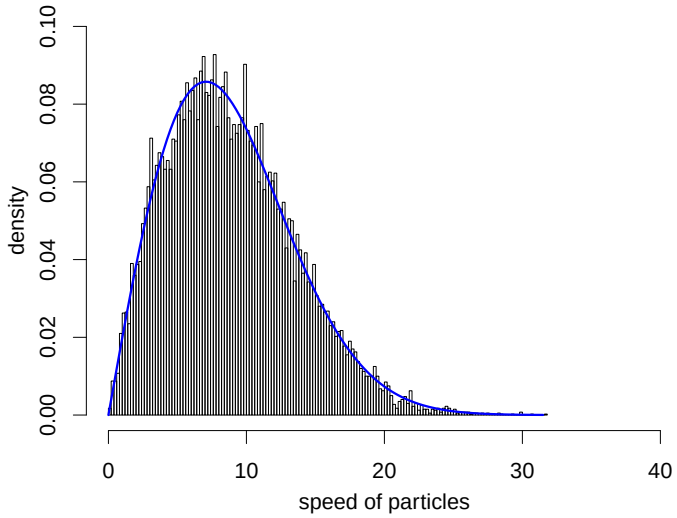


Figure III.20.: Empirical probability distribution of particle speeds generated by the agent-based model (bars) with the theoretical Maxwell-Boltzmann distribution (blue line).

Advanced plotting functionalities

R and its packages deliver a wide variety of plotting capabilities. As an example, we present a three-dimensional plot in combination with a contour map. We use the "Urban Site - Sprawl Effect" model [Felsen and Wilensky, 2007] from NetLogo's Models Library. This model simulates the growth of cities and urban sprawl. Seekers (agents) look for patches with high attractiveness and also increase the attractiveness of the patch they stay on. Therefore, the attractiveness of the patches is a state variable of the model, which can be plotted in R.

First, we initialize the RNetLogo package (if not already done) and load the model:

```
R> library("RNetLogo")
R> nl.path <- file.path(getwd(), "NetLogo 5.0.5")
R> NLStart(nl.path, gui = FALSE)
R> model.path <- file.path("models", "Curricular Models",
   "Urban Suite")
R> model.name <- "Urban Suite - Sprawl Effect.nlogo"
R> NLLoadModel(file.path(nl.path, model.path, model.name))
```

We resize NetLogo's World and set the parameter values:

```
R> NLCommand("resize-world -20 20 -20 20")
R> NLCommand("set smoothness 10",
+ "set max-attraction 5",
+ "set population 500",
+ "set seeker-search-angle 200",
+ "set seeker-patience 15",
+ "set wait-between-seeking 5")
```

Then, we set up the simulation and run it for 150 ticks:

```
R> NLCommand("setup")
R> NLDoCommand(150, "go")
```

Next, we get the value of the variable `attraction` from all patches as a matrix as well as the dimensions of NetLogo's World:

```
R> attraction <- NLGetPatches("attraction", as.matrix = TRUE)
R> pxcor <- NLReport(c("min-pxcor", "max-pxcor"))
R> pycor <- NLReport(c("min-pycor", "max-pycor"))
```

Now, we define the advanced plotting function with a three-dimensional plot and a contour map (adapted from Francois, 2011):

```
R> kde2dplot <- function(d, ncol = 50, zlim = c(0, max(z)),
+ nlevels = 20, theta = 30, phi = 30)
+ {
+ z <- d$z
+ nrz <- nrow(z)
+ ncz <- ncol(z)
+ colors <- tail(topo.colors(trunc(1.4 * ncol)), ncol)
+ fcol <- couleurs[trunc(z / zlim[2] * (ncol - 1)) + 1]
+ dim(fcol) <- c(nrz, ncz)
+ fcol <- fcol[-nrz, -ncz]
+ par.default <- par(no.readonly = TRUE)
+ par(mfrow = c(1, 2), mar = c(0, 0, 0, 0), cex = 1.5)
+ persp(d, col = fcol, zlim = zlim, theta = theta, phi = phi,
+ zlab = "attraction", xlab = "x", ylab = "y")
+
+ par(mar = c(2, 2, 0.5, 0.5))
+ image(d, col = colors)
+ contour(d, add = TRUE, nlevels = nlevels)
+ box()
+ par(par.default)
+ }
```

We merge the data and execute the plot function (Figure III.21):

```
R> d <- list(x = seq(pxcor[[1]], pxcor[[2]]),
+ y = seq(pycor[[1]], pycor[[2]]),
+ z = attraction)
R> kde2dplot(d)
```

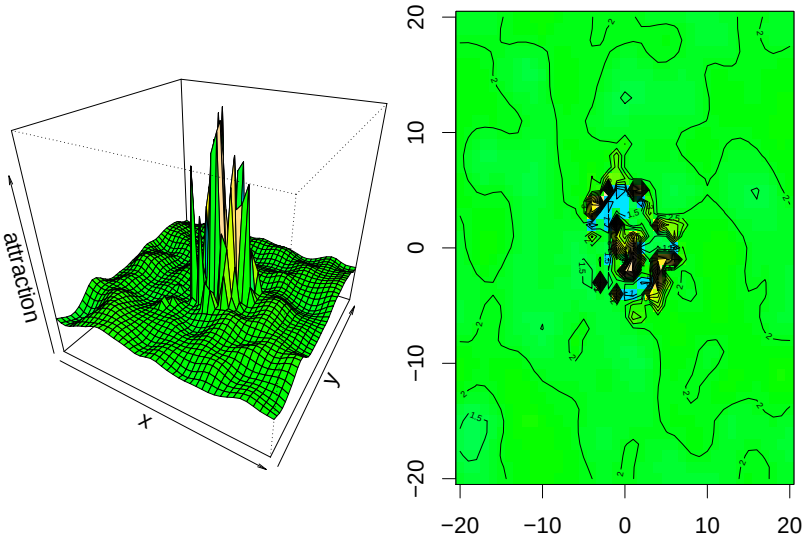


Figure III.21.: Spatial distribution of attractiveness of patches after 150 simulation steps. 3D plot (left) and contour plot (right).

Time sliding visualization

As agent-based models are often very complex, more than three dimensions could be relevant for their analysis. With the RNetLogo package it is possible to save the output of a simulation in R for every tick and then click through, or animate, the time series of these outputs, for example a combination of the model's View and distributions of state variables. As a prototype, we write a function to implement a timeslider to plot turtles. This function can be extended to visualize a panel of multiple plots by tick. With a slider we can browse through the simulation steps. To give an example, we use the Virus model [Wilensky, 1998] from NetLogo's Models Library to visualize the spatial distribution of infected and immune agents as well as boxplots of the time period of infection and the age in one plot panel.

We first load the required package rpanel [Bowman et al., 2007] and define a helper function to set the plot colours for the logical variables (sick, immune) of the turtles:

```
R> library("rpanel")
R> color.func <- function(color.var, colors, timedata) {
+ color <- NULL
+ if (!is.null(color.var)) {
+ index.color <- which(names(timedata) == color.var)
+ color <- timedata[[index.color]]
+ color[color == FALSE] <- colors[1]
+ color[color == TRUE] <- colors[2]
+ }
+ return(color)
+ }
```

Next, we define the main function containing the slider and what to do if we move the slider. The input is a list containing data frames for every tick. When the slider is moved, we send the current position of the slider (i.e., the requested tick) to the plotting function,

extract the corresponding data frame from the `timedata` list and draw a panel of four plots using this data frame.

```
R> plottimedata <- function(timedata.list, x.var, y.var,
   boxplot.var1,
+ boxplot.var2, color.var1 = NULL,
+ colors1 = "black", color.var2 = NULL,
+ colors2 = "black", mains = NULL, ...)
+ {
+ timeslider.draw <- function(panel) {
+ index.x <- which(names(timedata.list[[panel$t]]) == x.var)
+ index.y <- which(names(timedata.list[[panel$t]]) == y.var)
+ index.b1 <- which(names(timedata.list[[panel$t]]) ==
   boxplot.var1)
+ index.b2 <- which(names(timedata.list[[panel$t]]) ==
   boxplot.var2)
+
+ color1 <- color.func(color.var1, colors1,
   timedata.list[[panel$t]])
+ color2 <- color.func(color.var2, colors2,
   timedata.list[[panel$t]])
+
+ par(mfrow = c(2, 2), oma = c(0, 0, 1, 0))
+ plot(timedata.list[[panel$t]][[index.x]],
+ timedata.list[[panel$t]][[index.y]],
+ col = color1, main = mains[1], ...)
+ plot(timedata.list[[panel$t]][[index.x]],
+ timedata.list[[panel$t]][[index.y]],
+ col = color2, main = mains[2], ...)
+ boxplot(timedata.list[[panel$t]][[index.b1]], main =
   mains[3])
+ boxplot(timedata.list[[panel$t]][[index.b2]], main =
   mains[4])
+ title(paste("at time ",panel$t), outer = TRUE)
+ panel
+ }
+ panel <- rp.control()
+ rp.slider(panel, resolution = 1, var = t, from = 1,
+ to = length(timedata.list), title = "Time",
+ showvalue = TRUE, action = timeslider.draw)
+ }
```

In the third step, we initialize and run the NetLogo simulation and collect the results into the `timedata` list. Here, we run 100 ticks and use the `NLGetAgentSet` function to collect data from the turtles.

```
R> library("RNetLogo")
R> nl.path <- file.path(getwd(), "NetLogo 5.0.5")
```

```
R> model.path <- file.path("models", "Sample Models",
   "Biology", "Virus.nlogo")
R> NLStart(nl.path)
R> NLLoadModel(file.path(nl.path, model.path))
R> NLCommand("setup")
R> nruns <- 100
R> timedata <- list()
R> for(i in 1:nruns) {
+ NLCommand("go")
+ timedata[[i]] <- NLGetAgentSet(c("who", "xcor", "ycor",
   "age",
+ "sick?", "immune?", "sick-count"),
+ "turtles")
+ }
```

In the last step, we collect the dimension of the NetLogo World to use it for the axis extent of the plot and define the colours used for the variables `sick` (`green = FALSE, red = TRUE`) and `immune` (`red = FALSE, green = TRUE`). Finally, we call the above-defined `plottimedata` function to create the timeslider.

```
R> world.dim <- NLReport(c("(list min-pxcor max-pxcor)",
+ "(list min-pycor max-pycor)"))
R> colors1 <- c("green", "red")
R> colors2 <- c("red", "green")
R> plottimedata(timedata.list = timedata, x.var = "xcor",
   y.var = "ycor",
+ xlab = "x", ylab = "y", color.var1 = "sick?",
+ color.var2 = "immune?", boxplot.var1 = "sick-count",
+ boxplot.var2 = "age", colors1 = colors1, colors2 = colors2,
+ mains = c("Sick", "Immune", "Sick-count", "Age"),
+ xlim = world.dim[[1]], ylim = world.dim[[2]])
```

Then we can move the slider and the plot is updated immediately (Figure III.22).

III.4.6. Pitfalls

Amount of data

Please note that we are not able to stop the execution of a NetLogo command without closing our R session. Therefore, it is a good idea to think about the amount of data which should be transformed. For example, if we use the `NLGetPatches` function with the standard settings of the Forest Fire model from NetLogo's Models Library, we are requesting 63001 patch values. If we ask for the pxcor, pycor and pcolor values, we are requesting for $63001 \cdot 3 = 189003$ values. All these values have to be transformed from NetLogo data type to Java and from Java to R. This may take a while. For technical reasons, we are not informed about the progress of data transformation. Therefore, it looks like the program crashed, but if we are patient, the program will return with the result after some time. That's why it is always a good idea to test our code with a very small example (i.e., small worlds, low number of agents etc.). As mentioned in Section III.4.4, NetLogo 5.0.x is much faster at transferring data than NetLogo 4.x.

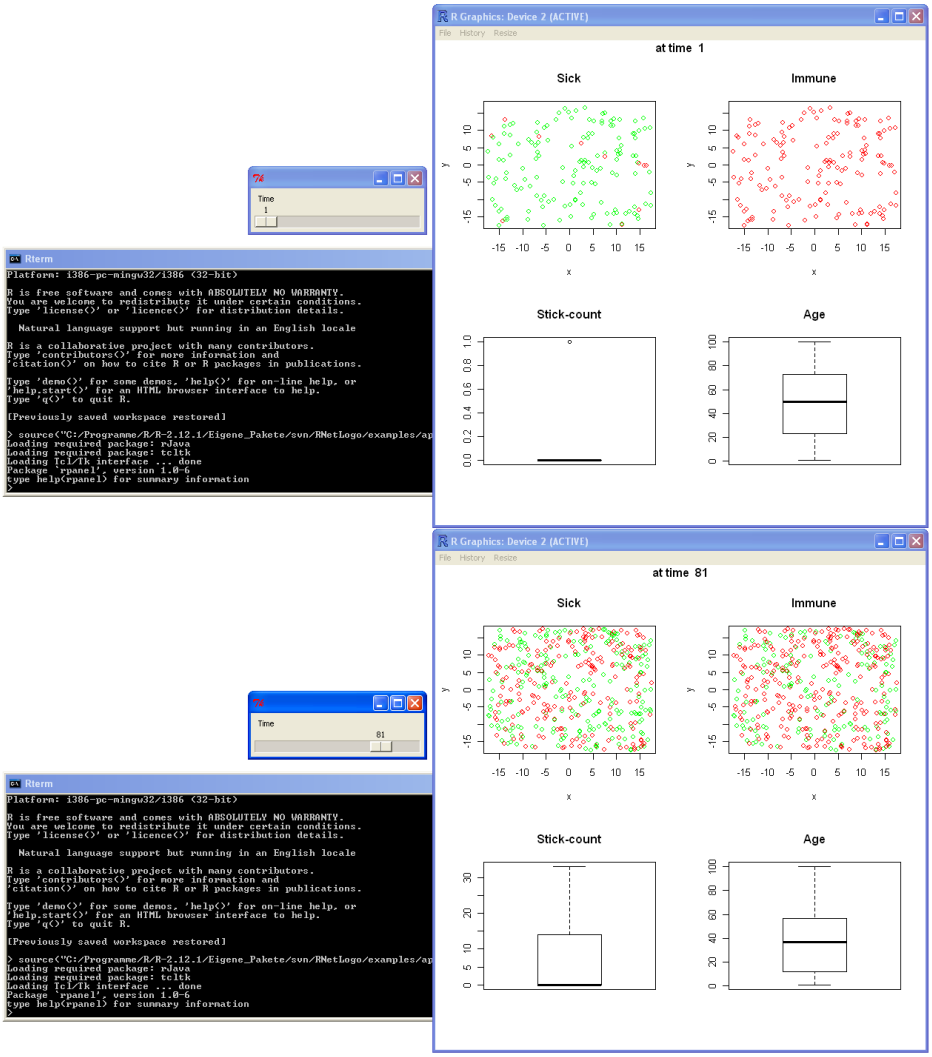


Figure III.22.: Timeslider example using the Virus model.

Table III.5.: Mapping from NetLogo data types to R data types.

NetLogo	R
Boolean	Boolean
String	String
Number	Double
List of strings	Vector of strings
List of booleans	Vector of booleans
List of numbers	Vector of doubles
Nested list (one nesting)	List of vectors
Nested list (multi-level nesting)	List of lists, lowest level: vectors

Endless loops

If we use the functions `NLDoCommandWhile` and `NLDoReportWhile`, we should double check our while-condition. Are we sure that the condition will be met some time? To prevent endless loops, these functions take an argument `max.minutes` with a default value of 10. This means that the execution of these functions will be interrupted if it takes longer than the submitted number of minutes. If we are sure that we do not submit something that will trigger an endless loop, we can switch off this functionality by using a value of 0 for the `max.minutes` argument. This will speed up the operation because the time checking operation will not be applied.

Data type

The general mapping of NetLogo data types to R data types in RNetLogo is given in Table III.5.

We should think about the data types we are trying to combine. For example, an R vector takes values of just one data type (e.g., string, numeric/double or logical/boolean) unlike a NetLogo list, which can contain different data types. Here are some examples.

First, we get a NetLogo list of numbers:

```
R> NLReport("(list 24 23 22)")
```

Second, we get a NetLogo list of strings:

```
R> NLReport("(list \"foo1\" \"foo2\" \"foo3\")")
```

Third, we try to get a NetLogo list of combined numbers and string:

```
R> NLReport("(list 24 \"foo\" 22)")
```

The first two calls of `NLReport` will run as expected but the last call will throw an error, because `NLReport` tries to transform a NetLogo list into an R vector, which will fail due to the mixed data types. This is also relevant for the columns of `data.frames`.

Table III.6.: Examples of results of `NLDoReport` with different NetLogo data structures. The Forest Fire model is used with a world of only 3 x 3 patches and a density of 99 percent. The model is reset before each example.

Call	Output of `str(<Call>)`

`NLDoReport(2, "go",`
`"(list count fires count embers)")`
	`List of 2`
	`$ : num [1:2] 2 2`
	`$ : num [1:2] 0 4`

`NLDoReport(2, "go",`
`c("count fires", "count embers"))`
	`List of 2`
	`$ :List of 2`
	`..$ : num 2`
	`..$ : num 2`
	`$ :List of 2`
	`..$ : num 0`
	`..$ : num 4`

Data structure

Since RNetLogo does not restrict how NetLogo reporters are combined, it is very flexible but makes it necessary to think very carefully about the data structure that will be returned. How a NetLogo value is transformed in general is already defined in Table III.5.

But this becomes more complex for iteration functions like `NLDoReport` where the return values of one iteration are combined with the results of another iteration, especially when requesting the result as a data frame instead of a list.

For example, it makes a difference in the returned data structure when we request two values as a NetLogo list or as two single reporters in a vector (Table III.6). Requesting the values as a NetLogo list returns a top-level list containing a vector of two values for all requested iterations. Requesting two single reporters returns these in a list as an entry of a top-level list. Therefore, this results in a nested list structure. There is not a wrong or preferred solution, it just depends on what we want to do with the result.

Requesting the result of `NLDoReport` as a data frame converts the top-level list to a data frame in a way that the top-level list entries become columns of the data frame and one iteration is represented by a row. This becomes problematic when nested NetLogo lists are requested (Table III.7). In such a case, the nested NetLogo lists are transformed into R lists and the resulting data frame contains lists in its columns. Such a data structure is a valid, but uncommon, data frame and some functions, like `write.table`, can operate only with a data frame that contains just simple objects in its columns. To make a data frame with nested lists fit for functions like `write.table` we have to use the `I(x)` function for the affected columns to treat them "as is" (see `help(I)` for details, e.g., `my.df$col1 <- I(my.df$col1)`).

Furthermore, using an agentset in an `NLDoReport` iteration with data frame return value can become problematic. As long as the number of members of the agentset does not change, it can be requested without problems in a data frame. The data frame contains one column for each agent and one row for each iteration. If the number of agents changes during the iterations the resulting data frame is not correct as it contains entries that do not exist. The number of columns equals the maximum number of agents over all iterations.

Table III.7.: Examples of results of `NLDoReport` with different NetLogo data structures. The Forest Fire model is used with a world of only 3 x 3 patches and a density of 99 percent. The model is reset before each example.

Call	Output of `str(<Call>)`

`NLDoReport(2, "go",`
`"(list count fires count embers)",`
`as.data.frame=TRUE)`
	`'data.frame':`
	`2 obs. of 2 variables:`
	`$ X1: num 0 0`
	`$ X2: num 4 4`

`NLDoReport(2, "go",`
`c("count fires", "count embers"),`
`as.data.frame=TRUE)`
	`'data.frame':`
	`2 obs. of 2 variables:`
	`$ X1: num 0 0`
	`$ X2: num 4 4`

`NLDoReport(2, "go",`
`c("count turtles",`
`"(list count fires count embers)"),`
`as.data.frame=TRUE)`
	`'data.frame':`
	`2 obs. of 2 variables:`
	`$ X1: num 4 4`
	`$ X2:List of 2`
	`..$ : num 0 4`
	`..$ : num 0 4`

For those iterations that contain less agents the columns of the data frame are filled with copied information from a former column. In short, the information is wrong. The following example illustrates this. The Forest Fire model is used with a world of only 5 x 3 patches.

```
R> res <- NLDoReport(3, "go", "[who] of turtles",
   as.data.frame = TRUE)
R> str(res)
```

Output:

```
'data.frame': 3 obs. of 7 variables:
 $ X1: num 2 4 0
 $ X2: num 0 2 6
 $ X3: num 3 0 4
 $ X4: num 1 3 1
 $ X5: num 2 1 5
 $ X6: num 0 4 3
 $ X7: num 3 2 2
```

The first iteration contains four turtles, the second five and the third seven turtles. The returned data frame therefore contains seven columns. Entries in columns for the first and the second row (i.e., iteration) are repeated from the first columns. But fortunately we are warned by R that the length of the vectors differ. When we cannot be sure that the number of return values is always the same over the iterations we should use the default list data structure instead of the data frame return structure. Furthermore, if we want to request an agentset, we should better use the `NLGetAgentSet` function in an R loop, as shown in

Section III.4.5, because it returns the requested values in a sorted order; for agents by their `who` number and in case of patches from upper left to lower right.

These examples illustrate that it is necessary to think about the data structure that is required for further analyses and which function can process such a data structure.

Working directory

We should avoid changing the working directory of R manually, because NetLogo needs to have the working directory pointed to its installation path. As the R working directory and the Java working directory depend on each other, changing the R working directory can result in unexpected behaviour of NetLogo. Therefore, we should use absolute paths for I/O processes in R instead of submitting `setwd(...)`. Note that the RNetLogo package changes the working directory automatically when loading NetLogo and changes back to the former working directory when the last active NetLogo instance is closed with `NLQuit`.

III.4.7. Discussion

This article gave a theoretical and practical introduction to the RNetLogo package. The reader should be well-prepared to start his/her own projects based on RNetLogo after studying the examples. Since there are so many interesting packages available in R with connections to many other programs, it is really amazing what this connection offers to both, R users and NetLogo users.

Note that there are code samples for all functions in the example folder (`RNetLogo/examples/ code_samples`) of the RNetLogo package. Furthermore, there are some example applications in the example folder, similar to those presented here.

As presented the RNetLogo package successfully links the statistical computing environment R with the agent-based modelling platform NetLogo. Thereby it brings together the world of statistics and data analysis with the world of agent-based modelling. From the viewpoint of an R user it opens the possibility to access a rule-based modelling language and environment. Therefore, (nearly) all types of agent-based and system-dynamics models can be easily embedded into R. NetLogo's Models Library gives a nice impression of what kind of models can be built, from deterministic to stochastic, from non-spatial to spatial models, from 2D to 3D, from cellular automata over network models and artificial neural networks to L-Systems and many others more.

Bringing simulation models to R is not entirely new. There are, on the one hand, other modelling environments, like Repast [North et al., 2006], that open the possibility to send data to R. But the ability to control simulation experiments from R is new for such modelling tools. NetLogo was selected because it is very easy to learn, very well designed, and much better documented than other ABM platforms. It has a very active user community and seems to be the most appropriate basis for all kinds of modellers, from beginners to professionals and from ecology over social sciences to informatics. On the other hand, there are packages available to build simulation models directly in R, like simecol [Petzoldt and Rinke, 2007]. Especially simecol is fast and very flexible and a good choice in comparison to implementations in pure R but it does not provide specific support for making model development and simulation efficient as agent-based model environments like NetLogo and Repast do.

Some first use-cases of RNetLogo have been presented in this article. Beside the advanced

visualization possibilities and connections to other software an important application area is the design and analysis of simulation experiments in a systematic, less ad-hoc, way. R delivers all necessary functions of the design of experiments (DoE) principles. With RNetLogo the technical connection between all kinds of DoE and ABM is available.

There are already ready-to-use solutions for model analysis/DoE techniques available for agent-based modelling, like BehaviorSearch [Stonedahl and Wilensky, 2013], MEME [Iványi et al., 2007], and openMOLE [Reuillon et al., 2010], but they are less flexible and adaptable than R. Often, for one task, several packages in R are available and if not, writing own functions is flexible and fast, especially because many scientists know R already from its application for data analysis. Since RNetLogo does not restrict the user to predefined analysis functions it opens up a large flexibility. But RNetLogo can only check the submitted NetLogo code strings at runtime. This is a disadvantage, although the NetLogo code strings are typically simple and lack of automated checking encourages well-designed analysis. Nevertheless, RNetLogo requires the user to understand data types and structures of both NetLogo and R.

RNetLogo pushes the documentation and therefore the reproducibility of agent-based modelling studies, a key feature of science, to a new level. Using RNetLogo in conjunction with tools like Sweave [Leisch, 2002], odfWeave [Kuhn et al., 2012] or SWord [Baier, 2009] will contribute to replicability and reproducibility of agent-based simulation studies by automatic and self-documented report generation. For example, Sweave can embed R code in a LaTex text document. When compiling the Sweave document, the R code is evaluated and the results (not only numeric but also images) can be embedded automatically in the LaTex document. The RNetLogo package opens up the possibility to embed not only results of R, but also the result of a NetLogo simulation. We can create a self-documented report with NetLogo simulations and R analytics (with or without source code). For an example see the Sweave code of this article.

Since models become more complex their computational requirements are increasing as well. A lot of these requirements are compensated by increasing computational power, but the use of modern model development and analysis techniques for stochastic models, like Bayesian calibration methods, make a large number of repeated simulations necessary. Using RNetLogo includes, of course, an overhead when converting model results from NetLogo to R and vice versa, but there are already techniques available to spread such repetitions to multi-cores and computer clusters (see the RNetLogo package vignette Parallel Processing with RNetLogo, Appendix H).

To sum up, I expect that this contribution will make agent-based modelling with NetLogo more popular and easier in the R community and will support the methodological developments towards rigorous model development, testing and analysis in the ABM community.

III.4.8. Acknowledgements

I would like to thank two anonymous reviewers as well as Volker Grimm and Winfried Kurth for their very valuable comments on an earlier version of the manuscript.

III.4.9. References

J. Arifovic. Genetic Algorithm Learning and the Cobweb Model. *Journal of Economic Dynamics and Control*, 18(1):3–28, 1994.

T Baier. *SWordInstaller: SWord: Use R in Microsoft Word (Installer). R package version 1.0-2*, 2009. URL http://cran.r-project.org/src/contrib/Archive/SWordInstaller/. (last accessed 2014/01/06).

E. Bakshy and U. Wilensky. Turtle Histories and Alternate Universes: Exploratory Modeling with NetLogo and Mathematica. In M.J. North, c.M. Macal, and D.L. Sallach, editors, *Proceedings of the Agent 2007 Conference on Complex Interaction and Social Emergence*, pages 147–158. IL: Argonne National Laboratory and Northwestern University, 2007a.

E. Bakshy and U. Wilensky. *NetLogo-Mathematica Link*. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL, 2007b. URL http://ccl.northwestern.edu/netlogo/mathematica.html. (last accessed 2014/01/06).

F.C. Billari and A. Prskawetz. *Agent-Based Computational Demography: Using Simulation to Improve Our Understanding of Demographic Behaviour*. Contributions to Economics. Physica, 2003.

A. Bowman, E. Crawford, G. Alexander, and R.W. Bowman. rpanel: Simple Interactive Controls for R Functions Using the tcltk Package. *Journal Of Statistical Software*, 17(9): 1–23, 2007.

C. Carpenter and L. Sattenspiel. The Design and Use of an Agent-Basedased Model to Simulate the 1918 Influenza Epidemic at Norway House, Manitoba. *American Journal of Human Biology*, 21(3):290–300, 2009.

R. Conte. From Simulation to Theory (and Backward). In F. Squazzoni, editor, *Epistemological Aspects of Computer Simulation in the Social Sciences, Second International Workshop, EPOS 2006, Brescia, Italy, October 5-6*, volume 5466 of *Lecture Notes in Computer Science*, pages 29–47. Springer, 2006.

R. Conte, N. Gilbert, and J.S. Sichman. MAS and Social Simulation: A Suitable Commitment. In J.S. Sichman, R. Conte, and N. Gilbert, editors, *Multi-Agent Systems and Agent-Based Simulation, First International Workshop, MABS '98, Paris, France, July 4-6*, volume 1534 of *Lecture Notes in Computer Science*, pages 1–9. Springer, 1998.

J. Conway, D. Eddelbuettel, T. Nishiyama, S.K. Prayaga, and N. Tiffin. *RPostgreSQL: R Interface to the PostgreSQL Database System. R package version 0.4*, 2012. URL http://CRAN.R-project.org/package=RPostgreSQL. (last accessed 2014/01/06).

M.J. Crawley. *Statistics: An Introduction Using R*. John Wiley & Sons, 2005.

G. Csárdi and T. Nepusz. The igraph Software Package for Complex Network Research. *InterJournal*, Complex Systems:1695, 2006. URL http://igraph.sf.net. (last accessed 2014/01/06).

D.L. DeAngelis and W.M. Mooij. Individual-Based Modeling of Ecological and Evolutionary Processes. *Annual Review of Ecology, Evolution, and Systematics*, 36:147–168, 2005.

J.M. Epstein and R. Axtell. *Growing Artificial Societies: Social Science from the Bottom Up*. The Brookings Institution, Washington, DC, 1996.

M. Felsen and U. Wilensky. NetLogo Urban Suite - Sprawl Effect Model, 2007. URL http://ccl.northwestern.edu/netlogo/models/UrbanSuite-SprawlEffect. (last accessed 2014/01/06).

J. Ferrer, C. Prats, and D. López. Individual-Based Modelling: An Essential Tool for Microbiology. *Journal of Biological Physics*, 34(1-2):19–37, 2008.

R. Francois. 2D Kernel Density Estimator: Perspective Plot and Contour Plot, 2011. URL http://web.archive.org/web/20120706042750/http://addictedtor.free.fr/graphiques/RGraphGallery.php?graph=1. (last accessed 2014/01/06).

N. Gilbert. Simulation: A New Way of Doing Social Science. *American Behavioral Scientist*, 40:1485–1487, 1999.

N. Gilbert. *Agent-Based Models*. Quantitative Applications in the Social Sciences. Sage, Los Angeles, CA, 2007.

R. Goedman, G. Grothendieck, S. Højsgaard, and A. Pinkus. *Ryacas: R Interface to the Yacas Computer Algebra System. R package version 0.2-11*, 2012. URL http://CRAN.R-project.org/package=Ryacas. (last accessed 2014/01/06).

S. Green, L. Hurst, B. Nangle, P. Cunningham, F. Somers, and R. Evans. Software Agents: A Review. Technical Report TCD-CS-1997-06, Trinity College Dublin, Department of Computer Science, 1997.

A.F. Griffin and C. Stanish. An Agent-Based Model of Prehistoric Settlement Patterns and Political Consolidation in the Lake Titicaca Basin of Peru and Bolivia. *Structure and Dynamics*, 2:1–46, 2007.

V. Grimm. Ten Years of Individual-Based Modelling in Ecology: What Have We Learned and What Could We Learn in the Future? *Ecological Modelling*, 115:129–148, 1999.

V. Grimm and S.F. Railsback. *Individual-Based Modeling and Ecology*. Princeton University Press, Princeton, N.J., 2005.

S. Heckbert, T. Baynes, and A. Reeson. Agent-Based Modeling in Ecological Economics. *Annals of the New York Academy of Sciences*, 1185:39–53, 2010.

C. Hewitt. *Viewing Control Structures as Patterns of Passing Messages*. A.I.Memo 410. MIT Press, 1976.

D.R. Hipp. About SQLite, 2012. URL http://www.sqlite.org/about.html. (last accessed 2014/01/06).

M.D. Iványi, L. Gulyás, R. Bocsi, G. Szemes, and R. Mészáros. Model Exploration Module. In M.J. North, C.M. Macal, and C.L. Sallach, editors, *Proceedings of the Agent 2007 Conference on Complex Interaction and Social Emergence*, pages 207–215. IL: Argonne National Laboratory and Northwestern University, 2007.

D.A. James. *RSQLite: SQLite Interface for R. R package version 0.11-4*, 2013. URL http://CRAN.R-project.org/package=RSQLite. (last accessed 2014/01/06).

D.A. James and S. DebRoy. *RMySQL: R Interface to the MySQL Database. R package version 0.9-3*, 2012. URL http://CRAN.R-project.org/package=RMySQL. (last accessed 2014/01/06).

N.R. Jennings. On Agent-Based Software Engineering. *Artificial Intelligence*, 117:277–296, 2000.

R. Kabacoff. *R in Action*. Manning, 2010.

R.I. Kabacoff. *Quick-R: Acccessing the Power of R*, 2013. URL http://www.statmethods.net/. (last accessed 2014/01/06).

M. Kuhn, S. Weston, N. Coulter, P. Lenon, and Z. Otles. *odfWeave: Sweave Processing of Open Document Format (ODF) Files. R package version 0.8.2*, 2012. URL http://CRAN.R-project.org/package=odfWeave. (last accessed 2014/01/06).

B. LeBaron. Agent-Based Computational Finance: Suggested Readings and Early Research. *Journal of Economic Dynamics and Control*, 24(5-7):679–702, 2000.

F. Leisch. Sweave: Dynamic Generation of Statistical Reports Using Literate Data Analysis. In W. Härdle and B. Rönz, editors, *Compstat 2002 - Proceedings in Computational Statistics*, pages 575–580. Physica, 2002.

R. Leombruni and M. Richiardi. *Industry and Labor Dynamics: The Agent-based Computational Economics Approach: Proceedings of the Wild@ace2003 Workshop, Torino, Italy, 3-4 October 2003*. World Scientific, 2004.

M. Luck, P. McBurney, and C. Preist. *Agent Technology: Enabling Next Generation Computing (A Roadmap for Agent Based Computing)*. AgentLink, Southampton: University of Southampton, 2003.

M.W. Macy and R. Willer. From Factors to Actors: Computational Sociology and Agent-Based Modeling. *Annual Review of Sociology*, 28:143–166, 2002.

J.H. Maindonald. *Using R for Data Analysis and Graphics: Introduction, Code and Commentary*, January 2008. URL http://cran.r-project.org/doc/contrib/usingR.pdf. (last accessed 2014/01/06).

N. Malleson, A. Heppenstall, and L. See. Crime Reduction Through Simulation: An Agent-Based Model of Burglary. *Computers, Environment and Urban Systems*, 34(3):236–250, 2010.

R.B. Matthews, N.G. Gilbert, A. Roach, J.G. Polhill, and N.M. Gotts. Agent-Based Land-Use Models: A Review of Applications. *Landscape Ecology*, 22(10):1447–1459, 2007.

J.M. Moonen. *Multi-Agent Systems for Transportation Planning and Coordination*. ERIM Ph.D. Series Research in Management. Erasmus Research Institute of Management (ERIM), Erasmus University Rotterdam, 2009.

D. Mukhin, D.A. James, and J. Luciani. *ROracle: OCI Based Oracle Database Interface for R. R package version 1.1-10*, 2013. URL http://CRAN.R-project.org/package=ROracle. (last accessed 2014/01/06).

M.J. North, N.T. Collier, and J.R. Vos. Experiences Creating Three Implementations of the Repast Agent Modeling Toolkit. *ACM Transactions on Modeling and Computer Simulation*, 16(1):1–25, 2006.

M.J. North, C.M. Macal, J.S. Aubin, P. Thimmapuram, M.J. Bragen, J. Hahn, J. Karr, N. Brigham, M.E. Lacy, and D. Hampton. Multiscale Agent-Based Consumer Market Modeling. *Complexity*, 15(5):37–47, 2010.

R. Oh, S. Sanchez, T. Lucas, H. Wan, and M. Nissen. Efficient Experimental Design Tools for Exploring Large Simulation Models. *Computational & Mathematical Organization Theory*, 15:237–257, 2009.

E. Oliveira. Applications of Intelligent Agent-Based Systems. In *Proceedings of SBAI - Simpósium Brasileiro de Automação Inteligente. 1999: São Paulo*, pages 51–58, 1999.

Oracle. Java Native Interface, 2013. URL http://docs.oracle.com/javase/7/docs/technotes/guides/jni/index.html. (last accessed 2014/01/06).

W.J. Owen. *The R Guide*, 2010. URL http://cran.r-project.org/doc/contrib/Owen-TheRGuide.pdf. (last accessed 2014/01/06).

E.J. Pebesma. Multivariable Geostatistics in S: The gstat Package. *Computers & Geosciences*, 30:683–691, 2004. URL http://www.gstat.org/. (last accessed 2014/01/06).

E.J. Pebesma and R.S. Bivand. Classes and Methods for Spatial Data in R. *R News*, 5 (2), 2005. URL http://cran.r-project.org/doc/Rnews/. (last accessed 2014/01/06).

T. Petzoldt and K. Rinke. Simecol: An Object-Oriented Framework For Ecological Modeling in R. *Journal of Statistical Software*, 22 (9):1–31, 2007. URL http://www.jstatsoft.org/v22/i09. (last accessed 2014/01/06).

M. Piacentini. *SQLite Database Browser*, 2013. URL http://sourceforge.net/projects/sqlitebrowser/. (last accessed 2014/01/06).

A. Pinkus, S. Winitzki, and J. Niesen. *YACAS Computer Algebra System*, 2007. URL http://yacas.sourceforge.net/homepage.html. (last accessed 2014/01/06).

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2014a. URL http://www.R-project.org. (last accessed 2014/01/06).

R Core Team. *R Language Definition: Version 3.1.0 (2014-04-10) DRAFT*, 2014b. URL http://cran.r-project.org/doc/manuals/R-lang.pdf. (last accessed 2014/06/10).

R Core Team. *R Installation and Administration 3.1.0 (2014-04-10)*, 2014c. URL http://cran.r-project.org/doc/manuals/R-admin.pdf. (last accessed 2014/06/10).

S.F. Railsback and V. Grimm. *Agent-Based and Individual-Based Modeling: A Practical Introduction*. Princeton University Press, 2012.

R. Reuillon, F. Chuffart, M. Leclaire, T. Faure, N. Dumoulin, and D. Hill. Declarative Task Delegation in OpenMOLE. In W.W. Smari and J.P. McIntire, editors, *Proceedings of the 2010 International Conference on High Performance Computing and Simulation (HPCS)*, pages 55–62, Caen, France, 2010.

B. Ripley. *RODBC: ODBC Database Access. R package version 1.3-10*, 2013. URL http://CRAN.R-project.org/package=RODBC. (last accessed 2014/01/06).

RWiki. rJava - R/Java interface, 2006. URL http://rwiki.sciviews.org/doku.php?id=packages:cran:rjava. (last accessed 2014/01/06).

T.C. Schelling. Models of Segregation. *The American Economic Review*, 59(2):488–493, 1969.

B. Shargel and U. Wilensky. *BehaviorSpace*. Northwestern University, Evanston, IL, 2002. URL http://ccl.northwestern.edu/netlogo/docs/behaviorspace.html. (last accessed 2014/01/06).

W. Shen, Q. Hao, H.J. Yoon, and D.H. Norrie. Applications of Agent-Based Systems in

Intelligent Manufacturing: An Updated Review. *Advanced Engineering Informatics*, 20(4): 415–431, 2006.

F. Squazzoni. The Impact of Agent-Based Models in the Social Sciences After 15 Years of Incursions. *History of Economic Ideas*, XVIII/2010/2:197–233, 2010.

F. Stonedahl and U. Wilensky. BehaviorSearch, 2013. URL http://behaviorsearch.org. (last accessed 2014/01/06).

K.P. Sycara. Multiagent Systems. *AI Magazine*, 19(2):79–92, 1998.

L. Tesfatsion. Agent-Based Computational Economics: A Constructive Approach to Economic Theory. In L. Tesfatsion and K.L. Judd, editors, *Handbook of Computational Economics*, volume 2, chapter 16, pages 831–880. Elsevier, 2006.

J.C. Thiele. RNetLogo: Provides an Interface to the Agent-Based Modelling Platform NetLogo. R package version 1.0-0, 2014. URL http://CRAN.R-project.org/package=RNetLogo. (last accessed 2014/06/15).

J.C. Thiele and V. Grimm. NetLogo Meets R: Linking Agent-Based Models with a Toolbox for Their Analysis. *Environmental Modelling & Software*, 25(8):972–974, 2010.

J.C. Thiele, W. Kurth, and V. Grimm. Agent- and Individual-Based Modeling with NetLogo: Introduction and New NetLogo Extensions. In K. Römisch, A. Nothdurft, and U. Wunn, editors, *22. Tagung der Sektion Forstliche Biometrie und Informatik des Deutschen Verbandes Forstlicher Forschungsanstalten und der Arbeitsgemeinschaft Ökologie und Umwelt der Internationalen Biometrischen Gesellschaft - Deutsche Region, 20-21th September 2010 in Göttingen (Germany)*, Die Grüne Reihe, pages 68–101, 2011.

S. Tisue. Controlling API, 2012. URL https://github.com/NetLogo/NetLogo/wiki/Controlling-API. (last accessed 2014/01/06).

S. Urbanek. *RJDBC: Provides Access to Databases Through the JDBC Interface. R package version 0.2-3*, 2013a. URL http://CRAN.R-project.org/package=RJDBC. (last accessed 2014/01/06).

S. Urbanek. *rJava: Low-level R to Java Interface. R package version 0.9-6*, 2013b. URL http://CRAN.R-project.org/package=rJava. (last accessed 2014/01/06).

W.N. Venables and B.D. Ripley. *Modern Applied Statistics with S*. Springer, New York, 4th edition, 2002.

W.N. Venables, D.M. Smith, and R Core Team. An Introduction to R." Notes on R: A Programming Environment for Data Analysis and Graphics, Version 3.1.0 (2014-04-10),, 2014. URL http://cran.r-project.org/doc/manuals/R-intro.pdf. (last accessed 2014/06/10).

G. Weiss. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, Cambridge, MA, 1999.

U. Wilensky. NetLogo GasLab Free Gas Model, 1997a. URL http://ccl.northwestern.edu/netlogo/models/GasLabFreeGas. (last accessed 2014/01/06).

U. Wilensky. NetLogo Fire Model, 1997b. URL http://ccl.northwestern.edu/netlogo/models/Fire. (last accessed 2014/01/06).

U. Wilensky. NetLogo Virus Model, 1998. URL http://ccl.northwestern.edu/netlogo/

models/Virus. (last accessed 2014/01/06).

U. Wilensky. NetLogo. Center for Connected Learning and Computer-Based Modeling, 1999. URL http://ccl.northwestern.edu/netlogo. (last accessed 2014/01/06).

U. Wilensky. Netlogo user manual, version 5.0.5. center for connected learning and computer-based modeling, northwestern university, evanston, il., 2013. URL http://ccl.northwestern.edu/netlogo/docs/. (last accessed 2014/06/10).

U. Wilensky and W. Rand. *An Introduction to Agent-Based Modeling: Modeling Natural, Social and Engineered Complex Systems with NetLogo*. MIT Press, 2014. Forthcomming.

M.J. Wooldridge. *An Introduction to Multiagent Systems*. John Wiley & Sons, New York, NY, 2005.

A.F. Zuur, E.N. Ieno, and E. Meesters. *A Beginner's Guide to R*. Use R. Springer, 2009.

CHAPTER IV

Facilitating Parameter Estimation and Sensitivity Analysis of Agent-Based Models: A Cookbook Using NetLogo and R

This manuscript is published as: JC Thiele, W Kurth, and V Grimm [2014]. Facilitating Parameter Estimation and Sensitivity Analysis of Agent-Based Models: A Cookbook Using NetLogo and R. Journal of Artificial Societies and Social Simulation 17 (3) 11 <http://jasss.soc.surrey.ac.uk/17/3/11.html>.

Authorship

- Winfried Kurth supported the writing of the manuscript.
- Volker Grimm wrote parts of the introductions and discussions and supported the writing of the rest of the manuscript.

IV.1. Abstract

Agent-based models are increasingly used to address questions regarding real-world phenomena and mechanisms; therefore, the calibration of model parameters to certain data sets and patterns is often needed. Furthermore, sensitivity analysis is an important part of the development and analysis of any simulation model. By exploring the sensitivity of model output to changes in parameters, we learn about the relative importance of the various mechanisms represented in the model and how robust the model output is to parameter uncertainty. These insights foster the understanding of models and their use for theory development and applications. Both steps of the model development cycle require massive repetitions of simulation runs with varying parameter values. To facilitate parameter estimation and sensitivity analysis for agent-based modellers, we show how to use a suite of important established methods. Because NetLogo and R are widely used in agent-based modelling and for statistical analyses, we use a simple model implemented in NetLogo as an example, packages in R that implement the respective methods, and the RNetLogo package, which links R and NetLogo. We briefly introduce each method and provide references for further reading. We then list the packages in R that may be used for implementing the methods, provide short code examples demonstrating how the methods can be applied in R, and present and discuss the corresponding outputs. The Supplementary Material includes full, adaptable code samples for using the presented methods with R and NetLogo. Our overall aim is to make agent-based modellers aware of existing methods and tools for parameter estimation and sensitivity analysis and to provide accessible tools for using these methods. In this way, we hope to contribute to establishing an advanced culture of relating agent-based models to data and patterns observed in real systems and to foster rigorous and structured analyses of agent-based models.

IV.2. Introduction

In agent-based models (ABMs), individual agents, which can be humans, institutions, or organisms, and their behaviours are represented explicitly. ABMs are used when one or more of the following individual-level aspects are considered important for explaining system-level behaviour: heterogeneity among individuals, local interactions, and adaptive behaviour based on decision making [Grimm, 2008]. The use of ABMs is thus required for many, if not most, questions regarding social, ecological, or any other systems comprised of autonomous agents. ABMs have therefore become an established tool in social, ecological and environmental sciences [Gilbert, 2007, Thiele et al., 2011, Railsback and Grimm, 2012].

This establishment appears to have occurred in at least two phases. First, most ABMs in a certain field of research are designed and analysed more or less ad hoc, reflecting the fact that experience using this tool must accumulate over time. The focus in this phase is usually more on how to build representations than on in-depth analyses of how the model systems actually work. Typically, model evaluations are qualitative, and fitting to data is not a major issue. Most models developed in this phase are designed to demonstrate general mechanisms or provide generic insights. The price for this generality is that the models usually do not deliver testable predictions, and it remains unclear how well they really explain observed phenomena.

The second phase in agent-based modelling appears to begin once a critical mass of models for certain classes of questions and systems has been developed, so that attention shifts

from representation and demonstration to obtaining actual insights into how real systems are working. An indicator of this phase is the increased use of quantitative analyses that focus on both a better mechanistic understanding of the model and on relating the model to real-world phenomena and mechanisms. Important approaches during this stage are sensitivity analysis and calibration (parameter fitting) to certain data sets and patterns.

The use of these approaches is, however, still rather low with agent-based modelling. A brief survey of papers published in the *Journal of Artificial Societies and Social Simulation* and in *Ecological Modelling* in the years 2009-2010 showed that the percentages of simulation studies including parameter fitting were 14 and 37%, respectively, while only 12 and 24% of the published studies included some type of systematic sensitivity analysis (for details of this survey, see Supplement SM1). There are certainly many reasons why quantitative approaches for model analysis and calibration are not used more often and why the usage of these approaches appears to differ between social simulation and ecological modelling, including the availability of data and generally accepted theories of certain processes, a focus on theory or application, and the complexity of the agents' decision making (e.g., whether they are humans or plants).

There is, however, a further important impediment to using more quantitative methods for analysing models and relating them more closely to observed patterns [Grimm et al., 2005a, Railsback and Grimm, 2012] and real systems: most modellers in ecology and social sciences are amateurs with regard to computer science and the concepts and techniques of experimental design [Lorscheid et al., 2012]. They often lack training in methods for calibration and sensitivity analysis and for implementing and actually using these methods. Certainly, comprehensive monographs on these methods exist [e.g., Saltelli et al., 2004, Kleijnen, 2008], but they tend to be dense and therefore not easily accessible for many modellers in social sciences and ecology. Moreover, even if one learns how a certain method works in principle, it often remains unclear how it should actually be implemented and used.

We therefore in this article introduce software and provide scripts that facilitate the use of a wide range of methods for calibration, the design of simulation experiments, and sensitivity analysis. We do not intend to give in-depth introductions to these methods but rather provide an overview of the most important approaches and demonstrate how they can easily be applied using existing packages for the statistical software program R [R Core Team, 2014] in conjunction with RNetLogo [Thiele et al., 2012], an R package that allows a NetLogo [Wilensky, 1999] model to be run from R, sends data to that model, and exports the model output to R for visualisation and statistical analyses.

R is a free, open-source software platform that has become established as a standard tool for statistical analyses in biology and other disciplines. An indicator of this is the rapidly increasing number of textbooks on R or on certain elements of R; currently, there are more than 30 textbooks on R available [R Core Team, 2013], e.g., Crawley [2005], Dalgaard [2008], Zuur et al. [2009]. R is an open platform, i.e., users contribute packages that perform certain tasks. RNetLogo is one of these packages.

NetLogo [Wilensky, 1999] is a free software platform for agent-based modelling that was originally designed for teaching but is increasingly used for science [Railsback and Grimm, 2012, Wilensky and Rand, 2014]. Learning and using NetLogo requires little effort due to its easy and stable installation, the excellent documentation and tutorials, a simple but powerful programming language, and continuous support by its developers and users via an active user forum on the internet. Moreover, NetLogo's source code was made available

to the public in 2011, which might lead to further developments and improvements, in particular regarding computational power, which can sometimes be limiting for large, complex models.

NetLogo comes with BehaviorSpace [Wilensky and Shargel, 2002], a convenient tool for running simulation experiments, i.e., automatically varying parameters, running simulations, and writing model outputs to files. However, for more complex calibrations, simulation experiments, or sensitivity analyses, it would be more efficient to have a seamless integration of NetLogo into software where modules or packages for these complex methods exist and can easily be used and adapted. Such a seamless link has been developed for Mathematica [Bakshy and Wilensky, 2007] and, more recently, also for R [RNetLogo, Thiele et al., 2012]. RNetLogo merges the power of R with the power and ease of use of NetLogo.

The software tool BehaviorSearch calibrates ABMs implemented in NetLogo [Stonedahl and Wilensky, 2013]; it implements some of the calibration methods that we describe below and appears to be powerful and robust [for an example use, see Radchuk et al., 2013]. Still, for many purposes, the professional agent-based modeller might need to take advantage of the wider range of calibration methods available via R packages and to control the operation of these methods in more detail. We recommend using the "cookbook" presented here in combination with BehaviorSearch. For models implemented in languages other than NetLogo, the scripts in our cookbook can be adapted because they are based on R, whereas BehaviorSearch cannot be used.

In the following, we will first explain how R, NetLogo, and RNetLogo are installed and how these programs can be learned. We introduce a simple example model, taken from population ecology, which will be used for all further demonstrations. We then present a wide range of techniques of model calibration, starting with a short general introduction and closing with an intermediate discussion. Afterwards, we do the same for sensitivity analysis techniques. Our main purpose is to provide an overview, or "cookbook", of methods so that beginners in parameter estimation and sensitivity analysis can see which approaches are available, what they can be used for, and how they are used in principle using R scripts. These scripts can also be adapted if platforms other than NetLogo are used for implementing the ABM, but then the users must replace the "simulation function" in the R scripts, where the data exchange between R and NetLogo occurs, with an appropriate alternative. All source codes, i.e., the NetLogo model implementation and the R/RNetLogo scripts, are available in the Supplementary Material.

We will not discuss the backgrounds of the methods in detail, as there is already a large body of literature on calibration and sensitivity analysis methods [e.g., Saltelli et al., 2000, 2004, 2008, Helton et al., 2006, Kleijnen, 1995, 2008, Cournède et al., 2013, Gan et al., 2014]. We will therefore refer to existing introductions to the respective methods. We also did not fine-tune the methods to our example model and did not perform all of the checks required for thorough interpretation of results, e.g., convergence checks. Therefore, the methods presented have the potential to produce more detailed and robust results, but for simplicity, we used default settings whenever possible. The purpose of this article is not to present the optimal application of the methods for the example model but to provide a good starting point to apply the methods to readers' own models. As with a real cookbook, readers should benefit from reading the general sections but might decide to browse through the list of approaches demonstrated and selectively zoom into reading specific "recipes".

Readers not familiar with R will not understand the R scripts in detail but can still see how easily R packages can be employed to perform even complex tasks. Readers not familiar with

statistical methods, e.g., linear regression, will not understand the interpretation of some of the results presented, but they should still grasp the general idea. Again, as with a real cookbook, you will not be able to follow the recipe if you never learned the basics of cooking. However, hopefully this article will convince some readers that learning these basics might pay off handsomely.

IV.2.1. Software Requirements

The model used and described in the next section is implemented in NetLogo [Wilensky, 1999]. NetLogo can be downloaded from http://ccl.northwestern.edu/netlogo/. Parameter fitting and sensitivity analysis is performed in R [R Core Team, 2014]. R can be downloaded from http://cran.r-project.org/. Because RNetLogo is available on CRAN, installation from within an R shell/RGUI can be performed by typing `install.packages("RNetLogo")`. For further details see the RNetLogo manual, available at http://cran.r-project.org/web/packages/RNetLogo/index.html. When RNetLogo is installed, loading the example model works in this way (path and version names might need to be adapted):

```
# 1. Load the package.
library(RNetLogo)

# 2. Initialize NetLogo.
nl.path <- "C:/Program Files/NetLogo 5.0.4"
NLStart(nl.path, nl.version=5, gui=FALSE, obj.name="my.nl1")

# 3. Load the NetLogo model.
model.path <- "C:/models/woodhoopoe.nlogo"
NLLoadModel(model.path, nl.obj="my.nl1")
```

This code was used in all application examples. The source codes of all examples as well as the R workspaces with simulation results are in the Supplementary Material. In many of the examples presented in this article, additional R packages are used. In most cases the installation is equivalent to the installation of RNetLogo. References are provided where these packages are used.

IV.2.2. The Example Model

The model description following the ODD protocol [Grimm et al., 2010] is adopted from Railsback and Grimm [2012]. Because they are simple, a description of the submodels is included in the section Process overview and scheduling. The source code of the NetLogo model is included in the Supplementary Material.

The model represents, in a simplified way, the population and social group dynamics of a group-living, territorial bird species with reproductive suppression, i.e., the alpha couple in each group suppresses the reproduction of subordinate mature birds. A key behaviour in this system is the subordinate birds' decision as to when to leave their territory for so-called scouting forays, on which they might discover a free alpha, or breeding, position somewhere else. If they do not find such a position, they return to their home territory. Scouting forays come with an increased mortality risk due to raptors. The model provides a laboratory for developing a theory for the scouting foray decision, i.e., alternative submodels of the decision to foray can be implemented and the corresponding output of the full model compared

to patterns observed in reality. Railsback and Grimm [2012] use patterns generated by a specific model version, and the educational task they propose is to identify the submodel they were using. In this article, we use the simplest model version, where the probability of subordinates undertaking a scouting foray is constant.

Purpose. - The purpose of the model is to explore the consequences of the subordinate birds' decisions to make scouting forays on the structure and dynamics of the social group and the entire population.

Entities, state variables, and scales. - The entities of the model are birds and territories. A territory represents both a social group of birds and the space occupied by that group. Territories not occupied by a group are empty. Territories are arranged in a one-dimensional row of 25 NetLogo patches with the boundary territories "wrapped" so that the model world is a ring. The state variables of a territory are the coordinate of its location and a list of the birds occupying it. The state variables of birds are their sex, age (in months), and whether they are alpha. The time step of the model is one month. Simulations run for 22 years, and the results from the initial two years are ignored.

Process overview and scheduling. - The following list of processes is executed in the given order once per time step. The order in which the birds and territories execute a process is always randomised, and state variables are updated immediately after each operation.

- Date and ages of birds are updated.
- Territories try to fill vacant alpha positions. If a territory lacks an alpha but has a subordinate adult (age > 12 months) of the right sex, the oldest subordinate becomes the new alpha.
- Birds undertake scouting forays. Subordinate adults decide whether to scout for a new territory with a vacant alpha position. If no other non-alpha is in the current territory, a subordinate adult definitely stays. If there are older non-alphas on the current home territory, a subordinate adult scouts with probability *scout-prob*. If the bird scouts, it randomly moves either left or right along the row of territories. Scouting birds can explore up to scouting-distance territories in their chosen direction. Of those territories, the bird occupies the one that is closest to its starting territory and has no alpha of its sex. If no such territory exists, the bird goes back to its starting territory. All birds that scout (including those that find and occupy a new territory) are then subjected to a predation mortality probability of *1.0 - scouting-survival*.
- Alpha females reproduce. In the last month of every year, alpha females that have an alpha male in their territory produce two offspring. The offspring have their age set to zero months and their sex chosen randomly with equal probability.
- Birds experience mortality. All birds are subject to stochastic background mortality with a monthly survival probability of *survival-prob*.
- Output is produced.

Design concepts.

- Emergence. The results we are interested in are the three patterns the model is supposed to reproduce (see Observation); they all emerge from the decision making for scouting but also may strongly depend on other model parameters, such as reproduction and mortality rates.
- Adaptation. There is only one adaptive decision: to undertake a scouting foray or not.
- Objectives. The subordinate birds' objective is to become an alpha so they can reproduce. If the individual stays at its home territory, all the older birds of its sex must die before the individual is able to become an alpha. If the individual scouts, to succeed it must find a vacant alpha position and it must survive predation during scouting.
- Sensing. We assume that birds know nothing about other territories and can only sense whether an alpha position is open in another territory by actually going there. Birds know both the status and age of the other birds in their group.
- Collectives. The social groups are collectives. Because the model's "territory" entities represent the social groups as well as their space, the model treats the behaviours of the social groups (promoting alphas) as behaviours of the territories.
- Observation. In addition to visual displays to observe individual behaviour, three characteristic patterns are observed at different hierarchical levels of the model: the long-term mean number of birds (mean or abundance criterion), the standard deviation from year to year in the annual number of birds (variation criterion) and the average percentage of territories that lack one or both alpha animals (vacancy criterion). The observational data are collected in November of each year, i.e., the month before reproduction occurs.

Initialisation. - Simulations begin in January (month 1). Every territory begins with two male and two female birds, with ages chosen randomly from a uniform distribution of 1 to 24 months. The oldest adult of each sex becomes alpha.

Input data. - The model does not include any external input.

Submodels. - Because all submodels are very simple, their full descriptions are already included in the process overview above. The model includes five parameters, which are listed in Table IV.1.

Table IV.1.: Model parameters. *Base values used by Railsback and Grimm [2012]

Parameter	Description	Base value*
fecundity	Number of offspring per reproducing female	2
scouting-distance	Distance over which birds scout	5
scouting-survival	Probability of surviving a scouting trip	0.8
survival-prob	Probability of a bird to survive one month	0.99
scout-prob	Probability to undertake a scouting trip	0.5

IV.3. Parameter Estimation and Calibration

Typically, ABMs include multiple submodels with several parameters. Parameterisation, i.e., finding appropriate values of at least some of these parameters, is often difficult due to the uncertainty in, or complete lack of, observational data. In such cases, parameter fitting or calibration methods can be used to find reasonable parameter values by combining sampling or optimisation methods with so-called inverse modelling, also referred to as pattern-oriented parameterisation/modelling [POM; Grimm and Railsback, 2005], or Monte Carlo Filtering, as the patterns are applied as filters to separate good from bad sets of parameter values [Grimm and Railsback, 2005]. The basic idea is to find parameter values that make the model reproduce patterns observed in reality sufficiently well.

Usually, at least a range of possible values for a parameter is known. It can be obtained from biological constraints (e.g., an adult human will usually be between 1.5 and 2.2 metres tall), by checking the variation in repeated measurements or different data in the literature, etc. During model development, parameter values are often chosen via simple trial and error "by hand" because precision is not necessary at this stage. However, once the design of the model is fixed and more quantitative analyses are planned, the model must be run systematically with varying parameter values within this range and the outcome of the model runs compared to observational data.

If the parameters are all independent, i.e., the calibrations of different parameters do not affect each other, it is possible to perform model calibration separately for all unknown parameters. Usually, though, parameters interact because the different processes that the parameters represent are not independent but interactive. Thus, rather than single parameters, entire sets of parameters must be calibrated simultaneously. The number of possible combinations can become extremely large and may therefore not be processable within adequate time horizons.

Therefore, more sophisticated ways of finding the right parameter values are needed. This can also be the case for independent parameters or if only one parameter value is unknown, if the model contains stochastic effects and therefore needs to be run multiple times (Monte Carlo simulations) for each parameter combination [Martínez et al., 2011] or if the model runs very slow. Therefore, efficient sampling or optimisation methods must be applied. Here, "sampling" refers to defining parameter sets so that the entire parameter space, i.e., all possible parameter combinations, is explored in a systematic way.

To assess whether a certain combination of parameter values leads to acceptable model output, it is necessary to define one, or better yet multiple, fitting criteria, i.e., metrics that allow one to quantify how well the model output matches the data. Such criteria should be taken from various hierarchical levels and possibly different spatial or temporal scales, e.g., from single individuals over social groups, if possible, to the whole population.

Two different strategies for fitting model parameters to observational data exist: best-fit and categorical calibration. The aim of calibration for the first strategy is to find the parameter combination that best fits the observational data. The quality measure is one exact value obtained from the observational data, so it is easy to determine which parameter set leads to the lowest difference. Of course, multiple fitting criteria can be defined, but they must be aggregated to one single measure, for example, by calculating the sum of the mean square deviation between the model and the observational data for all fitting criteria. An example for the application of such a measure can be found in Wiegand et al. [1998]. The problem with best-fit calibration is that even the best parameter set may not be able to

reproduce all observed patterns sufficiently well. Furthermore, aggregating different fitting criteria to one measure makes it necessary to think about their relation to each other, i.e., are they all equally important or do they need to be weighted?

These questions are not that important for the second strategy, categorical calibration. Here, a single value is not derived from the observational data, but rather, a range of plausible values is defined for each calibration criterion. This is particularly useful when the observational data are highly variable or uncertain by themselves. In this case, the number of criteria met is counted for a parameter set, i.e., the cases when the model result matches the defined value range. Then, the parameter set that matches all or most criteria is selected. Still, it is possible that either no parameter combination will match the defined value range or that multiple parameter sets will reproduce the same target patterns. In such a case, the fitting criteria (both their values and importance) and/or the model itself need to be adjusted. For further details on practical solutions to such conceptual problems, see, for example, Railsback and Grimm [2012].

IV.3.1. Preliminaries: Fitting Criteria for the Example Model

We assume, following Railsback and Grimm [2012], that the two parameters *survival-prob* and *scout-prob* have been identified as important parameters with uncertain values. We assume that reasonable value ranges for these two parameters are as follows:

- *scout-prob*: 0.0 to 0.5
- *survival-prob*: 0.95 to 1.0

These parameters should be fitted against the three response variables (fitting criteria/ patterns) described in the Observation section of the model description.

Railsback and Grimm [2012] define categorical calibration criteria. The acceptable value ranges derived from observational data they used are as follows:

- mean abundance (abundance criterion): 115 to 135 animals
- annual std. dev. (variation criterion): 10 to 15 animals
- mean territories lacking one or both alpha animals (vacancy criterion): 15 to 30%

When using these categorical fitting criteria, each criterion is fulfilled when the corresponding metric falls within the desired range.

Some of the calibration methods used below require a single fitting criterion, the best-fit criterion. To keep the patterns as they are, we use a hybrid solution by defining conditional equations to transform the categorical criteria to a best-fit criterion. The following is just a simple example of how such an aggregation can be performed. For more powerful approaches, see, for example, Soetaert and Herman [2009] for a function including a measure of accuracy of observational data. However, such transformations always involve a loss of more or less important information that can be non-trivial. Furthermore, differences in the results of different methods, for example, using categorical criteria versus using best-fit criteria, can have their origin in the transformation between these criteria.

To calculate the three above-defined categorical criteria as quantitative measures, we use conditional equations based on squared relative deviations to the mean value of the acceptable value range and sum them over the different criteria as follows:

$$abundance_{crit}(x) = \begin{cases} 0, & \text{if } 115 \leq x \leq 135 \\ \left(\frac{mean(115,135)-x}{mean(115,135)}\right)^2, & else \end{cases} \quad \text{(IV.1)}$$

$$variation_{crit}(y) = \begin{cases} 0, & \text{if } 10 \leq y \leq 15 \\ \left(\frac{mean(10,15)-y}{mean(10,15)}\right)^2, & else \end{cases} \quad \text{(IV.2)}$$

$$vacancy_{crit}(z) = \begin{cases} 0, & \text{if } 15 \leq z \leq 30 \\ \left(\frac{mean(15,30)-z}{mean(15,30)}\right)^2, & else \end{cases} \quad \text{(IV.3)}$$

$$cost(x,y,z) = abundance_{crit}(x) + variation_{crit}(y) + vacancy_{crit}(z) \quad \text{(IV.4)}$$

with *x,y,z* being the corresponding simulation result, e.g., x is the mean abundance of the simulation as mentioned above. If the simulation result is within the acceptable range, the *cost* value of the criteria becomes 0; otherwise, it is the squared relative deviation. By squaring the deviations, we keep the values positive and weigh large deviations disproportionately higher than low deviations (Eqs. IV.1-IV.3). This has an important effect on Eq. IV.4. For example, if we find small deviations in all three criteria, the overall *cost* value (Eq. IV.4) still stays low, but when only one criterion's deviation is rather high, its criterion value and therefore also the overall *cost* value becomes disproportionately high. We use this approach here because we think that small deviations in all criteria are less important than a single large deviation.

Alternatives to this are multi-criterion approaches where each criterion is considered separately and a combined assessment is performed by determining Pareto optima or fronts [Miettinen, 1999, de Weck, 2004]. See, for example, the package mco [Trautmann et al., 2013] for Pareto optimisation with a genetic algorithm. Multi-criterion approaches, however, have their own challenges and limitations and are much less straightforward to use than the *cost* function approach that we used here.

Because the example model includes stochastic effects, we repeat model runs using the same initialisation and average the output. Following Martínez et al. [2011] and Kerr et al. [2002], we ran 10 repetitions for every tested parameter set. However, for "real" simulation studies it is advisable to determine the number of repetitions by running the model with an increasing number of repetitions and calculating the resulting coefficient of variation (CV) of the simulation output. At the number of repetitions where the CV remains (nearly) the same, convergence can often be assumed [Lorscheid et al., 2012]. However, in cases of non-linear relationships between the input parameters and simulation output, this assumption may not be fulfilled.

If we have replicates of observational data and a stochastic model, which is not the case for our example, we should compare distributions of the results rather than using a single value comparison, as recommended in Stonedahl and Wilensky [2010]. The calculation of the `pomdev` measure [Piou et al., 2009], which is already implemented in the R package Pomic [Piou et al., 2009], could be of interest in such a case.

If we want to compare whole time series instead of single output values, the aggregation procedure can become more difficult. Such data could also be compared by mean square deviations [see, for example, Wiegand et al., 1998], but then small differences in all measurement points can yield the same deviation as a strong difference in one point. Hyndman [2013] provides an overview of R packages helpful for time series analysis.

IV.3.2. Full Factorial Design

A systematic method for exploring parameter space is (full) factorial design, known from Design of Experiments (DoE) methodology [Box et al., 1978]. It can be applied if the model runs very quickly or if the numbers of unknown parameters and possible parameter values are rather small. For example, Jakoby et al. [2014] ran a deterministic generic rangeland model that includes nine parameters and a few difference equations for one billion parameter sets.

In DoE terminology, the independent variables are termed "factors" and the dependent (output) variables "responses". For full factorial design, all possible factor combinations are used. The critical point here is to define the possible factor levels (parameter values) within the parameter ranges. For parameter values that are integer values by nature (e.g., number of children) this is easy, but for continuous values it can be difficult to find a reasonable step size for the factor levels. This can be especially difficult if the relationship between the factor and the response variables is non-linear.

For our example model, we assume the following factor levels [taken from Railsback and Grimm, 2012]:

- *scout-prob*: 0.0 to 0.5 with step size 0.05
- *survival-prob*: 0.95 to 1.0 with step size 0.005

There are several packages available in R for supporting DoE; for a collection, see, for example, Groemping [2013b]. To run a full factorial design in R, the function `expand.grid` from the base package can be used as follows:

```
# 1. Define a function that runs the simulation model
# for a given parameter combination and returns the
# value(s) of the fitting criteria. See Supplementary
# Material (simulation_function1.R) for an
# implementation example using RNetLogo.
sim <- function(params) {
   ...
   return(criteria)
}

# 2. Create the design matrix.
full.factorial.design <- expand.grid(scout.prob =
   seq(0.0,0.5,0.05), survival.prob = seq(0.95,1.0,0.005))

# 3. Iterate through the design matrix from step 2 and call
# function sim from step 1 with each parameter combination.
sim.results <- apply(full.factorial.design, 1, sim)
```

We include this example because it is a good starting point for proceeding to more sophisticated methods, such as the fractional factorial design. If you want to use a classical full factorial design with NetLogo, we recommend using NetLogo's BehaviorSpace [Wilensky and Shargel, 2002].

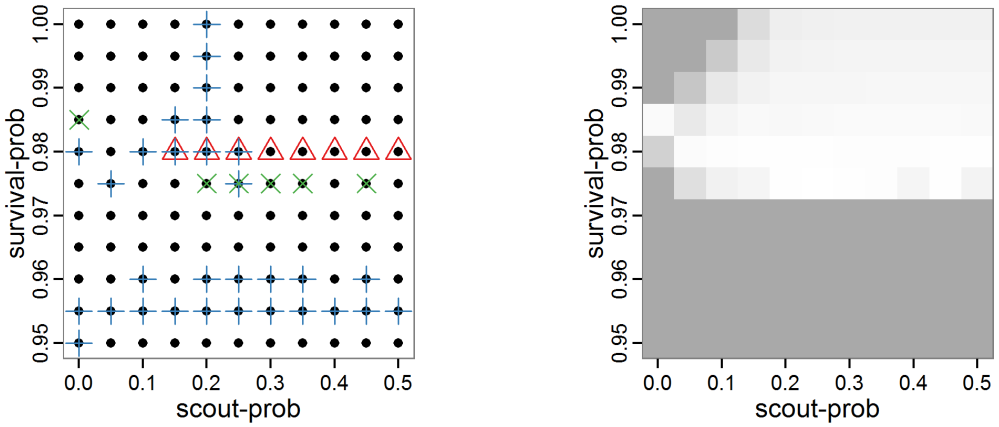


Figure IV.1.: Left: Results of the full factorial design (121 parameter sets) using categorical evaluation criteria. Black points symbolise the tested parameter combinations, and the three different symbols show whether the evaluation criteria were met (red triangle: abundance criterion, blue cross: variation criterion, green x: vacancy criterion). Right: Results of the full factorial design based on conditional best-fit equations (Eqs. IV.1- IV.3), summed up to the one metric *cost* (Eq. IV.4). The *cost* values are truncated at 10.0 (max. *cost* value was 842). Grey cells indicate high *cost* values, and white cells represent low values, i.e., better solutions. One cell represents one simulation with parameter values from the cell's midpoint corresponding to the left panel.

The results of the model calibration procedure with categorical calibration criteria can be explored visually. Figure IV.1 (left panel) shows such a figure for the example model. We see that none of the 121 tested parameter combinations met the three calibration criteria simultaneously. Still, the figure provides some useful hints. The central region of parameter space appears to be promising. The failure to meet all three criteria simultaneously might be due to the step width used.

Using the same output data that underlie Figure IV.1 (left panel), we can calculate the conditional best-fit equations (Eq. IV.4) and plot the results as a raster map (Figure IV.1, right panel). This plot shows more directly than the left panel where the most promising regions of parameter space are located (the white cells), which can then be investigated in more detail.

Overall, full factorial design is useful for visually exploring model behaviour regarding its input parameters in a systematic way but only if the number of parameters and/or factor levels is low. To gain a coarse overview, full factorial design can be useful for calibrating a small number of parameters at a time because the number of combinations can be kept small enough. Fractional factorial designs can be used for a larger number of parameters by selecting only a subset of the parameter combinations of a full factorial design [Box et al.,

1978]; the application in R is very similar, see the above-mentioned collection by Groemping [2013b].

IV.3.3. Classical Sampling Methods

One common method of avoiding full factorial designs, both in simulations and in real experiments, is the usage of sampling methods. The purpose of these methods is to strongly reduce the number of parameter sets that are considered but still scan parameter space in a systematic way. Various algorithms exist to select values for each single parameter, for example, random sampling with or without replacement, balanced random design, systematic sampling with random beginning, stratified sampling etc.

Simple Random Sampling

The conceptually and technically simplest, but not very efficient, sampling method is simple random sampling with replacement. For each sample of the chosen sample size (i.e., number of parameter sets), a random value (from an a priori selected probability density function) is taken for each parameter from its value range. The challenge here, and also for all other (random) sampling methods, is finding an appropriate probability density function (often just a uniform distribution is used) and the selection of a useful sample size. Applying a simple random sampling in R can look like the following:

```
# 1. Define a simulation function (sim) as done for
# Full factorial design.

# 2. Create the random samples from the desired random
# distribution (here: uniform distribution).
random.design <- list('scout-prob'=runif(50,0.0,0.5),
            'survival-prob'=runif(50,0.95,1.0))

# 3. Iterate through the design matrix from step 2 and call
# function sim from step 1 with each parameter combination.
sim.results <- apply(as.data.frame(random.design), 1, sim)
```

Despite the fact that this simple random sampling is not an efficient method and could rather be considered a trial-and-error approach to exploring parameter space, it is used quite often in practice [e.g., Molina et al., 2001]. We nevertheless included the source code for this example in the Supplementary Material because it can be easily adapted to other, related sampling methods.

The results of an example application using 50 samples with categorical calibration criteria are shown in Figure IV.2. The sampling points are distributed over the parameter space but leave large gaps. The overall pattern of the regions where the different criteria were met looks similar to that of the full factorial design (Figure IV.1). In this example application, we were not able to find a parameter combination that meets all three evaluation criteria. In general, the simple random sampling method is not an efficient method for parameter fitting.

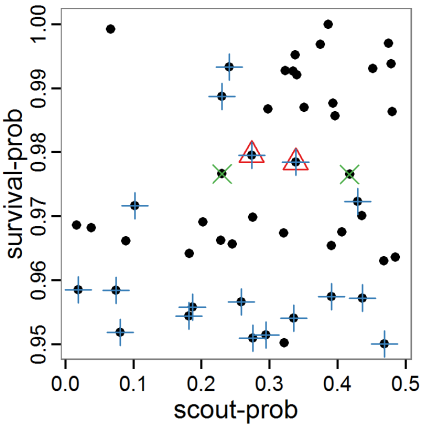


Figure IV.2.: Results of using simple random sampling based on categorical calibration criteria with 50 samples. Black points symbolise the tested parameter combinations, and the three different symbols show whether the evaluation criteria were met (red triangle: abundance criterion, blue cross: variation criterion, green x: vacancy criterion).

Latin Hypercube Sampling

As a more efficient sampling technique to scan parameter spaces, Latin hypercube sampling (LHS) [McKay et al., 1979] is widely used for model calibration and sensitivity analysis as well as for uncertainty analysis [e.g., Marino et al., 2008, Blower and Dowlatabadi, 1994, Frost et al., 2009, Meyer et al., 2009]. LHS is a stratified sampling method without replacement and belongs to the Monte Carlo class of sampling methods. It requires fewer samples to achieve the same accuracy as simple random sampling. The value range of each parameter is divided into N intervals (= sample size) so that all intervals have the same probability. The size of each interval depends on the used probability density distribution of the parameter. For uniform distributions, they all have the same size. Then, each interval of a parameter is sampled once [Marino et al., 2008]. As there are some packages for creating Latin hypercubes available in R, such as tgp [Gramacy and Taddy, 2013] and lhs [Carnell, 2012], it is easy to use this sampling method. For our small example model, the code for generating a Latin hypercube with the tgp package is as follows:

```
# 1. Define a simulation function (sim) as done for
# Full factorial design.

# 2. Create parameter samples from a uniform distribution
# using the function lhs from package tgp.
param.sets <- lhs(n=50, rect=matrix(c(0.0,0.95,0.5,1.0), 2))

# 3. Iterate through the parameter combinations from step 2
# and call function sim from step 1 for each parameter
# combination.
sim.results <- apply(as.data.frame(param.sets), 1, sim)
```

As with simple random sampling, the challenge of choosing appropriate probability density distributions and a meaningful sample size remains. Using, as shown above, a uniform random distribution for the two parameters of our example model, the results using categorical criteria for 50 samples are shown in Figure IV.3. We have been lucky and found one parameter combination (*scout-prob*: 0.0955, *survival-prob*: 0.9774) that met all three criteria. The source code for creating a Latin hypercube in the Supplementary Material also includes an example of applying a best-fit calibration (Eq. IV.4).

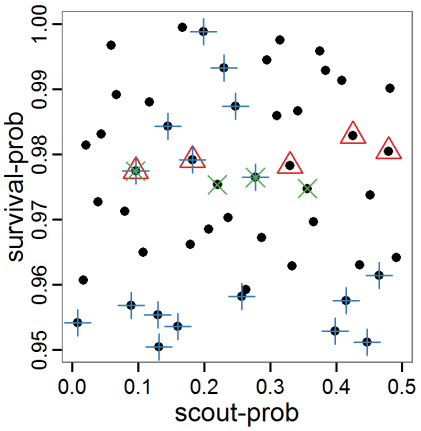


Figure IV.3.: Results from the Latin hypercube sampling using categorical evaluation criteria with 50 samples. Black points symbolise tested parameter combinations, and the three different symbols show whether the evaluation criteria were met (red triangle: abundance criterion, blue cross: variation criterion, green x: vacancy criterion).

IV.3.4. Optimisation Methods

In contrast to the sampling methods described above, optimisation methods create parameter sets not before the simulations are started but in a stepwise way based on the results obtained with one or more previously used parameter sets. These methods are used in many different disciplines, including operations research, physics etc. [Aarts and Korst, 1989, Bansal, 2005]. As the relationships between the input parameters and the output variables in ABMs are usually non-linear, non-linear heuristic optimisation methods are the right choice for parameter fitting. We will give examples for gradient and quasi-Newton methods, simulated annealing and genetic algorithms. There are, however, many other optimisation methods available, such as threshold accepting, ant colony optimisation, stochastic tunnelling, tabu search etc.; several packages for solving optimisation problems are available in R. See Theussl [2013] for an overview.

Gradient and Quasi-Newton Methods

Gradient and quasi-Newton methods search for a local optimum where the gradient of change in parameters versus change in the fitting criterion is zero. In cases where multiple local optima exist, the ability to find the global optimum depends on the starting conditions

[Sun and Yuan, 2006]. A popular example of gradient methods is the so-called conjugate gradient method (CG). Because the standard CG is designed for unconstrained problems (i.e., the parameter space cannot be restricted to a specific value range), it is not useful to apply it to parameter estimation problems of ABMs. Quasi-Newton methods instead are based on the Newton method but approximate the so-called Hessian matrix and, therefore, do not require the definition of the second derivative [Biethahn et al., 2004]. An introduction to these methods can be found in Fletcher [1987]. The implementation of both the gradient and quasi-Newton methods requires a gradient function to be supplied, which is often difficult in ABMs. Implementations in R can often approximate the gradient numerically. Here, we selected the L-BFGS-B method by Byrd et al. [1995], which is a variant of the popular Broyden-Fletcher-Goldfarb-Shanno (BFGS) method [Bonnans et al., 2006] because it is the only method included in the function `optim` of R's stats package [R Core Team, 2014], which can take value ranges (upper and lower limits) for the parameters into account. The strength of the L-BFGS-B method is the ability to handle a large number of variables. To use the L-BFGS-B method with our example ABM, we must define a function that returns a single fitting criterion for a submitted parameter set. For this, we use the single fitting criterion defined in Eq. IV.4. The usage of this method is as follows:

```
# 1. Define a function that runs the simulation model
# for a given parameter combination and returns the
# value of the (aggregated) fitting criterion. See
# Supplementary Material (simulation_function2.R) for
# an implementation example using RNetLogo.
sim <- function(params) {
...
return(criterion)
}

# 2. Run L-BFGS-B. Start, for example, with the maximum of
# the possible parameter value range.
result <- optim(par=c(0.5, 1.0),
          fn=sim, method="L-BFGS-B",
          control=list(maxit=200),
          lower=c(0.0, 0.95), upper=c(0.5, 1.0))
```

Other packages useful for working with gradient or quasi-Newton methods are Rcgmin [Nash, 2013], optimx [Nash and Varadhan, 2011] and BB [Varadhan and Gilbert, 2009]. The source code, including the L-BFGS-B, is in the Supplementary Material and can easily be adapted for other gradient or quasi-Newton methods.

The variation of the aggregated value of the conditional equations (*cost* value) over the 80 model evaluations (including evaluations for gradient approximation) of the L-BFGS-B method is shown in Figure IV.4 (upper panel). The algorithm checks the *cost* value of the start parameter set and the parameter values at the bounds of the valid parameter values, resulting in strong variations of the *cost* value over the number of iterations. Another source of the strong variation is intermediate simulation for the approximation of the gradient function. As we are only interested in the regions with low *cost* values, we truncated the high *cost* values in the graph to obtain a more detailed look at the variation of the lower *cost* values over the iterations. The lower panels of Figure IV.4 show the histograms of the tested parameter values. We see that the search by the L-BFGS-B method for an optimal value for

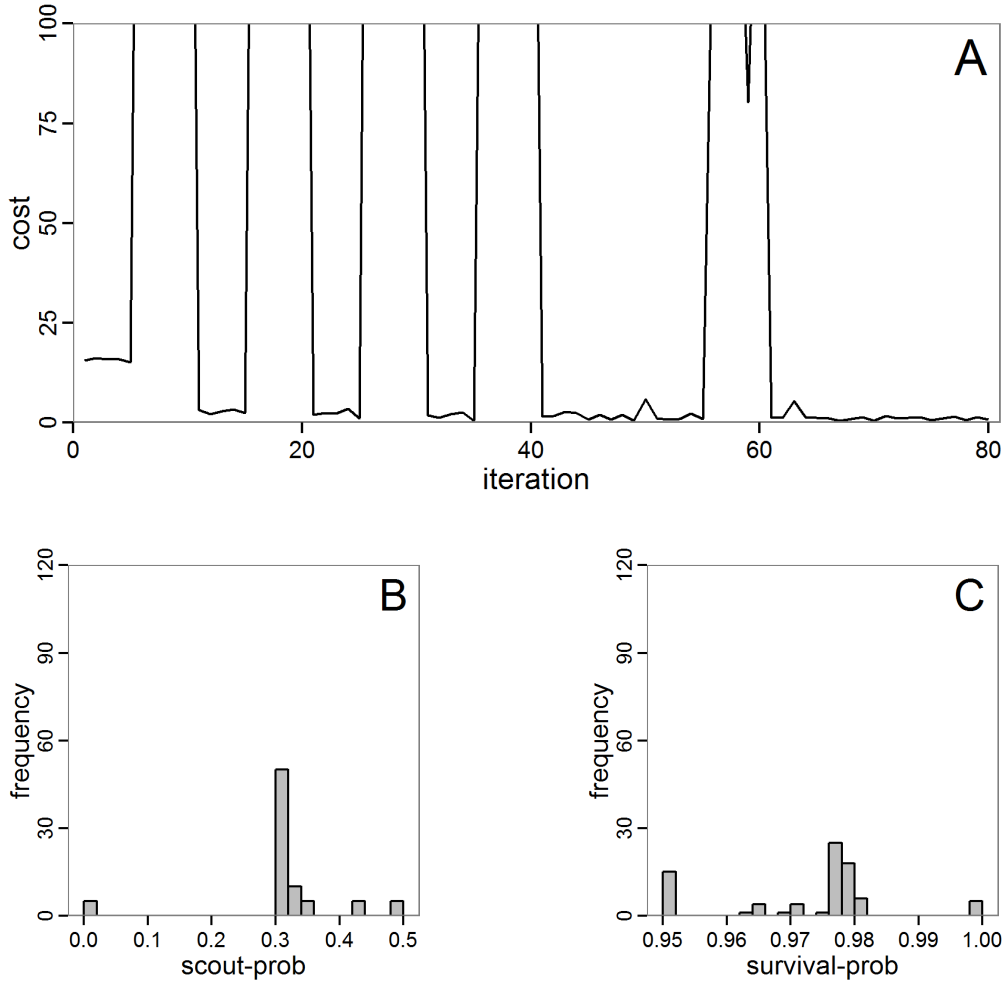


Figure IV.4.: Results of the L-BFGS-B method (includes intermediate simulations, e.g., simulations for gradient approximation). A: Performance of the *cost* value (Eq. IV.4) over the calls of the simulation function (x-axis, truncated at *cost* value 100, max. *cost* value was 317). B: Histogram of the tested parameter values for parameter *scout-prob*. C: Histogram of the tested parameter values for parameter *survival-prob*.

both model parameters, in the configuration used here, shortly checked the extreme values of the value range but focused on the middle range of the parameter space. We see that the method stopped quickly and left large areas out of consideration, which is typical for methods searching for local optima without the ability to also accept solutions with higher *costs* during the optimisation. For *survival-prob*, the minimum possible value is checked more precisely and smaller parts of the parameter space remain untested. The best fit was found with parameter values of 0.3111 for *scout-prob* and 0.9778 for *survival-prob*, which resulted in a *cost* value of 1.1. This *cost* value indicates that the three categorical criteria were not matched simultaneously; otherwise the *cost* value would be zero. However, keep in mind that the application of the method was not fine-tuned and a finite-difference approximation was used for calculating the gradient.

Simulated Annealing

In simulated annealing, temperature corresponds to a certain probability that a local optimum can be left. This avoids the problem of optimisation methods becoming stuck in local, but not global, optima. Simulated annealing is thus a stochastic method designed for finding the global optimum [Michalewicz and Fogel, 2004].

There are several R packages that include simulated annealing functions, for example, stats [R Core Team, 2014], subselect [Cerdeira et al., 2013] or ConsPlan [VanDerWal and Januchowski, 2010]. As for the gradient and quasi-Newton methods, to use simulated annealing with an ABM we must define a function that returns a single fitting criterion for a submitted parameter set. Using the GenSA package [Gubian et al., 2013], which allows one to define value ranges for the parameters, running a simulated annealing optimisation looks like this:

```
# 1. Define a simulation function (sim) as done for the
# L-BFGS-B method.

# 2. Run SA algorithm. Start, for example, with the maximum
# of the possible parameter value range.
result <- GenSA(par=c(0.5,1.0), fn=sim,
            lower=c(0.0, 0.95), upper=c(0.5, 1.0))
```

As with the gradient and quasi-Newton methods, the choice of the starting values as well as the number of iterations can be challenging. Furthermore, specific to simulated annealing, the selection of an appropriate starting temperature is another critical point.

The result of an application example with 256 model evaluations is shown in Figure IV.5. In the upper panel, we see the variation of the *cost* value over the iterations, i.e., the simulation function calls. The algorithm found a good solution very fast, but then the algorithm leaves this good solution and also accepts less good intermediate solutions. Because we are primarily interested in the regions with low *cost* values, i.e., good adaptations of the model to the data, we truncated the graph to obtain a better view of the variation in the region of low *cost* values. In the lower panels, we see that more of the parameter space is tested than with the previous L-BFGS-B method (Figure IV.4). The focus is also in the middle range of the parameter space, which is the region where the best solution was found and which is the most promising part of the parameter space, as we already know from the full factorial design. The minimum *cost* value found in this example was 0.65 with corresponding param-

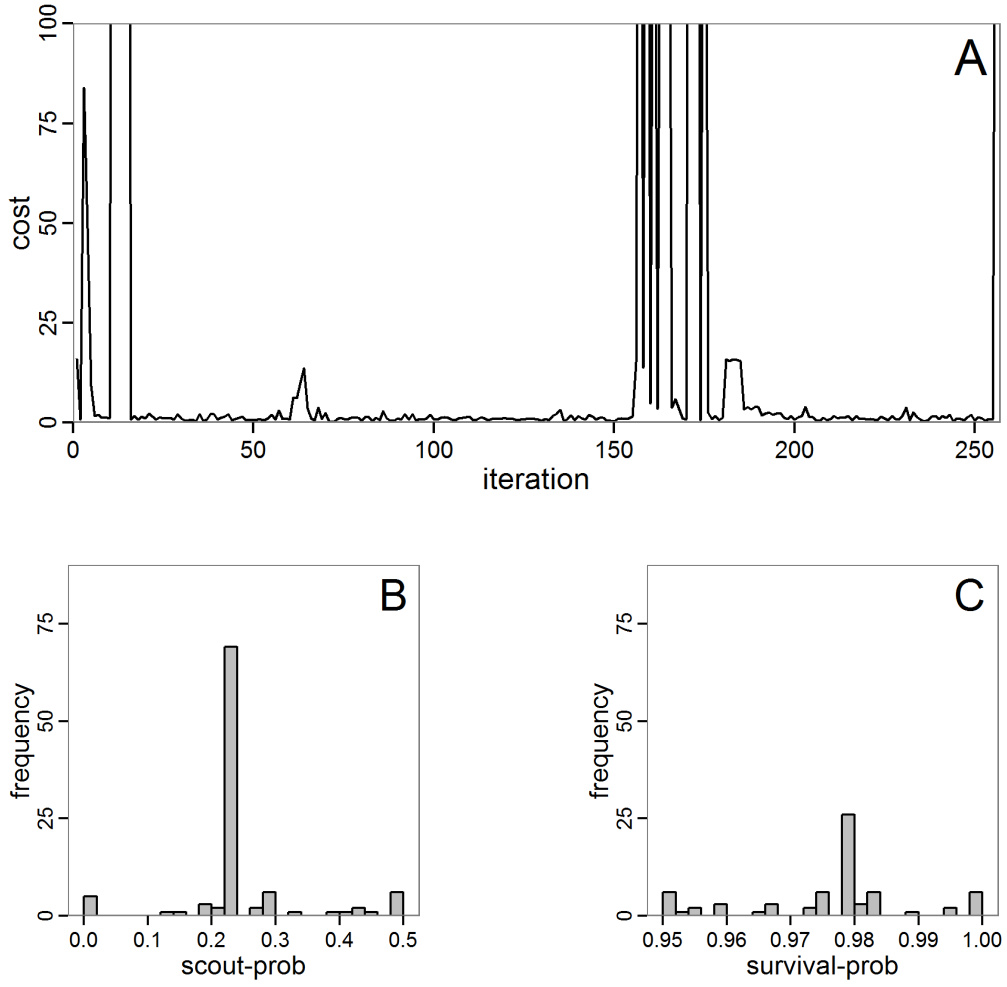


Figure IV.5.: Results of the simulated annealing method. A: Performance of the *cost* value (Eq. IV.4) over the calls of the simulation function (x-axis, truncated at *cost* value 100, max. *cost* value was 317). B: Histogram of the tested parameter values for parameter *scout-prob*. C: Histogram of the tested parameter values for parameter *survival-prob*.

eter values of 0.225 for *scout-prob* and 0.9778 for *survival-prob*. As with the quasi-Newton method, the *cost* value indicates that the three criteria have not been met simultaneously.

Evolutionary or Genetic Algorithms

Evolutionary algorithms (EA) are inspired by the natural process of evolution and feature inheritance, selection, mutation and crossover. Genetic algorithms (GA) are, like evolutionary strategies, genetic programming and some other variants, a subset of EAs [Pan and Kang, 1997]. GAs are often used for optimisation problems by using genetic processes such as selection or mutation on the parameter set. The parameter set, i.e., a vector of parameter values (genes), corresponds to the genotype. A population is formed by a collection of parameter sets (genotypes). Many books and articles about this methodology are available, e.g., Mitchell [1998], Holland [2001], or Back [1996]. Application examples of EA/GA for parameter estimation in the context of ABMs can be found in Duboz et al. [2010], Guichard et al. [2010], Calvez and Hutzler [2006], or Stonedahl and Wilensky [2010].

There are several packages for evolutionary and genetic algorithms available in R; see the listing in Hothorn [2013]. Using the package genalg [Willighagen, 2012] enables us to take ranges of permissible values for the parameters into account. The `rbga` function of this package requires a function that returns a single fitting criterion, as we have also used for the quasi-Newton and simulated annealing methods. The procedure in R is as follows:

```
# 1. Define a simulation function (sim) as done for the
# L-BFGS-B method.

# 2. Run the genetic algorithm.
result <- rbga(stringMin=c(0.0, 0.95),
         stringMax=c(0.5, 1.0),
         evalFunc=sim, iters=200)
```

Challenges with EAs/GAs include selecting an appropriate population size and number of iterations/generations, as well as meaningful probabilities for various genetic processes, such as mutation.

The fitting process using the genalg package with 290 function evaluations resulted in a best *cost* value of 0.35 with *scout-prob* of 0.3410 and *survival-prob* of 0.9763. The performance of the *cost* value over the model evaluations is shown in the upper panel of Figure IV.6. We find a very strong oscillation because successive runs are more independent than in the other methods above, e.g., by creating a new population. Therefore, this graph looks much more chaotic, and truncating the vertical axis to zoom into the region of low *cost* values is less informative in this specific case. As we can see in the lower panels of the figure, a wide range of parameter values has been tested, with slightly higher frequency around the best parameter value for *scout-prob* and a more diffuse pattern for *survival-prob*. However, the best parameter combination found is very similar to the one found by the other optimisation methods. In general, it appears that the promising value range of *survival-prob* is much smaller than that for *scout-prob*. The values of (sub-) optimal solutions for *survival-prob* are always close to 0.977, whereas the corresponding value of *scout-prob* varies on a much wider value range with only a small influence on the *cost* value. This pattern was already shown in Figure IV.1 (right panel). For investigating such a pattern in more detail, Bayesian methods can be very help- and powerful, as presented below.

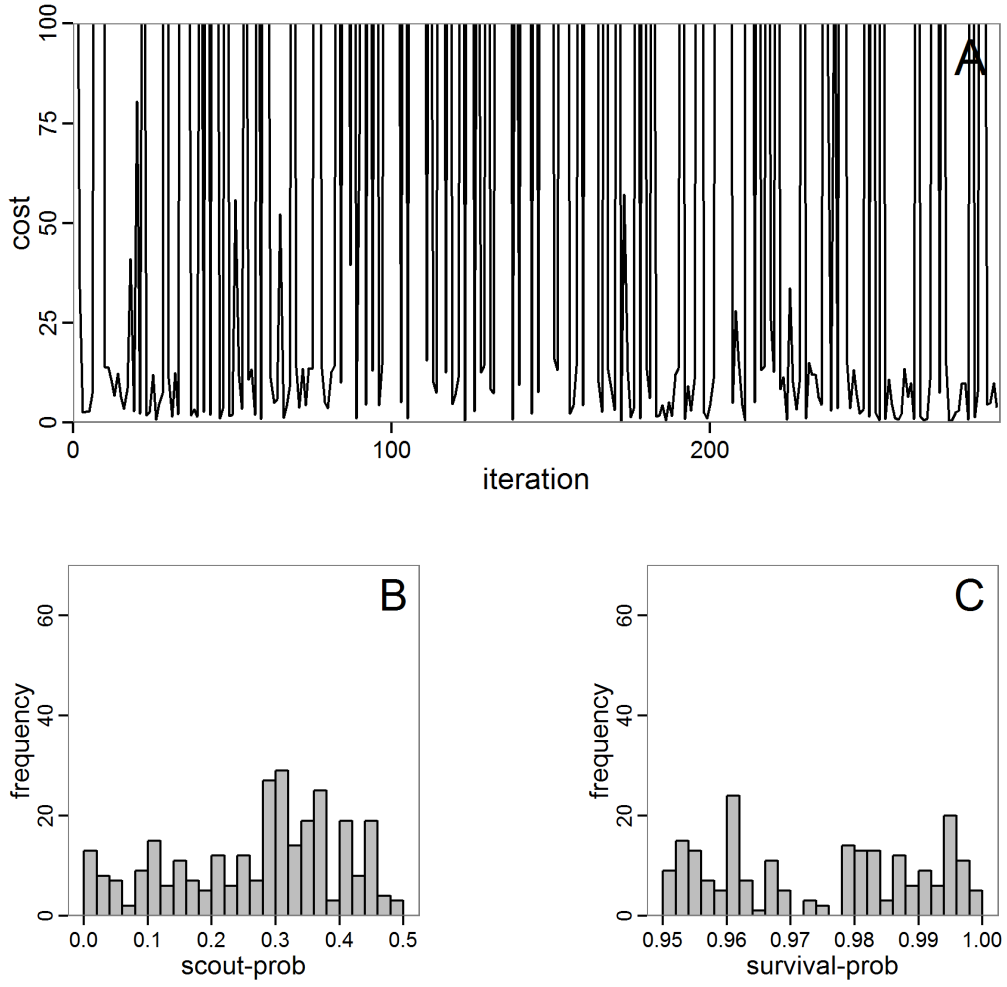


Figure IV.6.: Results of the genetic algorithm method for 10 populations and 50 generations. A: Performance of the *cost* value (Eq. IV.4) over the calls of the simulation function (x-axis, truncated at *cost* value 100, max. *cost* value was 2923). B: Histogram of the tested parameter values for parameter *scout-prob*. C: Histogram of the tested parameter values for parameter *survival-prob*.

IV.3.5. Bayesian Methods

Classical statistical maximum likelihood estimation for model parameters cannot be applied to complex stochastic simulation models; the likelihood functions are either intractable or hard to detect because this is computationally too expensive [Jabot et al., 2013]. By using the Bayesian strategy, the true/posterior probability density functions of parameters are calculated by point-wise likelihood approximations across the parameter space. The basic idea, as described in Jabot et al. [2013], is to run the model a very large number of times with different parameter values drawn from distributions we guess are underlying (prior distributions). Then, the simulation results and the observational data are compared using so-called summary statistics, which are some aggregated information calculated from the simulation and the observational data to reduce the dimensionality of the data. Only those parameter values where the difference between the simulated and the observed summary statistics is less than a defined threshold (given by a tolerance rate) are kept. At the end, an approximation of the posterior distribution is formed by the retained parameter values. Such methods, called Approximate Bayesian Computing (ABC), have been increasingly used for simulation models in recent years [e.g., May et al., 2013, Martínez et al., 2011, Sottoriva and Tavaré, 2010; and review by Hartig et al., 2011]. They not only deliver a best estimate for the parameter values but also provide measures of uncertainty with consideration of correlations among the parameters [Martínez et al., 2011]. Introductions to Bayesian statistical inference using ABC can be found in Beaumont [2010], Van Oijen [2008], or Hartig et al. [2011].

A list of useful R packages around Bayesian inference can be found in Park [2013]. The most relevant packages regarding ABC are abc [Csillery et al., 2012a], EasyABC [Jabot et al., 2013], pomp [King et al., 2013], FME [Soetaert and Petzoldt, 2010] and MCMChybridGP [Fielding, 2011]. If a specific method is not available in an out-of-box package, there are several R packages that can assist in developing custom implementations, such as the MCMC package [Geyer and Johnson, 2013] with its Markov chain Monte Carlo Metropolis-Hastings algorithm or the coda package [Plummer et al., 2006] for the analysis of Markov chain Monte Carlo results. An R interface to the openBUGS software [Lunn et al., 2009] comes with the BRugs package [Thomas et al., 2006] and enables the advanced user to define models and run Bayesian approximations in openBUGS, which is beyond the scope of this paper. MCMC in an ABC framework can also be used to compute some measures of model complexity [Piou et al., 2009]. The Pomic package [Piou et al., 2009] is based on an adaptation of the DIC measure [Spiegelhalter et al., 2002] to compare the goodness of fit and complexity of ABMs developed in a POM context.

Note that Bayesian methods require deeper knowledge and understanding than the other methods presented above to be adapted properly to a specific model. The methods presented above could be understood in principle without previous knowledge, but this is not the case for Bayesian methods. We recommend first reading Wikipedia or other introductions to these methods before trying to use the methods described in this section. Nevertheless, even for beginners, the following provides an overview of the inputs, packages, and typical outputs of Bayesian calibration techniques.

Rejection and Regression Sampling

The easiest variant of ABC regarding the sampling scheme is rejection sampling. Here, the user first defines a set of summary statistics, which are used to compare observations with simulations. Furthermore, a tolerance value, which is the proportion of simulation points whose parameter set is accepted, must be selected. Then, parameter sets are drawn from a user-defined prior distribution and tested for acceptance. At the end, the posterior distribution is approximated from the accepted runs [Beaumont et al., 2002].

Such sampling can be performed in R using the package abc [Csillery et al., 2012a] or the package EasyABC [Jabot et al., 2013]. The abc package offers two further improvements to the simple rejection method based on Euclidean distances: a local linear regression method and a non-linear regression method based on neural networks. Both add a further step to the approximation of the posterior distribution to correct for imperfect matches between the accepted and observed summary statistics [Csillery et al., 2012b].

Because the abc package expects random draws from the prior distributions of the parameter space, we must create such an input in a pre-process. For this, we can, for simplicity, reuse the code of the Latin hypercube sampling with separate best-fit measures for all three fitting criteria used as summary statistics (see Eqs. IV.1-IV.3). Also for simplicity, we apply a non-informative uniform (flat) prior distribution [for further reading see Hartig et al., 2011]. As summary statistics, we use the three criteria defined in the model description. We assume that the observed summary statistic is calculated as the mean of the minimum and the maximum of the accepted output value range, i.e., we assume that the value range was gained by two field measurements and we use the mean of these two samples to compare it with the mean of two simulated outputs. The procedure of using the `abc` function in R (for simple rejection sampling) is as follows:

```
# 1. Run a Latin hypercube sampling as performed above.
# The result should be two variables, the first
# containing the parameter sets (param.sets) and
# the second containing the corresponding summary
# statistics (sim.sum.stats).

# 2. Calculate summary statistics from observational data
#(here: using the mean of value ranges).
obs.sum.stats <- c(abundance=mean(c(115,135)),
          variation=mean(c(10,15)),
            vacancy=mean(c(0.15,0.3)))

# 3. Run ABC using observations summary statistics and the
# input and output of simulations from LHS in step 1.
results.abc <- abc(target=obs.sum.stats, param=param.sets,
         sumstat=sim.sum.stats,
            tol=0.3, method="rejection")
```

The results, i.e., the accepted runs that form the posterior distribution of simple rejection sampling and of the local linear regression method, can be displayed using histograms, as presented in Figure IV.7 (upper and middle panels). These results are based on Latin hypercube sampling with 30,000 samples and a tolerance rate of 30 per cent, which defines the percentage of accepted simulations. The histograms can be used to estimate the form

of the probability density function (kernel density), which is shown in the lower panels of Figure IV.7. These density estimations can be taken subsequently to gain distribution characteristics for the two input parameters, as shown in Table IV.2. Afterwards, the density estimations can be used to run the model not only with the mean or median of the parameter estimation but also for an upper and lower confidence value, which would result not only in one model output but in confidence bands for the model output. See Martínez et al. [2011] for an example.

We see that there are considerable differences in results between the simple rejection sampling and the rejection sampling with local linear regression correction. For example, the mean value of *scout-prob* is much lower with the regression method than with the simple rejection method. Furthermore, the posterior distribution of *survival-prob* is narrower with the regression method than with the simple rejection method.

Table IV.2.: Posterior distribution characteristics for the two parameters gained from the ABC rejection sampling (first and third columns) and the local linear regression method (second and fourth columns; weighted).

	scout-prob		*survival-prob*	
	rej. sampl.	loc. lin. reg.	rej. sampl.	loc. lin. reg.
Minimum	0.0003	-0.1151	0.9695	0.9746
5% percentile	0.0622	-0.0185	0.9733	0.9774
Median	0.2519	0.1423	0.9817	0.9803
Mean	0.2596	0.1485	0.9826	0.9803
Mode	0.2270	0.0793	0.9788	0.9805
95% percentile	0.4666	0.3296	0.9946	0.9832
Maximum	0.5000	0.5261	0.9999	0.9863

Looking at the joint posterior density in Figure IV.8, we see how the densities of the two parameters are related to each other. Not surprisingly, we see strong differences for the two methods, as we already know from Figure IV.7 that the distributions differ. However, we also see that the additional local linear regression condenses the distribution very much and uncovers a linear correlation between the two parameters. A Spearman correlation test between the two parameter samples delivers a ρ of 0.66 (p-value $<$2.2e-16) for the method with the additional local linear regression, whereas ρ is only 0.02 (p-value $=$0.11) for the simple rejection sampling. This result for the method with the additional local linear regression is in good accordance with the results of the following method and has many similarities to the pattern we know from the results of the full factorial design (Figure IV.1, right panel).

Because the application of the additional local linear regression method is based on the same simulation results as the simple rejection sampling, it comes with no additional computational costs. Therefore, it is a good idea to run both methods and check their convergence.

Markov Chain Monte Carlo

Markov chain Monte Carlo (MCMC) is an efficient sampling method where the selection of the next parameter combination depends on the last parameter set and the resulting deviation between the simulation and the observation [Hartig et al., 2011]. Therefore,

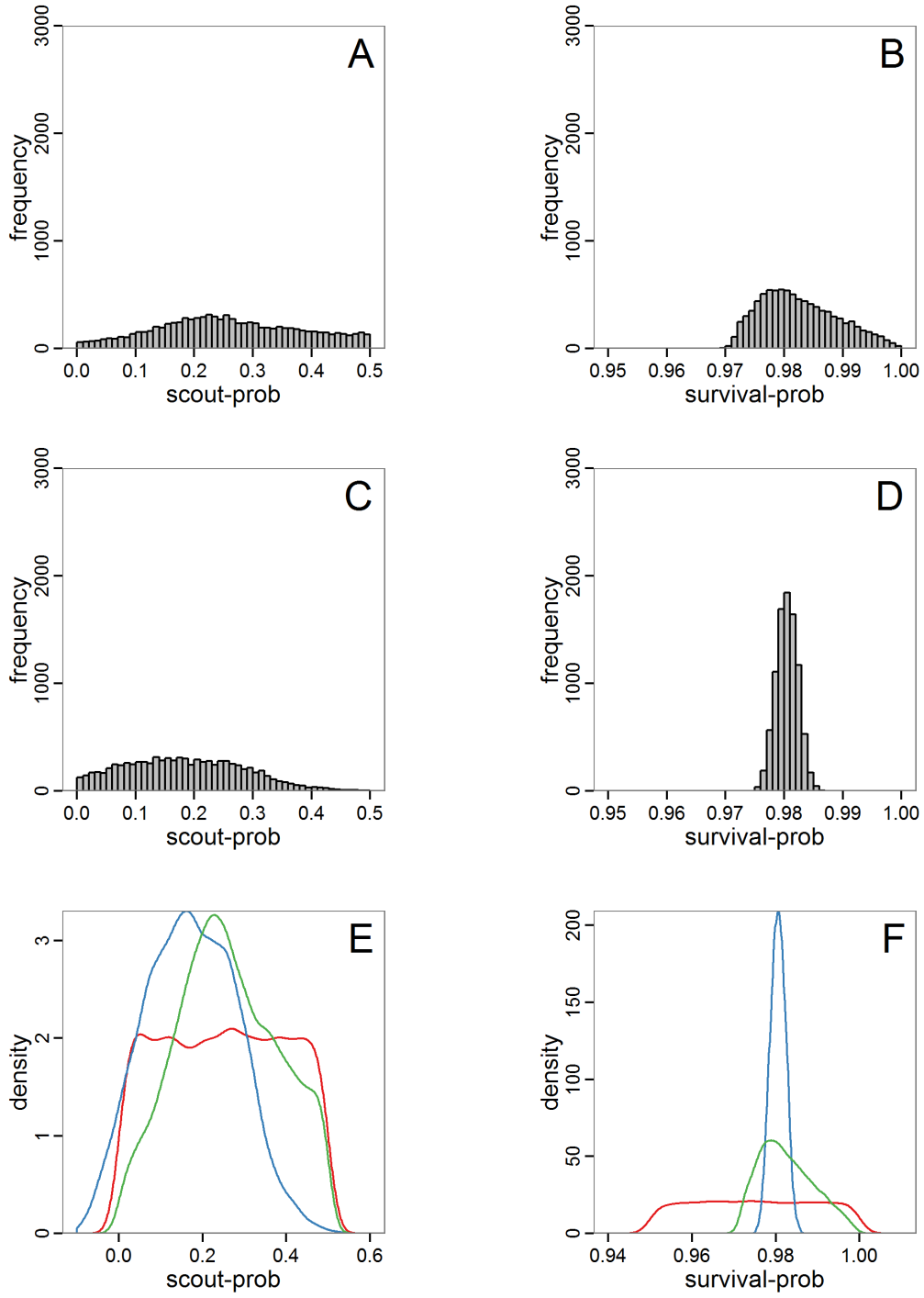


Figure IV.7.: Posterior distribution generated with the ABC rejection sampling method as well as the rejection sampling method followed by additional local linear regression. A: Histogram of accepted runs for *scout-prob* using rejection sampling. B: Histogram of accepted runs for *survival-prob* using rejection sampling. C: Histogram of accepted runs for *scout-prob* using the local linear regression method. D: Histogram of accepted runs for *survival-prob* using the local linear regression method. E & F: Density estimation for *scout-prob* (E) and *survival-prob* (F) by rejection sampling (green line), by the local linear regression method (blue line) and from a prior distribution (red line).

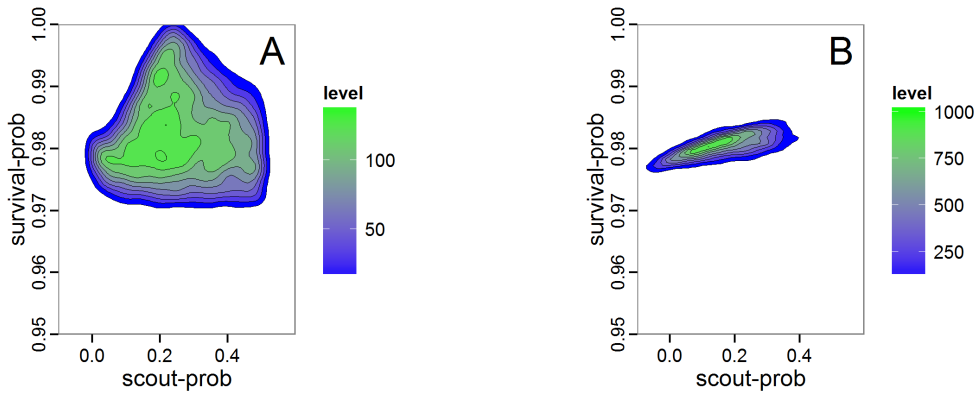


Figure IV.8.: Joint posterior density estimation based on A: ABC rejection sampling, B: rejection sampling with additional local linear regression. Both with 10-90% highest density contours. The last contour is invisible without zooming in, meaning high density is concentrated in a small area.

sampling is concentrated in the regions with high likelihood. This makes the method more efficient in comparison with rejection sampling. Only the initial parameter set is drawn from the prior distribution. In the long run, the chain of parameter sets will converge to the posterior distribution. The advantage of the MCMC methods over the rejection sampling methods is that it does not sample from the prior distribution. See, for example, Beaumont [2010] for further reading.

The R package EasyABC [Jabot et al., 2013] delivers several different algorithms for performing coupled ABC-MCMC schemes. The usage in R looks like this:

```
# 1. Define a function that runs the simulation model for a
# given parameter combination and returns all summary
# statistics.
# See Supplementary Material (simulation_function4.R)
# for an implementation example using RNetLogo.
sim <- function(params) {
   ...
   return(sim.sum.stats)
}

# 2. Calculate summary statistics from observational data
# (here using the mean of ranges).
obs.sum.stats <- c(abundance=mean(c(115,135)),
         variation=mean(c(10,15)),
         vacancy=mean(c(0.15,0.3)))

# 3. Generate prior information.
prior <- list('scout-prob'=c("unif",0.0,0.5),
           'survival-prob'=c("unif",0.95,1.0))

# 4. Run ABC-MCMC.
```

```
results.MCMC <- ABC_mcmc(method="Marjoram", model=sim,
            prior=prior, summary_stat_target=obs.sum.stats)
```

The results of applying the ABC-MCMC scheme to the example model with a total of 39,991 function calls and 3,000 samples in the posterior distribution are prepared in the same manner as the results of the rejection method and are shown in Figure IV.9. The results are very similar to those of the rejection sampling method with local linear regression correction but with even narrower posterior distributions. The numerical characteristics of the posterior distributions are processed using the package coda [Plummer et al., 2006] and are given in Table IV.3.

The ABC-MCMC scheme is more efficient than rejection sampling, but there are many more fine-tuning possibilities, which can also make its use more complicated.

Table IV.3.: Posterior distribution characteristics for the two parameters gained from the ABC-MCMC algorithm.

	scout-prob	*survival-prob*
Minimum	0.0034	0.9758
5% percentile	0.0280	0.9769
Median	0.1392	0.9785
Mean	0.1474	0.9785
Mode	0.0863	0.9758
95% percentile	0.2840	0.9802
Maximum	0.4296	0.9817

Sequential Monte Carlo

A sequential Monte Carlo (SMC) method, such as ABC-MCMC, is also used to concentrate the simulations to the zones of the parameter space with high likelihood [Jabot et al., 2013], i.e., to make the sampling more efficient compared to the rejection method. In contrast to MCMC, each step contains not only one parameter set but a sequence of sets (also called a particle or population). A sequence depends on its predecessor, but the simulations within a sequence are independent. The first sequence contains points from the prior distribution and performs a classical rejection algorithm. The successive sequences are then concentrated to those points of the former sequence with the highest likelihood, i.e., points that are nearest to the observed data [Jabot et al., 2013]. Therefore, the sequences converge to the posterior distribution based on, in contrast to ABC-MCMC, independent samples. The risk of getting stuck in areas of parameter space that share little support with the posterior distribution is lower than in ABC-MCMC [Hartig et al., 2011]. For further reading see Hartig et al. [2011], Jabot et al. [2013] and references therein.

The EasyABC package [Jabot et al., 2013] for R delivers four variants of SMC. The usage is very similar to the application of ABC-MCMC:

```
# 1. Define a simulation function (sim) as done for the
# ABC-MCMC method.

# 2. Generate observational summary statistics
# (using the mean of ranges).
```

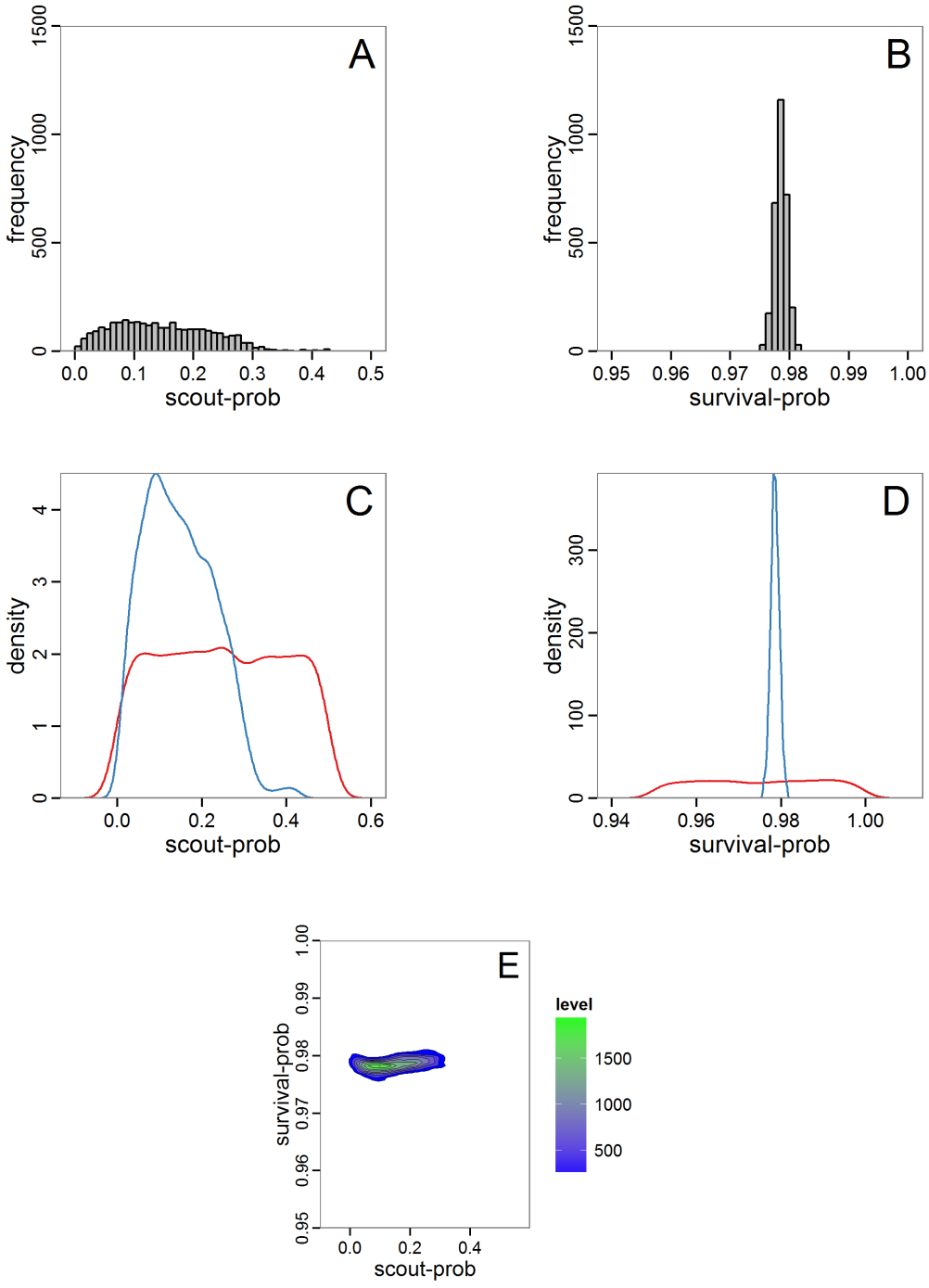


Figure IV.9.: Posterior distribution generated with ABC-MCMC. A: Histogram of accepted runs for *scout-prob*. B: Histogram of accepted runs for *survival-prob*. C & D: Density estimation for *scout-prob* (C) and *survival-prob* (D) by MCMC sampling (blue line) and from prior distribution (red line). E: joint posterior density estimation of *scout-prob* and *survival-prob* by SMC sampling with 10-90% highest density contours.

```
obs.sum.stats <- c(abundance=mean(c(115,135)),
          variation=mean(c(10,15)),
          vacancy=mean(c(0.15,0.3)))

# 3. Generate prior information.
prior <- list('scout-prob'=c("unif",0.0,0.5),
              'survival-prob'=c("unif",0.95,1.0))

# 4. Define a sequence of decreasing tolerance thresholds for
# the accepted (normalised) difference between simulated and
# observed summary statistics (in case of multiple summary
# statistics, like here, the deviations are summed and
# compared to the threshold); one value for each step, first
# value for the classical rejection algorithm, last value for
# the max. final difference.
tolerance <- c(1.5,0.5)

# 5. Run SMC.
results.MCMC <- ABC_sequential(method="Beaumont", model=sim,
                prior=prior, summary_stat_target=obs.sum.stats,
                tolerance_tab=tolerance, nb_simul=20)
```

The results of an application of the ABC-SMC scheme to the example model, prepared in the same manner as for the other ABC schemes, are given in Figure IV.10 and Table IV.4. They are based on 11,359 function calls and 3,000 retained samples for the posterior distribution. The distributions share some similarities with the resulting posterior distributions of the other ABC schemes regarding the value range but have different shapes. The posterior distribution of *scout-prob* does not have its peak at the very small values and is not that different from the prior distribution. The posterior distribution of *survival-prob* is also broader than with the other schemes. These differences from the other schemes could be the result of the lower sample size, differences in the methodologies, and missing fine-tuning. The multiple fine-tuning options in particular make this method complex for satisfactory application.

Table IV.4.: Posterior distribution characteristics for the two parameters gained from the ABC-SMC algorithm.

	scout-prob	*survival-prob*
Minimum	0.0003	0.9700
5% percentile	0.0785	0.9742
Median	0.2968	0.9825
Mean	0.2852	0.9826
Mode	0.0003	0.9700
95% percentile	0.4741	0.9925
Maximum	0.4999	0.9975

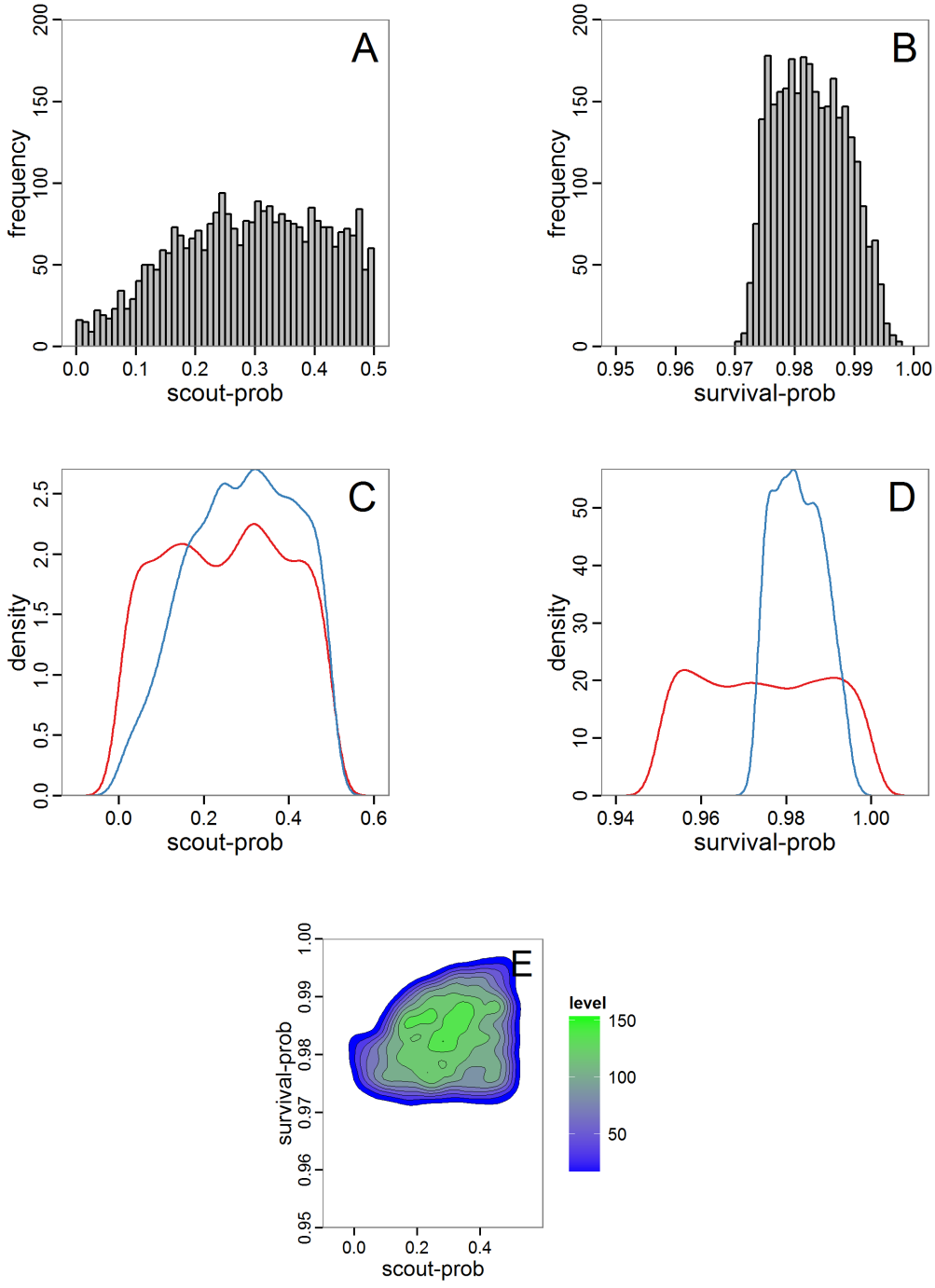


Figure IV.10.: Posterior distribution generated with ABC-SMC. A: Histogram of accepted runs for *scout-prob*. B: Histogram of accepted runs for *survival-prob*. C & D: Density estimation for *scout-prob* (C) and *survival-prob* (D) by SMC sampling (blue line) and from the prior distribution (red line). E: Joint posterior density estimations of *scout-prob* and *survival-prob* by SMC sampling with 10-90% highest density contours.

IV.3.6. Costs and Benefits of Approaches to Parameter Estimation and Calibration

We presented a range of methods for parameter calibration; from sampling techniques over optimisation methods to Approximate Bayesian Computation. Not only the knowledge required to properly apply these methods but also the efficiency of parameter estimation increase in exactly this order. Those modellers who are not willing to become familiar with the details of the more complex methods and are satisfied with less accurate/single parameter values should use the approved Latin hypercube sampling. Those who are interested in very good fits but do not need to worry too much about distributions and confidence bands for the parameter values should take a closer look into the various optimisation methods. The details can become tricky, but methods such as genetic algorithms and simulated annealing are widely used methods with lots of documentation. The ABC methods deliver the most recent approach to parameter calibration but require a much deeper statistical understanding than the other methods as well as sufficient computational power to run a very large number of simulations; on the other hand, these methods deliver much more information than the other methods by constructing a distribution of parameter values rather than one single value. This field is currently quickly evolving, and we see an ongoing development process of new approaches especially designed for the parameterisation of complex dynamic models [e.g., Hartig et al., 2013].

It is always a good idea to start with a very simple approach, such as Latin hypercube sampling, to acquire a feel for the mechanisms of the model and the response to varying parameter values. From there, one can decide whether more sophisticated methods should be applied to the fitting problem. This sequence avoids the situation in which a sophisticated method is fine tuned first, and it is later realised that the model was not able to produce the observed patterns, requiring a return to model development. Furthermore, it can be interesting to first identify those unknown/uncertain parameters that have a considerable influence on the model results. Then, intensive fine tuning of model parameters can be restricted to the most influential ones. For such an importance ranking, the screening techniques presented in the next section about sensitivity analysis can be of interest.

As an attempt to rank the methods, we plotted their costs versus the combination of the information generated by the method and the efficiency with which the method generates them (Figure IV.11). Under costs, we summarised the amount of time one would need to understand the method and to fine-tune its application as well as the computational effort. The most desirable method is the ABC technique, but its costs are much higher than those of the other methods. In the case of large and computationally expensive models, however, the application of ABC techniques may be impossible. Then, the other techniques should be evaluated. For pre-studies, we recommend the application of LHS because it is very simple, can be set up very quickly based on the scripts delivered in the Supplementary Material and can be easily parallelised.

For more complex ABMs, runtime might limit the ability to take full advantage of the methods presented here because the models cannot just be run several thousand times. Here, submodels could at least be parameterised independently. For example, in an animal population model, a submodel describing the animals' energy budget could be parameterised independent of the other processes and entities in the full model [e.g., Martin et al., 2013]. A limitation of this "divide and conquer" method of parameterisation is that interactions between submodels might in fact exist, which might lead to different parameter values than if the full ABM were parameterised.

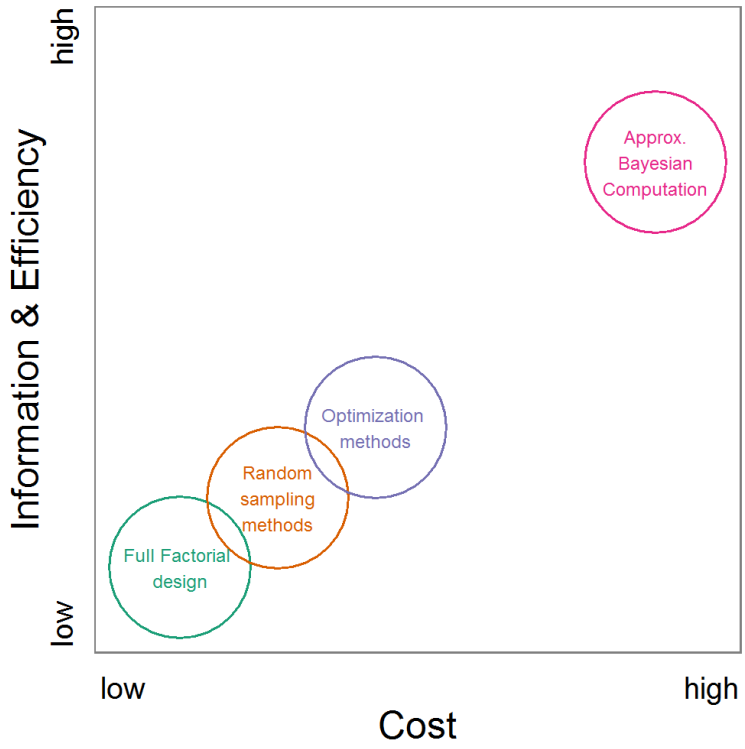


Figure IV.11.: A rough categorisation of the parameter fitting/calibration methods used regarding their cost vs. information and efficiency. Cost includes the computational costs as well as the time required for understanding and fine-tuning the methods. Information and efficiency includes aspects of the type of output and the way to reach it.

IV.4. Sensitivity Analysis

Sensitivity analysis (SA) is used to explore the influence of varying inputs on the outputs of a simulation model [Ginot et al., 2006]. The most commonly analysed inputs are model parameters. SA helps identify those parameters that have a strong influence on model output, which indicates which processes in the model are most important. Moreover, if inputs of the model are uncertain, which is usually the case, sensitivity analysis helps assess the importance of these uncertainties. If the model is robust against variations in the uncertain parameters, i.e., model output does not vary strongly when the parameter values are varied, the uncertainties are of low importance. Otherwise, the parameter values should be well-founded on empirical values [Bar Massada and Carmel, 2008, Schmolke et al., 2010]. SA is therefore closely related to uncertainty analysis [Ginot et al., 2006]. In addition to model parameters, entire groups of parameters, initial values of state variables, or even different model structures can also be considered as inputs to be analysed in SA [Schmolke et al., 2010].

With sensitivity analysis, three approaches are differentiated: screening, local and global sensitivity analysis [Saltelli, 2000; sometimes screening methods are added to global SA

methods, see, for example, Cariboni et al., 2007]. Screening methods are used to rank input factors by their importance to differentiate between more and less important inputs. These methods are often useful for computationally expensive models because they are very fast in identifying the important parameters, which should be analysed in more detail, but they cannot deliver a quantification of the importance [Saltelli, 2000].

Originating from the analysis of models based on ordinary differential equations, local sensitivity analysis quantifies the effect of small variations in the input factors [Soetaert and Herman, 2009, Marino et al., 2008]. Classical local sensitivity analysis is often performed as ceteris paribus analysis, i.e., only one factor is changed at a time (the so-called one-factor-at-time approach, OAT) [Bar Massada and Carmel, 2008]. In contrast, in global sensitivity analysis, input factors are varied over broader ranges. This is of special importance if the inputs are uncertain [Marino et al., 2008], which is mostly the case for ABMs. Furthermore, in global sensitivity analysis several input factors are often varied simultaneously to evaluate not only the effect of one factor at a time but also the interaction effect between inputs; the sensitivity of an input usually depends on the values of the other inputs.

IV.4.1. Preliminaries: Experimental Setup for the Example Model

For simplicity, we will restrict the following examples of sensitivity analyses to three parameters: *scout-prob*, *survival-prob*, and *scouting-survival*. The value ranges and base values used (e.g., for local sensitivity analysis) for the three parameters covered by the sensitivity analysis are listed in Table IV.5.

To control stochasticity in the simulation model, we apply the same approach with 10 repeated model runs as described for parameter fitting.

Table IV.5.: Model parameters used in sensitivity analysis. *Base values used by Railsback and Grimm [2012]

Parameter	Description	Base value*	Base value used here	Min. value	Max. value
scout-prob	Probability of undertaking a scouting trip	0.5	0.065	0.0	0.5
survival-prob	Probability of a bird surviving one month	0.99	0.978	0.95	1.0
scouting-survival	Probability of surviving a scouting trip	0.8	0.8	0.5	1.0

IV.4.2. Local Sensitivity Analysis

As mentioned above, local sensitivity analysis is often performed as a one-factor-at-time analysis with small variations of the input values. With ABMs, this is often achieved by varying the selected inputs by a specified percentage around their nominal, or default, value. This method provides only limited information regarding model sensitivity, but it is still often used and could be considered a first step in a more detailed analysis when applying a multi-step approach, as proposed, for example, by Railsback et al. [2006]. However, when performing such analyses, it should always be kept in mind that interactions between

parameters are ignored and that local sensitivities might be completely different if another set of nominal, or default, parameter values were chosen.

A local sensitivity analysis procedure in R can work in this way:

```
# 1. Define a simulation function (sim) as done for
# Full factorial design.

# 2. Run simulation using sim function defined in step 1
# for the standard input factor values.
base.param <- c(0.065, 0.978, 0.8)
sim.result.base <- sim(base.param)

# 3. Define a function for changing one of the parameter
# values (here with min and max constraints).
change <- function(i, base, multiplier) {
   mod <- base
   mod[i] <- min(max(mod[i] * multiplier, 0.0), 1.0)
   return(mod)
}

# 4. Create lists of parameter sets with reduced and
# increased parameter values (90% and 110%).
min.params <- lapply(1:length(base.param), change,
                base=base.param, multiplier=0.9)
max.params <- lapply(1:length(base.param), change,
                base=base.param, multiplier=1.1)

# 5. Run separate simulations (function in step 1) with
# input factor values varied by +-10%.
sim.results <- list()
sim.results$min <- lapply(min.params, sim)
sim.results$max <- lapply(max.params, sim)

# 6. Calculate the deviation between the base model output
# and the outputs using 90% and 110% of the standard value.
dev.min <- sapply(sim.results$min, function(x) {
  return((x-sim.result.base)/sim.result.base * 100)})
dev.max <- sapply(sim.results$max, function(x) {
  return((x-sim.result.base)/sim.result.base * 100)})
```

We selected a variation of 10% for the values, which results, for example, in values for *survival-prob* of 0.8802 and 1.0 (truncated because the probability cannot exceed 100 per cent). This means that we ran the simulation with three different values of *survival-prob*: 0.8802, 0.978 and 1.0. To measure the sensitivity of a parameter, we calculate here the change in the output relative to the output with base parameter values. Therefore, we obtain a dimensionless sensitivity measure for all tested parameters that can be easily compared against each other. Of course, there are other sensitivity measures possible. Examples include the calculation of the partial derivative by dividing the change in the output by

the change in the parameter value or the calculation of the standard deviation of the output over multiple replications [Railsback and Grimm, 2012].

Table IV.6.: Results of a local sensitivity analysis. The columns list the different model outputs and the rows the different input variables (parameters). Values shown are per cent deviations of output values. When changing one parameter, all other parameters are kept constant. Negative output values indicate a reduction of the output value.

	abundance	variation	vacancy
scout-prob.min	-1.0	-6.2	13.7
scout-prob.max	4.0	-8.0	15.4
survival-prob.min	-99.9	-97.3	468.6
survival-prob.max	199.0	456.1	-100.0
scouting-survival.min	-0.3	-3.7	-1.4
scouting-survival.max	4.9	32.4	-9.8

Table IV.6 lists the result of this simple local sensitivity analysis for the example model. It shows that the three outputs are relatively insensitive to small variations in the parameters *scout-prob* and *scouting-survival*. In contrast, the model outputs are highly sensitive to variations in parameter *survival-prob*. However, this conclusion is based on the base values used. Choosing other base values or a different variation percentage could result in completely different conclusions, as the dependence of model output on a single input could be non-linear. Furthermore, we have only learned something about main effects and nothing about the interaction effects when two or more inputs are varied at the same time.

IV.4.3. Screening Methods

Screening methods try to answer the question which of a large set of potentially important inputs actually have a strong influence on the simulation output of interest. They are designed to be computationally efficient and able to explore a large set of inputs. Often, one-factor-at-time approaches are applied but, in contrast to local sensitivity analysis, with variations of the inputs over a wide range of values [Campolongo et al., 2000b]. We restrict ourselves here to Morris's elementary effects screening, which appears to be the most important suitable method for ABMs. Other well-known methods are often not suitable for ABMs. For example, to use Bettonvil's sequential bifurcation, available in package sensitivity [Pujol et al., 2013], the user needs to know the sign of the main effects of all tested parameters in advance, which often cannot be logically deduced in ABMs, as the relationships between parameter values and model outputs are complex and non-linear.

Morris's Elementary Effects Screening

The Morris method was developed to explore the importance of a large number of input factors in computer models that cannot be analysed by classical mathematical methods [Morris, 1991]. Furthermore, the method is free of assumptions about the model, for example, the signs of effects [Saltelli et al., 2004]. Based on individually randomised one-factor-at-a-time

designs, it estimates the effects of changes in the input factor levels, i.e., the parameter values, which are called elementary effects (EEs). The EEs are statistically analysed to measure their relative importance. The results of the Morris method are two measures for every investigated input factor: μ, the mean of the elementary effects, as an estimate of the overall influence of an input factor/parameter, and σ, the standard deviation of the elementary effects, as an estimate of higher order effects, i.e., non-linear and/or interaction effects [Campolongo et al., 2007]. Still, this method does not identify interactions between specific input factors but instead delivers only lumped information about the interaction effect of one factor with the rest of the model [Campolongo et al., 2000a]. Examples for the application of the Morris screening method in the context of ABM can be found in Imron et al. [2012], Vinatier et al. [2013], and Vinatier et al. [2009].

The Morris screening method is available in R through the sensitivity package [Pujol et al., 2013]. The function `morris` includes the Morris method with improvements in sampling strategy and the calculation of μ^* to avoid Type-II errors [Campolongo et al., 2007]. When positive and negative effects occur at different factor levels, they can cancel each other out in the calculation of μ, whereas μ^* can handle this case. The application of the `morris` function in R can be performed as follows:

```
# 1. Define a simulation function (sim) as done for
# Full factorial design.

# 2. Create an instance of the class morris and define min
# (referred to as binf) and max (bsup) values of the
# parameters to be tested, the sampling design (design, here
# oat = Morris One-At-a-Time design with 5 levels), and the
# number of repetitions of the design (r).
mo <- morris(model = NULL, factors = 3, r = 4,
         design = list(type = "oat", levels = 5,
       grid.jump = 3), binf = c(0.0, 0.95, 0.5),
             bsup = c(0.5, 1.0, 1.0), scale=TRUE)

# 3. Get simulation model responses for sampling points using
# the sim function defined in step 1.
sim.results <- apply(mo$X, 1, sim)

# 4. Add simulation results to the morris object.
tell(mo, sim.results)
```

For the interpretation of the results, Saltelli et al. [2004] recommend comparing the values of σ and μ, or better μ^*. High values of μ indicate that a factor has an important overall influence on the output and that this effect always has the same sign [Saltelli et al., 2004]. In contrast, when there is a high value of μ^* and a low value of μ, it indicates that there is a non-monotonic effect on the output. High values of σ indicate that the elementary effects strongly depend on the choice of the other input factors, whereas a high μ or μ^* and a low σ indicate that the elementary effect is almost independent of the values of the other factors, which means that it is a first-order/main effect. In summary, the Morris screening method delivers measures of relative importance but cannot quantify the strength of the effects.

The results of an application of the Morris screening method to our example model with 40 function calls is shown in Figure IV.12. The plots on the left side (panels A, C, and

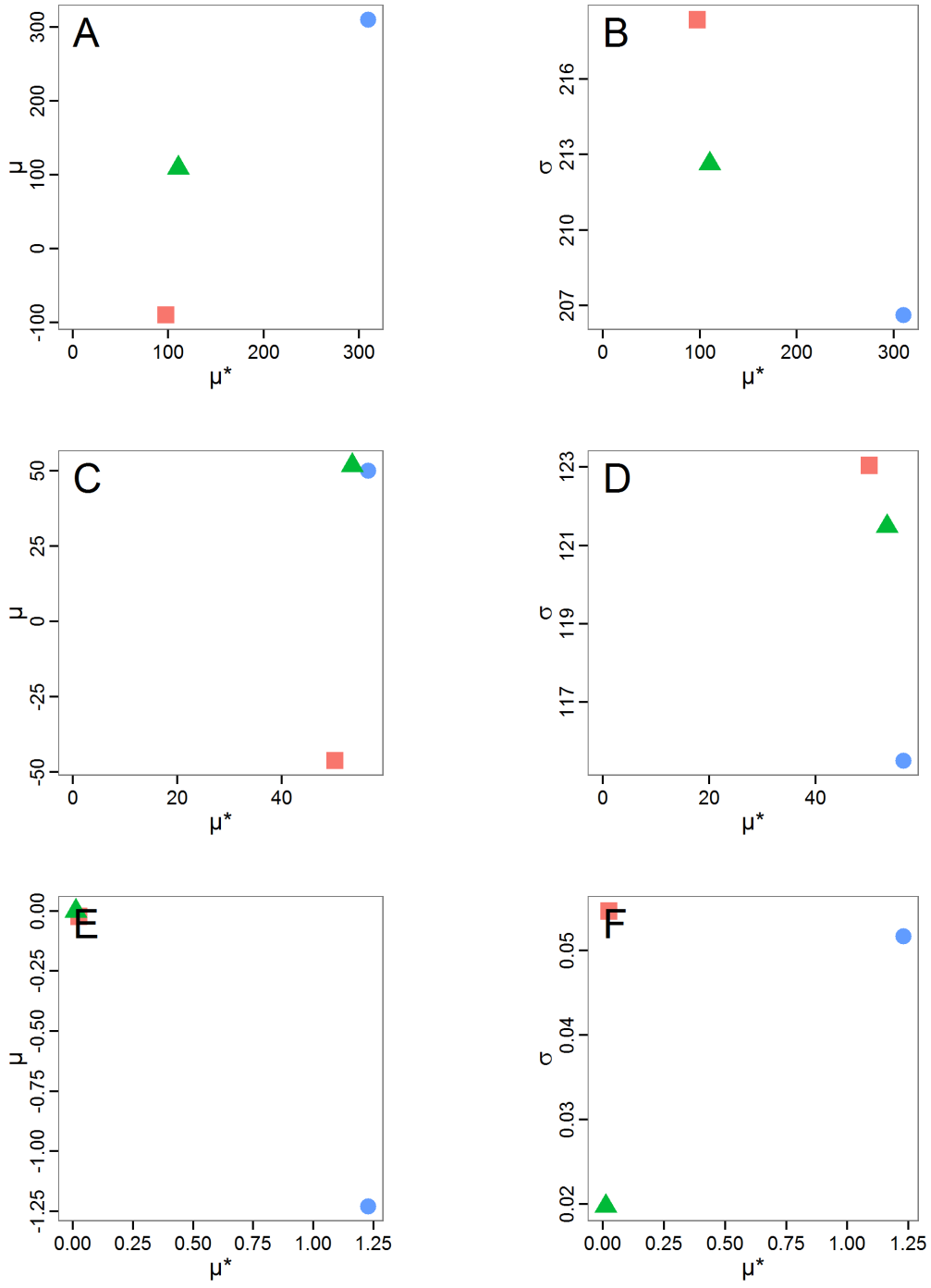


Figure IV.12.: Results of the Morris screening method. Red rectangle: *scout-prob*, blue point: *survival-prob*, green triangle: *scouting-survival*. A: Test for overall importance (high μ) and non-monotonic relationship (low μ and high μ*) for the abundance output. B: Test for interaction (high σ) and main effect (high μ* and low σ) for the abundance output. C: same as A for variation output. D: same as B for variation output. E: same as A for vacancy output. D: same as B for vacancy output.

E) indicate high overall influences of *survival-prob* for all three outputs, the abundance, variation, and vacancy criteria. The other two factors, *scout-prob* and *scouting-survival*, have a less important influence, but still are influential, on the abundance output (panel A). On the variation output (panel C), all three input factors are of nearly the same importance, and no clear importance ranking can be performed. From panel E, we learn that the other two factors, *scout-prob* and *scouting-survival*, are irrelevant for the vacancy output. Moreover, in panel A we see that the absolute values of μ and μ^* are the same for *scout-prob* but with a negative sign for μ. This leads to the conclusion that the influence of *scout-prob* on the abundance output is monotonic but negative. The same also applies to the variation output and to the influence of *survival-prob* on the vacancy output. No case of low μ but high μ^* value can be found; therefore, we can conclude that all effects are mostly monotonic. Panels B, D, and F deliver information about non-linear and/or interaction effects. Because all three factors are important for the abundance and variation outputs, we should check all three of these factors for non-linear/interaction effects. Panel B shows that the values of σ are highest for *scout-prob* and *scouting-survival*, although their μ^* values are relatively low. This leads to the conclusion that the influence of these two factors on the abundance output is strongly non-linear and/or dependent on the values of the other factors. Although the σ value for *survival-prob* is relatively low, compared to the other factors, it remains high ($\sim$66% of μ^*). This means that there is also an interaction and/or non-linear effect of *survival-prob* on the abundance output, but with a lower contribution to the overall influence relative to the other two factors. From the variation output in panel D we see that there is a very strong interaction/non-linear effect for *scout-prob* and *scouting-survival* and a lesser, but still very strong, interaction/non-linear effect detected for *survival-prob*. For the vacancy output, we do not have to interpret the σ values for *scout-prob* and *scouting-survival* because these factors have been identified as unimportant. For *survival-prob*, the only important factor for the vacancy output, the σ value is low (approx. 4% of μ^*). Therefore, we can conclude that it is mostly a main-/first-order effect.

Overall, we learned a great deal about the example model by applying the Morris screening method. Furthermore, a graphical analysis of the relationship between μ^* and μ as well as between μ^* and σ is simple but very useful. If one is only interested in ranking the input factors by their importance, a comparison of the values of μ^* should be sufficient.

IV.4.4. Global Sensitivity Analysis

In global sensitivity analysis, input variables are varied over the full range of their possible values. This distinguishes these methods from local sensitivity analysis. Moreover, in global sensitivity analysis effects are quantified, which is different from screening methods, which deliver only a ranking of the input variables by their importance for the output without quantification.

For some methods described below, such as partial correlation coefficients, it can be useful to perform a graphical analysis first to obtain a basic idea of the relationships between inputs and outputs. For such graphical analyses, for example, scatter and interaction plots, full factorial design and Latin hypercube sampling can be appropriate.

Excursion: Design of Experiment

The Design of Experiment (DoE) methodology, which was first formulated for experiments in agriculture [Lorscheid et al., 2012], can be used for sensitivity analysis. Introductions to the application of classical DoE for ABMs can be found, for example, in Campolongo and Saltelli [2000], Happe [2005], or Lorscheid et al. [2012].

As a first step, an experimental design must be selected. For simplicity, we use a full factorial design of the two extreme values of each of the k inputs (2^k, i.e., eight function calls for the example), which has also been used by Lorscheid et al. [2012] and Happe [2005]. Then, we run the simulation for the design points, i.e., parameter sets, and add the (averaged, in the case of stochastic models) simulation results to the so-called design matrix. This procedure can be run using package FrF2 [Groemping, 2013a] or DoE.base [Groemping, 2013c] and could look like this:

```
# 1. Define a function that runs the simulation model
# for a given input factor combination and returns the
# simulation output (averaged) as well as the output of
# each iteration into anova.df in case of repeated
# simulations for stochastic models. See Supplementary
# Materials (simulation_function5.R) for an
# implementation example using RNetLogo.
sim <- function(input, anova.df.name=="anova.df") {
   ...
   return(output)
}

# 2. Create full factorial design (2^k, i.e., k = number of
# parameters; therefore, only extreme values are tested).
ff <- FrF2(nruns=2^3, nfactors=3, randomize=False,
        factor.names=c('scout-prob',
                   'survival-prob',
                   'scouting-survival'))

# 3. Get simulation model response (sim.results and
# anova.df) for sampling points (ff) using sim function
# defined in step 1.
anova.df <- data.frame()
sim.results <- apply(as.data.frame(ff), 1, sim,
                anova.df.name="anova.df")

# 4. Add simulation model response to the design matrix.
ffr <- add.response(ff, response=sim.results)
```

Happe [2005] analysed the results by fitting a linear regression model (so-called metamodel) with the simulation inputs as independent variables and the simulation output as a dependent variable. Not performed by Happe [2005], but also known in DoE, is the usage of the metamodel to predict the results of input value combinations that have not been simulated (look for a `predict` function for the method you used to produce the metamodel in R). Non-parametric alternatives to linear or non-linear regression models for prediction as

recommended by Helton et al. [2006] are generalised additive models [GAM, see R package gam, Hastie, 2013], locally weighted regression (LOESS, see function `loess` in R's base package) and projection pursuit regression (function `ppr` in R's base package). For example, GAM was used by Han et al. [2012] for analysing a complex, stochastic functional-structural fruit tree model. Kriging, a geostatistical method, has also been applied for metamodel construction in ABMs; see, for example, Salle and Yildizoglu [2012]. Kriging methods are available in several R packages, see, for example, DiceEval [Dupuy and Helbert, 2013], kriging [Olmedo, 2011], or RandomFields [Schlather et al., 2013]. The approach of using metamodels to predict the model output of non-simulated input factor combinations is strongly related to the response surface method [RSM, see R package rsm, Lenth, 2009] [Ascough II et al., 2005].

The procedure described in Happe [2005] can be realised in R as follows:

```
# 5. Create effect plots.
MEPlot(ffr)
IAPlot(ffr)

# 6. Perform stepwise fitting of a (linear) regression model
# to the data of step 4 (stepAIC requires package MASS,
# Venables and Ripley, 2002).
in.design <- cbind(as.data.frame(ff), sim.results)
min.model <- lm(abundance ~ scout_prob, data=in.design)
max.model <- lm(abundance ~ scout_prob * survival_prob *
            scouting_survival, data=in.design)
lms <- stepAIC(min.model, scope=list(lower=min.model,
            upper=max.model))
```

The resulting main effect and two-way interaction effect plots are shown in Figure IV.13 and IV.13, respectively. The strongest main effects for all outputs are detected for the input factor *survival-prob*, with positive signs for abundance and variation and a negative sign for the vacancy output. For the two other input factors, no main effects for the vacancy output and a small one for the other two outputs were detected, with negative signs for *scout-prob* and positive signs for *scouting-survival*. The two-way interaction effect plots indicate interaction effects if the lines for a factor combination are not parallel. The less parallel the lines are, the higher is the expected interaction effect. We see interaction effects for all factor combinations for the abundance and the variation outputs but no two-way interaction effects for the vacancy output. These results are in accordance with the results from the Morris screening method.

Looking at Table IV.7, we see the results of a stepwise linear regression fitting (metamodelling). *Survival-prob* was retained in all three of the final regression models for the different outputs and was statistically significant (see column $Pr(>|t|)$). Furthermore, the sign of the estimate for *survival-prob* is in accordance with the visual findings. The other main effects found in the visual detection have not been retained in or added to the metamodel because they were not able to improve the model regarding the Akaike information criterion (AIC). In contrast, the main effect of *scouting-survival* has been added to the metamodel for the vacancy output, which was not detected in the visual analysis. However, this predictive variable has no statistically significant effect on the metamodel and should therefore be removed in a further step.

Our visual findings about the interaction effects have not been selected for the metamodel by the stepwise regression fitting. Only for the vacancy output has the interaction effect between *survival-prob* and *scouting-survival* been included, but this effect also has no significant effect on the metamodel but just improves the AIC. Therefore, it should be removed. These results can strongly depend on the selection measure used - here AIC, but R^2 and others are also possible - and on the direction of the stepwise search, here "forward" selection.

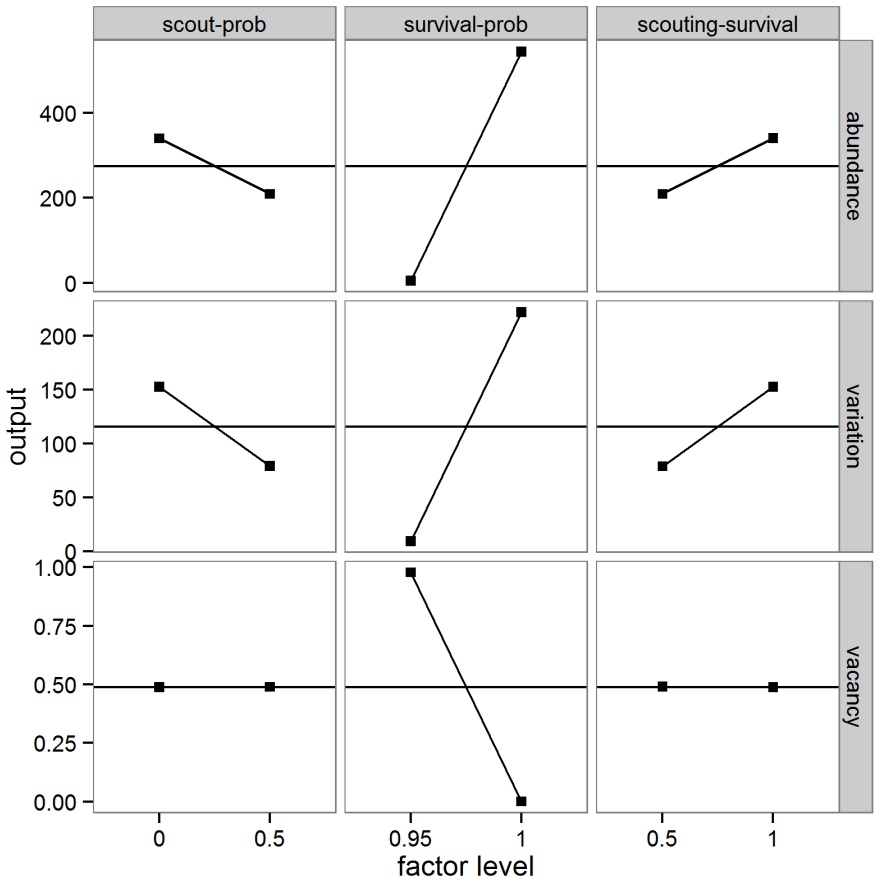


Figure IV.13.: Main effect plots (based on linear regression model). Parameters in columns (left: *scout-prob*, middle: *survival-prob*, right: *scouting-survival*) and outputs in rows (top: abundance, middle: variation, bottom vacancy). Horizontal lines (without rectangles) in rows visualise mean values. Right rectangle higher than left rectangle indicates a main effect with a positive sign and vice versa. Rectangles on the same output value (y-axis) indicate no main effect.

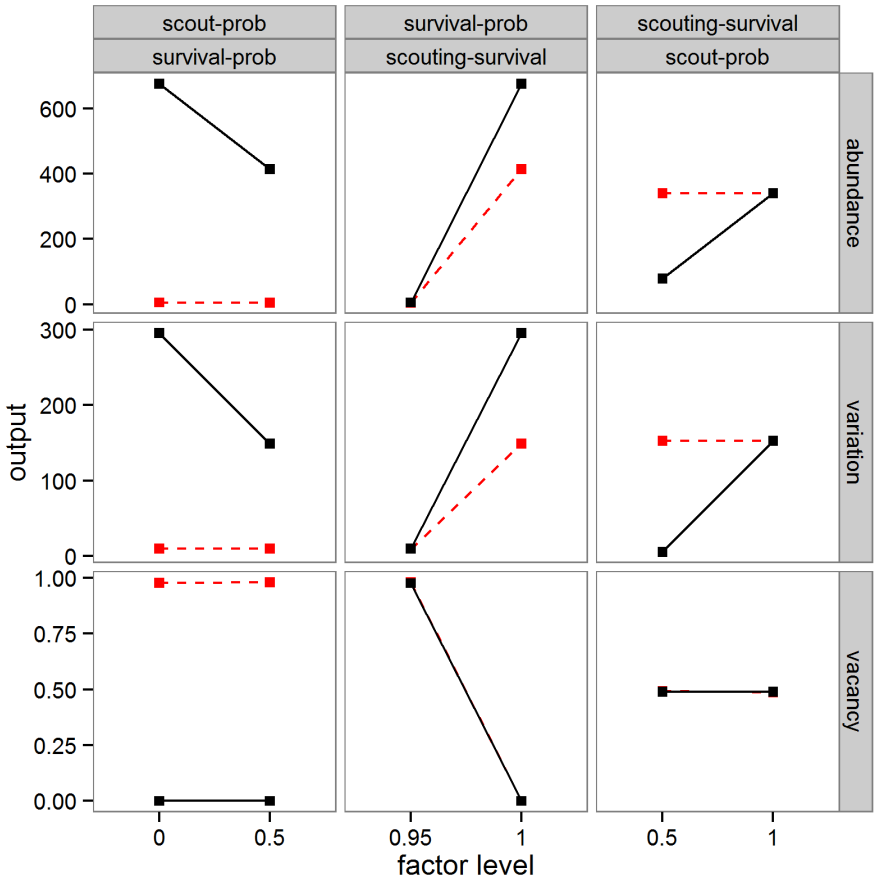


Figure IV.14.: Interaction effect plots (based on linear regression). Left column: *scout-prob* interacting with *survival-prob*. Red dotted line: value of *survival-prob* is 0.95, black solid line: value of *survival-prob* is 1.0. Middle column: *survival-prob* interacting with *scouting-survival*. Red dotted line: value of *scouting-survival* is 0.5, black solid line: value of *scouting-survival* is 1.0. Left column: *scouting-survival* interacting with *scout-prob*. Red dotted line: value of *scout-prob* is 0.0, black solid line: value of *scout-prob* is 0.5. Output in rows (top: abundance, middle: variation, bottom: vacancy). Lines in parallel indicate no interaction effect.

Table IV.7.: Results of a stepwise linear regression fitting based on 2^k-design. Names in the first column are the input factors. Single names are main effects. Names combined with a colon are interaction effects.

	Estimate	Std. Error	t-Value	$Pr(>\|t\|)$
a) abundance output				
Final Model: abundance ~*survival-prob* (adj. R^2: 0.6958)				
Intercept	-10238	2550	-4.02	0.00699 **
survival-prob	10783	2614	4.13	0.00618 **
b) variation output				
Final Model: variation ~*survival-prob* (adj. R^2: 0.5127)				
Intercept	-4034	1436	-2.81	0.0307 *
survival-prob	4257	1472	2.89	0.0276 *
c) vacancy output				
Final Model: vacancy ~*survival-prob* + *scouting-survival* + *survival-prob*:*scouting-survival* (adj. R^2: 1.0)				
Intercept	19.70	0.09	230.31	2.13e-09 ***
survival-prob	-19.70	0.09	-224.63	2.36e-09 ***
scouting-survival	-0.20	0.11	-1.81	0.14
survival-prob:*scouting-survival*	0.20	0.11	1.77	0.15

An alternative approach to that used by Happe [2005] was used by Lorscheid et al. [2012]. They calculated an effect matrix that delivers a description of the main and interaction effects. In a preliminary step, the results of all simulations (data.frame `anova.df` of step 3 in the R code above) are analysed for significant main and interaction effects using an ANOVA (or non-parametric substitute). If needed, this process can be run iteratively as a "cascaded DoE" for complex inputs, i.e., factors that represent several sub-factors, as described in Lorscheid et al. [2012]. A non-iterative procedure, which could be easily adapted to an iterative one, can be realised in R as follows (steps 1-4 correspond to the previous R code example):

```
# 5. Calculate ANOVA with anova.df data.frame from step 3
glm(formula = output ~ scout_prob * survival_prob *
   scout_survival, data=anova.df)

# 6. Define a function that calculates the effects between
# parameters after Saltelli et al. [2000][1].

# 7. Calculate main and interaction effects using the
# function defined in step 6 and data in ffr from step 4 up
# to the desired interaction level. Use desnum(ffr) to obtain
# the level signs needed for the effect calculation. See
# Supplementary Materials (SM14b_DoE_effect_matrix.R) for an
# implementation example.
```

[1] $E_j = \frac{\sum_{i=1}^{n}(S_{ij}y_j)}{(n/2)}$

Table IV.8.: Results of the ANOVA. Signif. codes: '***' 0.001 '**' 0.01 '*' 0.05. Names in the first column are the input factors. Single names are main effects. Names combined with a colon are interaction effects.

	Df	Sum Sq.	Mean Sq.	F value	$Pr(>\|t\|)$
1) abundance output					
scout-prob	1	342330	342330	309953	< 2e-16 ***
survival-prob	1	5813439	5813439	5263611	< 2e-16 ***
scouting-survival	1	343050	343050	310604	< 2e-16 ***
scout-prob:*survival-prob*	1	341010	341010	308757	< 2e-16 ***
scout-prob:*scouting-survival*	1	341153	341153	308887	< 2e-16 ***
survival-prob:*scouting-survival*	1	340292	340292	308107	< 2e-16 ***
scout-prob:*survival-prob*:					
scouting-survival	1	342186	342186	309822	< 2e-16 ***
Residuals	72	80	1		
2) variation output					
scout-prob	1	108419	108419	52704	< 2e-16 ***
survival-prob	1	905960	905960	440401	< 2e-16 ***
scouting-survival	1	109103	109103	53036	< 2e-16 ***
scout-prob:*survival-prob*	1	108228	108228	52611	< 2e-16 ***
scout-prob:*scouting-survival*	1	108216	108216	52606	< 2e-16 ***
survival-prob:*scouting-survival*	1	107547	107547	52280	< 2e-16 ***
scout-prob:*survival-prob*:					
scouting-survival	1	108430	108430	52710	< 2e-16 ***
Residuals	72	148	2		
3) vacancy output					
scout-prob	1	0	0	1.03e-1	0.749
survival-prob	1	19.11	19.11	2.33e+5	< 2e-16 ***
scouting-survival	1	0	0	1.46e-1	0.231
scout-prob:*survival-prob*	1	0	0	1.03e-2	0.749
scout-prob:*scouting-survival*	1	0	0	8-33e-1	0.364
survival-prob:*scouting-survival*	1	0	0	1.46e-1	0.231
scout-prob:*survival-prob*:					
scouting-survival	1	0	0	8.33e-1	0.364
Residuals	72	0.006	0		

The results of the ANOVA for the identification of significant effects are given in Table IV.8. For the abundance and variation outputs, all terms are highly significant, whereas only the input factor *survival-prob* is significant for the vacancy output.

Table IV.9.: DoE main/first-order and second-order effect matrix.

	scout-prob	*survival-prob*	*scouting-survival*
1) abundance			
scout-prob	-130.830	-130.578	130.605
survival-prob		539.140	130.440
scouting-survival			130.968

2) variation			
scout-prob	-73.627	-73.562	73.558
survival-prob		212.833	73.330
scouting-survival			73.859
3) vacancy			
scout-prob	0.0007	-0.0007	0.0019
survival-prob		-0.9776	0.0025
scouting-survival			-0.0025

When comparing the ANOVA results (Table IV.8) with the effect matrix (Table IV.9, calculated in step 7 of the R code sample above), we see that the findings from the ANOVA correspond to the values in the effect matrix, i.e., we find high main effect values (on the diagonal of each sub-matrix) for the abundance and variation outputs for all parameters, and for vacancy, a considerable effect (with negative sign) only for the main effect of *survival-prob*. The main effect of *scout-prob* and the interaction between *scout-prob* and *survival-prob* have negative signs for the abundance and variation outputs, whereas the other effects have positive signs. These findings correspond to the main and interaction plots based on linear regressions from the previous method.

A 2^k -design, as used by Happe [2005] and Lorscheid et al. [2012], will only be appropriate for linear and monotonic relationships between inputs and outputs. Of course, different sampling designs with space-filling curves like Latin hypercube sampling (LHS) could be applied, and in case of metamodelling, non-linear regression models, splines, neural networks or Kriging methods can be used [Siebertz et al., 2010]. However, methods that build on the fundamentals of DoE and were especially developed for sensitivity analysis are already available and more efficient. For non-linear and non-monotonic responses, see, for example, the Sobol' method (see below) as an adaption of DoE principles to computational experiments. Nevertheless, the adapted DoE methods presented here are relatively easy to understand and communicate and do not require extensive computations (depending on the sampling scheme). Therefore, if the model characteristics fit the methods' requirements, they can be used for a quick but still useful first global sensitivity analysis.

Regression-Based Methods

Partial (rank) correlation coefficient Correlation techniques measure the strength of a linear relationship between an input and an output variable. Partial correlation techniques enable the user to measure the strength of the relationship of more than one input variable [Campolongo et al., 2000b]. Therefore, if linear relationships are expected, the partial correlation coefficient (PCC) can be applied as a sensitivity measure. Instead, if non-linear but monotonic associations are expected, partial rank correlation coefficients (PRCC) are used to measure the strength of the relationship. Both methods are robust measures as long as input factors are uncorrelated [Marino et al., 2008]. An example of the application of PRCC to an ABM can be found in Marino et al. [2008].

For both PCC and PRCC, a sample of model outputs must be created first. It is preferable to use a Latin hypercube sampling (LHS) scheme [Blower and Dowlatabadi, 1994], but other sampling schemes could also be applied. Both PCC and PRCC are implemented in the

sensitivity package for R [Pujol et al., 2013]. Therefore, calculating the PCC/PRCC in R can look like this:

```
# 1. Define a simulation function (sim) as done for
# Full factorial design.

# 2. Create parameter samples from, for example, a uniform
# distribution using function lhs from package tgp
# [Gramacy and Taddy, 2013].
param.sets <- lhs(n=100, rect=matrix(c(0.0,0.95,0.5,1.0), 2))

# 3. Iterate through the parameter combinations from step 2
# and call function sim from step 1 for each parameter
# combination.
sim.results <- apply(as.data.frame(param.sets), 1, sim)

# 4. Calculate the partial (rank) correlation coefficient
# based on the simulation results of step 3.
pcc.result <- pcc(x=param.sets, y=sim.results, nboot = 100,
              rank = FALSE)
```

The result of an application of PCC/PRCC on the example model with 200 samples is shown in Figure IV.15. Obviously, there is a very strong positive linear relationship between the *survival-prob* input factor and the abundance output as well as a strong negative linear relationship between the same input factor and the vacancy output. Because the (absolute) values for the PRCC for *scout-prob* and *scouting-survival* in panel C are greater than for PCC, this could indicate a non-linear but monotonic relationship between these two factors and the vacancy output. For the abundance output (panel A), there is a weak linear relationship detected for the input factors *scout-prob* (with a negative sign) and *scouting-survival* (with a positive sign). For the variation output (panel B) there is no obvious importance ranking. Either there is only a small influence of the input factors on the output, the relationship is non-monotonic or the input factors are not independent (which is actually the case, as we will see later using variance decomposition).

In summary, the PCC and especially the PRCC are often used as importance measures. They are relatively easy to understand, interpret and communicate and are a quantitative alternative to qualitative, visual sensitivity analysis using, for example, scatterplots [Hamby, 1995].

Standardised (rank) regression coefficient The methods of standardised regression coefficient (SRC) and standardised rank regression coefficient (SRRC) deliver similar results to those of PCC/PRCC but are more strongly influenced by the distribution from which the tested parameter values are drawn [Campolongo et al., 2000b]. In a first step, fitting a linear regression model to the simulation data delivers measures of the relationship between the inputs and output of the simulation model. The regression coefficients are standardised by multiplication with the ratio between standard deviations of input factor and output value. In SRRC, the original values are replaced by ranks. As with PCC and PRCC, SRC is only able to measure linear relationships, whereas SRRC can also be used for non-linear but monotonic associations between input and output variables when little or no correlation between the input factors exists [Marino et al., 2008].

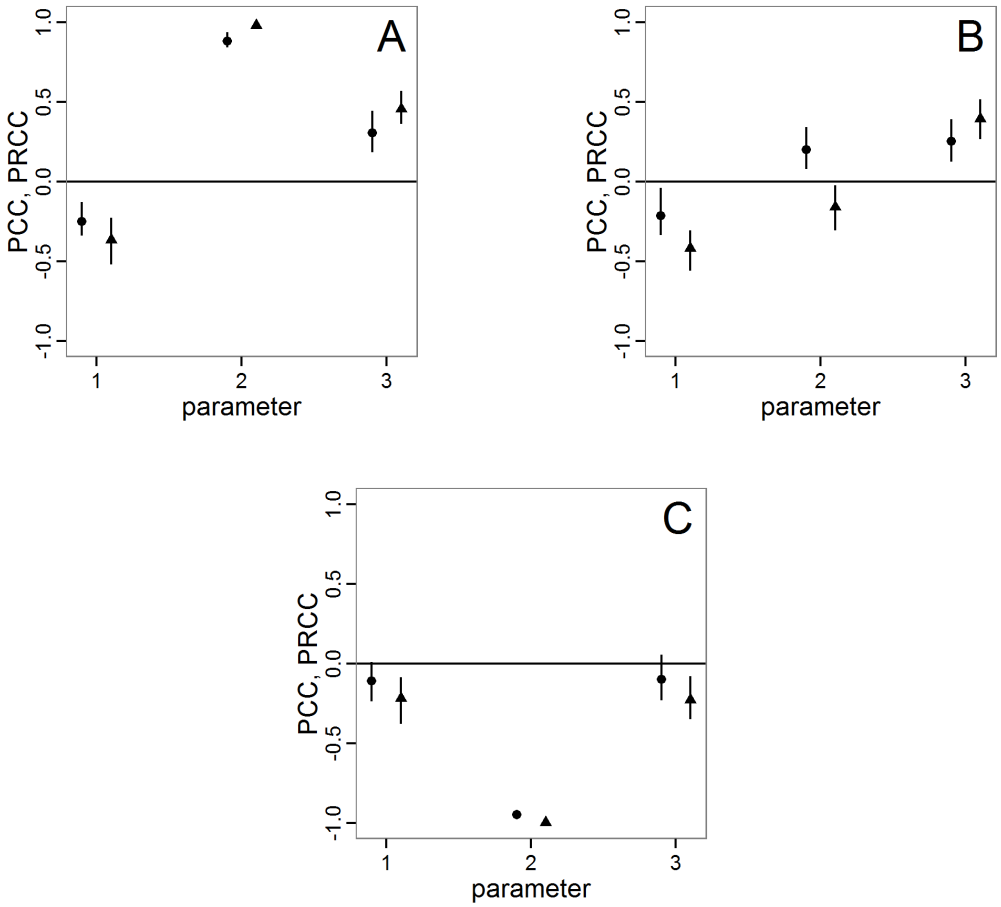


Figure IV.15.: Results of the PCC/PRCC. A: for abundance output. B: for variation output. C: for vacancy output. Circles (to the left of each x-axis tick) show original PCC values (measure of linear relationship). Triangles (to the right of each x-axis tick) show original PRCC values (measure of linear or non-linear but monotonic relationship). Sticks show bootstrapped 95% confidence intervals of corresponding sensitivity indices. Numbers on x-axis for all plots: 1: *scout-prob*, 2: *survival-prob*, 3: *scouting-survival*.

These methods are also implemented in the sensitivity package for R [Pujol et al., 2013], and the application is equivalent to that of PCC/PRCC (therefore not shown here; see PCC/PRCC and replace the function call `pcc` with `src`).

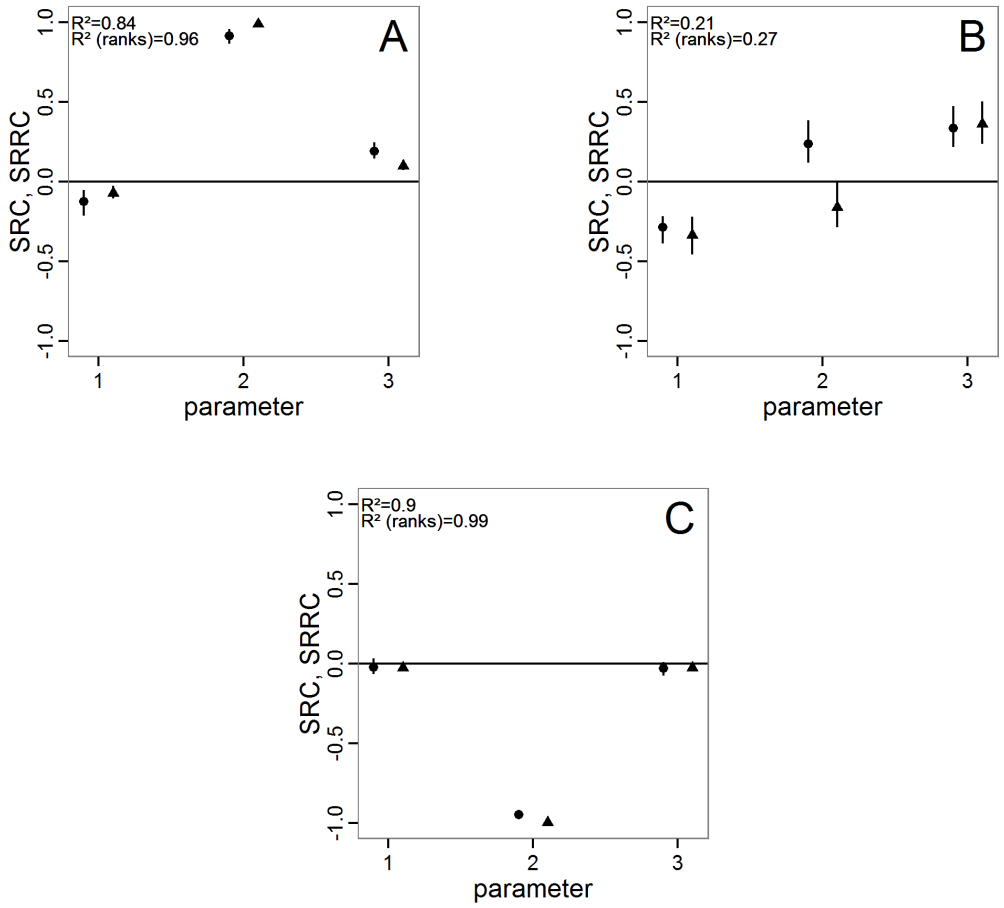


Figure IV.16.: Results of the SRC/SRRC. A: for abundance output. B: for variation output. C: for vacancy output. Circles (to the left of each x-axis tick) show original SRC values (measure of linear relationship). Triangles (to the right of each x-axis tick) show original SRRC values (measure of linear or non-linear but monotonic relationship). Sticks show bootstrapped 95% confidence intervals of corresponding sensitivity indices. Numbers on x-axis for all plots: 1: *scout-prob*, 2: *survival-prob*, 3: *scouting-survival*. Top R^2 in each plot corresponds to SRC and bottom R^2 belongs to SRRC, giving the proportion of variation in the data captured by the regression model.

The results of an application of the SRC/SRRC method to the example model based on Latin hypercube sampling with 200 samples drawn from a uniform distribution for all parameters are given in Figure IV.16. Campolongo et al. [2000b] recommend calculating the coefficient of determination (R^2), which is not processed by the sensitivity package. Therefore, we wrote a small function to calculate it (see Supplementary Materials) because it tells us how well the linear regression model reproduces the output, i.e., how much of the out-

put's variance is explained by the regression model [Campolongo et al., 2000b]. In the case of non-linear relationships, using rank transformation (SRRC) can improve the R^2 but will also alter the model because it becomes more additive and therefore includes less of the interaction effects [Campolongo et al., 2000b]. In general, when there are strong interaction effects or non-monotonic relationships, R^2 will be low and the application of SRC/SRRC is not very useful. Saltelli et al. [2008] recommend the usage of these methods only for models where R^2 is greater than or equal to 0.7. As shown in Figure IV.16, this condition is met for the abundance and vacancy outputs but not for the variation output (panel B). Therefore, we should discard the results for the variation output. For the remaining two outputs, there is a strong dominance of the *survival-prob* input factor. There is only a small change in the coefficient values when using SRRC instead of SRC, and the R^2s for SRC are already very high. This leads to the conclusion that there is a strong linear effect of *survival-prob* on these two outputs, with a positive sign for the abundance (SRC: 0.9865) and a negative sign for the vacancy output (SRC: -0.9987). Note that the absolute value of the SRC or SRRC gives a measure of the effect strength, and the sign defines the direction of the effect. All in all, the results are very similar to the findings with PCC/PRCC.

Variance Decomposition Methods

For the investigation of non-linear and non-monotonic relationships between the inputs and outputs one should apply variance decomposition methods [Marino et al., 2008], but they can also be applied to models with monotonic and/or linear relationships. The three methods presented here are so-called total sensitivity indices (T_{S_i}) because they quantify the parameters' main effects as well as all interaction effects of any order [Ascough II et al., 2005]. These methods are, compared to the other sensitivity methods presented so far, computationally expensive. Therefore, it is recommended to first identify the important parameters by using, for example, Morris screening, and then restrict the variance decomposition methods to the most important parameters [Campolongo et al., 2000b].

In analogy to ANOVA, the methods use techniques for the decomposition of the total variance of model output into first- (main) and higher-order (interaction) effects for each input factor [Confalonieri et al., 2010]. When model input is varied, model output varies too, and the effect is measured by the calculation of statistical variance. Then, the part of the variance that can be explained by the variation of the input is determined [Marino et al., 2008].

Sobol´ method The Sobol´ method delivers a quantitative measure of the main and higher-order effects. It is very similar to effect calculation in DoE theory [Saltelli et al., 1999] and can be considered the adaptation of classical DoE to computer simulations. The idea is that the total variance is composed of the variance of the main and the interaction effects. Therefore, multiple integrals for the partial effect terms of different orders are extracted by decomposition and evaluated using Monte-Carlo methods instead of using factor levels, as is performed in classical DoE. For further details see, for example, the original work of Sobol' [1990] or that of Chan et al. [2000].

An implementation of the Sobol´ method can be found in the sensitivity package for R [Pujol et al., 2013]. The `sobol` function is used in this way:

```
# 1. Define a simulation function (sim) as done for
# Full factorial design.
```

```
# 2. Create two random input factor samples (input.set.1,
# input.set.2) as required by the Sobol´ method, for
# example two Latin hypercube samples
# (see also Partial (rank) correlation coefficient).

# 3. Create an instance of the class sobol using the result
# of step 2. Choose the order of variance decomposition:
so <- sobol(model = NULL, X1 = input.set.1,
         X2 = input.set.2, order = 2, nboot = 100)

# 4. Get the simulated model response for all input factor
# combinations needed by the sobol method by calling the
# function defined in step 1.
sim.results <- apply(so$X, 1, sim)

# 5. Add the simulation results to the sobol instance.
tell(so, sim.results)
```

A modification of the Sobol´ method exists that reduces the required number of model evaluations and delivers a main and total effect index, similar to the eFAST method. The implementation of this optimised method is also included in the sensitivity package [Pujol et al., 2013]. The call of the `sobol2007` function is similar to the `sobol` function with the exception that the number of orders of effects that should be calculated cannot be defined because this method cannot deliver single two- or higher-order effects explicitly as the original method can. Note that there are further modified versions of Sobol´ algorithms available in the package sensitivity (see functions `sobol2002`, `sobolEff` and `soboljansen`).

The results of a usage example of the `sobol2007` function with 3000 simulation function calls are shown in Figure IV.17. For every output and every input factor there are two candle sticks, the left one for the 95% confidence interval of the main/first-order effect index and the right one for the 95% confidence interval of the total effect index. When the confidence bands are very large, this can be an indicator of too-small samples. Furthermore, it can happen that the index values are negative or exceed 1 due to numerical errors in the calculation, but these can be treated as 0 and 1, respectively [Saltelli et al., 2008]. The values of the indices are percentage measures of the contribution of the input factor to the variance of the output.

For pure additive models, the sum of the main effect indices (S_i) is 1; for others, it is below 1 and can never exceed 1 (only due to numerical errors). For example, the abundance output (panel A in Figure IV.17) sums up to approx. 0.74 (bias corrected), which means that the model is largely additive, i.e., only rather small interactions between the parameters occur, with a very strong contribution of the main effect of *survival-prob* (responsible for approx. 72% of the variation of the output). However, the quantification of the main effect contains considerable uncertainty, as indicated by the confidence interval. For the vacancy output (panel C), the contribution of *survival-prob* to the variance of the output is approx. 95%, i.e., the output variation depends nearly completely on the main effect of *survival-prob*. For the variation output (panel B), the main effects contribute only a very small proportion to the variance of the output. They sum up to approx. 13%.

The total sensitivity index (S_{T_i}) contains the main effect and all interaction effects with the

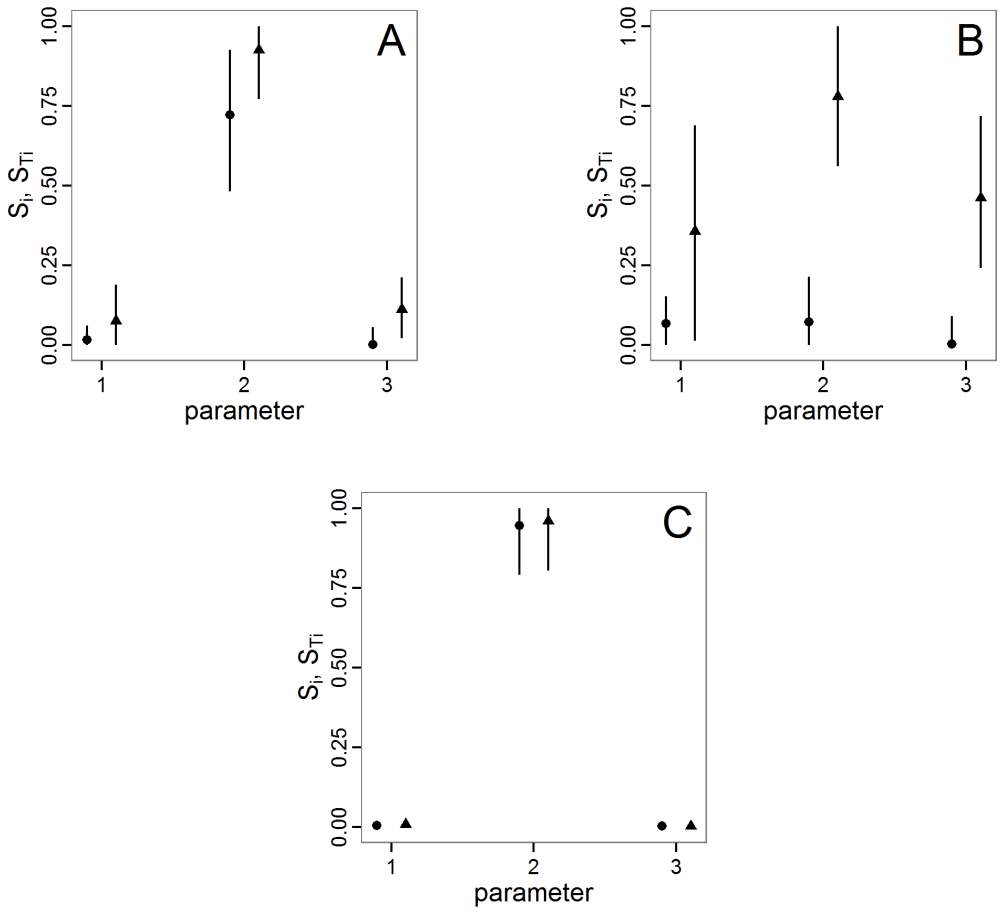


Figure IV.17.: Results of the Sobol´ method with modifications by Saltelli (`sobol2007` function). A: for abundance output. B: for variation output. C: for vacancy output. Circles (to the left of each x-axis tick) show bias-corrected original first-order sensitivity index values (S_i), i.e., main effects. Triangles (to the right of each x-axis tick) show bias-corrected original total sensitivity index values (S_{T_i}), i.e., main and all interaction effects. Sticks show bootstrapped 95% confidence intervals of corresponding sensitivity indices. Negative values and values above 1, due to numerical errors, are set to 0 and 1, respectively. Numbers on x-axis for all plots: 1: *scout-prob*, 2: *survival-prob*, 3: *scouting-survival*.

other input factors. Therefore, it is often greater than the main effect index (S_i). Equality between S_i and S_{T_i} means that the input factor has no interaction terms with the other input factors. In contrast to S_i, the sum of all indices of S_{T_i} is often greater than 1. Only if the model is perfectly additive, i.e., no interactions between the parameters exist, the sum is equal to 1 [Saltelli et al., 2008].

In Figure IV.17, we see that there is nearly no interaction effect for the vacancy output (panel C), and the model is strongly additive. In contrast, strong interaction effects are uncovered by the S_{T_i} for the variation output (panel B). The most important factor is *survival-prob*, but the confidence bands for all factors are large for S_{T_i}, i.e., the conclusions are very uncertain. For the abundance output (panel A), we see an interaction effect explaining approx. 20% of the variance of the output and small interaction effects for the other two factors. Ostromsky et al. [2010] provide some rules of thumb for the categorisation of input factors. They classified input factors with total sensitivity index (S_{T_i}) values of greater than 0.8 as very important, those with values between 0.5 and 0.8 as important, those with values between 0.3 and 0.5 as unimportant and those with values less than 0.3 as irrelevant. Using this classification, we come to the conclusion that the input factors *scout-prob* and *scouting-survival* are irrelevant for the abundance output, whereas *survival-prob* is very important. For the variation output, the input factor *survival-prob* is important whereas *scout-prob* and *scouting-survival* are unimportant (with high uncertainty), and for the vacancy output *survival-prob* is very important whereas the other two input factors are irrelevant.

Applying the standard Sobol´ method also enables us to analyse the higher-order effects separately. We calculated the Sobol´ index up to the second order with 3500 simulation function calls (Figure IV.18). The results for the main effects exhibit the same pattern as described for the `sobol2007` function but are numerically not exactly identical because this calculation is based on a new data sample. Taking into account second-order effects (S_{ij}) shows that the main part of the interaction effects previously identified for *survival-prob* on the abundance output (panel A) results from interactions with both of the other input factors, *scout-prob* and *scouting-survival*, whereas there is no interaction effect between *scout-prob* and *scouting-survival*. For the variation output (panel B) we find the same pattern but with a much higher proportion of the interaction between *scout-prob* and *survival-prob*. This explains the very large differences between the main effect index and the total sensitivity index from function `sobol2007` for the variation output (panel B). In accordance with the results of the `sobol2007` function, we see no remarkable second-order sensitivity index value for the vacancy output in panel C.

As we have seen, the Sobol´ method is able to deliver all information needed for a comprehensive global sensitivity analysis. The method, however, is computationally more demanding than the other methods presented so far, and understanding the method in detail can be challenging.

Extended Fourier amplitude sensitivity test Extended Fourier amplitude sensitivity test (eFAST) uses, as the name suggests, Fourier series expansion to measure the strength of the influence of uncertain inputs on the model output. Being an extension of the original FAST [Cukier et al., 1973], eFAST adds the calculation of a total effects index (STi), which is the sum of main and all higher-order effects [Saltelli et al., 1999]. For details on this method see Saltelli et al. [1999] or Chan et al. [2000]. eFAST delivers the same measures of sensitivity

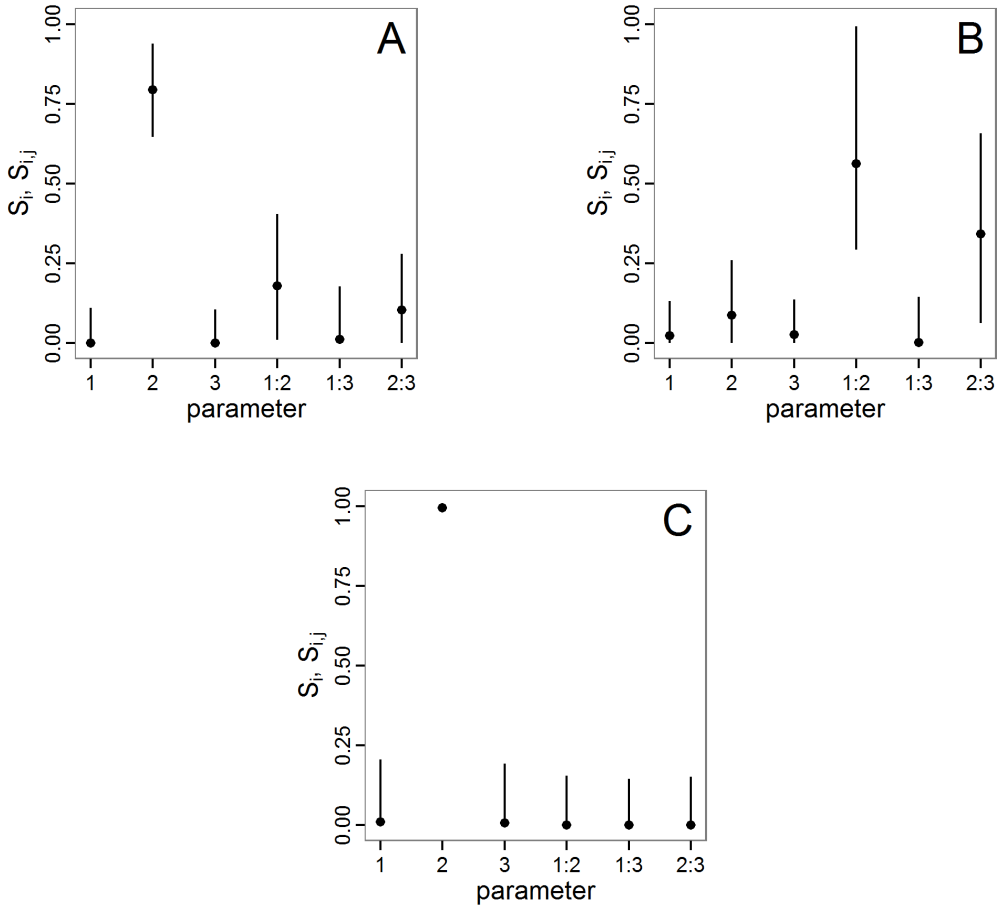


Figure IV.18.: Results of the Sobol´ method. A: for abundance output. B: for variation output. C: for vacancy output. Circles show bias-corrected original sensitivity index values. Sticks show bootstrapped 95% confidence intervals of corresponding sensitivity indices. Negative values and values above 1, due to numerical errors, are set to 0 and 1, respectively. Numbers on x-axis for all plots: 1: *scout-prob*, 2: *survival-prob*, 3: *scouting-survival*. Single numbers are main/first-order effects (S_i) whereas numbers combined with colons show second-order interaction effects (S_{ij}).

as the Sobol´ method (except specific interaction indices) but requires fewer simulations and is therefore computationally less expensive than the classical Sobol´ method [Ravalico et al., 2005]. Furthermore, it is more robust, especially at low sample sizes, to the choice of sampling points [Saltelli et al., 1999].

The classical FAST algorithm is available in package fast for R [Reusser, 2013], and the eFAST algorithm is included in the package sensitivity [Pujol et al., 2013]. Using the `fast99` function of the sensitivity package in R works in this way:

```
# 1. Define a simulation function (sim) as done for
# Full factorial design.

# 2. Create an instance of class fast99 with quantile
# functions for all input factors.
f99 <- fast99(model = NULL, factors = 3, n = 1000,
          q = c("qunif","qunif","qunif"),
          q.arg = list(list(min=0.0,max=0.5),
               list(min=0.95,max=1.0),
               list(min=0.5,max=1.0)))

# 3. Get the simulated model response for all input factor
# combinations needed by the fast99 method by calling the
# function defined in step 1.
sim.results <- apply(f99$X, 1, sim)

# 4. Add the simulation results to the fast99 instance.
tell(f99, sim.results)
```

The interpretation of the results of the eFAST method is the same as for the Sobol´ method (function `sobol2007`). The method returns a main/first-order sensitivity index (S_i) as well as a total sensitivity index (S_{T_i}) that also contains interaction effects. Figure IV.19 shows the results based on 600 simulation function calls. The results are very similar to the results of the Sobol´ method. There is a strong effect of input factor *survival-prob* for the abundance (panel A) and vacancy (panel C) outputs. There are nearly no interaction effects for the vacancy output and just minor interaction effects for the abundance output, whereas the main reasons for variance in the variation output are interaction effects. In contrast to the results of the Sobol´ method from the previous section, here the input factor *scout-prob* explains slightly more of the variance in the variation output than the *survival-prob* input factor, whereas with the Sobol´ method it was the other way around and the S_{T_i} of *scouting-survival* was larger than the S_{T_i} of *scout-prob*. The reasons for this could be due to differences in the methods themselves or to differences in the sampling schemes, too-small sample sizes, and/or a lack of convergence regarding stochastic effects, i.e., not enough replications for one parameter set. We have already observed that the confidence bands calculated for the Sobol´ indices have been large, i.e., the results are uncertain. Therefore, it is not surprising that the results for the variation output differ. Nevertheless, the overall picture is the same between the Sobol´ and eFAST methods. The choice of one or the other method depends primarily on the question of whether someone is interested only in main- and total-effect indices (then select eFAST; if confidence intervals are of interest then select the optimised Sobol´ method), or if one wants also to know the second- and higher-order effect indices

explicitly (then select the original Sobol´ method, but keep in mind that it will not deliver total-effect indices).

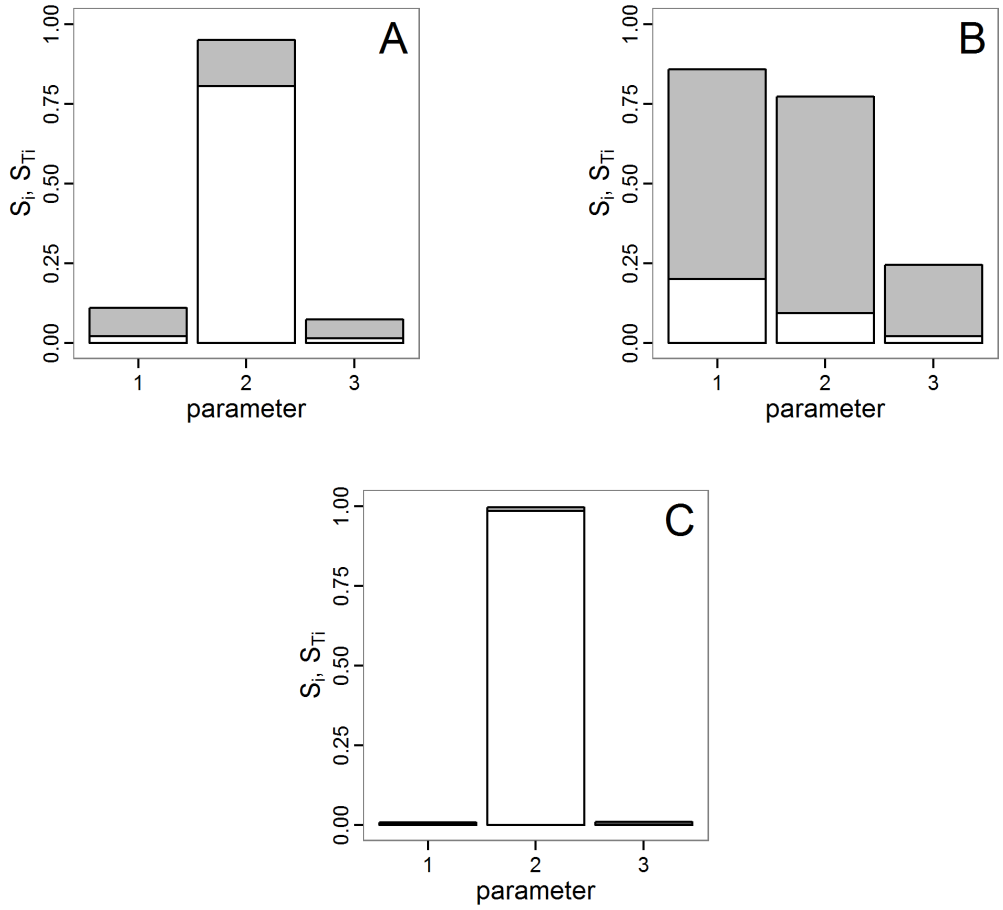


Figure IV.19.: Results of the eFAST method. A: for abundance output. B: for variation output. C: for vacancy output. Numbers on x-axis for all plots: 1: *scout-prob*, 2: *survival-prob*, 3: *scouting-survival*. White bar: first-order effects (S_i), i.e., main effects. Sum of white and grey bars: total effect (S_{T_i}), i.e., main and all interaction effects.

FANOVA decomposition with Gaussian Process The method of fitting a Gaussian process (GP) to replace a computationally expensive model is related to the metamodel approach in DoE. A GP is a stochastic process, i.e., a generalisation of a probability distribution to functions of random variables, where each finite collection follows a joint (multivariate) normal distribution [Grimmett and Stirzaker, 1993, Rasmussen and Williams, 2006, Snelson, 2007]. GPs are used for interpolation, extrapolation and smoothing of time series [Wiener, 1950] and spatial data. The geostatistical Kriging method, for example, is based on GPs [Snelson, 2007]. A GP, as a surrogate or emulator of a computer model, can be further used for a Functional Analysis of Variance (FANOVA), as performed by Dancik et al. [2010]. The R package mlegpFULL [Dancik, 2009, available on request from the developer

of the online available mlegp package] can perform the fitting procedure and the FANOVA decomposition. The following shows how this works:

```
# 1. Run a Latin hypercube sampling as done above
# (see Partial (rank) correlation coefficient).
# The result should be two variables: the first containing
# the input factor combinations (param.sets) and the second
# containing the corresponding model responses (sim.results).

# 2. Fit a Gaussian process to the simulation results.
fit <- mlegp(param.sets, sim.results, param.names =
          c('scout-prob',
            'survival-prob',
            'scout-survival'))

# 3. Calculate the FANOVA decomposition.
FANOVADecomposition(fit, verbose=FALSE)
```

We used a Latin hypercube sampling with only 500 samples, i.e., only 500 simulation function calls, to fit the Gaussian process. The results of the FANOVA decomposition using this Gaussian process are shown in Table IV.10. They contain results for main/first-order as well as second-order interaction effects but do not deliver total sensitivity indices as eFAST and Sobol´ with modification by Saltelli do. Nevertheless, the results here are in good accordance with the results of the eFAST and Sobol´ methods. The variation in the abundance output is mostly determined by the variation of the input factor *survival-prob* (ca. 81%), and there are only small second-order interaction effects. The second-order interaction effects here are primarily determined under participation of *survival-prob* with the other two input factors. There is nearly no interaction effect between *scout-prob* and *scouting-survival*, as we already know from the Sobol´ method. The variance in variation output is determined to a large extent by the interaction effects of *survival-prob* with the other two input factors. The main effects are less important, but the ranking of importance of the three input factors differs in comparison to the eFAST and Sobol´ methods. From the Sobol´ method we already know that the results for the variation output are uncertain. Therefore, this difference is not surprising. For the vacancy output the results here again fully match the results of the eFAST and Sobol´ methods. The most important factor is *survival-prob*, explaining approx. 99% of the variance with its first-order effect (eFast: 98%, Sobol': 99%, optimised Sobol': 95%).

Table IV.10.: Results of the FANOVA decomposition method using a fitted Gaussian process. Names in the first column are the input factors. Single names are main effects. Names combined with a colon are (second-order) interaction effects.

	abundance	variation	vacancy
scout-prob	1.733	10.984	0.107
survival-prob	80.561	13.272	98.828
scouting-survival	2.487	12.160	0.022
scout-prob:*survival-prob*	5.820	31.400	0.653
scout-prob:*scouting-survival*	0.134	2.862	0.022
survival-prob:*scouting-survival*	4.655	26.914	0.530

Overall, the results fit very well with the results of the other variance decomposition methods even though we used fewer simulation runs. Nevertheless, FANOVA decomposition can only be as good as the Gaussian process is, and the goodness of fit is limited by how well the sample represents the simulation model behaviour with respect to the varied input factors, which depends primarily on the sampling design and the number of samples. Furthermore, the GP is only useful in cases where the response surface is a smooth function of the parameter space [Dancik et al., 2010].

IV.4.5. Costs and Benefits of Approaches to Sensitivity Analysis

In the previous section we presented different sensitivity analysis techniques. Despite the fact that it is impossible to list all existing methods, we presented, from our point of view, the most commonly used and interesting ones in the context of ABMs and provided starting points for adaptations to others. In Figure IV.20, we compare the methods presented here in the same manner as the fitting methods in the intermediate discussion (see Figure IV.11). There is quite some overlap between the different methods, and the ranking in terms of costs and gains is not as clear as with the fitting methods. Furthermore, the purposes of the methods as well as their requirements for the models are different.

As with the fitting methods, for sensitivity analysis it is often useful to start with a simple method and then apply a more informative but computationally more demanding method to explore only the most important input factors. It is always a good idea to start with a graphical method, e.g., scatterplots, to obtain a rough feeling for the relationships of inputs and outputs and their linearity or non-linearity. In a next step, one can, for example, use Morris's elementary effects screening to identify the most important factors and apply the Sobol´ method afterwards to these factors or, as performed by Marino et al. [2008], apply the partial (rank) correlation coefficient method first and use the eFAST method afterwards.

IV.5. Discussion

One might argue that most ABMs are too complex and computationally expensive to run them hundreds or thousands of times with varying parameter values and starting conditions for parameter fitting or sensitivity analysis. However, the way for ABM to become an accepted research method is not to make the models as realistic and complex as possible. An important, if not decisive, design criterion for ABMs, as well as any other simulation models, is that they must run quickly enough to be amenable to comprehensive and systematic analysis. This requires that a single run should be fast enough to allow for both interactive cursory and automatic systematic model analysis. Thus, it is essential to test submodels and make a great effort to find the simplest ones that still serve their purpose.

There can be limits to simplification, for example if a model is no longer able to reproduce multiple patterns simultaneously [Grimm and Railsback, 2012, Railsback and Grimm, 2012]; in such cases, using computing clusters, which currently are often available, can help to still perform the high number of simulations required by some of the methods described here.

A point we have so far not discussed in detail is the variability of simulation outputs caused by stochastic components, which are usually included in ABMs. A single simulation

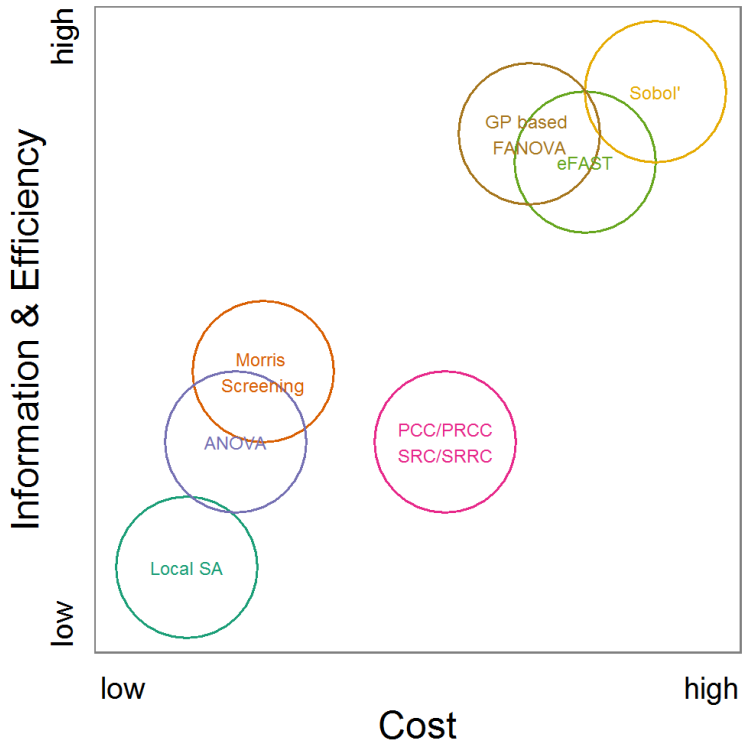


Figure IV.20.: Rough categorisation of sensitivity methods used regarding their cost vs. information and efficiency. Cost includes the computational costs (i.e., number of simulations, which depend itself on the model and the number of parameters) as well as the time consumed by fine-tuning of the methods. Information and efficiency includes aspects about the type of output and the way reaching it. Partly adapted from Campolongo et al. [1999].

run may thus not be representative for the spectrum of possible simulation outputs. Using different seeds for the function generating random numbers can result in completely different dynamics.

This issue is often addressed in textbooks about agent-based modelling [e.g., North and Macal, 2007, Railsback and Grimm, 2012, Squazzoni, 2012, Tesfatsion and Judd, 2006]. It is recommended to run a model with the same configuration repeatedly with different random seeds. Then, the mean of the model outputs is calculated, as we did here; confidence intervals should also be calculated. However, this solution implies two further issues. First, the question of how many iterations are needed must be answered. The classical textbooks do not answer this question. It is often solved by an ad hoc definition that, for example, 10 or 50 replications are sufficient for a specific model [e.g., Kerr et al., 2002, Arifin et al., 2013, Martínez et al., 2011, Squazzoni, 2012, Railsback and Grimm, 2012]. Very likely, just 10 iterations, as used here, will often not be enough. However, an established general accepted strategy for finding an adequate number of repetitions is missing. Nikolic et al. [2013] recommend performing a LHS over the parameter space with 100 replicated runs for each parameter set to identify the most variable metric and parameter value set. Then, this parameter value set producing the most variable metric is used to estimate the number of replications needed to gain a defined confidence level. A similar approach was applied by Kelso and Milne [2011]. In combination with the method suggested by Lorscheid et al. [2012], discussed at the beginning of this article, it is a good starting point to avoid ad hoc assumptions about appropriate replication numbers while a general strategy is missing. This approach becomes less reliable when non-linear processes are involved but is the best approach currently available in terms of the cost-benefit ratio.

The second issue affects the interpretation of the output of ABMs: variation in model output represents variation in reality, i.e., in environmental drivers and the properties and behaviours of a model's entities. Ignoring this variation by only considering averaged model outputs might thus be misleading. Still, averages should capture overall trends, and sensitivity analyses should thus still be informative of the relative importance of processes. The sensitivity of model output to uncertainties in model inputs, though, might be blurred by the stochasticity inherent to the system to be represented. Real uncertainty might thus be higher than the uncertainty detected by sensitivity analyses, which is focused on averaged model outputs. Therefore, it might often be advisable to consider not only the sensitivity of the averaged outputs but also that of the variance of the outputs.

Sensitivity analyses can help understand how a model works, but it should be noted that at least two more general approaches will usually be required for full understanding: simulation experiments and some type of regression analysis. In simulation experiments, one parameter at a time is varied over its full range, and the response of one or more output metrics is studied. These experiments are usually designed like real experiments: only one or a few factors are varied systematically, whereas all other factors are kept constant. Simulation experiments are basically designed to test simple hypotheses, i.e., the model settings are "controlled" to such a degree that we can make predictions of how the model should behave. Important typical experiments include the use of extreme parameter values; turning processes on and off; exploring smaller systems; exploring submodels separately; making drivers constant; making the environment spatially more homogeneous, etc.

With regression analysis, we here refer to statistical methods that explore how much certain processes, represented by their corresponding parameters, affect one or more model outputs. Typical approaches for this include analysis of variance, generalised linear models,

regression trees, path analysis, and many more. For all of these methods, packages exist in R, so they can in principle be used for analysing ABMs in the same way as done here for sensitivity analysis.

Some readers, if not the majority, might have been lost while reading about the more complex methods described here, primarily because they are not familiar with the statistical methods employed for interpreting the results of the sensitivity analyses. Still, we always tried to describe in detail what the output of the analyses means in terms of the sensitivity of single parameters, or parameter interactions, and all this for different model outputs. After such detailed and partly technical considerations, it is always important to step back and ask yourself the following: what have we learned about the relative importance of processes, how processes interact, and how uncertainties in parameter values would affect simulation results?

For our example model, the main lessons learned are as follows: All three tested parameters have a considerable influence on the model results, or, expressed the other way round, the model is sensitive to variations in these parameter values. However, the influence strongly differs regarding both the output measure considered and the parameters. Therefore, it is important to not analyse simulation results on the basis of a single output measure.

In all analyses, *survival-prob* has been identified as the most important parameter. This is not surprising, as this survival probability affects all individuals every month. We varied this probability by only 5 per cent. This means, on the one hand, that the population is very vulnerable regarding its survival probability, and on the other hand, that the more uncertain the value of *survival-prob* is, the more uncertain the models' outputs are, for example, assessments of extinction risks. Is this sensitivity real, or is it an artefact of the selected model structure? In reality, individuals are different, so some individuals should have a higher survival chance than others. It has been shown that this variability serves as a "buffer mechanism", reducing extinction risk [Grimm et al., 2005b].

Still, even if we focus on survival probability as represented in the example model, improving survival to reduce extinction risk might not be sufficient because there are considerable interaction effects, especially regarding the standard deviation of the annual abundance (variation criterion). Although the main-effects of *scout-prob* and *scouting-survival* were unimportant, their influence caused by interactions with *survival-prob* was very important for the variation criterion. Furthermore, we observed both linear effects and non-linear effects of parameter variations.

We based the technical implementation on two commonly used software tools, NetLogo and R. R in particular, with its many user-contributed packages, delivers all commonly used methods for parameter fitting and sensitivity analysis. Nevertheless, we would again like to remind readers of other tools for parameter fitting and/or sensitivity analysis for models implemented in NetLogo. The most prominent ones are BehaviorSearch [Stonedahl and Wilensky, 2013], openMole [Reuillon et al., 2010], MASS/MEME [Iványi et al., 2007], and Simlab [Simlab, 2012]. Other general-purpose statistical/mathematical software systems, such as MatLab [The MathWorks Inc., 2010] and Mathematica [Wolfram Research Inc., 2013], also provide many of the described methods, but these systems are mostly proprietary.

What we wanted to foster with this "cookbook" is to avoid re-inventing the wheel. In young research areas, such as the field of agent-based modelling, it is common for users to implement their own solutions when standards are missing. Often, however, several com-

ponents have already been implemented. We believe that re-inventing the wheel in every ABM project is a major waste of time and money and liable to introduce errors. Rather than trying to implement existing methods from scratch, or trying something new but untested, one should try and use existing tested software tools. This is in contrast to the everyday practice of most agent-based modellers, who are accustomed to programming virtually everything from scratch. With regard to parameterisation and sensitivity analysis, this ad hoc approach would be inefficient and highly uncertain.

We hope that our cookbook lowers the threshold for using fitting and sensitivity analysis methods in ABM studies and delivers a contribution towards rigorous agent-based modelling. Nevertheless, this paper cannot (and is not intended to) replace the intensive study of more detailed literature about these topics and the specific methods. It was our intention to give a rough overview of the most popular methods available to make modellers aware of them. Furthermore, we wanted to show what the methods can bring to modellers and how to apply the methods to an agent-based model in a reusable way. We based the technical implementation on two commonly used software tools, NetLogo and R, to achieve a less steep learning curve. Moreover, both NetLogo and R are supported by large user groups and have established discussion forums on the internet, where beginners can post questions regarding the methods presented here or browse the forums' archives for similar questions.

Still, reading a cookbook does not make you a chef; it only shows you how to start and gives you an idea of what you could achieve if you work hard enough. We hope that this contribution helps more ABM cooks to produce wonderful meals: models that aid in understanding, in a predictive way, and managing real agent-based complex systems.

IV.6. Supplemental Materials

Supplemental Materials are available at:
http://sourceforge.net/projects/calibrationsensitivityanalysis/.

IV.7. Acknowledgement

We thank Klaus G. Troitzsch, Nigel Gilbert, Gary Polhill and Cyril Piou for their helpful comments on earlier drafts. Furthermore, we thank Gilles Pujol for his work on the sensitivity package and Garrett Dancik for his work on the mlegp/mlegpFULL package.

IV.8. References

E.H.L. Aarts and J. Korst. *Simulated Annealing and Boltzmann Machines*. Wiley, Chichester, 1989.

S.M.N. Arifin, G.R. Madey, and F.H. Collins. Examining the Impact of Larval Source Management and Insecticide-Treated Nets Using a Spatial Agent-Based Model of Anopheles Gambiae and a Landscape Generator Tool. *Malaria Journal*, 12(1):290, 2013.

J.C. Ascough II, T.R. Green, L. Ma, and L.R. Ahjua. Key Criteria and Selection of Sensitivity Analysis Methods Applied to Natural Resource Models. In A. Zerger and R.M. Argent, editors, *MODSIM 2005 International Congress on Modelling and Simulation*, pages 2463–2469, Melbourne, December 2005. URL http://www.mssanz.org.au/modsim05/papers/ascough_2.pdf. (last accessed 2014/01/06).

T. Back. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, Oxford, 1996.

E. Bakshy and U. Wilensky. Turtle Histories and Alternate Universes: Exploratory Modeling with NetLogo and Mathematica. In M.J. North, c.M. Macal, and D.L. Sallach, editors, *Proceedings of the Agent 2007 Conference on Complex Interaction and Social Emergence*, pages 147–158. IL: Argonne National Laboratory and Northwestern University, 2007.

R.C. Bansal. Optimization Methods for Electric Power Systems: An Overview. *International Journal of Emerging Electric Power Systems*, 2(1), 2005.

A. Bar Massada and Y. Carmel. Incorporating Output Variance in Local Sensitivity Analysis for Stochastic Models. *Ecological Modelling*, 213(3-4):463–467, 2008.

M. Beaumont. Approximate Bayesian Computation in Evolution and Ecology. *Annual Review of Ecology, Evolution, and Systematics*, 41(1):379–406, 2010.

M. Beaumont, W. Zhang, and D.J. Balding. Approximate Bayesian Computation in Population Genetics. *Genetics*, 162(4):2025–2035, December 2002.

J. Biethahn, A. Lackner, and M. Range. *Optimierung und Simulation*. Oldenbourg, München, 2004.

S.M. Blower and H. Dowlatabadi. Sensitivity and Uncertainty Analysis of Complex Models of Disease Transmission: An HIV Model, as an Example. *International Statistical Review / Revue Internationale de Statistique*, 62(2):229–243, 1994.

J.-F. Bonnans, J.C. Gilbert, C. Lemarechal, and C.A. Sagastizábal. *Numerical Optimization: Theoretical and Practical Aspects*. Springer, Berlin/Heidelberg, 2nd ed. edition, 2006.

G.E.P. Box, W.G. Hunter, and J.S. Hunter. *Statistics for Experimenters: An Introduction to Design, Data Analysis, and Model Building*. Wiley Series in Probability and Mathematical Statistics. Wiley, New York, 1978.

R.H. Byrd, P. Lu, J. Nocedal, and C. Zhu. A Limited Memory Algorithm for Bound Constrained Optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995.

B. Calvez and G. Hutzler. Automatic Tuning of Agent-Based Models Using Genetic Algorithms. In J.S. Sichman and L. Antunes, editors, *Multi-Agent Based Simulation VI*, number 3891 in Lecture Notes in Computer Science, pages 41–57. Springer, Berlin/Heidelberg, 2006.

F. Campolongo and A. Saltelli. Design of Experiments. In A. Saltelli, , K. Chan, and E.M. Scott, editors, *Sensitivity Analysis*, pages 51–63. Wiley, Chichester etc., 2000.

F. Campolongo, S. Tarantola, and A. Saltelli. Tackling Quantitatively Large Dimensionality Problems. *Computer Physics Communications*, 117(1-2):75–85, 1999.

F. Campolongo, J.P.C. Kleijnen, and T. Andres. Screening Methods. In A. Saltelli, , K. Chan, and E.M. Scott, editors, *Sensitivity Analysis*, pages 65–80. Wiley, Chichester etc., 2000a.

F. Campolongo, A. Saltelli, T. Sorensen, and S. Tarantola. Hitchhiker's Guide to Sensitivity Analysis. In A. Saltelli, , K. Chan, and E.M. Scott, editors, *Sensitivity Analysis*, pages 17–47. Wiley, Chichester etc., 2000b.

F. Campolongo, J. Cariboni, and A. Saltelli. An Effective Screening Design for Sensitivity

Analysis of Large Models. *Environmental Modelling & Software*, 22(10):1509–1518, 2007.

J. Cariboni, D. Gatelli, R. Liska, and A. Saltelli. The Role of Sensitivity Analysis in Ecological Modelling. *Ecological Modelling*, 203(1-2):167–182, 2007.

R. Carnell. lhs: Latin Hypercube Samples. R package version 0.10, 2012. URL http://cran.r-project.org/web/packages/lhs/. (last accessed 2014/01/06).

J.O. Cerdeira, P.D. Silva, J. Cadima, and M. Minhoto. subselect: Selecting Variable Subsets. R package version 0.12-3, 2013. URL http://cran.r-project.org/web/packages/subselect. (last accessed 2014/01/06).

K. Chan, S. Tarantola, A. Saltelli, and I.M. Sobol'. Variance-Based Methods. In A. Saltelli, , K. Chan, and E.M. Scott, editors, *Sensitivity Analysis*, pages 167–197. Wiley, Chichester etc., 2000.

R. Confalonieri, G. Bellocchi, S. Bregaglio, M. Albert, and M. Acutis. Comparison of Sensitivity Analysis Techniques: A Case Study with the Rice Model WARM. *Ecological Modelling*, 221(16):1897–1906, 2010.

P.-H. Cournède, Y. Chen, Q. Wu, C. Baey, and B. Bayol. Development and Evaluation of Plant Growth Models: Methodology and Implementation in the PYGMALION platform. *Mathematical Modelling of Natural Phenomena*, 8(4):112–130, 2013.

M.J. Crawley. *Statistics: An Introduction Using R*. John Wiley & Sons, 2005.

K. Csillery, M. Blum, and O. Francois. abc: Tools for Approximate Bayesian Computation (ABC). R package version 1.8, 2012a. URL http://cran.r-project.org/web/packages/abc. (last accessed 2014/01/06).

K. Csillery, O. Francois, and M.G.B. Blum. Approximate Bayesian Computation (ABC) in R: A Vignette, 2012b. URL http://cran.r-project.org/web/packages/abc/vignettes/abcvignette.pdf. (last accessed 2014/01/06).

R.I. Cukier, C.M. Fortuin, K.E. Shuler, A.G. Petschek, and J.H. Schaibly. Study of the Sensitivity of Coupled Reaction Systems to Uncertainties in Rate Coefficients. I Theory. *The Journal of Chemical Physics*, 59:3873–3878, 1973.

P. Dalgaard. *Introductory Statistics with R*. Springer, New York, 2nd edition, 2008.

G.M. Dancik. mlegpFULL: Maximum Likelihood Estimates of Gaussian Processes, 2009. Available from developer on request (see http://cran.r-project.org/web/packages/mlegp/index.html, last accessed 2014/01/06).

G.M. Dancik, D.E. Jones, and K.S. Dorman. Parameter Estimation and Sensitivity Analysis in an Agent-Based Model of Leishmania Major Infection. *Journal of Theoretical Biology*, 262(3):398–412, 2010.

O.L. de Weck. Multiobjective Optimization: History and Promise. In *The Third China-Japan-Korea Joint Symposium on Optimization of Structural and Mechanical Systems*, Kanazawa, Japan, 2004.

R. Duboz, D. Versmisse, M. Travers, E. Ramat, and Y.-J. Shin. Application of an Evolutionary Algorithm to the Inverse Parameter Estimation of an Individual-Based Model. *Ecological Modelling*, 221(5):840–849, 2010.

D. Dupuy and C. Helbert. DiceEval: Construction and Evaluation of Metamodels. R package version 1.2, 2013. URL http://CRAN.R-project.org/package=DiceEval. (last accessed 2014/01/06).

M.J. Fielding. MCMChybridGP: Hybrid Markov Chain Monte Carlo Using Gaussian Processes. R package version 4.3, 2011. URL http://cran.r-project.org/web/packages/MCMChybridGP/. (last accessed 2014/01/06).

R. Fletcher. *Practical Methods of Optimization*. Wiley, Chichester, 2nd ed. edition, 1987. ISBN 0471915475.

C.J. Frost, S.E. Hygnstrom, A.J. Tyre, K.M. Eskridge, D.M. Baasch, J.R. Boner, G.M. Clements, J.M. Gilsdorf, T.C. Kinsell, and K.C. Vercauteren. Probabilistic Movement Model with Emigration Simulates Movements of Deer in Nebraska, 1990-2006. *Ecological Modelling*, 220(19):2481–2490, 2009.

Y. Gan, Q. Duan, W. Gong, C. Tong, Y. Sun, W. Chu, A. Ye, C. Miao, and Z. Di. A Comprehensive Evaluation of Various Sensitivity Analysis Methods: A Case Study With a Hydrological Model. *Environmental Modelling & Software*, 51:269–285, 2014.

C.J. Geyer and L.T. Johnson. mcmc: Markov Chain Monte Carlo. R package version 0.9-2, 2013. URL http://cran.r-project.org/web/packages/mcmc/. (last accessed 2014/01/06).

N. Gilbert. *Agent-Based Models*. Quantitative Applications in the Social Sciences. Sage, Los Angeles, CA, 2007.

V. Ginot, S. Gaba, R. Beaudouin, F. Aries, and H. Monod. Combined Use of Local and ANOVA-Based Global Sensitivity Analyses for the Investigation of a Stochastic Dynamic Model: Application to the Case Study of an Individual-Based Model of a Fish Population. *Ecological Modelling*, 193(3-4):479–491, 2006.

R.B. Gramacy and M.A. Taddy. tgp: Bayesian Treed Gaussian Process Models. R package version 2.4-9, 2013. URL http://cran.r-project.org/web/packages/tgp/. (last accessed 2014/01/06).

V. Grimm. Individual-Based Models. In S.E. Jørgensen, editor, *Ecological Models*, pages 1959–1968. Elsevier, Oxford, 2008.

V. Grimm and S.F. Railsback. *Individual-Based Modeling and Ecology*. Princeton University Press, Princeton, N.J., 2005.

V. Grimm and S.F. Railsback. Pattern-Oriented Modelling: A 'Multi-scope' for Predictive Systems Ecology. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 367 (1586):298–310, 2012.

V. Grimm, E. Revilla, U. Berger, F. Jeltsch, W.M. Mooij, S.F. Railsback, H.-H. Thulke, J. Weiner, T. Wiegand, and D.L. DeAngelis. Pattern-Oriented Modeling of Agent-Based Complex Systems: Lessons from Ecology. *Science*, 310(5750):987–991, 2005a.

V. Grimm, E. Revilla, J. Groeneveld, S. Kramer-Schadt, M. Schwager, J. Tews, M.C. Wichmann, and F. Jeltsch. Importance of Buffer Mechanisms for Population Viability Analysis. *Conservation Biology*, 19(2):578–580, 2005b.

V. Grimm, U. Berger, D.L. DeAngelis, J.G. Polhill, J. Giske, and S.F. Railsback. The ODD Protocol: A Review and First Update. *Ecological Modelling*, 221:2760–2768, 2010.

G.R. Grimmett and D.R. Stirzaker. *Probability and Random Processes*. Clarendon Press, Oxford [u.a.], 2. ed., reprint. edition, 1993.

U. Groemping. Package 'FrF2' Manual. R package version 1.6-8, 2013a. URL http://cran.r-project.org/web/packages/FrF2/. (last accessed 2014/01/06).

U. Groemping. CRAN Task View: Design of Experiments (DoE) & Analysis of Experimental Data. Version 2013-03-20, 2013b. URL http://cran.r-project.org/web/views/ExperimentalDesign.html. (last accessed 2014/01/06).

U. Groemping. DoE.base: Full Factorials, Orthogonal Arrays and Base Utilities for DoE Packages. R package version 0.25-3, 2013c. URL http://cran.r-project.org/web/packages/DoE.base/. (last accessed 2014/01/06).

S. Gubian, Y. Xiang, B. Suomela, and J. Hoeng. Generalized Simulated Annealing for Efficient Global Optimization: the GenSA Package for R. R package version 1.1.3. (last accessed 2014/01/06), 2013. URL http://cran.r-project.org/web/packages/GenSA/.

S. Guichard, D.J. Kriticos, J.M. Kean, and S.P. Worner. Modelling Pheromone Anemotaxis for Biosecurity Surveillance: Moth Movement Patterns Reveal a Downwind Component of Anemotaxis. *Ecological Modelling*, 221(23):2801–2807, 2010.

D.M. Hamby. A Comparison of Sensitivity Analysis Techniques. *Health Physics*, 68(2):195–204, 1995.

L. Han, D. Da Silva, F. Boudon, T. Cokelaer, C. Pradal, R. Faivre, and E. Costes. Investigating the Influence of Geometrical Traits on Light Interception Efficiency of Apple Trees: A Modelling Study with MAppleT. In M. Kang, Y. Dumont, and Y. Guo, editors, *Fourth International Symposium on Plant Growth Modeling, Simulation, Visualization and Applications*, pages 152–159. IEEE Press, 2012.

K. Happe. Agent-Based Modelling and Sensitivity Analysis by Experimental Design and Metamodelling: An Application to Modelling Regional Structural Change. 2005 International Congress, August 23-27, 2005, Copenhagen, Denmark 24464, European Association of Agricultural Economists, 2005. URL http://ideas.repec.org/p/ags/eaae05/24464.html.

F. Hartig, J.M. Calabrese, B. Reineking, T. Wiegand, and A. Huth. Statistical Inference for Stochastic Simulation Models - Theory and Application. *Ecology Letters*, 14(8):816–827, 2011.

F. Hartig, C. Dislich, T. Wiegand, and A. Huth. Technical Note: Approximate Bayesian Parameterization of a Complex Tropical Forest Model. *Biogeosciences Discuss.*, 10(8):13097–13128, 2013.

T. Hastie. *Package 'gam' Manual. R package version 1.09*, 2013. URL http://cran.r-project.org/web/packages/gam/. (last accessed 2014/01/06).

J.C. Helton, J.D. Hiermer, C.J. Sallaberry, and C.B. Storlie. Survey of Sampling-Based Methods for Uncertainty and Sensitivity Analysis. *Reliability Engineering & System Safety*, 91 (10-11):1175–1209, 2006.

J.H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, 6th edition,

2001.

T. Hothorn. CRAN Task View: Machine Learning & Statistical Learning. Version 2013-12-12, 2013. URL http://cran.r-project.org/web/views/MachineLearning.html. (last accessed 2014/01/06).

R.J. Hyndman. CRAN Task View: Time Series Analysis. Version 2013-12-22, 2013. URL http://cran.r-project.org/web/views/TimeSeries.html. (last accessed 2014/01/06).

M.A. Imron, A. Gergs, and U. Berger. Structure and Sensitivity Analysis of Individual-Based Predator-Prey Models. *Reliability Engineering & System Safety*, 107:71–81, 2012.

M.D. Iványi, L. Gulyás, R. Bocsi, G. Szemes, and R. Mészáros. Model Exploration Module. In M.J. North, C.M. Macal, and C.L. Sallach, editors, *Proceedings of the Agent 2007 Conference on Complex Interaction and Social Emergence*, pages 207–215. IL: Argonne National Laboratory and Northwestern University, 2007.

F. Jabot, T. Faure, and N. Dumoulin. EasyABC: Performing Efficient Approximate Bayesian Computation Sampling Schemes. R package version 1.2.2, 2013. URL http://CRAN.R-project.org/package=EasyABC. (last accessed 2014/01/06).

O. Jakoby, V. Grimm, and K. Frank. Pattern-Oriented Parameterization of General Models for Ecological Application: Towards realistic Evaluations of Management Approaches. *Ecological Modelling*, 275:78–88, 2014.

J.K. Kelso and G.J. Milne. Stochastic Indiviual-Based Modelling of Influenza Spread for the Assessment of Public Health Interventions. In *19th International Congress on Modelling and Simulation*, Perth, Austrialia, December 2011. URL http://www.mssanz.org.au/modsim2011/B2/kelso.pdf. (last accessed 2014/01/06).

B. Kerr, M.A. Riley, M.W. Feldman, and B.J.M. Bohannan. Local Dispersal Promotes Biodiversity in a Real-Life Game of Rock-Paper-Scissors. *Nature*, 418(6894):171–174, 2002.

A.A. King, E.L. Ionides, C.M. Breto, S. Ellner, M.J. Ferrari, B.E. Kendall, M. Lavine, D.C. Reuman, H. Wearing, and S.N. Wood. pomp: Statistical inference for partially observed Markov processes . R package version 0.43-8, 2013. URL http://cran.r-project.org/web/packages/pomp/index.html. (last accessed 2014/01/06).

J.P.C. Kleijnen. Sensitivity Analysis and Related Analysis: A Survey of Statistical Techniques, 1995. URL http://econpapers.repec.org/paper/dgrkubrem/1995706.htm. (last accessed 2014/01/06).

J.P.C. Kleijnen. *Design and Analysis of Simulation Experiments*. Springer, New York, NY, 2008.

R.V. Lenth. Response-Surface Methods in R, Using rsm. *Journal of Statistical Software*, 32 (7):1–17, 2009. URL http://www.jstatsoft.org/v32/i07/. (last accessed 2014/01/06).

I. Lorscheid, B.-O. Heine, and M. Meyer. Opening the 'Black Box' of Simulations: Increased Transparency and Effective Communication Through the Systematic Design of Experiments. *Computational & Mathematical Organization Theory*, 18(1):22–62, 2012.

D. Lunn, D. Spiegelhalter, A. Thomas, and N. Best. The BUGS Project: Evolution, Critique and Future Directions. *Statistics in Medicine*, 28(25):3049–3067, 2009.

S. Marino, I.B. Hogue, C.J. Ray, and D.E. Kirschner. A Methodology for Performing Global Uncertainty and Sensitivity Analysis in Systems Biology. *Journal of Theoretical Biology*, 254(1):178–196, 2008.

B.T. Martin, T. Jager, R.M. Nisbet, T.G. Preuss, and V. Grimm. Predicting Population Dynamics From the Properties of Individuals: A Cross-Level Test of Dynamic Energy Budget Theory. *American Naturalist*, 181(4):506–519, 2013.

I. Martínez, T. Wiegand, J.J. Camarero, E. Batllori, and E E. Gutiérrez. Disentangling the Formation of Contrasting Tree-Line Physiognomies Combining Model Selection and Bayesian Parameterization for Simulation Models. *American Naturalist*, 177(5):136–152, 2011.

F. May, I. Giladi, M. Ristow, Y. Ziv, and F. Jeltsch. Metacommunity, Mainland-island System or Island Communities? Assessing the Regional Dynamics of Plant Communities in a Fragmented Landscape. *Ecography*, 36(7):842–853, 2013.

M.D. McKay, R.J. Beckman, and W.J. Conover. A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code. *Technometrics*, 21(2):239–245, 1979.

K.M. Meyer, M. Vos, W.M. Mooij, W.H. Gera Hol, A.J. Termorshuizen, L.E.M. Vet, and W.H. Van Der Putten. Quantifying the Impact of Above- and Belowground Higher Trophic Levels on Plant and Herbivore Performance by Modeling. *Oikos*, 118(7):981–990, 2009.

Z. Michalewicz and D.B. Fogel. *How to Solve It*. Springer, Berlin, 2nd rev. and extended ed. edition, 2004.

K. Miettinen. *Nonlinear Multiobjective Optimization*. Springer, 1999.

M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, 1998.

J.A.E. Molina, C.E. Clapp, D.R. Linden, R.R. Allmaras, M.F. Layese, R.H. Dowdy, and H.H. Cheng. Modeling the Incorporation of Corn (Zea Mays L.) Carbon from Roots and Rhizodeposition into Soil Organic Matter. *Soil Biology and Biochemistry*, 33(1):83–92, 2001.

M.D. Morris. Factorial Sampling Plans for Preliminary Computational Experiments. *Technometrics*, 33:161–174, 1991.

J.C. Nash. Rcgmin: Conjugate Gradient Minimization of Nonlinear Functions with Box Constraints. R package version 2013-02.20, 2013. URL http://cran.r-project.org/web/packages/Rcgmin/. (last accessed 2014/01/06).

J.C. Nash and R. Varadhan. Unifying Optimization Algorithms to Aid Software System Users: optimx for R. *Journal of Statistical Software*, 43(9):1–14, 2011. URL http://www.jstatsoft.org/v43/i09/. (last accessed 2014/01/06).

I. Nikolic, K.H. Van Dam, and J. Kasmire. Practice. In K.H. Van Dam, I. Nikolic, and Z. Lukszo, editors, *Agent-Based Modelling of Socio-Technical Systems*, number 9 in Agent-Based Social Systems, pages 73–140. Springer, Dordrecht; New York, 2013.

M.J. North and C.M. Macal. *Managing Business Complexity: Discovering Strategic Solutions with Agent-Based Modeling and Simulation*. Oxford University Press, Oxford etc., 2007.

O.E. Olmedo. kriging: Ordinary Kriging. R package version 1.0.1, 2011. URL http://CRAN.R-project.org/package=kriging. (last accessed 2014/01/06).

T. Ostromsky, I. Dimov, R. Georgieva, and Z. Zlatev. Sensitivity Analysis of a Large-Scale Air Pollution Model: Numerical Aspects and a Highly Parallel Implementation. In I. Lirkov, S. Margenov, and J. Wasniewski, editors, *Large-Scale Scientific Computing: 7th International Conference, LSSC 2009*, pages 197–205. Springer, Berlin, Heidelberg, 2010.

Z. Pan and L. Kang. An Adaptive Evolutionary Algorithm for Numerical Optimization. In X. Yao, J.-H. Kim, and T. Furuhashi, editors, *Simulated Evolution and Learning*, volume 1285, pages 27–34. Springer, Berlin/Heidelberg, 1997.

J.H. Park. CRAN Task View: Bayesian Inference. Version 2013-12-12, 2013. URL http://cran.r-project.org/web/views/Bayesian.html. (last accessed 2014/01/06).

C. Piou, U. Berger, and V. Grimm. Proposing an Information Criterion for Individual-Based Models Developed in a Pattern-Oriented Modelling Framework. *Ecological Modelling*, 220 (17):1957–1967, 2009.

M. Plummer, N. Best, K. Cowles, and K. Vines. CODA: Convergence Diagnosis and Output Analysis for MCMC. *R News*, 6(1):1–1, 2006. URL http://CRAN.R-project.org/doc/Rnews/Rnews_2006-1.pdf. (last accessed 2014/01/06).

G. Pujol, B. Iooss, and A. Janon. sensitivity: Sensitivity Analysis. R package version 1.7, 2013. URL http://cran.r-project.org/web/packages/sensitivity/index.html. (last accessed 2014/01/06).

R Core Team. Book Related To R, 2013. URL http://www.r-project.org/doc/bib/R-books.html. (last accessed 2014/01/06).

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2014. URL http://www.R-project.org. (last accessed 2014/01/06).

V. Radchuk, K. Johst, J. Groeneveld, V. Grimm, and N. Schtickzelle. Behind the Scenes of Population Viability Modeling: Predicting Butterfly Metapopulation Dynamics under Climate Change. *Ecological Modelling*, 259:62–73, 2013.

S.F. Railsback and V. Grimm. *Agent-Based and Individual-Based Modeling: A Practical Introduction*. Princeton University Press, 2012.

S.F. Railsback, P.C. Cunningham, and R.H. Lamberson. A Strategy for Parameter Sensitivity and Uncertainty Analysis of Individual-Based Models, 2006. URL http://www.humboldt.edu/ecomodel/documents/SensAnalysis_DRAFT.pdf. (last accessed 2014/01/06).

C.E. Rasmussen and C.K.I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

J.K. Ravalico, H.R. Maier, G.C. Dandy, J.P. Norton, and B.F.W. Croke. A Comparison of Sensitivity Analysis Techniques for Complex Models for Environment Management. In *Proceedings*, pages 2533–2539, Melbourne, 2005. URL http://www.mssanz.org.au/modsim05/papers/ravalico.pdf. (last accessed 2014/01/06).

R. Reuillon, F. Chuffart, M. Leclaire, T. Faure, N. Dumoulin, and D. Hill. Declarative Task Delegation in OpenMOLE. In W.W. Smari and J.P. McIntire, editors, *Proceedings of the 2010 International Conference on High Performance Computing and Simulation (HPCS)*, pages 55–62, Caen, France, 2010.

D. Reusser. fast: Implementation of the Fourier Amplitute Sensitivity Test (FAST). R package version 0.63, 2013. URL http://cran.r-project.org/web/packages/fast/index.html. (last accessed 2014/01/06).

I. Salle and M. Yildizoglu. Efficient Sampling and Metamodeling for Computational Economic Models. Technical Report 2012-18, Groupe de Recherche en Economie Théorique et Appliquée, 2012. URL http://ideas.repec.org/p/grt/wpegrt/2012-18.html. (last accessed 2014/01/06).

A. Saltelli. What Is Sensitivity Analysis? In A. Saltelli, K. Chan, and E.M. Scott, editors, *Sensitivity Analysis*, pages 3–13. Wiley, Chichester etc., 2000.

A. Saltelli, S. Anders, and K. Chan. A Quantitative Model-Independent Method for Global Sensitivity Analysis of Model Output. *Technometrics*, 41(1):39–56, 1999.

A. Saltelli, K. Chan, and E.M. Scott. *Sensitivity Analysis*. Wiley, 2000.

A. Saltelli, S. Tarantola, F. Campolongo, and M. Ratto. *Sensitivity Analysis in Practice: A Guide to Assessing Scientific Models*. Wiley, 2004.

A. Saltelli, M. Ratto, T. Andres, F. Campolongo, J. Cariboni, D. Gatelli, M. Saisana, and S. Tarantola. *Global Sensitivity Analysis: The Primer*. Wiley-Interscience, 2008.

M. Schlather, A. Malinowski, M. Oesting, D. Boecker, K. Strokorb, S. Engelke, J. Martini, P. Menck, S. Gross, K. Burmeister, J. Manitz, R. Singleton, B. Pfaff, and R Core Team. RandomFields: Simulation and Analysis of Random Fields. R package version 3.0.5, 2013. URL http://CRAN.R-project.org/package=RandomFields. (last accessed 2014/01/06).

A. Schmolke, P. Thorbek, D.L. DeAngelis, and V. Grimm. Ecological Modelling Supporting Environmental Decision Making: A Strategy for the Future. *Trends in Ecology and Evolution*, 25:479–486, 2010.

K. Siebertz, D. van Bebber, and T. Hochkirchen. *Statistische Versuchsplanung: Design of Experiments (DoE)*. Springer Berlin etc., 2010.

Simlab. Simlab: Simulation Environment for Uncertainty and Sensitivity Analysis, 2012. URL http://simlab.jrc.ec.europa.eu/. (last accessed 2014/01/06).

E.L. Snelson. *Flexible and Efficient Gaussian Process Models for Machine Learning*. PhD thesis, University College London, 2007. URL http://www.gatsby.ucl.ac.uk/~snelson/thesis.pdf. (last accessed 2014/01/06).

I.M. Sobol'. On Sensitivity Estimation for Nonlinear Mathematical Models. *Matem. Mod.*, 2 (1):112–118, 1990.

K. Soetaert and P.M.J. Herman. *A Practical Guide to Ecological Modelling: Using R as a Simulation Platform*. Springer, Dordrecht, 2009.

K. Soetaert and T. Petzoldt. Inverse Modelling, Sensitivity and Monte Carlo Analysis in R Using Package FME. *Journal of Statistical Software*, 33(3):1–28, 2010. URL http://www.jstatsoft.org/v33/i03/. (last accessed 2014/01/06).

A. Sottoriva and S. Tavaré. Integrating Approximate Bayesian Computation with Complex Agent-Based Models for Cancer Research. In G. Saporta and Y. Lechevallier, editors, *COMPSTAT 2010 - Proceedings in Computational Statistics*, pages 57–66. Springer,

Berlin/Heidelberg, 2010.

D.J. Spiegelhalter, N.G. Best, B.P. Carlin, and A. Van Der Linde. Bayesian Measures of Model Complexity and Fit. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 64(4):583–639, 2002.

F. Squazzoni. *Agent-Based Computational Sociology*. John Wiley & Sons, 2012.

F. Stonedahl and U. Wilensky. Evolutionary Robustness Checking in the Artificial Anasazi Model. In *Proceedings of the 2010 AAAI Fall Symposium on Complex Adaptive Systems*, Arlington, VA, 2010. URL http://www.aaai.org/ocs/index.php/FSS/FSS10/paper/download/2181/2622. (last accessed 2014/01/06).

F. Stonedahl and U. Wilensky. BehaviorSearch, 2013. URL http://behaviorsearch.org. (last accessed 2014/01/06).

W. Sun and Y.-X. Yuan. *Optimization Theory and Methods: Nonlinear Programming*. Springer, New York, 2006.

L. Tesfatsion and K.L. Judd. *Handbook of Computational Economics, Volume 2: Agent-Based Computational Economics*. North Holland, 2006.

The MathWorks Inc. MATLAB, 2010.

S. Theussl. CRAN Task View: Optimization and Mathematical Programming. Version 2013-02-14, 2013. URL http://cran.r-project.org/web/views/Optimization.html. (last accessed 2014/01/06).

J.C. Thiele, W. Kurth, and V. Grimm. Agent- and Individual-Based Modeling with NetLogo: Introduction and New NetLogo Extensions. In K. Römisch, A. Nothdurft, and U. Wunn, editors, *22. Tagung der Sektion Forstliche Biometrie und Informatik des Deutschen Verbandes Forstlicher Forschungsanstalten und der Arbeitsgemeinschaft Ökologie und Umwelt der Internationalen Biometrischen Gesellschaft - Deutsche Region, 20-21th September 2010 in Göttingen (Germany)*, Die Grüne Reihe, pages 68–101, 2011.

J.C. Thiele, W. Kurth, and V. Grimm. RNetLogo: An R Package for Running and Exploring Individual-based Models Implemented in NetLogo. *Methods in Ecology and Evolution*, 3: 480–483, 2012.

A. Thomas, B. O'Hara, U. Ligges, and S. Sturtz. Making BUGS Open. *R News*, 1(6):12–17, 2006.

H. Trautmann, D. Steuer, and O. Mersmann. mco: Multi Criteria Optimization Algorithms and Related Functions. R package version 1.0.12, 2013. URL http://CRAN.R-project.org/package=mco. (last accessed 2014/01/06).

M. Van Oijen. Bayesian Calibration (BC) and Bayesian Model Comparison (BMC) of Process-Based Models: Theory, Implementation and Guidelines. Report, Natural Environment Research Council - Centre for Ecology & Hydrology, 2008. URL http://nora.nerc.ac.uk/6087/1/BC%26BMC_Guidance_2008-12-18_Final.pdf. (last accessed 2014/01/06).

J. VanDerWal and S. Januchowski. ConsPlan: Conservation Planning Tools. R package version 0.1, 2010. URL http://www.rforge.net/ConsPlan/index.html. (last accessed 2014/01/06).

R. Varadhan and P. Gilbert. BB: An R Package for Solving a Large System of Nonlinear Equations and for Optimizing a High-Dimensional Nonlinear Objective Function. *Journal of Statistical Software*, 32(4):1–26, 2009. URL http://www.jstatsoft.org/v32/i04/. (last accessed 2014/01/06).

W.N. Venables and B.D. Ripley. *Modern Applied Statistics with S*. Springer, New York, 4th edition, 2002.

F. Vinatier, P. Tixier, C. Le Page, P.-F. Duyck, and F. Lescourret. COSMOS, a Spatially Explicit Model to Simulate the Epidemiology of Cosmopolites Sordidus in Banana Fields. *Ecological Modelling*, 220(18):2244–2254, 2009.

F. Vinatier, M. Gosme, and M. Valantin-Morison. Explaining Host-Parasitoid Interactions at the Landscape Scale: A New Approach for Calibration and Sensitivity Analysis of Complex Spatio-Temporal Models. *Landscape Ecology*, 28(2):217–231, 2013.

T. Wiegand, J. Naves, T. Stephan, and A. Fernandez. Assessing the Risk of Extinction for the Brown Bear (Ursus arctos) in the Cordillera Cantabrica, Spain. *Ecological Monographs*, 68 (4):539–570, 1998.

N. Wiener. *Extrapolation, Interpolation, and Smoothing of Stationary Time Series: With Engineering Applications*. Technology Press Books. Wiley, New York, NY etc., 2nd edition, 1950.

U. Wilensky. NetLogo. Center for Connected Learning and Computer-Based Modeling, 1999. URL http://ccl.northwestern.edu/netlogo. (last accessed 2014/01/06).

U. Wilensky and W. Rand. *An Introduction to Agent-Based Modeling: Modeling Natural, Social and Engineered Complex Systems with NetLogo*. MIT Press, 2014. Forthcomming.

U. Wilensky and B. Shargel. BehaviorSpace, 2002. URL http://ccl.northwestern.edu/netlogo/behaviorspace.html. (last accessed 2014/01/06).

E. Willighagen. genalg: R Based Genetic Algorithm. R package version 0.1.1, 2012. URL http://cran.r-project.org/web/packages/genalg/index.html. (last accessed 2014/01/06).

Wolfram Research Inc. *Mathematica, Version 9.0*, 2013. URL http://www.wolfram.com/mathematica/. (last accessed 2014/01/06).

A.F. Zuur, E.N. Ieno, and E. Meesters. *A Beginner's Guide to R*. Use R. Springer, 2009.

CHAPTER V

Modellers in Ecology: Replicate!

This manuscript is submitted for publication as: JC Thiele and V Grimm [minor revisions]. Modellers in Ecology: Replicate! Oikos.

Authorship

- Volker Grimm re-wrote the manuscript based on a former version of Jan C. Thiele.

V.1. Highlights

- Computational modellers should more often start from the replication of an existing model than from scratch.
- Replication facilitates robustness analysis, which explores where model mechanisms explaining given observations break down.
- A culture of replication would lead to increased credibility, coherence, and efficiency of computational modelling and thereby facilitate theory development.

V.2. Abstract

There are two major limitations to the potential of computational models in ecology for producing general insights: their design is path-dependent, reflecting different underlying questions, assumptions, and data, and there is too little robustness analysis exploring where the model mechanisms explaining certain observations break down. We here argue that both limitations could be overcome if modellers in ecology would more often replicate and modify existing models. Replication comprises the re-implementation of an existing model and the replication of its results. The benefits of replication include less effort being spent to enter the iterative stage of model development and having more time for systematic robustness analysis. A culture of replication would lead to increased credibility, coherence, and efficiency of computational modelling and thereby facilitate theory development.

V.3. Introduction

Computational models have become indispensable in ecology and any other discipline dealing with agent-based complex systems [Grimm et al., 2005]. They allow essential features of the real world that have to be ignored in mathematical models to be taken into account [Evans et al., 2013]. General insights, however, are often hard to distil from computational models. One reason is their higher complexity. To limit the degrees of freedom in model structure and parameters, models must often be tied to specific real systems. We here argue, though, that an equally important reason is that computational modellers in ecology almost exclusively develop models from scratch.

The prevalence of *de novo* computational models is largely a consequence of the path-dependence of modelling. Models do not represent systems *per se*, but only with regard to specific questions or problems. Thus, modellers start from different questions and make different model assumptions, depending on their conceptual understanding of the system, on the experts and data involved, on their preferred modelling approach, and even on such mundane things as the available software and hardware. Each assumption constrains the potential of a model to reproduce observations in a different way, leading to different subsequent model assumptions, etc. The final models represent chains, or paths, of submodels and parameter values that were needed to obtain sufficiently realistic output. Even for the same system, they can be completely different. This leads to models of, e.g., savannahs, forest fire ecosystems, or cycling voles that are so different that they are harder to compare than apples and oranges, which is a major obstacle to distilling general insights.

There is nothing wrong in principle with the path-dependent assembly of models. It cannot be avoided, and it is usually along the path that modellers develop understanding.

Nevertheless, once a model reproduces observations, to distinguish signal from noise in the model, the path must be conceptually reversed [Grimm and Railsback, 2005] to systematically explore alternative submodels and simplified environments in a robustness analysis. There seems to be a psychological barrier, though, to reversing the path [Grimm, 1999]: constructing a realistic model can be a long and thorny enterprise. It can sometimes take years to make a model produce realistic output. It is understandable that the modeller then uses the model as given for its intended purpose, which again can take a long time. Sooner or later, as Francis Crick noticed [Crick, 1988], modellers fall in love with their models, which makes it unlikely that they themselves will ever perform a robustness analysis.

If we had a culture of model replication where modellers do not always start from scratch but also from existing models that they replicated, robustness analysis and distilling general insights would become a community task. If you replicated a model, there is no psychological barrier against radical scrutinising, modification, and simplification. Moreover, by replication, we would multiply the manpower available for learning from interesting models.

So far, the replication of computational models and its benefits have mostly been discussed in the social sciences [Axtell et al., 1996, Edmonds and Hales, 2003, Rouchier et al., 2008]. In ecology, where models tend to be more complex and more closely linked to real systems, replication is virtually absent. We want to change this by arguing here that ecological modelling indeed needs a culture of replication. We summarise arguments for replication, list tools, and emphasise the important role of open source software. Most importantly, we will argue that replication is not only good for scientific practice, as it increases the credibility and robustness of models, but provides direct benefits to those who replicate models. But, first let us discuss in more detail what replication involves.

V.4. What Does Model Replication Involve?

Model replication can be defined as the re-implementation of a model that has been developed and implemented by others. By ïmplementationẅe refer to a computer program. Usually, a replication is based on the verbal or conceptual description of the original model. If the code of the original implementation is also available, details of the original model, which can be ambiguous in verbal descriptions, can be adopted from the original code. However, usually and preferably, the re-implementation uses a different programming language so that the details of model algorithms are also implemented independently. Replication should also include differences in one or more of the hardware used, the software tools such as program libraries, and the algorithms employed [Wilensky and Rand, 2007].

Replication is tested by comparing the output of the re-implementation to the output of the original implementation. For comparison, three different replication standards of decreasing selectivity were proposed [Axtell et al., 1996]: numerical identity, distributional equivalence, and relational equivalence. Numerical identity is hard to achieve, sometimes even with the same computer and implementation, as stochasticity and the details of floating-point algorithms may lead to differences. Distributional equivalence means that the output of many runs of the two different models cannot be distinguished statistically. Relational equivalence means that although there are distributional differences, the relations between the input and output of the original and replicated models are qualitatively the same.

Successful replication according to one or more of these standards increases the credibility

of both the original and replicated model, which can be important by itself as the credibility of computational models is often doubted. Replication is considered the hallmark of the scientific method, although in practice is it quite rare [Collins, 1985, Giles, 2006, Goodstein, 2010, Jasny et al., 2011]. The reason for this, both in real and simulation experiments, is that merely increasing the credibility of a general approach (for example, agent-based modelling,) or of a specific model or experiment usually does not provide sufficient incentive for scientists to invest time and energy into replication.

V.5. Robustness Analysis

The more interesting aspect of replication is, as we call it here, "robustness analysis", which involves checking whether the results of the original model are "robust to substantial extensions over time, explanatory variables, and/or alternative estimation procedures" [Burman et al., 2010]. These criteria were formulated for statistical models in econometrics but can be extended to simulation models as well: how robust are the output and the conclusions from the original model if we change its temporal and spatial resolution and extent, modify its representation of model entities and processes, analyse the model output with other methods, or explore additional model output? These modifications should particularly include systematic simplification, including more homogeneous settings for environmental variables, such as habitat features or environmental drivers.

Replication is thus a means for exploring where the explanations and predictions provided by a given model break down. The resulting robustness analysis is a critical tool for overcoming one of the main limitations of computational models: the path-dependence of their development and, hence, their analysis. Nevertheless, replication of computational models is still rare, because not only scientists but also journal editors and reviewers seem not to be sufficiently aware of the benefits of replicating computational models.

V.6. Benefits of Replication

Replication can provide considerable returns to the replicator. When modelling from scratch, even if modellers scan existing models and copy their designs to some degree, it can take a long time for a model to first be conceptually formulated, then be implemented, tested, and analysed. A direct benefit of replication is that the modeller starts from existing hypotheses and submodels and therefore enters the "modelling cycle" [Grimm and Railsback, 2005] of iterative model formulation, implementation, simulation, and analysis much more quickly. This leaves more time for the critical but time-consuming task of in-depth model analysis, which is often performed only superficially because too much time had been spent developing the first conceptual model.

Some modellers argue, though, that the gradual development of a conceptual model from scratch is important, as the modeller gradually gets familiar with the system and the question to be addressed and develops conceptual understanding. This is a valid argument, but replication is different from just running an existing model, which indeed can be quite limited for gaining new insights. Replication stands between modelling from scratch and the mere use of existing models, as it requires that we understand the conceptual model, and its rationale, in all detail before we can re-implement [Grimm et al., 2010]. Thus, gradually developing understanding is also an integral part of replication, but it is faster than modelling from scratch, as it starts from existing building blocks.

Starting a new modelling project with the replication of one or more existing models also facilitates publication, for several reasons: in introductions, it can be easier to use the hypotheses and findings of existing models as the point of departure, as they provide a seamless link to existing work and knowledge; by showing that we start with an implementation that replicates the output of the original model, we add considerable credibility to our own model and its implementation; by explicitly referring to robustness analysis, we indicate that we will not present just another case study, but contribute to the development of theory, which is also important for broadening the scope of applied models.

Last but not least, even if we decide, after replication, to design an entirely new model, we very likely can use some building blocks of the replicated model and its implementation. Many animal models, for example, include home range dynamics [Wang and Grimm, 2007, Börger et al., 2008]. As these should follow some general principles, it is likely that we can transfer and adapt home range submodels to our own model. This example also shows that we do not always have to replicate an entire model. If the original submodel was tested separately, as it should be [Grimm et al., 2014], we can compare the output of the original and the replicated submodels.

All these direct benefits to replicators also imply indirect benefits via improving scientific conduct. Computational models are put into a more rigorous framework, where artefacts from erroneous code or model assumptions are detected early on; re-implemented submodels, which are used in different contexts, can be assembled into a library of tested and useful submodels of certain behaviours, which in turn facilitates developing new models and reviewing them; and robustness analysis contributes to theory development by overcoming the path-dependence of modelling.

V.7. Examples

In contrast to ecology, the replication of computational models has been discussed for almost two decades in the social sciences [Axtell et al., 1996, Edmonds and Hales, 2003, Rouchier et al., 2008]. A reason might be that modelling is a relatively new approach in social sciences, so it was important to convince modelling sceptics that simulation is a rigorous approach. Ecology has a long history of mathematical modelling. Moreover, the rigorousness, testing, and predictions of models were rarely discussed, because most mathematical models were designed for demonstrating concepts rather than making testable predictions [Grimm and Railsback, 2005, Evans et al., 2013].

Often, the replication of social models was discussed in the broader context of "model alignment", wherein the question is under what conditions two or more models with different structures and histories would make the same, or similar, predictions when addressing the same scenario and issue [Axtell et al., 1996, Edmonds and Hales, 2003, Rouchier et al., 2008].

A good example of replication and its benefits started with the agent-based model of Macy and Sato [2002] which explored how trust can be build among distant people. Will and Hegselmann [2008] failed to replicate the results of this model. A specific rule was then identified in the source code of the original model that was missing in the model description but was essential for replicating the published results. Will [2009] considered the effect of this rule an artefact and developed a revised model. Macy and Sato [22], in turn, claimed that in fact the revised model supported their original hypothesis, if parameter values were

varied more systematically. Further examples of replication can be found in Edmonds and Hales [2003], Merlone et al. [2008], Miodownik et al. [2010], and Radax and Rengs [2010].

In ecology, Mooij and DeAngelis [1999] were not able to reproduce the results regarding the sensitivity of dispersal success to dispersal mortality. The original authors [Ruckelshaus et al., 1997] varied mortality between 2, 8, 16, 24 and 32 per cent and found extremely high sensitivity already for 2 per cent, which would render any prediction of spatially explicit simulation models too uncertain. This work was therefore frequently cited as an argument against the usefulness of spatially explicit, individual-based population models.

Only after obtaining the original code from the original authors [Ruckelshaus et al., 1997] did Mooij and DeAngelis [1999] realise that not mortality, but survival per movement step was varied between 2 and 32 per cent, leading to unrealistically high mortalities which had caused the extreme sensitivities observed. With the correct implementation of mortality, sensitivity to uncertainty did not completely disappear, but was much lower.

A further example is Cortés-Avizanda et al. [2014], who re-implemented a model of vultures searching for carcasses which they considered unrealistic [Jackson et al., 2008], and then confronted this, and alternative models, with patterns observed in field experiments. The original model, which was designed to explore ideas rather than making predictions, produced numbers that were far too high of vultures arriving at a carcass shortly after it had been found by the first vulture. Box 1 provides another example where the replication of a simple model, which was performed in less than a day, allowed the parameter space analysed with the original model to be extended, which immediately led to questions challenging the original conclusions.

Box 1: Example replication of a C-S-R coexistence model

Kerr et al. [2002] presented a theoretical grid-based model of three competing species (C-S-R) inspired by *Escherichia coli* bacterial systems. The model is used to analyse how coexistence depends on the localness of interactions and dispersal under varying parameter values. It is based on simple stochastic rules in the style of a rock-paper-scissors (R-P-S) game. The model world consists of cells arranged in a 250 x 250 grid, each initialised randomly by assigning it to one species. The model is run for 10,000 time steps. In each step, an asynchronous update of each cell occurs. The probability of state transition of a cell depends on its own state and the states of its neighbours.

For the state transition rule, Kerr et al. [2002] applied a local Moore neighbourhood rule as well as a global neighbourhood rule, and showed that coexistence only emerges under the local neighbourhood rule. Furthermore, they demonstrated that the emergence of coexistence is robust to variations in the parameters Δ_R (death probability of R) and τ (toxicity of C).

We re-implemented the model of citetKerrEtAl2002 in NetLogo [Wilensky, 1999] and produced a model that was relationally equivalent to the original. Then, we reviewed the rule given by Kerr et al. [2002], $\Delta_{S,0} < \Delta_R < \Delta_C < \frac{\Delta_{S,0}+\tau}{1+\tau}$, which ensures that S displaces R, R displaces C, and C displaces S. From this rule, the permitted parameter value range of Δ_R and τ under fixed values of $\Delta_{S,0} = \frac{1}{4}$ and $\Delta_C = \frac{1}{3}$ can be derived:

- min. $\Delta_R : \frac{1}{4}$, max. $\Delta_R : \frac{1}{3}$
- min. $\tau : \frac{1}{8}$, max. τ : Inf. or $\frac{3}{4}$ (because of $\Delta_{S,0} + \tau * f_C$; with f_C being the proportion of neighbouring cells occupied by species C)

Using this full range of permitted values for a robustness analysis reveals that coexistence is not as robust against the changes in the interaction and dispersal regime as was suggested by Kerr et al. [2002]. The proportion of runs with coexistence after 10,000 steps decreases from 61 % with the parameter range used by Kerr et al. [2002] to only 30 % with the full parameter value range used here (Figure V.1). This does not imply that the coexistence mechanism reported by Kerr et al. [2002] does not exist, but that it is less general than they claim.

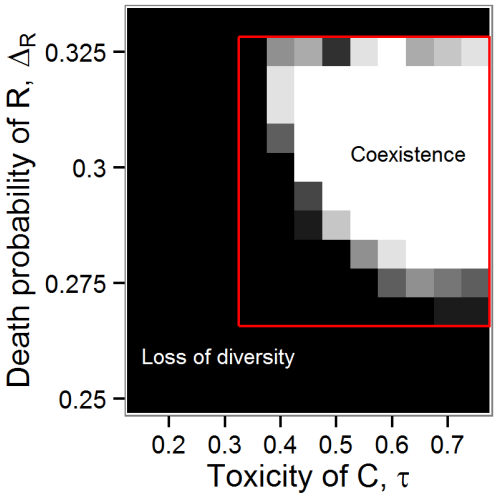


Figure V.1.:
Robustness of a coexistence mechanism based on rock-paper-scissor interactions between three species of bacteria to variations in two model parameters, Δ_R (mortality of "rock" species) and τ (toxicity of "scissors" species). The red box delineates the originally explored parameter ranges Kerr et al. [2002]. For each parameter set, 10 repeated simulations with 10,000 time steps were run. Grayscale indicates the number of runs with coexistence; i.e., the lighter, the higher the number of runs with coexistence (white: 10 out of 10, i.e., always coexistence; black: 0 out of 10, i.e., always extinction). Results reported from the red box indicate that a quite general coexistence mechanism was found, suggesting that this might also be a robust explanation of coexistence observed in reality, whereas results from the entire parameter space show that the coexistence mechanism is less robust and, in turn, is likely to be less relevant in reality.

V.8. Tools Supporting Replication

A major obstacle to replication used to be the lack of a common format for model descriptions. Fortunately, this situation is now changing, as use of the ODD (Overview, Design concepts, Details) protocol for describing computational models [Grimm et al., 2006, 2010] has been increasing over the last several years. It is estimated that more than 60 % of the descriptions of new agent-based models in ecology currently follow this protocol. The ODD protocol makes it easier to identify the structure, processes, and the details of a submodel's implementation; it is designed as a verbal description, but it can include equations, pseudo-code, and, for very complex models [e.g., Becher et al., 2014], links to short pieces of code. Replication of a model described via ODD still requires that the replicator fully understand the original model. Even if the original model description does not follow ODD, rewriting it to the ODD format has turned out to be useful for replication [M. Janssen, pers. communication, C. LePage, pers. communication].

A further means for facilitating replication is the increased use of the same software plat-

form for the same type of models, rather than using general-purpose programming languages such as C++ or Java. In agent-based modelling, NetLogo [Wilensky, 1999, Railsback and Grimm, 2012] already dominates in social sciences and is gaining ground in ecology. Thus, when replication based on a model description fails, checking the code is easier if we all use the same programming language. However, being able to program in more than one programming language is still important, as subtle mistakes are often specific to the design of a certain language. Because most of the software testing for computational models will focus on submodels, which are usually relatively simple, simple languages available in spreadsheets or statistical software will be sufficient for producing independent implementations.

Ultimately, we might assemble libraries of tested submodels representing specific behaviours or processes. If these have been independently tested and used in many projects, replication would turn to a mixture of writing one's own code and using existing procedures from software libraries. This workflow is already associated with standard models representing interactions between plants, e.g., vertical competition for light in individual-based models of forests ["gap models", Botkin et al., 1972, Bugmann, 2001], or horizontal competition via overlapping "zones of influence" [Weiner et al., 2001], as well as for energy budgets [Kooijman, 2010, Martin et al., 2013], but it might also work for certain behavioural "primitives" [Ginot et al., 2002], such as habitat selection, foraging, or dispersal.

Standard formats for model description, standard software platforms, and standard submodels are not sufficient, but must be complemented by routinely making code implementing a model available on permanent and non-proprietary websites. This is the only way to ensure successful replication some indefinite time after a model is first published. Currently, code is not routinely made available, and if so, links to websites are often outdated. For example, of the agent-based or individual-based models published in the journals *Ecological Modelling* and *Environmental Modelling & Software* in the years 2009 to 2010, in 12 per cent of the studies the source code or a link/URL to it was given in articles published in *Ecological Modelling*, and 43 per cent for *Environmental Modelling & Software*, but only 21 per cent of those weblinks were still working as of today.

If journals would require that authors add their source code to their submissions as part of the supplementary material, replication and all its benefits would be considerably facilitated. Most journals, though, are proprietary, and subscription to the journal is required to have access to the supplements. One important initiative has already started to change this situation: the COMSES network (www.openabm.org) maintains a library of models, consisting of a model description, the code, and all files and information needed to re-run simulations [Rollins et al., 2014]. Authors can require certification, which is obtained after peer-review checking the documentation, completeness, and readability of the source code. Certified models are made available under a permanent web address, which is similar to the commercial DOI (digital object identifiers) system.

Facilitation requires that we have free access to the software that was used to implement the original model, because when replication fails, we need to compare the model output of the original and replicated models in detail. If the original model was implemented using proprietary software, this may not work because the underlying algorithms are not accessible or because not everyone might have the resources to buy that software. Using non-proprietary software platforms is advisable for many reasons [Greve, 2003], not the least being that it opens a model to replication.

V.9. The Way to Proceed

Having a culture of replication would have many benefits, both for replicators and for their science in general. How can we establish such a culture? Both a bottom-up and a top-down approach are needed. Bottom-up means that if modellers are convinced that replication is a good starting point for their project, then they should be bold and submit their work without any excuses or justification for why they did not start from scratch. If the work leads to new insights, its chances of getting printed and having impact are not different from work based on *de novo* models; actually, the chances might be higher. Moreover, using libraries such as the one maintained by the COMSES network (www.openabm.org) allows us to turn even the replication itself, without any further analysis, into a citeable product, which could be categorised under "peer-reviewed software"; this could be particularly valuable for modelling neophytes.

The top-down approach starts from the type of work a journal is willing to accept. We would encourage journal editors, as Palmer [2000] has already done for ecology in general, to create a new category of "Model Replication" where modellers can publish their replication exercises if they led to new and important insights. In econometrics, journals have already defined such a category. Burman et al. [2010] call for replication papers and envisage three types of replication: positive, where the replication fully worked; negative (type 1), where replication only worked after contacting the original authors; and negative (type 2), where replication worked, but the results were not found to be robust.

Having "replication" as an explicit category would encourage replication, and make "community modelling" an explicit part of modelling and theory development. Of course, none of these proposals deny the role of models developed from scratch. Ideally, modelling in ecology and other fields dealing with agent-based complex systems would encompass the full spectrum of models, from those fully developed from scratch to those fully based on a replicated model.

Acknowledgments

We would like to thank U. Berger, B. Martin and S. Railsback for their comments.

V.10. References

R. Axtell, R. Axelrod, J.M. Epstein, and M.D. Cohen. Aligning Simulation Models: A Case Study and Results. *Computational and Mathematical Organization Theory*, 1:123–141, 1996.

M.A. Becher, V. Grimm, P. Thorbek, J. Horn, P.J. Kennedy, and J.L. Osborne. BEEHAVE: A Systems Model of Honey Bee Colony Dynamics and Foraging to Explore Multifactorial Causes of Colony Failure. *Journal of Applied Ecology*, 51:470–482, 2014.

D.B. Botkin, J.F. Janak, and J.R. Wallis. Some Ecological Consequences of a Computer Model of Forest Growth. *Journal of Ecology*, 60(3):849–872, 1972.

L. Börger, B.D. Dalziel, and J.M. Fryxell. Are There General Mechanisms of Animal Home Range Behaviour? A Review and Prospects for Future Research. *Ecology Letters*, 11:637–650, 2008.

H. Bugmann. A Review of Forest Gap Models. *Climate Change*, 51:259–305, 2001.

L.E. Burman, W.R. Reed, and J. Alm. A Call for Replication Studies. *Public Finance Review*, 38:787–793, 2010.

H.M. Collins. *Changing Order: Replication and Introduction in Scientific Practice*. Sage, 1985.

A. Cortés-Avizanda, R. Jovani, J.A. Donázar, and V. Grimm. Bird Sky Networks: How Do Avian Scavengers Use Social Information to Find Carrion? *Ecology*, 95:1799–1808, 2014.

F. Crick. *What Mad Pursuit: A Personal View of Scientific Discovery*. Basic Books, 1988.

B. Edmonds and D. Hales. Replication, Replication and Replication: Some Hard Lessons From Model Alignment. *Journal of Artificial Societies and Social Simulation*, 6 (4) 11, 2003. URL http://jasss.soc.surrey.ac.uk/6/4/11.html. (last accessed 2014/08/07).

M.R. Evans, V. Grimm, K. Johst, T. Knuuttila, R. de Langhe, C.M. Lessells, M. Merz, M.A. O'Malley, S.H. Orzack, M. Weisberg, D.J. Wilkinson, O. Wolkenhauer, and T.G. Benton. Do Simple Models Lead to Generality in Ecology? *Trends in Ecology and Evolution*, 28: 578–583, 2013.

J. Giles. The Trouble With Replication. *Nature*, 442:344–347, 2006.

V. Ginot, C. Le Page, and S. Souissi. A Multi-Agent Architecture to Enhance End-User Individual-Based Modelling. *Ecological Modelling*, 157:23–41, 2002.

D. Goodstein. *On Fact and Fraud: Cautionary Tales from the Front Lines of Science*. Princeton, 2010.

G.C.F. Greve. Brave GNU World. *Linux Magazine*, 12:89–91, 2003. URL http://www.linux-magazine.com/w3/issue/37/Brave_GNU_World.pdf. (last accessed 2014/08/07).

V. Grimm. Ten Years of Individual-Based Modelling in Ecology: What Have We Learned and What Could We Learn in the Future? *Ecological Modelling*, 115:129–148, 1999.

V. Grimm and S.F. Railsback. *Individual-Based Modeling and Ecology*. Princeton University Press, Princeton, N.J., 2005.

V. Grimm, E. Revilla, U. Berger, F. Jeltsch, W.M. Mooij, S.F. Railsback, H.-H. Thulke, J. Weiner, T. Wiegand, and D.L. DeAngelis. Pattern-Oriented Modeling of Agent-Based Complex Systems: Lessons from Ecology. *Science*, 310(5750):987–991, 2005.

V. Grimm, U. Berger, F. Bastiansen, S. Eliassen, V. Ginot, J. Giske, J. Goss-Custard, T. Grand, S.K. Heinz, G. Huse, A. Huth, J.U. Jepsen C. Jørgensen, W.M. Mooij, B. Müller, G. Pe'er, C. Piou, S.F. Railsback, A.M. Robbins, M.M. Robbins, E. Rossmanith, N. Rüger, E. Strand, S. Soissi, R.A. Stillman, R. Vabø, U. Visser, and D.L. DeAngelis. A Standard Protocol for Describing Individual-Based and Agent-Based Models. *Ecological Modelling*, 198:115–126, 2006.

V. Grimm, U. Berger, D.L. DeAngelis, J.G. Polhill, J. Giske, and S.F. Railsback. The ODD Protocol: A Review and First Update. *Ecological Modelling*, 221:2760–2768, 2010.

V. Grimm, J. Augusiak, A. Focks, B.M. Frank, F. Gabsi, A.S.A. Johnston, C. Liu, B.T. Martin, M. Meli, V. Radchuk, P. Thorbek, and S.F. Railsback. Towards Better Modelling and Decision Support: Documenting Model Development, Testing, and Analysis Using TRACE. *Ecological Modelling*, 280:129–139, 2014.

A.L. Jackson, G.D. Ruxton, and D.C. Houston. The Effect of Socal Facilitation on foraging Success in Vultures: A Modelling Study. *Biology Letters*, 4:311–313, 2008.

B.R. Jasny, G. Chin, L. Chong, and S. Vignieri. Again, and Again, and Again ... *Science*, 334 (6060):1225, 2011.

B. Kerr, M.A. Riley, M.W. Feldman, and B.J.M. Bohannan. Local Dispersal Promotes Biodiversity in a Real-Life Game of Rock-Paper-Scissors. *Nature*, 418(6894):171–174, 2002.

S.A.L.M. Kooijman. *Dynamic Energy Budget Theory for Metabolic Organisation*. Cambridge University Press, 3rd edition, 2010.

M. Macy and Y. Sato. Trust, Cooperation and Market Formation in the US and Japan. *Proceedings of the National Academy of Sciences*, 99:7214–7220, 2002.

B.T. Martin, T. Jager, R.M. Nisbet, T.G. Preuss, and V. Grimm. Predicting Population Dynamics From the Properties of Individuals: A Cross-Level Test of Dynamic Energy Budget Theory. *American Naturalist*, 181(4):506–519, 2013.

U. Merlone, M. Sonnessa, and P. Terna. Horizontal and Vertical Multiple Implementations in a Model of Industrial Districts. *Journal of Artificial Societies and Social Simulation*, 11 (2) 5, 2008. URL http://jasss.soc.surrey.ac.uk/11/2/5.html. (last accessed 2014/08/07).

D. Miodownik, B. Cartrite, and R. Bhavnani. Between Replication and Docking: "Adaptive Agents, Political Institutions, and Civic Traditions" Revisited. *Journal of Artificial Societies and Social Simulation*, 13 (3) 1, 2010. URL http://jasss.soc.surrey.ac.uk/13/3/1.html. (last accessed 2014/08/07).

W.M. Mooij and D.L. DeAngelis. Error Propagation in Spatially Explicit Population Models: A Reassessment. *Conservation Biology*, 13:930–933, 1999.

A.R. Palmer. Quasi-Replication and the Contract of Error: Lessons from Sex Ratios, Heritabilities and Fluctuating Asymmetry. *Annual Review of Ecology and Systematics*, 31:441–480, 2000.

W. Radax and B. Rengs. Prospects and Pitfalls of Statistical Testing: Insights from Replicating the Demographic Prisoner's Dilemma. *Journal of Artificial Societies and Social Simulation*, 13 (4) 1, 2010. URL http://jasss.soc.surrey.ac.uk/13/4/1.html. (last accessed 2014/08/07).

S.F. Railsback and V. Grimm. *Agent-Based and Individual-Based Modeling: A Practical Introduction*. Princeton University Press, 2012.

N.D. Rollins, C.M. Barton, S. Bergin, and M.A. Janssen. A Computational Model Library for Publishing Model Documentation and Code. *Environmental Modelling & Software*, 61: 59–64, 2014.

J. Rouchier, C. Cioffi-Revilla, J.G. Polhill, and K. Takadama. Progress in Model-To-Model Analysis. *Journal of Artificial Societies and Social Simulation*, 11 (2) 8, 2008. URL http://jasss.soc.surrey.ac.uk/11/2/8.html. (last accessed 2014/08/07).

M. Ruckelshaus, C. Hartway, and P. Kareiva. Assessing the Data Requirements of Spatially Explicit Dispersal Models. *Conservation Biology*, 11:1298–1306, 1997.

M. Wang and V. Grimm. Home Range Dynamics and Population Regulation: An Individual-

Based Model of the Common Shrew. *Ecological Modelling*, 205:397–409, 2007.

J. Weiner, P. Stoll, H. Müller-Landau, and A. Jasentuliyana. The Effects of Density, Spatial Pattern, and Competitive Symmetry on Size Variation in Simulated Plant Populations. *American Naturalist*, 158:438–450, 2001.

U. Wilensky. NetLogo. Center for Connected Learning and Computer-Based Modeling, 1999. URL http://ccl.northwestern.edu/netlogo. (last accessed 2014/01/06).

U. Wilensky and W. Rand. Making Models Match: Replicating an Agent-Based Model. *Journal of Artificial Societies and Social Simulation*, 10 (4) 2, 2007. URL http://jasss.soc.surrey.ac.uk/10/4/2.html. (last accessed 2014/08/07).

O. Will. Resolving a Replication That Failed: News on the Macy & Sato Model. *Journal of Artificial Societies and Social Simulation*, 12 (4) 11, 2009. URL http://jasss.soc.surrey.ac.uk/12/4/11.html. (last accessed 2014/08/07).

O. Will and R. Hegselmann. A Replication That Failed - On the Computational Model in "Michael W. Macy and Yoshimichi Sato: Trust, Cooperation and Market Formation in the U.S. and Japan. Proceedings of the National Academy of Sciences, May 2002". *Journal of Artificial Societies and Social Simulation*, 11 (3) 3, 2008. URL http://jasss.soc.surrey.ac.uk/11/3/3.html. (last accessed 2014/08/07).

CHAPTER VI

Discussion and Outlook

The present thesis aims to contribute to the development of a rigorous framework of agent-based modelling by linking, extending and using existing software platforms. As shown, the thesis focuses more on technical solutions by a full-fledged integration of NetLogo and R. However, software tools can only be seen as successful when users accept and use them. Therefore, evaluating the objective can only be done by analysing the impact of the work in the community.

VI.1. Impact

The main output of this thesis are the R-/Rserve-Extension to NetLogo and the RNetLogo package for R. Since making these two software tools available to the public they have been used by many researches from various fields. A short overview of impacts and feedbacks is given in Table VI.1 for the R-/Rserve-Extension and in Table VI.2 for the RNetLogo package.

Furthermore, the paper Thiele et al. [2012b]: *Agent-based modelling: Tools for linking NetLogo and R* is placed on rank 8 of most viewed papers in Journal of Artificial Societies and Social Simulation with 785 views within eight weeks calculated on January 8th, 2014 [JASSS, 2014]. Therefore, it is the most-viewed paper published in 2012 of these eight weeks. Moreover, the paper Thiele et al. [2012a]: *RNetLogo: An R Package for Running and Exploring Individual-Based Models Implemented in NetLogo* is within the top ten per cent of all 1.1 million articles tracked by Altmetric measuring online attraction of scholarly articles on January 8th, 2014 [Altmetric, 2014].

ISI Web of Knowledge (requested on 2014/07/02) lists seven citations of the paper about the R-Extension in Environmental Modelling & Software [Thiele and Grimm, 2010], with two being self-citations and one refering to the R-Extension but using RNetLogo. Google Schoolar lists 17 citations (requested on 2014/07/03) with three self-citations and also the one incorrect citation.

The paper published in Methods in Ecology and Evolution about the RNetLogo package [Thiele et al., 2012a] was cited three times following ISI Web of Knowledge (requested on 2014/07/02), whereas Google Schoolar lists 9 citations (requested on 2014/07/03), one being a self-citation.

For the summary paper in Journal of Artificial Societies and Social Simulation about all

three tools to connect R and NetLogo, Thiele et al. [2012b], Google Schoolar registered three citations (requested on 2014/07/03).

However, it can be assumed that the tools are much more often used as indicated by Table VI.1 and Table VI.2 and citations do not appear because either the usage was not mentioned or the studies are not published until now. Therefore, citations are a weak measure of impact for software tools.

An example of the application of the R-Extension can be found in Ibarra and Janssen [2012]. They used the R-Extension to generate different landscape configurations with R as initial condition for their NetLogo patches. Fernández [2011] used the R-Extension to connect NetLogo with an open source software that models hydraulic and water quality in water distribution piping systems. [Falbo, 2011] used the R-Extension in conjunction with the spatstat package to analyse the spatial point pattern of positions of the individuals.

A use-case of the RNetLogo package can be found in Vinatier et al. [2013]. In this paper, the RNetLogo package was obviously used to run a NetLogo model systematically from R and collect seemlessly simulation results to conduct a sensitvity analysis with the Morris screening method, perform an ANOVA, and to calibrate parameter values. Thus, the RNetLogo package has been applied in the way that was proposed in Chapter IV. Frank and Baret [2013] also used the RNetLogo package to control the model run from RNetLogo and transfer data from NetLogo to R to perform a sensitivity analysis starting with the Morris screening method and analysing the most influential parameters in detail with the Sobol' method. Roberts and Lee [2012] used the RNetLogo package to fit general linear models to the simulation output data and to create advanced graphics for the results of a model of peer influence on the behavior among teenage drivers in a social network.

Beside the applications the tools presented have been stimulating others to develop similar tools. For example, MatNet, a Matlab-Extension to NetLogo, is partly built on the code of the R-Extension [Biggs, 2013, Biggs and Papin, 2013] and the pyNetLogo library providing a connection between Python and NetLogo re-uses code of the RNetLogo package [Kwakkel, 2013, 2014]. Moreover, the stats-Extension was inspired by the R-Extension [Stealin, 2012].

This impact and feedback overview suggests that this work already contributed towards rigorous agent-based modelling. It seems that it made the life of many agent-based modellers easier and saved time of implementation which can be invested into model analysis.

Table VI.1.: R-/Rserve-Extension. Impact and feedback.

1. Downloads

586 downloads of R-Extension version 1.2 within one year from sourceforge.

295 downloads of Rserve-Extension version 0.1beta within one year from sourceforge.

2. Books

Railsback and Grimm [2012, p. 313]

"In the other direction, a NetLogo R extension gives access to any R command from within NetLogo (Thiele and Grimm, in press). This extension makes R's very large library of mathematical and statistical functions work essentially as NetLogo primitives. For example, you can use R's spatial analysis packages to analyse movement of your turtles (how big of an area did each turtle use?); or you can use R to program more efficient simulation experiments, for example by using latin hypercube sampling to conduct parameter sensitivity analysis with fewer model runs."

O'Sullivan and Perry [2013, p. XXIV]

"Most of the figures were directly generated from these models using the excellent NetLogo-R extension described by Thiele and Grimm (2010)."

3. Courses

http://www.forst.tu-dresden.de/summerschool/index.php?page=2009-3 (last accessed 2014/01/08)

Summer School Individual- and Agent-based Modelling 2010 by Uta Berger, Volker Grimm, Steven Railsback, and Cyril Piou, Lecture 15: R-Extension.

http://torinor.net/wp-content/uploads/2012/12/Connecting_R_with_PythonNetLogo_20121129_PietroTerna.pdf (last accessed 2014/01/08)

Tutorial "Connecting R with..." by Pietro Terna, Connecting NetLogo and R (using Rserve tool).

http://simulatingcomplexity.wordpress.com/tag/r-extension/PythonNetLogo_20121129_PietroTerna.pdf (last accessed 2014/07/03)

Tutorial "R you experienced? Using the R extension for NetLogo" by Benjamin Davies.

4. Direct Feedbacks (subset)

N. Klepeis, USA (eMail, Sep. 2013)

"I love your R extension for NetLogo. I have succeeded in incorporating sophisticated numerical modeling into ABM's using my existing R scripts."

T. Filatova, The Netherlands (eMail, Feb. 2013)

"I am a big fan of both NetLogo and R and was very happy to find out about the R-extension for NetLogo you designed. Thanks for that!"

P. Smaldino, USA (eMail, Feb. 2013)

"I am taking a stab at using your NetLogo-R-Extension, which seems like it will be extremely valuable."

M. Fernandez (eMail, Sep. 2012)

"After following your excellent install instructions, probably the best I've seen ever, it works fine."

C. Liu, New Zealand (eMail, Oct. 2011)

"First thank you for your handy examples in R-NetLogo Extension. They are quite helpful to me."

F. Ascensao, Portugal (eMail, Jun. 2011)

"Before anything, my sincere thankfulness for your work with the NetLogo R extension. It's definitely going to improve the analysis I'm doing for my PhD."

U. Wilensky, USA (eMail, Feb. 2011)

"Just wanted to drop a note to express our appreciation for your commitment to keeping up the "R" extension. It's a very useful extension to many of our users."

P. Terna, Italy (eMail, Jan. 2011)

"Thanks for your wonderful initiative and effort in connecting NetLogo and R."

C. Piou, France (eMail, Apr. 2010)

"Just to tell you that your application to meet R and NetLogo got a new fan... that's great! Thanks for the hard work on connecting the 2 and the nice documentations (and paper!)."

E. Bruch, USA (eMail, Apr. 2010)

"I recently discovered your wonderful extension to NetLogo for R"

E.R. Crema, UK (eMail, Apr. 2010)

"Fristly, thanks a lot for your extension, its' a wonderful Idea and for sure It's the most useful thing I've ever seen in any ABM software."

D. Birks, Australia (eMail, Apr. 2010)

"Firstly, thank you for working on the R extension for NetLogo. [...] Currently all of my model outputs are batch processed by R, having read your paper, the ability to produce real-time advanced statistics through R within NetLogo is something I have long dreamt about."

Table VI.2.: RNetLogo. Impact and feedback.

1. Downloads

1583 downloads of version 0.9-6 between Apr. 2013 and Dec. 2013 from CRAN.

2. Books

Railsback and Grimm [2012, p. 313]

"Similarly, R (www.r-project.org) is a very popular platform for statistical programming and analysis. A new optional package for R allows users to execute NetLogo programs from within an R program (Thiele et al., in prep.)."

O'Sullivan and Perry [2013, p. XXIV]

"The analyses we present were all conducted using the freely available R environment (R-Development-Core-Team, 2012), some taking advantage of the RNetLogo library (Thiele et al., 2012) which allows NetLogo to run within R and so takes advantage of the latter's analytical capabilities."

3. Courses

http://www.forst.tu-dresden.de/summerschool/index.php?page=2012-2
(last accessed 2013/06/12)

Summer School Individual- and Agent-based Modelling 2012 by Uta Berger, Volker Grimm, and Cyril Piou, Lecture 14: RNetLogo.

http://www.demogr.mpg.de/de/ausbildungkarriere/international_advanced_studies_in_demography_3279/courses_3280/agent_based_modeling_and_simulation_abm_abs_3287/default.htm
(last accessed 2014/01/08)

Agent-based modeling and simulation (ABM-ABS), Course coordinator: Frans Willekens, Short course 2: Agent-based modeling using ODD protocol, the NetLogo programming platform and the RNetLogo package for linking NetLogo and R (instructors: Dr. Jürgen Groeneveld, Department Ökologische Systemanalyse, Helmholtz-Zentrum für Umweltforschung (UFZ), Leipzig, and Dr. Katrin Meyer, University of Goettingen).

4. Internet Resources

Review on openABM, 2012/07/12, http://www.openabm.org/story/rnetlogo-new-package-links-netlogo-r (last accessed 2014/01/08)

"RNetLogo, a New Package That Links NetLogo & R. [...]"

Note at RBloggers, 2012/01/23, http://www.r-bloggers.com/rnetlogo-a-package-for-running-netlogo-from-r/ (last accessed 2014/01/08)

"RNetLogo - A package for running NetLogo from R. [...]"

Blog Post, 2013/09/08, http://sgsong.blogspot.de/2013/09/rnetlogo.html (last accessed 2014/01/08)

"The RNetLogo package is a small piece of wonder. One can conduct agent-based simulation, try different model parameters, and then collect and conduct statistical analysis of the simulation results."

Message on Google Group Modelling4All, 2013/05/17, https://groups.google.com/forum/#!topic/modelling4all/DUjgmfpaQiU
(last accessed 2014/01/08)

"I found a couple of nice pages describing RNetLogo and I thought it would be worth mentioning since the topic is becoming popular..."

5. Direct Feedbacks (subset)

E. van der Vaart, UK (eMail, Aug. 2013)

"I just wanted to thank you for your wonderful RNetLogo package."

D. Worm (eMail, Jun. 2013)

"First of all, this package is exactly what I am looking for."

J. Lello, UK (eMail, Apr. 2013)

"Thank you for this great package and extremely kind support."

P. Galpern, Canada (eMail, Sep. 2012)

"I want to thank you for a superb interface to NetLogo. It is has increased my productivity substantively. I now treat NetLogo essentially as an interpreter/engine and design simulations/explore parameter spaces from R."

A. Kane (eMail, Nov. 2012)

"I'd like to thank you for your RNetLogo Package, it's proving to be really helpful."

M. Barton, USA (eMail, Jul. 2012)

"RNetLogo is an R package with a lot of potential. Thanks for doing it."

F. Rebaudo, France (eMail, Mar. 2012)

"First of all thanks for RNetLogo, I don't remember how it was possible to work without it!"

S.C. Robert, USA (eMail, Feb. 2012)

"I have been working with your RNetLogo package as a part of a class project. The package has made it extremely easy for me to use NetLogo and analyze the resulting data in R."

VI.2. Challenges and Shortcomings

The R-/Rserve-Extension and the RNetLogo package make several things easier and more rigorous. However, where there is light, there is also shadow: The new tools introduce new challenges to the users.

For the R-Extension I often observed problems with the installation. Setting environment variables seems to be challenging to some users. Thus, I further extended the documentations, provided detailed step-by-step guides and included an extension which can be used to evaluate the set up. This addressed also the problem that was introduced when R switched to a separate 32-/64-bit installation since R 2.12, whereas NetLogo comes with a 32-bit Java on Windows machines by default. Furthermore, when large amounts of data are transferred between NetLogo and R memory issues can arise. This is due to the fact that both, NetLogo and R, share a common system process and therefore a common memory. Whereas advanced users are able to increase the memory at start up of NetLogo this becomes difficult to inexperienced users. Furthermore, the amount of data is often underestimated. I saw, for example, cases where users created so much data that R vectors required several Gigabytes of memory which would become also critical when running R stand-alone.

Overcoming these challenges was one motivation for introducting the Rserve-Extension. Here, NetLogo and R run in separated system processes and do not share a common memory block. While this makes the installation easier it brings new drawbacks: running the BehaviorSpace with multiple processes in parallel is not supported. Furthermore, the user must be able to install and start an Rserve server, however, this is relatively easy and I observed problems with this very rarely.

The installation and set up of the RNetLogo package is much easier and can be automated via CRAN. However, on Linux and Macintosh machines the required rJava package does not run in pure R and needs to be started from JGR for example, currently. This is not special to the RNetLogo package but applies to all packages requiring rJava. Not recognizing this advice in the manual is an often observed issue. Furthermore, using a mediator software between R and NetLogo slows down the simulations. However, some improvements in the transfer strategy and a guideline showing how to use the RNetLogo package best reduced this problem as shown in a package vignette. In another package vignette, I showed how RNetLogo can be used in combination with the parallel package to run multiple NetLogo simulations distributed on several system processes and controlled by RNetLogo.

A challenge that applies to all the solutions coupling R and NetLogo is that users should have enough knowledge of both softwares. Necessary is at least a good knowledge of the data types and concepts. Many of the problems I saw resulted from sending lists with mixed data types, which is supported in NetLogo, to an R vector, which does not support such mixing. Maybe such pitfalls must be highlighted more strongly in documentations and need an automatic checking mechanism in future versions.

In general, such software tools are only successful when continuous developments and adaptations are done. During the last years several adaptations of the extensions and the package had to be done due to changes in NetLogo and R. Because the source code of the software tools is released under an open-source license and is given to the community, continuous updates can be done by everyone. Furthermore, the code can be used to inspire others for further developments.

VI.3. Open Issues

Although this thesis contributes a step ahead towards rigorous agent-based modelling and is already bearing fruits as shown in the impact analysis, it is just one piece of a larger puzzle. There are many other open topics not directly addressed in this thesis so far. Some of them are discussed and first concepts are given in the following. However, only the realization of a concept does not make it a standard. It takes a long way and depends on the continuous support by the developers of the software tools and the acceptance by the community, which is not always predictable.

VI.3.1. Analysis of ABMs

Many ABMs include stochastic effects, i.e., random variables, generated by random number generators, are used for some parameters, inputs or initializations instead of fixed values. This means that running the same model with same settings but a different random seed results in different simulation results [see also North and Macal, 2007]. However, often single runs of an ABM are interpreted. Of course, this is not wrong and makes sense to understand cause-reaction mechanisms but can be insufficient to derive general conclusions. Single runs can be non-representative and reflect extreme cases. Therefore, repeated simulations are needed additionally. Then, mean values, confidence bands and extreme values for each time step can be calculated and plotted. However, a general rule for choosing the number of repetitions to assume convergence is currently missing as shown in Chapter IV. The Bayesian methods (like Approximate Bayesian Computation) could help here, however, the large number of repetitions require a lot of computer power and can be limiting for large and processing-intensive models. Although Railsback and Grimm [2012], North and Macal [2007], Grimm and Railsback [2005], for example, already address challenges in stochastic models and model analysis briefly, a further sensitization of modellers to such problems and additional guidelines could help to overcome these challenges.

VI.3.2. Debugger

Integrated development environments (IDE) for software programming include convenient step-wise debuggers, nowadays. Such debuggers provide functionalities to set break points where the execution stops and the state of the system can be checked and changed. Furthermore, one can walk through code execution step-wise, go into called functions/methods, move back to the calling code and so on. Such a mechanism is currently provided in ABM development environments where the model is directly implemented by using "only" libraries of a higher, general-purpose programming language, like in Java with Repast or Maven. However, this is using the standard debugging provided with the programming language and, therefore, the debugger is not restricted to the user-defined model code but goes deeper to the linked libraries as well. For the averaged modeller with rather weak programming background, this is not very comfortable. For NetLogo there is currently no debugging mechanism available, although Railsback et al. [2006] already pointed to the importance of such a functionality. At the moment, one can only use "print"-statements, execute code blocks separately, or use agent monitors for debugging purposes.

For NetLogo a first concept was developed during preparation of this thesis. I called this concept a "NetLogo Pseudo-Debugger". The idea is to separate the debugger from the NetLogo compiler and use the Controlling API of NetLogo to run the simulation stepwise. Thus,

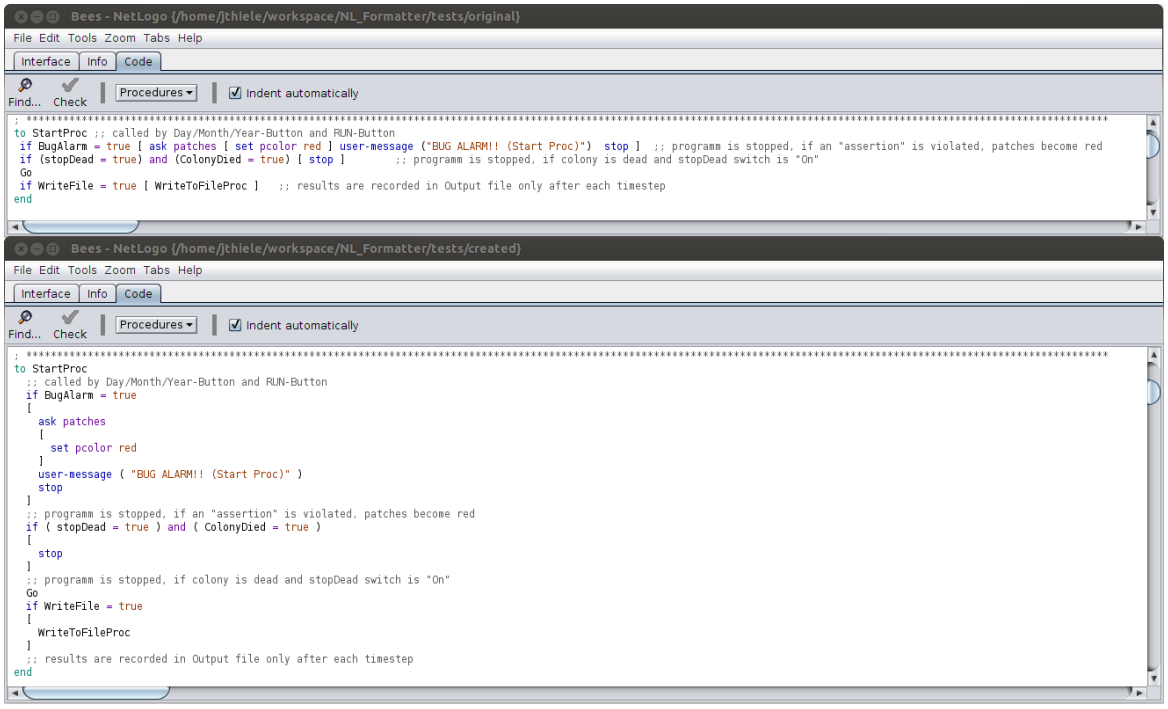


Figure VI.1.: Top: Original unformatted but valid NetLogo code. Bottom: The same piece of code after running the NLFormatter. Every independent block of code is in a separated line. Nesting hierarchies are identified by indentation levels. This structure is the basis for line-by-line executions.

the debugger is placed into a separate application. The window has, like NetLogo itself, several tabs; one with the NetLogo interface and another with the code.

The original idea was to write a domain-specific language (DSL) definition with Xtext for Eclipse [Foundation, 2013] to make it possible to use Eclipse for writing, debugging, and running NetLogo models. This requires a context-free grammar definition, for example in Backus-Naur form. However, there is no such a grammar definition of the NetLogo language available and difficult to create as the NetLogo language does not know line end specifiers, function end tokens nor a fixed number of function arguments [Harvey, 2006].

Therefore, I changed the concept to a slightly easier solution without Eclipse and the definition of a DSL. In this concept, the code written in the code tab is formatted specially; every block of commands that can be separated reasonably is written in a separate line (Figure VI.1). Opening and closing brackets are also in separate lines if they do not belong to a primitive or reporter. Nested blocks of commands, like in loops or if-clauses, are indented hierarchically. This enables the modeller to set breakpoints to specific lines. The programming work for this formatting task is already finished and called NLFormatter (available at http://sourceforge.net/projects/nlformatter/).

In a second step, the different code lines in the code tab are mapped to an internally "resolved" code. In this resolved code, calls of user-defined procedures are recognized and linked. Furthermore, loops are resolved and restructured so that they can be processed line-by-line for each iteration. For example, for an "ask turtles" call, first, an agentset of

the turtles is created. Afterwards, for each turtle each line of the belonging code block is processed separately, meaning the modeller can execute each iteration line-by-line.

Similar things are done for if-else-clauses. The code blocks are saved separately and, first, only the reporter is executed. Depending on the result, the cursor of the debugger jumps to the mapped code line for true or for false (else-block).

Using the Controlling API [Tisue, 2012] it is also possible to change variable values of agent variables as well as global variables during the stepwise model execution. With NetLogo's *org.nlogo.lite.InterfaceComponent* class [Tisue, 2012], the NetLogo interface tab can be included into the Pseudo-Debugger application window. The code tab will be written from scratch, because it needs the options of setting breakpoints, highlight the line that is currently processing and so on. An additional component has to be written to check and change values of agents and global variables equivalent to NetLogo's Command Center.

With this concept the full range of debugging functionalities can be realized without changing NetLogo itself. Of course, the code executing will become much slower in the Pseudo-Debugger by resolving all process flows and executing everything line-by-line through the Controlling API, however, executing speed does not matter during debugging. After the modeller finished the debugging and is happy with the model code she/he will switch back to the original NetLogo program.

The development of this concept started when the NetLogo code was not released under an open source license and was not available. In the recent time a growing number of additional tools and documentations is available. Therefore, it is maybe possible to reuse several things from the NetLogo developers, like the JFlex definition of the tokenizer and parser [Tisue, 2014b], the syntax highlighting [Tisue, 2014c] or the language testing functionalities [Tisue, 2014a], to simplify the implementation of the Pseudo-Debugger and, maybe, come back to the idea to implement it with Eclipse.

VI.3.3. ODD-Generator

Often, model code and ODD model description are written independently. The model code is written within the modelling environment and the description is written in a text editor. NetLogo comes with an info tab for model description, however, it does not follow the ODD protocol and it is still separated from the model code. However, it is desirable to integrate the model description into the model code to see any discrepancy very fast. Furthermore, some parts of the ODD documentation could be directly derived from the model code, like the dimension of the world, the simulation step size, the entities, the state variables and so on. To keep it comfortable, a functionality to see model and code side by side as well as hide one part should exist.

Ideally, one would write the model description and then insert, or where possible automatically generate, the corresponding model code. Similar mechanisms are already known from software development with documentation generators, like Javadoc [Oracle, 2014], Doxygen [van Heesch, 2014] or Sphinx [Brandl, 2014], and code generators in the context of model-driven software engineering, like the Eclipse Modeling Framework. Visual programming is already possible with Repast [Road, 2010], and the use of Unified Modelling Language (UML) for ABM model descriptions has been described by Bersini [2012]. Also the application of a documentation generator to produce an ODD description has been done by [Topping et al., 2010] by using Doxygen for a model written in C++. However, a community-accepted general strategy and a toolset for NetLogo or all frequently used ABM

toolkits is missing, currently. I am sceptical that UML can become a standard in ABM since the hurdles for non-programmers, which most of agent-based modellers from outside informatics are, are too high. However, the approach of Topping et al. [2010] should be pursued further. For NetLogo, adapting this approach is not easy since Doxygen does not support this language. I suggest to first define an implementation-independent definition of code markers for ODD protocol generation, meaning that there are annotations defined indicating the different parts of the ODD protocol. Afterwards, we need platform specific implementations of the ODD markers. Once this is done, one can write annotated comments into the code. The code editors should have buttons to switch off code or ODD parts, functions to export the ODD to various formats and insert details generated from code. In a further step, mechanisms adapted from tools like JDiff [Doar, 2007] should be implemented, which allow the modeller to generate reports of what has changed between two versions of the model/model description.

VI.3.4. Community-Based Submodel Collection

Currently, I have the impression that many ABMs are built from scratch. Former developments are maybe discussed in the introduction of an article, however, processes are often re-invented, although sometimes there are similarities in several submodels with other models. My idea is to establish a toolbox of commonly used submodels implemented in all commonly used languages. An institution, like the openABM Consortium, defines submodels with pen and paper, and a well-tested reference implementation in one language is provided alongside the submodel description. Later, users of other ABM toolkits can provide their implementations of the same submodels. The consortium will put them in incubation phase and after acceptance they get marked. This could reduce the often seen re-invention process and could lead to further rigour in ABM. Model description could contain only reference to the labels of the submodels and everybody in the community knows which submodel is referenced to, or can check the definition on the submodel documentation collection of the consortium. Furthermore, when somebody modifies an accepted submodel, the modeller can only document what was changed and, thus, give a clear description what was intended when using a different submodel. We already find first attempts to something similar with the model archive of the openABM Consortium [Janssen et al., 2008], but this is currently for whole models, not for model parts, and it is not put into a larger strategy of rigorous submodel collection with definitions and reference implementations. Also, some similarities can be found in the Modelling4All project [Kahn, 2014], where users can construct a NetLogo model by choosing from different submodels. However, such a project should be built on the shoulders of a consortium. This idea is inspired by the situation in the geospatial sector: The Open Geospatial Consortium (OGC) releasing implementation-independent technical definitions in the context of geographic information systems [OGC, 2014] and the Open Source Geospatial Foundation (OSGeo) supporting and promoting the collaborative development of such software [OSGeo, 2014]. When a software implements the technical definitions it is labelled as OGC-compliant and when it passes the incubation it is labelled as a OSGeo-project.

My dream is that a whole framework for rigorous agent-based modelling exists with conceptional and technical definitions, specification of interfaces, and corresponding technical implementations. Later on, also models could be labelled as, for example, openABM-compliant.

VI.3.5. Open Source Licensing Model for ABMs

As already discussed in Chapter V about replication of agent-based models, one piece of puzzle to make ABMs more reliable and increase their acceptance is to make their source code public, i.e., to assign an open-source license. Although there are many different open source licenses already available [see, e.g., Wikipedia, 2014, for a comparison] they all miss one point: When a modified version of the model is published there is no guarantee for being cited. Only the copyright notice in the source file must be given. Furthermore, as we look into the future and assume that models are commonly described with the ODD protocol [Grimm et al., 2010], it should be obligated to publish the modified model also by adapting the ODD protocol and make it public available. Therefore, it is desirable to create an open source licensing model based on one of the commonly used open source licenses already available and adapt it to the specific purposes of ABMs. Here, a cooperation with legal experts should be established.

VI.3.6. Outlook

So far, a step towards rigorous agent-based modelling was done with this thesis. A lot of users already use the presented tools. However, as partly shown a lot of steps are still missing. I hope that especially the aforementioned ideas will find other enthusiasts who are willing to help taking ABM to the next level of scientific acceptance.

VI.4. References

Altmetric. Article Details, 2014. URL http://www.webcitation.org/query?url=http%3A%2F%2Fwww.altmetric.com%2Fdetails.php%3Fdomain%3Donlinelibrary.wiley.com%26citation_id%3D565306%23&date=2014-01-08. (last accessed 2014/01/08).

H. Bersini. UML for ABM. *Journal of Artificial Societies and Social Simulation*, 15(1), 2012. URL http://jasss.soc.surrey.ac.uk/15/1/9.html.

M. Biggs. MatNet: The Matlab Extension for NetLogo, 2013. URL https://github.com/mbi2gs/netlogo-matlab-extension/wiki. (last accessed 2014/07/03).

M.B. Biggs and J.A. Papin. Novel Multiscale Modeling Tool Applied to Pseudomonas aeruginosa Biofilm Formation. *PLoS One*, 8(10):e78011, 2013.

G. Brandl. Sphinx. Python Documentation Generator, 2014. URL http://sphinx-doc.org/. (last accessed 2014/05/22).

M.B. Doar. JDiff - An HTML Report of API Differences, 2007. URL http://javadiff.sourceforge.net/. (last accessed 2014/05/22).

K.R. Falbo. An Individual Based Larval Dispersion model for the Hawaiian Hawksbill Sea Turtle in the Hawaiian Archipelago. Master's thesis, Humboldt State University, 2011.

A.M.H. Fernández. *Improving Water Network Management by Efficient Division into Supply Clusters*. PhD thesis, Universitat Politècnica de València, 2011.

The Eclipse Foundation. Xtext 2.5 Documentation, 2013. URL http://www.eclipse.org/Xtext/documentation/2.5.0/Xtext%20Documentation.pdf. (last accessed 2014/05/22).

B.M. Frank and P.V. Baret. Simulating Brown Trout Demogenetics in a River/Nursery Brook System. The Individual-Based Model DemGenTrout. *Ecological Modelling*, 248:184–202, 2013.

V. Grimm and S.F. Railsback. *Individual-Based Modeling and Ecology*. Princeton University Press, Princeton, N.J., 2005.

V. Grimm, U. Berger, D.L. DeAngelis, J.G. Polhill, J. Giske, and S.F. Railsback. The ODD Protocol: A Review and First Update. *Ecological Modelling*, 221:2760–2768, 2010.

B. Harvey. BNF Grammar for LOGO Programming Language, 2006. URL http://newsgroups.derkeiler.com/Archive/Comp/comp.lang.logo/2006-04/msg00015.html. (last accessed 2014/05/22).

I.P. Ibarra and M.A. Janssen. Mobility, Resource Harvesting and Robustness of Social-Ecological Systems. *CSID Working Paper Series*, 2012.

M.A. Janssen, L.N. Alessa, M. Barton, S. Bergin, and A. Lee. Towards a Community Framework for Agent-Based Modelling. *Journal of Artificial Societies and Social Simulation*, 11 (2) 6, 2008. URL http://jasss.soc.surrey.ac.uk/11/2/6.html. (last accessed 2014/01/06).

JASSS. Most Viewed Articles During the Past 8 Weeks, 2014. URL http://www.webcitation.org/query?url=http%3A%2F%2Fjasss.soc.surrey.ac.uk%2Fadmin%2Ftop20.html&date=2014-01-08. (last accessed 2014/01/08).

K. Kahn. Modelling4All - About, 2014. URL http://m.modelling4all.org/about/index.html. (last accessed 2014/05/22).

J. Kwakkel. Exploratory Modeling and Analysis, 2013. URL http://wiki.tudelft.nl/pub/Education/SPM955xABMofCAS/GuestLectueJanKwakkel/ema20lecture20agent20based1.pdf. (last accessed 2014/07/03).

J. Kwakkel. pyNetLogo, 2014. URL https://github.com/quaquel/pyNetLogo. (last accessed 2014/07/03).

M.J. North and C.M. Macal. *Managing Business Complexity: Discovering Strategic Solutions with Agent-Based Modeling and Simulation*. Oxford University Press, Oxford etc., 2007.

OGC. About OGC, 2014. URL http://www.opengeospatial.org/ogc. (last accessed 2014/05/22).

Oracle. Javadoc Tool, 2014. URL http://www.oracle.com/technetwork/java/javase/documentation/javadoc-137458.html. (last accessed 2014/05/22).

OSGeo. About the Open Source Geospatial Foundation, 2014. URL http://www.osgeo.org/content/foundation/about.html. (last accessed 2014/05/22).

D. O'Sullivan and G.L.W. Perry. *Spatial Simulation: Exploring Pattern and Process*. Wiley, 2013.

S.F. Railsback and V. Grimm. *Agent-Based and Individual-Based Modeling: A Practical Introduction*. Princeton University Press, 2012.

S.F. Railsback, S.L. Lytinen, and S.K. Jackson. Agent-Based Simulation Platforms: Review and Development Recommendations. *Simulation*, 82:609–623, 2006.

Road. Repast Simphony, 2010. URL http://repast.sourceforge.net/repast_simphony.html.

(last accessed 2014/01/06).

S.C. Roberts and J.D. Lee. Using Agent-Based Modeling to Predict the Diffusion of Safe Teenage Driving Behavior Through an Online Social Network. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 56:2271–2275, 2012.

C. Stealin. Version 1.2.0 of the stats extension - Message in the NetLogo users Group at 2012/05/08, 2012. URL http://netlogo-users.18673.x6.nabble.com/Version-1-2-0-of-the-stats-extension-td4960300.html. (last accessed 2014/07/04).

J.C. Thiele and V. Grimm. NetLogo Meets R: Linking Agent-Based Models with a Toolbox for Their Analysis. *Environmental Modelling & Software*, 25(8):972–974, 2010.

J.C. Thiele, W. Kurth, and V. Grimm. RNetLogo: An R Package for Running and Exploring Individual-based Models Implemented in NetLogo. *Methods in Ecology and Evolution*, 3: 480–483, 2012a.

J.C. Thiele, W. Kurth, and V. Grimm. Agent-Based Modelling: Tools for Linking NetLogo and R. *Journal of Artificial Societies and Social Simulation*, 15 (3) 8, 2012b. URL http://jasss.soc.surrey.ac.uk/15/3/8.html. (last accessed 2014/01/06).

S. Tisue. Controlling API, 2012. URL https://github.com/NetLogo/NetLogo/wiki/Controlling-API. (last accessed 2014/01/06).

S. Tisue. Language Tests, 2014a. URL https://github.com/NetLogo/NetLogo/wiki/Language-tests. (last accessed 2014/05/22).

S. Tisue. Compiler Architecture, 2014b. URL https://github.com/NetLogo/NetLogo/wiki/Compiler-architecture. (last accessed 2014/05/22).

S. Tisue. Syntax Highlighting, 2014c. URL https://github.com/NetLogo/NetLogo/wiki/Syntax-highlighting. (last accessed 2014/05/22).

C. Topping, T.T. Høyea, and C.R. Olesen. Opening the Black Box - Development, Testing and Documentation of a Mechanistically Rich Agent-Based Model. *Ecological Modelling*, 221:245–255, 2010.

D. van Heesch. Doxygen, 2014. URL http://www.stack.nl/~dimitri/doxygen/. (last accessed 2014/05/22).

F. Vinatier, M. Gosme, and M. Valantin-Morison. Explaining Host-Parasitoid Interactions at the Landscape Scale: A New Approach for Calibration and Sensitivity Analysis of Complex Spatio-Temporal Models. *Landscape Ecology*, 28(2):217–231, 2013.

Wikipedia. Comparison of Free and Open-Source software Licenses, 2014. URL http://en.wikipedia.org/wiki/Comparison_of_free_software_licenses. (last accessed 2014/05/22).

APPENDIX A

Documentation of MultiView-Extension

A.1. Introduction

The MultiView-Extension gives you the opportunity to visualize more than one patch variable at a time within different views. A second usage scenario is that you preserve the state of former simulation steps in different windows, to see the spatial and temporal (patch) pattern during the simulation.

It creates new windows, in which you see the NetLogo World/View (only the patches, without turtles). For each window you can define a patch variable which should be used for the colourization of the patches as it is done for the normal view with the `pcolor` variable. The new window(s) has/have a context menu which gives you the opportunity to inspect the patches and to export the view into a png image file.

A.2. Author and Copyright

Jan C. Thiele, 2010
University of Goettingen, Germany
Department of Ecoinformatics, Biometrics and Forest Growth

A.3. Caution

The MultiView-Extension is new and experimental. It is not yet tested extensively. Nonetheless, I hope it will find some users. Please let me know about your experiences (jthiele at gwdg dot de, http://www.uni-goettingen.de/en/72779.html).

A.4. Usage

Unzip the package directly (without creating a new folder) into the *extensions* folder of your NetLogo installation. You should find a folder *multiview* now and within this folder a file *multiview.jar* as well as some other folders. See folder *example* for a short and simple

demonstration of the functionality of the extension. To use the extension in your model, add a line to the top of your Procedures Tab:

```
extensions [multiview]
```

If your model already uses other extensions, then it already has an `extensions` line in it, so just add `multiview` to the list.

For more information on using NetLogo extensions, see the Extensions Guide.

A.4.1. Some Remarks

To use a new View Window you should declare a new global variable, for example like this:

```
globals[
  view1
]
```

You will need such a global variable for every View Window you want to use. Then define your patch variables, for example:

```
patches-own[
  pcolor2
]
```

In your `setup` procedure you will create the new window and save the (object) reference into the global variable. This could look like this:

```
to setup[
  ; create a new window with the title "visualization for
     pcolor2" which will use the patch variable "pcolor2" for
     visualization
  set view1 multiview:newView "visualization for pcolor2"
     "pcolor2"
]
```

Every time you want an update of the colourization of the patches of your new View Window you have to call the repaint primitive for every View Window. In most cases you will do this in the `go` procedure like this:

```
to go[
   ... ; do something
   multiview:repaint view1
]
```

Please note that you cannot call the `multiview:repaint` primitive for a window which you have already closed via a mouse click, because the reference saved in the global variable will not be found and the call of `multiview:repaint` will fail. If you close NetLogo, load a new model or call the `clear-all` primitive, all currently open View Windows will be closed.

A.5. Primitives

A.5.1. multiview:close

```
multiview:close variable
```

Will close (and destroy) the View Window (if open) specified in the variable argument. Please remember that it is not possible to call the repaint primitive for the variable again after execution finished. But you can create a new window, stored in the variable, again.

A.5.2. multiview:newView

```
multiview:newView title patch-variable
```

A reporter to create a new View Window. The window uses the settings defined for the NetLogo World/View (i.e., `max-pxcor`, `min-pxcor`, `patch-size` etc.) at the time of the call of the primitive. You cannot change these settings after creation of the View Window. To close the current window and create a new one will help. The first argument, `title`, is any string (surrounded by quotation marks) and will be used as title of the new window. The second argument, `patch-variable`, is the patch-variable which will be used for the colourization of the patches in the new View Window. This argument should be given as a string (surrounded by quotation marks). You should save the return value of this reporter into a global variable.

A.5.3. multiview:rename

```
multiview:rename variable title
```

Primitive to change the title of a View Window. The first argument is the variable which holds the reference to the window and the second argument is a string (surrounded by quotation marks) with the new window title.

A.5.4. multiview:repaint

```
multiview:repaint variable
```

Primitive to repaint/update a View Window saved in variable. This should be called in every simulation step, when the NetLogo World is updated/repainted. But it is also possible to save the state of a former simulation step in such a window. In this case, you should not call the `repaint` primitive in a `go` procedure.

APPENDIX B

Documentation of Pygments-Plug-In

This is a Pygments-Plug-In to support NetLogo syntax. It enables you to create syntax highlighting and formatting from NetLogo model files into different output formats, for example HTML, RTF or LaTeX.

B.1. Installation

- Install Python (http://www.python.org/download/)
- Install easy_install (http://pypi.python.org/pypi/setuptools)
- open Shell/MS DOS prompt and type (as administrator): `easy_install Pygments`
- Voila! If this runs without error, Pygments is successfully installed!
- Next step: Install the NetLogo-Plug-In.
- Unzip the *NetLogo-Plug-In.zip* file
- Open Shell/MS DOS prompt, navigate to the extracted Plug-In folder, where the *setup.py* file is located and type: `python setup.py install`
- If this runs without errors: Congratulations! The NetLogo Plug-In is installed and ready to use!

B.2. Usage

See Pygments Documentation (http://pygments.org/docs/) for details. The following examples are all command line examples.

1. Create an image file of the NetLogo code *test1.nlogo*:

   (Python Image Library (PIL) has to be installed! (http://www.pythonware.com/products/pil/))

   a) for bitmap:

```
pygmentize -l NetLogo -O full,style=NetLogo -f bmp -o
   test1.bmp test1.nlogo
```

b) for gif:

```
pygmentize -l NetLogo -O full,style=NetLogo -f gif -o
   test1.gif test1.nlogo
```

c) for png:

```
pygmentize -l NetLogo -O full,style=NetLogo -f png -o
   test1.png test1.nlogo
```

d) for jpeg:

```
pygmentize -l NetLogo -O full,style=NetLogo -f jpeg -o
   test1.jpeg test1.nlogo
```

2. Create an rtf file for NetLogo model *test1.nlogo*:

```
pygmentize -l NetLogo -O full,style=NetLogo -f rtf -o
   test1.rtf test1.nlogo
```

3. Create an html output for NetLogo model *test1.nlogo*:

a) with embedded css-style:

```
pygmentize -l NetLogo -O full,style=NetLogo -f html -o
   test1.html test1.nlogo
```

b) with extra css-file:

Create the html file:

```
pygmentize -l NetLogo -f html -o test1.html test1.nlogo
```

Export the style to css file:

```
pygmentize -f html -S NetLogo -a .syntax >
   netlogosyle.css
```

Finally paste header and footer around your html-code (*test1.html*):

header:

```
<html>
<head>
  <title>NetLogo Code highlighted with NetLogo-Plugin
     for Pygments</title>
  <link rel="stylesheet" href="netlogostyle.css">
  <meta http-equiv="content-type" content="text/html;
     charset=utf-8">
</head>
<body>
<div class="syntax"><pre>
```

footer:

```
</pre></div>
</body>
</html>
```

4. Create a latex file for NetLogo model *test1.nlogo*:

```
pygmentize -l NetLogo -O full,style=NetLogo -f latex -o
   test1.latex test1.nlogo
```

5. Create an svg file for NetLogo model *test1.nlogo*:

```
pygmentize -l NetLogo -O full,style=NetLogo -f svg -o
   test1.svg test1.nlogo
```

APPENDIX C

Documentation of R-Extension

Version: 1.3 (August 2013)
The R-Extension of NetLogo provides primitives to use the statistical software R (Gnu S) (see http://www.r-project.org/) within a NetLogo model. There are primitives to create R variables with values from NetLogo variables or agents and others to evaluate commands in R with and without return values.

There are two other projects related to this one:

- The Rserve-Extension: similar to this R-Extension but uses no direct connection to R. It communicates via an Rserve server. This has some advantages (e.g., easier installation -> no environment variables; independent R process -> own process that can be configured independently from NetLogo and can also run on a different computer) as well as disadvantages (e.g., no interactiveShell).
- The RNetLogo package for R: this is the other way around. R is the basis and NetLogo is started and controlled from R.

C.1. Installation/Configuration

Please follow the instructions here. If you have problems please see "Troubleshooting" and also the FAQ web page. To use this extension you will need this extension, NetLogo 5.x, R (>=3.0), Java (>=1.5), rJava package (>=0.9.4) and, if you want to use the plot device, JavaGD package.

Note that the rJava package can be installed for 32- or 64-bit architectures. You have to decide if you want to run ALL required software (NetLogo, R, rJava) with 32- or 64-bit architecture.

Attention: The environment variable names (*R_HOME*, *JRI_HOME*) are case-sensitive!

First, system-independent step: Copy the folder of the R-Extension (name is *r*) into the *extensions* folder of your NetLogo installation.

C.1.1. Windows

This manual assumes you have Windows Vista, 7, or 8. Note: There are two additional pdf files with detailed step-by-step guides for Vista/Win7/Win8 64-bit users. One, explaining how to setup NetLogo and the R-Extension running with 32-bit (the easy way) and another one showing how to run both in 64-bit mode (the slightly more tricky way, because you have to make sure that NetLogo runs with a 64-bit Java).

1. Download and install R (see http://www.r-project.org/).
2. Install the rJava package (and the JavaGD package if you want to use the included plot device, see also notes to Plotting) in R. There are two different ways to do this:
   a) Start the RGui from your program list, click on the item *Packages* in the menu bar and then on *Install Package(s)*. Select your favourite server and find rJava in the list of packages.
   b) Open a console, type R (Environment variable *PATH* has to contain R for this, see 3.). Then type `install.packages("rJava")` and follow the instructions (for further information see http://www.rforge.net/rJava/index.html).
3. Set Environment Variables of the operation system: *Control Panel -> System -> Advanced Environment Variables -> System Variables*:
   a) Create a new entry:
   Variable Name: *R_HOME*
   Variable Value: *<Path to folder of your R-Installation, e.g., C:\Program Files\R\R-3.0.1\>*
   b) Add a value to variable *PATH*:
   Variable Name: *PATH*
      i. for 32-bit: Variable Value: *...;%R_HOME%\bin\i386*
      ii. for 64-bit: Variable Value: *...;%R_HOME%\bin\x64*
      Go to the path and look if there is an *R.dll* file!
   c) Create a new entry:
   Variable Name: *JRI_HOME*
   Variable Value: *<Path to folder of jri in rJava in your R-library-Path, e.g., C:\Users\<username>\Documents\R\win-library\3.0\rJava\jri*
   Hint: If you do not know where the rJava package is installed, you can start R and type `library(rJava)` to load rJava and then type `path.package("rJava")` to get installation path. Now append */jri* and use this path for setting the *JRI_HOME* environment variable.

C.1.2. Linux

Java (JRE or JDK) and R must be installed. If you have installed it via a package manager, it should be callable after installation in the console when typing `R`. Make sure that Java and R are running in the same architecture: 32- or 64-bit.

If R is running, install the R-package rJava (and JavaGD if you want to use the included plot device), type `install.packages("rJava")` and follow the instructions (for further information see http://www.rforge.net/rJava/index.html). Then you have to set two global environment variables by typing:

1. `export R_HOME= <path to your R installation, e.g., /usr/lib/R>`
2. `export JRI_HOME= <path to the jri-folder of the rJava library, e.g., /usr/lib/R/library/rJava/jri or /usr/local/lib/R/site-packages/jri>`
   Hint: If you do not know where the rJava package is installed, you can start R and type `library(rJava)` to load rJava and then type `path.package("rJava")` to get installation path. Now append */jri* and use this path for setting the *JRI_HOME* environment variable.

Then start NetLogo from this shell. If you do not want to set the variables after a restart again, you can save the commands in your *.profile* file. If you have problems in get it running, see Troubleshooting and have a look into the two additional pdf file for Windows (32-bit, 64-bit). Especially section 9. can also be applied here to check, if rJava works.

C.1.3. Macintosh

Many thanks to Enrico R. Crema, Erich Neuwirth and Simone Gabbriellini for providing their solutions.

For Mac OS until Mountain Lion (submitted by Enrico R. Crema) First install R and the rJava package (and JavaGD package, if you want to use the included plot device) in R. Next, set the environment variables *R_HOME* and *JRI_HOME*. You need to edit the file *environment.plist* inside *~/.MacOSX* as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.
    apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
<key>JRI_HOME</key>
<string>/Library/Frameworks/R.framework/Resources/library/ rJava/
    jri</string>
<key>R_HOME</key>
<string>/Library/Frameworks/R.framework/Resources</string>
</dict>
</plist>
```

If you do not want to use the terminal for editing (and actually creating the *environment.plist* file) you can use this package: http://www.rubicode.com/Software/RCEnvironment/ which allows the creation and the management of the *environment.plist* file. See Figure C.1 for an example of the file with the settings for the R-Extension. The path to *R_HOME* and *JRI_HOME* should be the same for any user and if this works editing manually the *environment.plist* file should perfectly work. Hint: If you do not know where

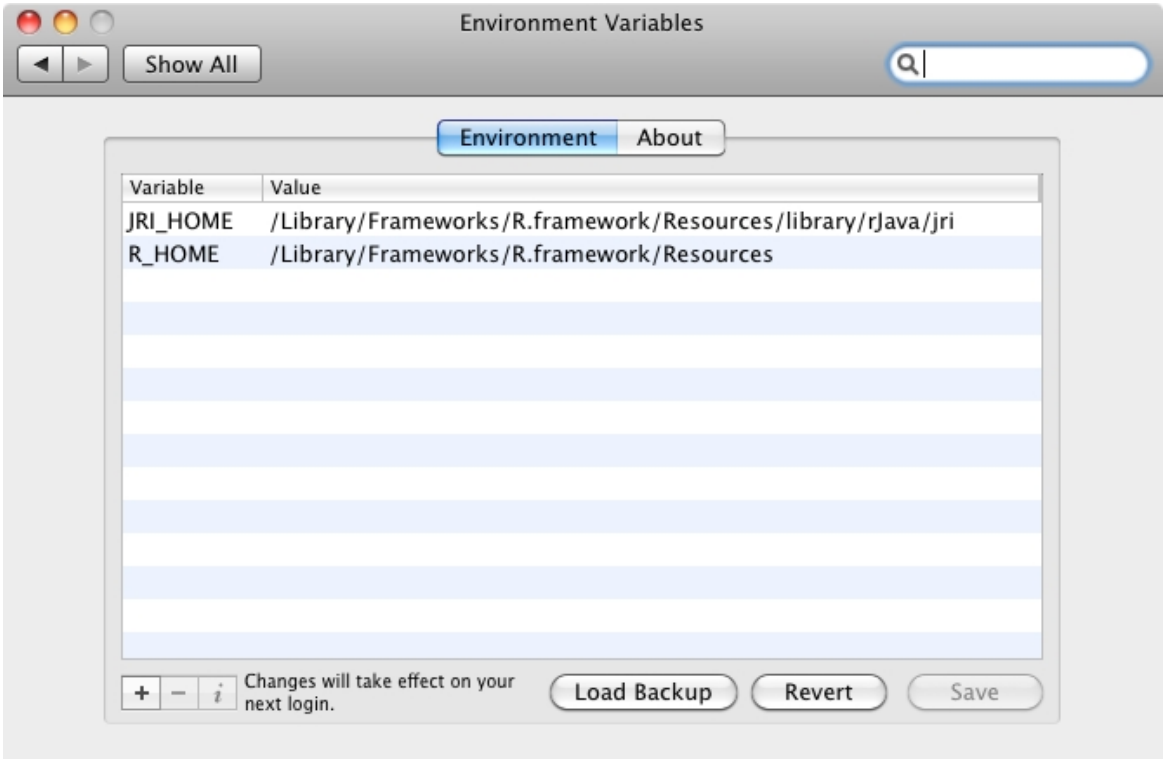


Figure C.1.

the rJava package is installed, you can start R and type `library(rJava)` to load rJava and then type `.path.package("rJava")` to get installation path. Now append */jri* and use this path for setting the *JRI_HOME* environment variable.

Since Mountain Lion (reported by Simone Gabbriellini) Open a Terminal Window.
Use the command `sudo nano /etc/launchd.conf`.
Enter your password.
Add the environment variables (*R_HOME* and *JRI_HOME*, see above) using the following syntax:

```
setenv NAME_OF_VARIABLE valueOfTheVariable
```

Press Ctrl + o to save the file.
Press enter to accept the name of the file.
Press Ctrl + x to exit the editor.

Other Solution (by Erich Neuwirth) Ctrl-Click on the NetLogo Application and select Show Package Contents. In the directory Contents you will find a file *info.plist* with the following last few lines:

```
<key>NSJavaRoot</key>
<string>..</string>
```

```
</dict>
</plist>
```

Add a few lines so that the last lines now are:

```
<key>NSJavaRoot</key>
<string>..</string>
<key>LSEnvironment</key>
<dict>
<key>JRI_HOME</key>
<string>/Library/Frameworks/R.framework/Resources/library/rJava/
    jri</string>
<key>R_HOME</key>
<string>/Library/Frameworks/R.framework/Resources</string>
</dict>
</dict>
</plist>
```

If XCode is installed on your machine, there is another way of adding the environment variables:

Ctrl-Click on the NetLogo Application and select Show Package Contents. In the directory Contents you will find a file *info.plist*. Double-click it, it will open with the PList editor of Xcode. On the Editor drop down menu, check the Show Raw Keys & Values item.

Select the list line in the editor window and select Add Item on the Editor drop down menu. A new line will appear; on the drop down of this line, select LSEnvironment. The editor then will set the type to dictionary. When the line is selected, you will see a + sign. Pressing it will allow you to define the two environment variables you need (*R_HOME* and *JRI_HOME*, see above).

Based on: http://groups.yahoo.com/group/netlogo-users/message/15188.

C.2. Troubleshooting

In the R-Extension folder you can find a folder *rsystemcheck*. Just copy this folder to your NetLogo *extensions* directory. Then, you can open the NetLogo model *Systemcheck.nlogo*, which is inside the *rsystemcheck* folder.

This model enables you to check the requirements of the R-Extension. Things to check are the Java version (used for NetLogo and R: yes, they can differ, since NetLogo comes with its own Java, e.g., on Windows), the availability of R from a terminal/shell, the availability of the rJava/JRI package and the availability of the JavaGD package (just in case you want to use the included plotting device).

Some notes on common errors:

1. NetLogo closes immediately after leaving the Code Tab or pressing the check button: This indicates that the extension was not able to start R. Please check, if you can start R from a terminal/shell. Please use the RSystemCheck-Extension!

2. You get a `java.lang.NoClassDefFoundError: org/rosuda/REngine/REngine at org.nlogo.extension.r.Entry.runOnce` error: The extension was not able to connect to the JRI-library. Please check, if you are able to load the rJava

package within an R shell (`library(rJava)`) and have a look at your *JRI_HOME* environment variable. Please use the RSystemCheck-Extension!

3. You are a Windows 64-bit user: Please note that R (since version 2.12) will be installed as dual 32-/64-bit version, while NetLogo comes with it is own Java (JRE, Java Runtime Environment) with 32-bit. You have to harmonize your NetLogo, Java in terminal/shell and R in terminal/shell and your R packages (rJava, JavaGD) to 32- or to 64-bit. Please use the RSystemCheck-Extension to get information about and test your configuration. You can configure your R to a 32-bit version (since R version 2.12) by changing the entry in your *PATH* environment variable from *%R_HOME%/bin* to *%R_HOME%/bin/i386*. To start NetLogo directly from a terminal/shell could enable you to run NetLogo with the same Java version as R does (and enables you to use 64-bit Java). Just open a terminal/shell/ms-dos prompt, navigate to your NetLogo installation folder and type `java -jar NetLogo.jar`.

Have a look into the two additional pdf files for Windows in the *doc* folder of the extension (also interesting for other OS). Especially section 9. can also be applied to check, if rJava works.

C.3. How to Use

To use the extension in your model, add a line to the top of your Procedures Tab:

```
extensions [r]
```

If your model already uses other extensions, then it already has an `extensions` line in it, so just add `r` to the list.

For more information on using NetLogo extensions, see the Extensions Guide.

For examples of the usage of the R-Extension, see the folder *examples* in the folder of this extension.

For a first description on how to use the extension in applets, see *r-extension-in-applets.html* in folder *doc* of the extension.

C.3.1. Some Tips

Plotting If you want to use the plot function of R, you can activate the JavaGD plot device via `r:setPlotDevice`, see *plot-example1.nlogo*. This is the preferred method! For this, the JavaGD package has to be available for the R-Extension. By default, only the global system-wide library path is available. Therefore, the JavaGD package has to be installed there (as Administrator/superuser) or you have to append the library path before activating the JavaGD device. This can be done for example by using the `.libPaths()` function (e.g., on Windows something like: `r:eval ".libPaths(c(\"C:/Users/username/Documents/R/win-library/3.0\", .libPaths()))"`). A third (permanent) option is to set the *R_LIBS* environment variable.

But you can also use the standard R device, but then, you have to give R some cpu time, e.g., by running an evaluation of `sys.sleep(0.01)` with a forever button. See the *plot-example2.nlogo*. (Many thanks to Thomas Petzold.). The creation of plots into files is also possible. See the *plot-into-file-example.nlogo* in the *examples* folder.

Load and Save data from/into file It is possible to load and save data from file directly in R via `r:eval "dataname <- read.table('<path to file>')"` and `r:eval "write.table(dataname, file='<filename>')"`, respectively.

Data.frame with vector in cells Normally, a data.frame cell contains only a single value. Each column is represented as a vector and if you will put a vector of vectors to a data.frame, it will be split into several columns. With the R-Extension it is possible to put a vector into a data.frame cell, when you assign a NetLogo list to a column which contains nested NetLogo lists for each row. If you want, for example, to use `write.table` on this data.frame, you have to mark this column as `class="AsIs"`. You can do this by using the `I(x)`-function. Example: If the column of interest has the name `col1` of the data.frame `df1` you can execute `r:eval "df1$col1 <- I(df1$col1)"`. Call `help(I)` from within an R terminal for further details.

Load an R-Script Furthermore, you can define functions in an R-Script, load it, and use the functions. Load R-files via `r:eval "source('<path to r-file>')"`.

Load a Package It is also possible to load R packages via `r:eval "library(<name of package>)"`.

Note that the underlying R session does not know user-specific library paths. If the package you want to load is installed in such a user-specific library path you have to add it. You can check the current library path with the R function `.libPaths()`. Just add your user-specific library path, for example under Windows, like this: `r:eval ".libPaths(c("C:/Users/<username>/Documents/R/win-library/3.0", .libPaths()))"`.

When you compile your code containing `extensions [r]` you will create a new R workspace. Until you reload the extension, open a new model or submit the primitive `r:clear`, all R variables assigned in this session will be available like you would use R from the command line or in the R Console.

interactiveShell You can open an Interactive R Shell via `r:interactiveShell`. This shell is a port to the underlying R instance. This shell works on the global environment (see R environments) while the extension itself works on a custom local environment. But there is one automatic variable `nl.env` in the global environment, which is a reference to the local environment of the extension. Do not delete this variable! You can access a variable created by the extension via `get("<variable name>",nl.env)`, for example `myvar <- get("myvar",nl.env)`. If you want to plot from the interactiveShell you should use the included JavaGD plot device (see `r:setPlotDevice`). You can save and load the history of entered R commands via a right mouse button context menu. Please read the notes at the top of the output text area after opening the shell! On Linux OS it can happen that you see an error message from X11. Please check, if everything worked correctly. If so, you can ignore these messages. If not, please write a report to jthiele @ gwdg.de.

R Environments When you load a model the R-Extension creates a new R environment. When you create an R variable using the R-Extension, this variable is created in the local

R environment. Furthermore, all calls from the R-Extension work on this local environment. This new environment concept enables you to use the extension in BehaviorSpace experiments. Therefore, you do not have to care about the environment while you are not using the interactiveShell or other tools, which work on the global environment. You can explicitly assign a variable to the global environment by using the «- operator or by executing `assign(<name>,<value>,envir=.GlobalEnv)`. If you work with the interactiveShell, see the notes at the top of the output text area after opening the shell. Type `help(environment)` in an R shell to learn more about environments.
You can/should clear (i.e., remove all variable and free memory) the local environment via `r:clearLocal`. If you want to clear also the global environment (the whole workspace), call `r:clear`.

Memory With the R-Extension you can load R into the process of NetLogo. Because of the architecture of R, both softwares share one system process and therefore the memory given to NetLogo.

In some circumstances it can happen that you receive an out of memory error due to Java's heap space. You can increase the heap space before starting NetLogo by adapting the `-Xmx` JVM-parameter (see also http://ccl.northwestern.edu/netlogo/docs/faq.html#windowsmemory). But on 32-bit systems, this is very limited. Therefore, it is a good idea to use a 64-bit system if you want/need to use high amount of RAM. You can see the memory usage of R by starting the interactiveShell (`r:interactiveShell`) and type there: `memory.size(max=F)` and `memory.size(max=T)`. Furthermore, you can check the memory limit by typing: `memory.limit()`.
See also:

- http://stat.ethz.ch/R-manual/R-patched/library/base/html/memory.profile.html
- http://stat.ethz.ch/R-manual/R-patched/library/utils/html/object.size.html
- http://stat.ethz.ch/R-manual/R-devel/library/utils/html/memory.size.html

If you call the garbage collector in the interactiveShell by typing `gc()`, you will get some information about the current memory usage (see also http://stat.ethz.ch/R-manual/R-patched/library/base/html/gc.html). If you type `gc(nl.env)` you will see the percentage of memory used for cons cells and vectors. Do not forget to call the `r:gc` primitive after removing an R variables and do not forget to remove R variable you do not need anymore! See how the memory usage changes after removing variables and calling `r:gc`.

If you use too much memory, it can happen that NetLogo will close abruptly. In such a case, check if there is a way to reduce the memory used. If not, try to switch over to the Rserve-Extension. With the Rserve-Extension both softwares, NetLogo and R, run independently. There is, of cause, also a limit of transferable data amount with one request, but it is less restrictive.

One last note on this topic: Keep in mind that R is a vector-oriented language. Prevent mass calls with single values whenever possible and replace them by vector operations. This is much faster and more stable.

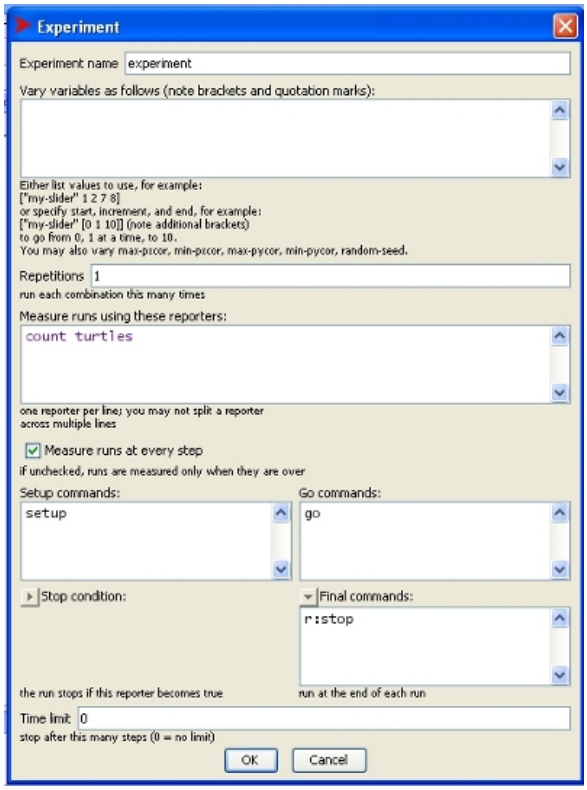


Figure C.2.

Headless Since R-Extension version 1.1 it is possible to use the extension when NetLogo is running in headless mode. This is for example the case, when you run BehaviorSpace experiments from the command line (see http://ccl.northwestern.edu/netlogo/docs/behaviorspace.html#advanced). The difference is that the interactiveShell is not initialized/instantiated. You can use the extension as you know it from GUI mode, but it is not possible to open the interactiveShell (`r:interactiveShell`) and to set the plot device (`r:setPlotDevice`). But one additional thing has to be done: You have to call `r:stop` finally when running NetLogo headless to stop the R engine. Otherwise NetLogo will not be closed and you will not get back your command line prompt. When setting up a BehaviorSpace experiment, there is the option to set final commands. This is a good place to add the `r:stop` command (see Figure C.2).

C.4. Primitives

C.4.1. r:clear

```
r:clear
```

It clears the R-Workspace. All variables in R will be deleted. It evaluates the R command `rm(list=ls())` and `rm(list=ls(nl.env))`. Therefore, it deletes variables created in global as well as local environment (see R Environments for details about environ-

ments). It is always a good idea to add this command to your `setup` procedure under your `clear-all` call.

```
;; clear the R workspace
r:clear
```

C.4.2. r:clearLocal

```
r:clearLocal
```

It clears the local R environment, which is used by the extension. All variables which have been created in the local environment will be deleted. It evaluates the R command `rm(list=ls(nl.env))`. See R Environments for details about environments. See `r:clear` for deleting all variables, i.e., the globals as well.

```
;; delete the local variables
r:clearLocal
```

C.4.3. r:eval

```
r:eval R-command
```

It evaluates the submitted R command. The R command should not return a value.

```
;; creates a new vector in R with a sequence from 1 to 10
r:eval "x <- seq(1,10)"
show r:get "x"
```

C.4.4. r:__evaldirect

```
r:__evaldirect R-command
```

It evaluates the submitted R command in the global environment (not in the local environment like `r:eval` does) and without a check (not using try-function internally). This can be necessary for some R packages, like gglopt2. Please note that you can produce name clashes when creating new variables using this primitive. The variable will be created into the global environment and will not overwrite variables with the same name that have been created into the local environment. If you request a variable with `r:get` it will search in the local environment first. Therefore, if there are variables with the same name in the local and the global environment, it will report the variable from the local environment and not the variable created via `r:__evaldirect`. If there is only one variable with the requested name in the global environment, everything will be fine - `r:get` will report the value of this variable. If you want to remove a variable created via `r:__evaldirect`, i.e., in the global environment, call `r:eval "rm(myvar, envir=.GlobalEnv)"`, replace `myvar` by the name of your variable. The R command should not return a value. This primitive is experimental.

```
;; creates a new vector in R with a sequence from 1 to 10
r:__evaldirect "x <- seq(1,10)"
show r:get "x"
```

C.4.5. r:gc

```
r:gc
```

Calls the garbage collector of Java (i.e., the R-Extension) and R. Call this primitive after removing an R variable to free the memory.

```
;; create a variable
r:eval "x <- 1:10"
;; remove the variable
r:eval "rm(x)"
;; call the garbage collector
r:gc
```

C.4.6. r:get

```
r:get R-command
```

Reports the return value of the submitted R command. Return type could be a string, number, boolean, NetLogo list or a NetLogo list of lists. R lists will be converted into a NetLogo list. If the R list itself contains further lists, it will be converted into a NetLogo list with nested NetLogo lists. Lists containing values of different data types are also supported (e.g., mixed strings, numbers and booleans/logicals). Data.frames will be converted into a NetLogo list with nested list for each column, but the column names will be lost (same for named R lists). R matrices can be received, but they are converted into one NetLogo list. `NULL` and `NA` values are not converted and will throw an error, because NetLogo has no corresponding value.

```
;; returns a list with 10 variables show
r:get "rnorm(10)"
```

C.4.7. r:interactiveShell

```
r:interactiveShell
```

Opens a window with two text areas. The upper one is the R output stream and in the lower one you can type R commands. This is the access to the underlying R session. You can type multi-line commands. To submit commands press Ctrl+Enter. With "PageUp" and "PageDown" in the input area you can browse through the history of submitted commands. With right mouse click context menu, you can save and load an RHistory (interchangeable with R terminal and other R GUIs).

Please note that the interactiveShell works on the global environment, while commands submitted from NetLogo live in a local environment. A reference to this local environment is automatically added to the global environment (named `nl.env`, please do not delete this variable. With a call of `r:clear` you can restore it but this will empty your workspace). You can use this to have access to variables which you have created from NetLogo by `get("<variable name>",nl.env)`. To copy for example a variable with the name `var1` from the local environment to the global environment, type `var <- get("var",nl.env)`. See section R Environments for details. If you just want to see the contents of a variable which lives in the local environment, you can submit your command, for example in the NetLogo Command Center, and the result will be shown in the output area of the interactiveShell. For example:

```
r:put "test" (list world-width world-height)
r:interactiveShell
```

```
r:eval "print(test)"
r:eval "str(test)"
```

Variables which have been created in the interactiveShell are available from NetLogo, even if they are created in the global environment. But if there is a variable with the same name in the local environment, you will get this variable in NetLogo instead the one from the global environment.

If you want to execute plot commands from the interactiveShell you should activate the integrated JavaGD plot device via `r:setPlotDevice` first.

```
;; opens interactiveShell
r:interactiveShell
```

C.4.8. r:put

```
r:put name value
```

Creates a new variable in R with the name name. The value can be a string, number, boolean or list. NetLogo lists are converted to R vectors, if all entries are of the same data type. If a NetLogo list contains different data types (mixed strings, numbers or booleans), it will be converted into an R list. If a NetLogo list contains other/nested NetLogo lists it will be converted into an R list and the nested lists are handled by the same rule (vectors if all items are of the same data type, ...).

```
;; creates an R variable "testvar" with the size of turtle 0
r:put "testvar" [size] of turtle 0
show r:get "testvar"
```

C.4.9. r:putagent

```
r:putagent name agent|agentset variable
(r:putagent name agent|agentset variable1 variabl2 ...)
```

Creates a new named list in R with the name `name`. `Variable` is repeatable and can be every variable of the agent or agentset. Names of the elements of the R list will be the same as the names of the agent variables. Turtles will be assigned in ascending order of their `who` variable. Patches will be assigned in lines from upper left to lower right. Since the arguments of this primitive are repeatable, do not forget the parentheses around the statement.

```
;; creates an R-list "agentlist1" with the size and the id of
   turtles, do not forget the parentheses
(r:putagent "agentlist1" turtles "size" "who")
show r:get "agentlist1$who"
;; creates an R-list "agentlist2" with the pcolor, pxcor and
   pycor of patches
(r:putagent "agentlist2" patches "pcolor" "pxcor" "pycor")
show r:get "agentlist2$pcolor"
```

C.4.10. r:putagentdf

```
r:putagentdf name agent|agentset variable
(r:putagentdf name agent|agentset variable1 variable2 ...)
```

Same as `r:putagent` but creates an R data.frame instead of a list. Please read the notes about data.frames, if one of your agent variables contains NetLogo lists.

```
;; creates an R-list "agentlist2" with the pcolor, pxcor and
   pycor of patches, do not forget the parentheses
(r:putagentdf "df1" patches "pcolor" "pxcor" "pycor")
show r:get "class(df1)"
```

C.4.11. r:putdataframe

```
r:putdataframe name varname value
(r:putdataframe name varname1 value1 varname2 value2 ... )
```

Same as `r:putnamedlist` but creates an R data.frame instead of a list. If you send more than one list to NetLogo and the lists are of different length, the smaller ones will be filled with `NA` values. If you send nested NetLogo lists (e.g., of type: `[ [ ] [ ] ... ]`) to one column please read the notes about data.frames with vectors in cells.

```
;; creates an R-list "agentlist2" with the pcolor, pxcor and
   pycor of patches, do not forget the parentheses
(r:putdataframe "df1" "v1" [12 13 14 15 16] "v2" ["foo1"
   "foo2" "foo3" "foo4" "foo5"] "v3" [1.1 2.2 3.3 4.4 5.5])
show r:get "df1$v3"
```

C.4.12. r:putlist

```
r:putlist name value
(r:putlist name value1 value2 ... )
```

Creates a new list in R with the name `name`. `Value` is repeatable and can be a number, boolean or list. Each `value` will get the name of its position (1, 2, 3,...). Since the arguments of this primitive are repeatable, do not forget the parentheses around the statement.

```
;; creates an R-list "list1", do not forget the parentheses
(r:putlist "list1" 25.5 [25 43 32 53] "testvalue" [44.3 32.32
   321.2 4.2])
show r:get "class(list1)"
show r:get "list1[[1]]"
show r:get "list1$'0'"
show r:get "list1[[2]]"
```

C.4.13. r:putnamedlist

```
r:putnamedlist name varname value
(r:putnamedlist name varname1 value1 varname2 value2 ... )
```

Creates a new named list in R with the name `name`. `Value` is repeatable and can be a number, boolean or list. Each variable will get the name `varname`. Since the arguments of this primitive are repeatable, do not forget to put the statement into parentheses.

```
;; creates an R-list "list1" , do not forget the parentheses
(r:putnamedlist "list1" "v1" 25.5 "v2" [25 43 32 53] "v3"
   "testvalue" "v4" [44.3 32.32 321.2 4.2])
```

```
show r:get "class(list1)"
show r:get "list1[[1]]"
show r:get "list1$v1"
```

C.4.14. r:setPlotDevice

```
r:setPlotDevice
```

To open an R plot in a window you can use the JavaGD plot device. With this primitive you can activate this device and all following calls of R plots will be printed with this device.

To use this device, you have to install the JavaGD package in R. Open an R terminal or the interactiveShell (see `r:interactiveShell`) and type `install.packages("JavaGD")`.

With this plot window you can save the plot to a file of various graphics formats and you can copy the plot to the clipboard. Please note that on Linux OS it can be necessary to allow to add images to the clipboard (e.g., in KDE you have to configure KLIPPER to allow images). The resolution for raster images depends on the size of the plot window. If you need high resolution maximize the window (and do not use jpeg, because the driver is bad) or better use a vector image format. Please see the notes about plotting for other details.

```
;; activate the JavaGD plot device
r:setPlotDevice
```

C.4.15. r:stop

```
r:stop
```

Stops the R engine. This is needed (only) if NetLogo is running in headless mode, for example when running BehaviorSpace experiments from the command line with something like this: `java -cp NetLogo.jar org.nlogo.headless.Main -model mymodel.nlogo -experiment exp1 -table outtab1.csv`. Should be the last call in headless simulation. See usage notes above for details.

```
r:stop
```

APPENDIX D

Documentation of Rserve-Extension

Version 0.1beta (August 2011)

The Rserve-Extension of NetLogo provides primitives to use the statistical software GNU R (see http://www.r-project.org/) via the Rserve package (based on TCP/IP connection) within a NetLogo model. There are primitives to create R variables with values from NetLogo variables or agents and others to evaluate commands in R with and without return values.

This extension is designed to work together with the RNetLogo package. If you do not want to use RNetLogo and just look for an extension to call R code from NetLogo without using a remote server we recommend to use the R-Extension based on rJava/JRI instead of this Rserve-Extension.

D.1. Installation/Configuration

To use this extension you will need this extension, NetLogo 5.x, R (>=2.12), Java (>=1.5) and Rserve package >= 0.6-5 (lower versions are untested).

All you have to do is to install NetLogo, R and the Rserve package (available on CRAN) from within R.

D.2. Usage

First, start the Rserve TCP/IP server (Note that it is possible to start the Rserve server in debug mode.). Make sure that no firewall is blocking the port. For information how to do this on your operation system please see the Rserve documentation. If you have not changed the default settings and run it on the same machine as your NetLogo program, the server will run on "localhost" on port 6311.

To use the extension in your model, add a line to the top of your Procedures Tab:

```
extensions [rserve]
```

If your model already uses other extensions, then it already has an `extensions` line in it, so just add `rserve` to the list.

For more information on using NetLogo extensions, see the Extensions Guide.

For examples of the usage of the Rserve-Extension, see the folder *examples* in the folder of this extension.

D.3. Some Tips

Plotting If the server supports plotting, you can open plot devices. But, depending on your operation system, you have to give some cpu time to R by executing `rserve:eval "Sys.sleep(0.01)"` with a forever button. Otherwise the plot frame will be locked. Plots from packages such as ggplot2 are not supported by Rserve and are therefore not available.

Load and Save data from/into file It is possible to load and save data from file directly in R via `rserve:eval "dataname <- read.table('<path to file>')"` and `rserve:eval "write.table(dataname, file='<filename>')"`, respectively. Please note that if you use a remote connection to R, the file has to be on the server and the Rserve server must support file operations.

Data.frame with vector in cells Normally, a data.frame cell contains only a single value. Each column is represented as a vector and if you put a vector of vectors to a data.frame, it will be split into several columns. With the R-Extension it is possible to put a vector into a data.frame cell, when you assign a NetLogo list to a column which contains nested NetLogo lists for each row. If you want, for example, to use write.table on this data.frame, you have to mark this column as `class="AsIs"`. You can do this by using the `I(x)`-function.
Example: If the column of interest has the name `col1` of the data.frame `df1` you can execute `rserve:eval "df1$col1 <- I(df1$col1)"`. Call `help(I)` from within an R terminal for further details.

Load an R-Script Furthermore, you can define functions in an R-Script, load it, and use the functions. Load R-files via `rserve:eval "source('<path to r-file>')"`. Please note that if you use a remote connection to R, the file has to be on the server and the Rserve server must support file operations.

Load a Package It is also possible to load R packages via `rserve:eval library(<name of package>)`. Please note that the library must be installed in the R used by Rserve.

R Environments All calls of `rserve:init` will create a new local environment. This should enable you to use the extension in BehaviorSpace experiments. But, as the Rserve developer mentioned on their project website, this concept does not work on Windows operation systems. Furthermore, on Windows, it is not possible to connect more than one NetLogo process to Rserve in parallel. You have to close one connection first before you can establish another. Otherwise, your NetLogo will crash. On Linux, using several connections in parallel works without problems. Currently, there are no experiences on Mac OS. Until you close the connection or execute rserve:clear, all R variables assigned in this session will be available like you would use R from the command line.

D.4. Primitives

D.4.1. rserve:clear

```
rserve:clear
```

It clears the R-Workspace. All variables in R will be deleted. It evaluates the R command `rm(list=ls())` and `rm(list=ls(nl.env))`. Therefore, it deletes variables created in global as well as local environment (see R Environments for details about environments). It is always a good idea to add this command to your `setup` procedure under your `clear-all` call.

```
;; clear the R workspace
rserve:clear
```

D.4.2. rserve:close

```
rserve:close
```

It closes the connection to Rserve server, if there is one.

```
;; close the connection/disconnect
rserve:close
```

D.4.3. rserve:eval

```
rserve:eval R-command
```

It evaluates the submitted R command. The R command should not return a value.

```
;; creates a new vector in R with a sequence from 1 to 10
rserve:eval "x <- seq(1,10)"
show rserve:get "x"
```

D.4.4. rserve:get

```
rserve:get R-command
```

Reports the return value of the submitted R command. Return type can be a string, number, boolean, NetLogo list or a NetLogo list of lists. R lists will be converted into a NetLogo list. If the R list itself contains further lists, it will be converted into a NetLogo list with nested NetLogo lists. Lists containing values of different data types are also supported (e.g., mixed strings, numbers and booleans/logicals). Data.frames will be converted into a NetLogo list with nested list for each column, but the column names will be lost (same for named R lists). R matrices can be received, but they are converted into one NetLogo list. `NULL` and `NA` values are not converted and will throw an error, because NetLogo has no corresponding value.

```
;; returns a list with 10 variables
show rserve:get "rnorm(10)"
```

D.4.5. rserve:init

```
rserve:init port host <username> <password>
```

Initializes a connection to an Rserve server. There can be only one connection at a time. Close the active connection first (via `rserve:close`) before initializing another. The first argument is the port number. The default port of Rserve is 6311. The second argument is the host name. If you run the server locally, i.e., on the same machine as your NetLogo, than it is `"localhost"`. The username and password are optional parameters. If the server you try to connect needs authentication, you can submit the required login information with these two last parameters. Do not forget parentheses around the whole primitive if you use these two optional parameters. Please note that Windows OS does not support more than one connection at a time, i.e., you cannot connect two NetLogo models at the same time. See Rserve documentation for details.

```
;; example for initialization without login
rserve:init 6311 "localhost"
;; example for initialization with login
(rserve:init 6311 "localhost" "myname" "secret")
```

D.4.6. rserve:isConnected

```
rserve:isConnected
```

Reports if there is a connection established.

```
;; is there a connection to an Rserve serve established?
print rserve:isConnected
```

D.4.7. rserve:put

```
rserve:put name value
```

Creates a new variable in R with the name name. The value can be a string, number, boolean or list. NetLogo lists are converted to R vectors, if all entries are of the same data type. If a NetLogo list contains different data types (mixed strings, numbers or booleans), it will be converted into an R list. If a NetLogo list contains other/nested NetLogo lists it will be converted into an R list and the nested lists are handled by the same rule (vectors if all items are of the same data type, ...).

```
;; creates an R variable "testvar" with the size of turtle 0
rserve:put "testvar" [size] of turtle 0
show rserve:get "testvar"
```

D.4.8. rserve:putagent

```
rserve:putagent name agent|agentset variable
(rserve:putagent name agent|agentset variable1 variabl2 ...)
```

Creates a new named list in R with the name `name`. `Variable` is repeatable and can be every variable of the agent or agentset. Names of the elements of the R list will be the same as the names of the agent variables. Turtles will be assigned in ascending order of their `who` variable. Patches will be assigned in lines from upper left to lower right. Since the arguments of this primitive are repeatable, do not forget the parentheses around the statement.

```
;; creates an R-list "agentlist1" with the size and the id of
   turtles, do not forget the parentheses
(rserve:putagent "agentlist1" turtles "size" "who")
show rserve:get "agentlist1$who"
;; creates an R-list "agentlist2" with the pcolor, pxcor and
   pycor of patches
(rserve:putagent "agentlist2" patches "pcolor" "pxcor" "pycor")
show rserve:get "agentlist2$pcolor"
```

D.4.9. rserve:putagentdf

```
rserve:putagentdf name agent|agentset variable
(rserve:putagentdf name agent|agentset variable1 variable2 ...)
```

Same as `rserve:putagent` but creates an R data.frame instead of a list. Please read the notes about data.frames, if one of your agent variables contains NetLogo lists.

```
;; creates an R-list "agentlist2" with the pcolor, pxcor and
   pycor of patches, do not forget the parentheses
(rserve:putagentdf "df1" patches "pcolor" "pxcor" "pycor")
show rserve:get "class(df1)"
```

D.4.10. rserve:putdataframe

```
rserve:putdataframe name varname value
(rserve:putdataframe name varname1 value1 varname2 value2 ... )
```

Same as `rserve:putnamedlist` but creates an R data.frame instead of a list. If you send more than one list to NetLogo and the lists are of different length, the smaller ones will be filled with `NA` values. If you send nested NetLogo lists (e.g., of type: `[ [ ] [ ] ... ]`) to one column please read the notes about data.frames with vectors in cells.

```
;; creates an R-list "agentlist2" with the pcolor, pxcor and
   pycor of patches, do not forget the parentheses
(rserve:putdataframe "df1" "v1" [12 13 14 15 16] "v2" ["foo1"
   "foo2" "foo3" "foo4" "foo5"] "v3" [1.1 2.2 3.3 4.4 5.5])
show rserve:get "df1$v3"
```

D.4.11. rserve:putlist

```
rserve:putlist name value
(rserve:putlist name value1 value2 ... )
```

Creates a new list in R with the name `name`. `Value` is repeatable and can be a number, boolean or list. Each `value` will get the name of its position (1, 2, 3,...). Since the arguments of this primitive are repeatable, do not forget the parentheses around the statement.

```
;; creates an R-list "list1", do not forget the parentheses
(rserve:putlist "list1" 25.5 [25 43 32 53] "testvalue" [44.3
   32.32 321.2 4.2])
show rserve:get "class(list1)"
show rserve:get "list1[[1]]"
show rserve:get "list1$'0'"
show rserve:get "list1[[2]]"
```

D.4.12. rserve:putnamedlist

```
rserve:putnamedlist name varname value
(rserve:putnamedlist name varname1 value1 varname2 value2 ... )
```

Creates a new named list in R with the name `name`. `Value` is repeatable and can be a number, boolean or list. Each variable will get the name `varname`. Since the arguments of this primitive are repeatable, do not forget to put the statement into parentheses.

```
;; creates an R-list "list1" , do not forget the parentheses
(rserve:putnamedlist "list1" "v1" 25.5 "v2" [25 43 32 53] "v3"
   "testvalue" "v4" [44.3 32.32 321.2 4.2])
show rserve:get "class(list1)"
show rserve:get "list1[[1]]"
show rserve:get "list1$v1"
```

D.4.13. rserve:setSendBufferSize

```
rserve:setSendBufferSize
```

Primitive to set the maximum buffer size for submissions (in bytes, min=32k, max=1GB). See Rserve documentation for details.

```
;; change maximum buffer size print
rserve:setSendBufferSize 2052
```

APPENDIX E

Quickstart Guide RNetLogo

E.1. Getting RNetLogo

Prerequisites To use RNetLogo you have to install Gnu R (currently tested with 2.12.x and 2.13.x), NetLogo (currently tested with 4.1.x and 5.0beta3 and 5.0beta4) and the rJava package for R (and maybe a Java Runtime Environment). Sources of dependent software:

1. Gnu R: http://cran.r-project.org/
2. rJava package for R: either install it via `install.packages("rJava")` or download it from http://cran.r-project.org/web/packages/rJava/index.html or http://www.rforge.net/rJava/ and install it manually via `install.packages("<path to rJava>",repos=NULL)`. It can be necessary to setup Java support for R via `R CMD javareconf`. See documentation of rJava and mailing list for details. Make sure that the rJava package works properly. Try to load rJava in an R session via `library(rJava)` and run `.jinit()` to test the initialization.
3. Java Runtime Environment (JRE): e.g., Oracle JRE http://java.com/. It can be necessary to set an environment variable *JAVA_HOME* to the jre directory such that R can find the JRE.
4. NetLogo: http://ccl.northwestern.edu/netlogo/download.shtml

Installation of RNetLogo RNetLogo can be installed like any other R package. Either you use the automatic installation from CRAN by typing `install.packages("RNetLogo")` into an R shell or you download the package manually from http://cran.r-project.org/web/packages/RNetLogo/index.html and type `install.packages("<path to RNetLogo>", repos=NULL)`.

E.2. Kick-Starting RNetLogo

1. Start R
2. Load the Package:

```
library(RNetLogo)
```

3. Start NetLogo session:

```
NLStart("C:/Program Files/NetLogo 4.1.2", gui=TRUE,
   nl.version=4)
```

4. Load a model:

```
NLLoadModel("C:/Program Files/NetLogo 4.1.2/models/Sample
   Models/Earth Science/Fire.nlogo")
```

5. Submit a command:

```
NLCommand("setup")
NLDoCommand(10, "go")
```

6. Get a value:

```
br.trees <- NLReport("burned-trees")
print(br.trees)
```

E.3. Manuals and Tutorials to RNetLogo

The functions of RNetLogo are documented on the manual pages (type `??RNetLogo` or/and `help(<function name>)`). Furthermore, there is a tutorial (*<RNetLogo installation folder>/tutorial/tutorial.pdf*) which comes with the RNetLogo package and gives an introduction to the different functions and shows some application examples. Furthermore, there is a folder *examples/code_samples* in the RNetLogo package, which gives a small usage example for all functions.

APPENDIX F

RNetLogo Manual

`RNetLogo-Package` *Provides an interface to the agent-based modelling platform NetLogo*

Description

Interface to use and access Wilensky's NetLogo (Wilensky 1999) from R (R Core Team 2013) using either headless (no GUI) or interactive GUI mode. Provides functions to load models, execute commands, and get values from reporters. Mostly analogous to the NetLogo Mathematica Link http://ccl.northwestern.edu/netlogo/docs/mathematica.html.

Details

Package:	RNetLogo
Type:	Package
Version:	0.9-6
Date:	2013-04-24
License:	GNU GPL v2
LazyLoad:	yes

Start by creating a NetLogo instance by using `NLStart`. Then load a model with the function `NLLoadModel` and then use commands and reporters to do what you like.

It is possible to use NetLogo 3D. Just set the `is3d` argument in `NLStart` to `TRUE`. This functionality is experimental. All RNetLogo functions should work in NetLogo 3D as they do in conventional 2D NetLogo except `NLSetPatches`, which is not implemented to work with NetLogo 3D properly. `NLSetPatchSet` delivers a similar functionality usable also with NetLogo 3D but uses a data.frame instead of a matrix.

Note for MAC users: If you want to run RNetLogo in headless mode (without GUI, i.e., setting argument `gui=FALSE` in `NLStart`) you have to disable AWT before loading the package. Just execute `Sys.setenv(NOAWT=1)` before executing `library(RNetLogo)`. If you want to run RNetLogo in GUI mode you have to start it from the JGR application (see http://cran.r-project.org/web/packages/JGR/index.html and the note at http://groups.yahoo.com/group/netlogo-users/message/14817). It can be necessary to run `Sys.setenv(NOAWT=1)` before loading the JGR package and run `Sys.unsetenv("NOAWT")` before starting JGR via `JGR()`.

Note for Linux users: If you want to run RNetLogo in GUI mode you should start RNetLogo from JGR (see http://cran.r-project.org/web/packages/JGR/index.html).

Note for Windows 32-bit users: Starting RNetLogo (in GUI mode) on 32-bit Windows (not 64-bit Windows running in 32-bit mode) may fail in R version 2.15.2 and 2.15.3 (see description here: https://stat.ethz.ch/pipermail/r-devel/2013-January/065576.html). The reason could be the increased C stack size in 2.15.2 and 2.15.3. If you execute `Cstack_info()` you can see how large the C stack size is. The problem seems to be resolved with 3.0.0. A workaround is to use R 2.15.1 or 3.x or to start RNetLogo from JGR (see http://cran.r-project.org/web/packages/JGR/index.html) or RStudio (see http://www.rstudio.com/).

If you want to increase the Java Heap Space and set other parameters of the Java Virtual Machine (JVM) see notes at `NLStart`.

See the paper located in folder *tutorial* in the installation path of the package for an introduction. Example codes for all functions can be found in the folder *examples* in the installation path of the package. For performance notes see Appendix G and for an introduction how to run RNetLogo in parallel on multicore computers or clusters/grids see Appendix H.

References

For NetLogo see http://ccl.northwestern.edu/netlogo. For R-Extension for NetLogo see http://r-ext.sourceforge.net/. For Rserve-Extension for NetLogo see http://rserve-ext.sourceforge.net/. The RNetLogo package is analogous to (and inspired by) the NetLogo Mathematica Link http://ccl.northwestern.edu/netlogo/docs/mathematica.html. Wilensky, U. (1999) NetLogo. http://ccl.northwestern.edu/netlogo/. Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL. R Core Team (2013) R: A Language and Environment for Statistical. R Foundation for Statistical Computing.Vienna, Austria. http://www.R-project.org.

See Also

`NLStart`, `NLLoadModel`, `NLQuit`, `rJava package`

Examples

```
## Not run:
library(RNetLogo)
nl.path <- "C:/Program Files/NetLogo 5.0.4"
```

```
NLStart(nl.path, nl.version=5)
model.path <- "/models/Sample Models/Earth
   Science/Fire.nlogo"
NLLoadModel(paste(nl.path,model.path,sep=""))
NLCommand("setup")
NLDoCommand(10, "go")
burned <- NLReport("burned-trees")
print(burned)
NLQuit()

## End(Not run)
```

`NLCommand` *Executes a command in the referenced NetLogo instance.*

Description

`NLCommand` executes a NetLogo command (submitted as a string) in the (submitted) NetLogo instance.

Usage

```
NLCommand(..., nl.obj=NULL)
```

Arguments

`...`	An undefined number of strings with the NetLogo command(s) to be executed. Vectors, lists and data.frames will be represented as NetLogo lists. To set a NetLogo list you can write `'set mylist',c(1,2,3)` if the current NetLogo model knows a list named `mylist`. Furthermore, you can execute multiple commands in series, e.g., `'setup','go'`
`nl.obj`	(optional) A string identifying a reference to a NetLogo instance created with `NLStart`.

Details

The command can be anything which can be submitted from the NetLogo Command Center. A command has no return value! If you want to return a value from NetLogo use `NLReport` and other report functions.

Value

No return value.

See Also

NLDoCommand, NLDoCommandWhile, NLReport

Examples

```
## Not run:
NLStart("C:/Program Files/NetLogo 5.0.4")
NLCommand("create-turtles 10")

## End(Not run)
```

NLDfToList — *Transforms a data.frame into a NetLogo list or multiple NetLogo lists (one for each column of the data.frame).*

Description

NLDfToList pushes the values of a data.frame into NetLogo lists. The column names of the data.frame are used as names for the NetLogo lists (but the lists must already exist in the current NetLogo model).

Usage

```
NLDfToList(in.data.frame, nl.obj=NULL)
```

Arguments

in.data.frame	The data.frame to fill the NetLogo lists.
nl.obj	(optional) A string identifying a reference to a NetLogo instance created with NLStart.

Details

Remember: There must be lists in the NetLogo model with the names of the columns of the submitted data.frame.

Value

No return value.

See Also

NLDoCommand, NLDoCommandWhile, NLReport

Examples

```
## Not run:
NLStart("C:/Program Files/NetLogo 5.0.4")
df1 <- data.frame(x=c(1,2,3,4),y=c(5,6,7,8))
# the current NetLogo model must have two variables ('x' and
   'y')
# add the variables
NLSourceFromString("globals [x y]", append.model=FALSE)
# set the variables to the data.frame
NLDfToList(df1)

## End(Not run)
```

`NLDoCommand` *Repeats execution of a command in the referenced NetLogo instance for a defined number of times.*

Description

`NLDoCommand` executes a NetLogo command (submitted as a string) in the submitted NetLogo instance more than one time. It works like `NLCommand`.

Usage

```
NLDoCommand(iterations, ..., nl.obj=NULL)
```

Arguments

`iterations`	An integer defining the number of times the command is executed.
`...`	An undefined number of string(s) with the NetLogo command(s) to be executed. See `NLCommand` for details.
`nl.obj`	(optional) A string identifying a reference to a NetLogo instance created with `NLStart`.

Details

This function is used to execute a command more than one time. It is usually used to call a procedure (e.g., `go`) for a defined number of times.

Value

No return value.

See Also

NLCommand, NLDoCommandWhile, NLReport

Examples

```
## Not run:
nl.path <- "C:/Program Files/NetLogo 5.0.4"
NLStart(nl.path)
model.path <- "/models/Sample Models/Earth
   Science/Fire.nlogo"
NLLoadModel(paste(nl.path,model.path,sep=""))
NLCommand("setup")
NLDoCommand(10, "go")

## End(Not run)
```

NLDoCommandWhile *Repeats a command in the referenced NetLogo instance while a reporter returns* TRUE.

Description

NLDoCommandWhile function executes a NetLogo command (submitted as a string) in the submitted NetLogo instance more than one time. It works like NLCommand but will be repeated as long as the reporter returns TRUE.

Usage

```
NLDoCommandWhile(condition, ..., max.minutes=10, nl.obj=NULL)
```

Arguments

condition	A string with a NetLogo conditional reporter.
...	An undefined number of string(s) with the NetLogo command(s) to be executed. See NLCommand for details.
max.minutes	(optional) If max.minutes > 0 the execution stops after the defined number of minutes (with an error). By default, all executions are stopped after 10 minutes, to prevent the execution of endless loops. If you need more time, increase the value. If you are sure what you do, you can set this value to 0. Then it will run while the condition is true (i.e., endlessly when the condition is never met. In GUI mode, you can press *Tools -> Halt* in the NetLogo menu to interrupt a running process.). This can speed up the execution, because the time checking is not applied in this case.
nl.obj	(optional) A string identifying a reference to a NetLogo instance created with NLStart.

Details

This function is used to execute a command for more than one time. It can be used, for example, to run a simulation (by calling `go`) while a variable is below some limit.

The condition is evaluated before the submitted commands are executed. If the condition is `FALSE` at the first evaluation, the commands will never be executed.

Attention: Make sure that the condition switches from `TRUE` to `FALSE` sometime, otherwise you will run an endless loop (which is stopped after 10 minutes by default, see argument `max.minutes`).

Value

No return value.

See Also

`NLCommand`, `NLDoCommandWhile`, `NLReport`

Examples

```
## Not run:
nl.path <- "C:/Program Files/NetLogo 5.0.4"
NLStart(nl.path)
model.path <- "/models/Sample Models/Earth
   Science/Fire.nlogo"
NLLoadModel(paste(nl.path,model.path,sep=""))
NLCommand("setup")
NLDoCommandWhile("burned-trees < 500", "go")

## End(Not run)
```

`NLDoReport` — *Repeats a command and a reporter in the referenced NetLogo instance for a defined number of times.*

Description

`NLDoReport` executes a NetLogo command (submitted as a string) in the NetLogo instance for more than one time, and executes the reporter after each iteration. It works like a combination of `NLReport` and `NLDoCommand`.

Usage

```
NLDoReport(iterations, command, reporter,
   as.data.frame=FALSE, df.col.names=NULL, nl.obj=NULL)
```

Arguments

`iterations`	An integer defining how many times the command is repeated.
`command`	A string with the NetLogo command to be executed.
`reporter`	A string containing a NetLogo reporter. This argument can also be an R vector containing multiple strings with different NetLogo reporters (separated by commas), like `c("count patches", "count turtles")`. (A similar effect can be reached by using a NetLogo reporter returning a NetLogo list, like `"(list count patches count agents)"` as a single string argument. But the result will not be an R list with nested R lists but an R list with nested R vectors because NetLogo lists are converted to R vectors.)
`as.data.frame`	(optional) If `TRUE` the function will return a data.frame instead of a list. Default is `FALSE`, which returns a list.
`df.col.names`	(optional) If `as.data.frame=TRUE`, it contains the names of the columns of the returned data.frame. The argument is a vector containing the names as strings in the same order as the submitted reporters.
`nl.obj`	(optional) A string identifying a reference to a NetLogo instance created with `NLStart`.

Details

This function is used to execute a command more than one time and reports a value or a number of values after each iteration. It is often used to call a procedure (e.g., `go`) for a defined number of times and to save the value of a state variable each time.

Value

A list/nested list or data.frame with the value(s) of the reporter after each execution of the command.

See Also

`NLDoCommand`, `NLReport`, `NLDoReportWhile`

Examples

```
## Not run:
nl.path <- "C:/Program Files/NetLogo 5.0.4"
NLStart(nl.path)
model.path <- "/models/Sample Models/Earth
   Science/Fire.nlogo"
NLLoadModel(paste(nl.path,model.path,sep=""))
NLCommand("setup")
```

```
burned10 <- NLDoReport(10, "go", "burned-trees")
initburned10 <- NLDoReport(10, "go",
   c("initial-trees","burned-trees"), as.data.frame=TRUE,
   df.col.names=c("initial","burned"))
str(initburned10)

## End(Not run)
```

`NLDoReportWhile` *Repeats execution of a command and a reporter in the referenced NetLogo instance while a conditional reporter returns* `TRUE`.

Description

`NLDoReportWhile` function executes a NetLogo command (submitted as a string) more than one time and executes the reporter after each iteration. It works like `NLDoReport` but will be repeated while the conditional reporter returns `TRUE`.

Usage

```
NLDoReportWhile(condition, command, reporter,
   as.data.frame=FALSE, df.col.names=NULL, max.minutes=10,
   nl.obj=NULL)
```

Arguments

`condition` A string with a NetLogo conditional reporter.

`command` A string with the NetLogo command to be executed.

`reporter` A string containing a NetLogo reporter. This argument can also be an R vector containing multiple strings with different NetLogo reporters (separated by commas), like `c("count patches", "count turtles")`. (A similar effect can be reached by using a NetLogo reporter returning a NetLogo list, like `"(list count patches count agents)"` as a single string argument. But the result will not be an R list with nested R lists but an R list with nested R vectors because NetLogo lists are converted to R vectors.)

`as.data.frame` (optional) If `TRUE` the function will return a data.frame instead a list. Default is `FALSE` which returns a list.

`df.col.names` (optional) If `as.data.frame=TRUE`, it defines the names of the columns of the returned data.frame. The argument is a vector containing the names as strings in the same order as the reporters.

`max.minutes` (optional) If `max.minutes > 0` the execution stops after the defined number of minutes (with an error and no return value). By default, all executions are stopped after 10 minutes, to prevent the execution of endless loops. If you need more time, increase the value. If you are sure what you do, you can set this value to 0. Then, it will run while the condition is true (i.e., endlessly when the condition is never met. In GUI mode, you can press *Tools -> Halt* in the NetLogo menu to interrupt a running process.). This can speed up the execution, because the time checking is not applied in this case.

`nl.obj` (optional) A string identifying a reference to a NetLogo instance created with `NLStart`.

Details

This function executes a command more than one time and reports a value or a number of values after each iteration. It is usually used to call a procedure (e.g., `go`) while a variable is below a boundary value and to save the value of a state variable each time. Attention: Make sure that the condition switches from `TRUE` to `FALSE` sometime, otherwise you will run an endless loop (which is stopped after 10 minutes by default, see argument `max.minutes`).

Value

A list/nested list with the value(s) of the reporter after each execution of the command.

See Also

`NLDoCommandWhile`, `NLReport`, `NLDoReport`

Examples

```
## Not run:
nl.path <- "C:/Program Files/NetLogo 5.0.4"
NLStart(nl.path)
model.path <- "/models/Sample Models/Earth
   Science/Fire.nlogo"
NLLoadModel(paste(nl.path,model.path,sep=""))
NLCommand("setup")
burnedLower2200 <- NLDoReportWhile("burned-trees < 2200",
   "go", "burned-trees")
str(burnedLower2200)

## End(Not run)
```

NLGetAgentSet — *Reports variable value(s) of one or more agent(s) as a data.frame (optional as a list or vector)*

Description

NLGetAgentSet is an easy way to access variable value(s) of one or more agent(s) (in a sorted way) by specifying the name of the agent or the name of an agentset containing the agents. An agent is a turtle, breed, patch, or link. An agentset is a collection of agents.

Usage

```
NLGetAgentSet(agent.var, agentset, as.data.frame=TRUE,
   agents.by.row=FALSE, as.vector=FALSE, nl.obj=NULL)
```

Arguments

agent.var	A string or vector/list of strings with the variable names of the agent(s).
agentset	A string specifying the agent or agentset to be queried.
as.data.frame	(optional) If TRUE (default) the function will return a data.frame with a column for each agent.var and a row for each agent. The column names are taken from the names of the agent.var argument. If FALSE the function will return a list instead of a data.frame (little bit faster when not using agents.by.row=TRUE).
agents.by.row	(optional) This argument has an effect only in combination with as.data.frame=FALSE, i.e. when a list is returned. If agents.by.row=FALSE (default) the returned list contains one list element for each agent.var. Each list element contains a vector with the values of the different agents (agentset). If agents.by.row=TRUE the returned list contains one list element for each agent. Each list element contains a vector with the values of the different requested agent variables (agent.var). Attention: agents.by.row=TRUE makes the function very slow, especially when many agents are requested.
as.vector	(optional) Set this argument to TRUE for getting the result as a simple vector in case of requesting only one agent variable. This is the fastest way to access one agent variable. It does not make sense to set this variable to TRUE together with as.data.frame=TRUE, but as.vector is processed first and will win the race if you accidentally set as.data.frame to TRUE as well. By default as.vector is FALSE.
nl.obj	(optional) A string identifying a reference to a NetLogo instance created with NLStart.

Details

It is possible to use all variables of an agent, which can be found in NetLogo's Agent Monitors. It is not possible to get values from different types of agents (i.e., turtles, patches, links) with one call of `NLGetAgentSet`. See Appendix G for performance details and notes on changes since version 0.9.3.

Value

Returns a data.frame (optionally a list) with the variable value(s) of an agent/agents of an agentset. One row for each agent and one column for each agent variable. The result is sorted in the same manner as using `sort agentset` in NetLogo, i.e., turtles are sorted by their `who` variable and patches from upper left to lower right.

To get the same result as with default settings until RNetLogo version 0.9.2 use: `as.data.frame=FALSE` and `agents.by.row=TRUE`.

See Also

`NLReport`, `NLGetPatches`, `NLGetGraph`

Examples

```
## Not run:
nl.path <- "C:/Program Files/NetLogo 5.0.4"
NLStart(nl.path)
# NLLoadModel(...)
NLCommand("create-turtles 10")

colors <- NLGetAgentSet(c("who","xcor","ycor","color"),
   "turtles with [who < 5]")
str(colors)

# or as a list (slightly faster):
colors.list <- NLGetAgentSet(c("who","xcor","ycor","color"),
   "turtles with [who < 5]", as.data.frame=FALSE)
str(colors.list)

# or as a list with one list element for each agent
# (very slow!, not recommended especially for large
   agentsets)
colors.list2 <-
   NLGetAgentSet(c("who","xcor","ycor","color"), "turtles
   with [who < 5]", as.data.frame=FALSE, agents.by.row=TRUE)
str(colors.list2)

# getting the ends of links is a little bit more tricky,
   because they store only the
# reference to the turtles and turtles cannot directly be
   requested.
```

```
# A way to go is:
# create some links
NLCommand("ask turtles [ create-links-with n-of 2 other
   turtles ]")
link.test <- NLGetAgentSet(c("[who] of end1","[who] of
   end2"),"links")
str(link.test)

## End(Not run)
```

`NLGetGraph` *Captures a network.*

Description

`NLGetGraph` converts a set of NetLogo Link agents into an igraph graph object (see package igraph for details on graph objects).

Usage

```
NLGetGraph(link.agentset="links", nl.obj=NULL)
```

Arguments

`link.agentset`
: (optional) A string defining an agentset of NetLogo Links. Default is `"links"`, which are all links.

`nl.obj`
: (optional) A string identifying a reference to a NetLogo instance created with `NLStart`.

Details

Saves a link network in a graph object of package igraph for network analysis.

Value

Returns a graph object of package igraph.

See Also

`NLGetAgentSet`

Examples

```
## Not run:
nl.path <- "C:/Program Files/NetLogo 5.0.4"
NLStart(nl.path)
model.path <-
"/models/Sample Models/Networks/Preferential
   Attachment.nlogo"
NLLoadModel(paste(nl.path,model.path,sep=""))
NLCommand("setup")
NLDoCommand(4, "go")
graph1 <- NLGetGraph()
plot(graph1, layout=layout.kamada.kawai,
   vertex.label=V(graph1)$name, vertex.shape="rectangle",
   vertex.size=20, asp=FALSE)

## End(Not run)
```

`NLGetPatches` *Reports the values of patch variables as a data.frame (optional as a list, matrix or simple vector)*

Description

`NLGetPatches` is an easy way to access variables of all patches (default) or of a subset of patches.

Usage

```
NLGetPatches(patch.var, patchset="patches", as.matrix=FALSE,
   as.data.frame=TRUE, patches.by.row=FALSE,
   as.vector=FALSE, nl.obj=NULL)
```

Arguments

`patch.var`	A string or vector/list of strings with the names of patch variables to report.
`patchset`	(optional) A string defining which patches to request. By default, values of all patches are returned.
`as.matrix`	(optional) If this variable is `TRUE` (default is `FALSE`), the function will return the result as a matrix representing the NetLogo world. (This option is only available if the argument `patchset` is not used, i.e., if you request all patches and there is only one patch variable, i.e., length of `patch.var` is 1.)

`as.data.frame`
: (optional) If `TRUE` (default) the function returns a data.frame with one column for each `patch.var` and one row for each patch. The column names are taken from the names of the `patch.var` argument. If `FALSE` the function will return a list instead of a data.frame (little bit faster, when not using `patches.by.row=TRUE`).

`patches.by.row`
: (optional) This argument has an effect only in combination with `as.data.frame=FALSE`, i.e., when a list is returned. If `patches.by.row=FALSE` (default) the returned list contains one list element for each `patch.var`. Each list element contains a vector with the values of the different patches (`patchset`). If `patches.by.row=TRUE` the returned list contains one list element for each patch. Each list element contains a vector with the values of the different requested patch variables (`patch.var`). Attention: `patches.by.row=TRUE` makes the function very slow, especially when many patches are requested.

`as.vector`
: (optional) Set this argument to `TRUE` for getting the result as a simple vector in case of requesting only one patch variable. This is the fastest way to access one patch variable. It does not make sense to set this variable to `TRUE` together with `as.data.frame=TRUE` or `as.matrix=TRUE`, but `as.vector` is processed first and will win the race if you accidentally set `as.data.frame` or `as.matrix` to `TRUE` as well. By default `as.vector` is `FALSE`.

`nl.obj`
: (optional) A string identifying a reference to a NetLogo instance created with `NLStart`.

Details

It is possible to use all the variables of a patch, which can be found in NetLogo's Agent Monitors. See Appendix G for performance details and notes on changes since version 0.9.3.

Value

Returns a data.frame (optionally a list) with the variable value(s) of a patch/patches of a patchset. One row for each patch and one column for each patch variable. The result is sorted (like using `sort patchset` in NetLogo), e.g., patches are sorted from upper left to lower right.

To get the same result as with default settings until RNetLogo version 0.9.2 use: `as.data.frame=FALSE` and `patches.by.row=TRUE`.

See Also

`NLReport`, `NLGetAgentSet`, `NLGetGraph`

Examples

```
## Not run:
nl.path <- "C:/Program Files/NetLogo 5.0.4"
NLStart(nl.path)
# NLLoadModel(...)

allpatches <- NLGetPatches(c("pxcor","pycor","pcolor"))
str(allpatches)

# only a subset of patches
subsetpatches <- NLGetPatches(c("pxcor","pycor","pcolor"),
   "patches with [pxcor < 5]")
str(subsetpatches)

# or as a list (slightly faster):
colors.list <- NLGetPatches(c("pxcor","pycor","pcolor"),
   "patches with [pxcor < 5]", as.data.frame=FALSE)
str(colors.list)

# or as a list with one list element for each patch
# (very slow!, not recommended especially for large
   patchsets)
colors.list2 <- NLGetPatches(c("pxcor","pycor","pcolor"),
   "patches with [pxcor < 5]", as.data.frame=FALSE,
   patches.by.row=TRUE)
str(colors.list2)

## End(Not run)
```

`NLLoadModel` *Loads a model into the NetLogo instance.*

Description

`NLLoadModel` loads a model (**.nlogo* file) into the submitted NetLogo instance.

Usage

```
NLLoadModel(model.path, nl.obj=NULL)
```

Arguments

`model.path`	A string containing either the absolute path to the model file (**.nlogo* file) or a relative path to the model file starting from the NetLogo installation directory specified in `NLStart`.

`nl.obj` (optional) A string identifying a reference to a NetLogo instance created with `NLStart`.

Value

No return value.

See Also

`NLStart`, `NLQuit`

Examples

```
## Not run:
nl.path <- "C:/Program Files/NetLogo 5.0.4"
NLStart(nl.path)
model.path <- "/models/Sample Models/Earth
   Science/Fire.nlogo"
absolute.model.path <- paste(nl.path,model.path,sep="")
NLLoadModel(absolute.model.path)

relative.model.path <- "models/Sample Models/Earth
   Science/Fire.nlogo"
NLLoadModel(relative.model.path)

## End(Not run)
```

`NLQuit` *Quits a NetLogo instance.*

Description

Quits the NetLogo workspace and closes the GUI window (if started with GUI).

Usage

```
NLQuit(nl.obj=NULL, all=FALSE)
```

Arguments

`nl.obj` (optional) A string identifying a reference to the NetLogo instance defined in `nl.obj` of `NLStart`.

`all` (optional) A boolean variable: If `TRUE` all active instances of NetLogo created with `NLStart` are closed. Then, `nl.obj` argument is not used.

Value

No return value.

Warning

There is currently no way to kill a NetLogo instance with GUI completely. After executing `NLQuit` on a GUI instance, you cannot run `NLStart` again. You have to quit your R session first and start a new one. The reason is that NetLogo quits via `System.exit` (and has no functionality to quit all threads manually) but executing `System.exit` will terminate the whole JVM which will also terminate rJava and finally R. But there is a trick to run RNetLogo in GUI mode multiple times described in Appendix H. It can happen that some memory is not released although you have executed `NLQuit`, because shutting down the running JVM via rJava and unloading the required libraries is not possible. Therefore, it is a good idea to start a new R session if possible when you load a new model.

See Also

`NLStart`

Examples

```
## Not run:
nl.path <- "C:/Program Files/NetLogo 5.0.4"
NLStart(nl.path)
NLQuit()

## End(Not run)
```

`NLReport` *Reports a value or list of values*

Description

`NLReport` reports NetLogo data back to R.

Usage

```
NLReport(reporter, nl.obj=NULL)
```

Arguments

`reporter`	A string containing a NetLogo reporter. (Or a vector of strings.)
`nl.obj`	(optional) A string identifying a reference to a NetLogo instance created with `NLStart`.

Details

Every reporter (commands which return a value) that can be called in the NetLogo Command Center can be called with `NLReport`.

Value

A vector of length one if only one value is returned. Otherwise it is a list or, if necessary, a nested list with the reported values.

See Also

`NLDoReport`, `NLDoReportWhile`, `NLGetPatches`, `NLGetAgentSet`

Examples

```
## Not run:
nl.path <- "C:/Program Files/NetLogo 5.0.4"
NLStart(nl.path)
model.path <- "/models/Sample Models/Earth
   Science/Fire.nlogo"
NLLoadModel(paste(nl.path,model.path,sep=""))
NLCommand("setup")
NLDoCommand(10, "go")
noburned <- NLReport("burned-trees")
str(noburned)

## End(Not run)
```

`NLSetAgentSet` *Sets a variable of one or more agent(s) to value(s) in a data.frame or vector.*

Description

`NLSetAgentSet` is an easy way to set the variable value(s) of one or more agent(s) (by specifying the name of the agent or the name of an agentset containing the agents) to the value(s) of a data.frame or vector.

Usage

```
NLSetAgentSet(agentset, input, var.name=NULL, nl.obj=NULL)
```

Arguments

`agentset`	A string specifying the agent or agentset for which values should be changed.
`input`	A data.frame or vector. If a data.frame, it must have one column with the corresponding agent variable name for each agent variable to be set and one row for each agent. The rows have to be sorted in the order NetLogo is processing the agentset with `sort agentset` (e.g., turtles are sorted by their who value). If a vector, only one agent variable can be set and the name has to be given by the optional argument `var.name`.
`var.name`	If `input` is a simple vector instead of a data.frame it gives the name of the agent variable as a string which should be set with the values of the vector submitted in `input`. With a vector you can only set one agent variable at a time.
`nl.obj`	(optional) A string identifying a reference to a NetLogo instance created with `NLStart`.

Details

The agent variable values contained as columns in the input data.frame are changed. The columns of the data.frame have to be named exactly like the agent variables which should get the values. The rows have to be sorted as NetLogo would process the agentset using the `sort` reporter.

Value

No return value.

See Also

`NLSetPatches`, `NLGetAgentSet`, `NLGetGraph`, `NLDfToList`

Examples

```
## Not run:
nl.path <- "C:/Program Files/NetLogo 5.0.4"
NLStart(nl.path)
# NLLoadModel(...)
ag <- NLGetAgentSet(c("xcor","ycor"),"turtles")
ag2 <- data.frame(xcor=ag$xcor, ycor=ag$xcor)
NLSetAgentSet("turtles", ag2)

## End(Not run)
```

NLSetPatches	*Sets a variable of all patches in the NetLogo world to the values in a matrix.*

Description

`NLSetPatches` is an easy way to set the values of all patches to the values of a matrix.

Usage

```
NLSetPatches(patch.var, in.matrix, nl.obj=NULL)
```

Arguments

`patch.var`	The name of the patch variable to set.
`in.matrix`	A matrix that represents the NetLogo world (has the same dimensions).
`nl.obj`	(optional) A string identifying a reference to a NetLogo instance created with `NLStart`.

Details

The matrix must have the same x- and y-dimensions as the NetLogo world, indices beginning with (1,1). The upper-left cell (1,1) of the matrix represents the upper-left patch of the NetLogo world, no matter where the origin of the NetLogo world is set. This function is not available when running NetLogo 3D. Use `NLSetPatchSet` instead.

Value

No return value.

See Also

`NLReport`, `NLGetAgentSet`, `NLGetGraph`, `NLDfToList`

Examples

```
## Not run:
nl.path <- "C:/Program Files/NetLogo 5.0.4"
NLStart(nl.path)
m1 <- matrix(1:1089 , 33)
NLSetPatches("pcolor", m1)

## End(Not run)
```

NLSetPatchSet	*Sets the variable value of one or more patch(es) to value(s) in a data.frame.*

Description

NLSetPatchSet is an easy way to set the variable value of one or more patch(es) to the value(s) of a data.frame.

Usage

```
NLSetPatchSet(patch.var, input, nl.obj=NULL)
```

Arguments

patch.var	This argument gives the name of the patch variable as a string which should be set to the values of the third (for NetLogo 2D) or fourth column (for NetLogo 3D) of the data.frame submitted in input.
input	A data.frame with columns giving the coordinates of a patch and the values for the patch variable to be changed. For conventional 2D NetLogo there has to be a pxcor and a pycor column , for NetLogo 3D there has to be a pxcor, a pycor, and a pzcor column. The name of the column that contains the new values for the patch variable has to be equal to the argument patch.var.
nl.obj	(optional) A string identifying a reference to a NetLogo instance created with NLStart.

Details

This function is used to update one patch variable for patches identified by their pxcor, pycor (and pzcor in case of NetLogo 3D) values based on values given in a data.frame.

Value

No return value.

See Also

NLSetPatches, NLGetAgentSet, NLGetGraph, NLDfToList

Examples

```
## Not run:
nl.path <- "C:/Program Files/NetLogo 5.0.4"
NLStart(nl.path)
# NLLoadModel(...)

# for NetLogo 2D:
input <- NLGetPatches(c("pxcor","pycor","pcolor"))
str(input)
# for NetLogo 3D:
input <- NLGetPatches(c("pxcor","pycor","pzcor","pcolor"))
str(input)

input$pcolor <- floor(abs(rnorm(nrow(input))*100))
patch.var <- "pcolor"
NLSetPatchSet(patch.var, input)

## End(Not run)
```

NLSourceFromString

Creates or appends NetLogo code from R.

Description

`NLSourceFromString` is a way to create/append a NetLogo model's source code dynamically from R.

Usage

```
NLSourceFromString(..., append.model=TRUE, nl.obj=NULL)
```

Arguments

`...`	An undefined number of strings containing NetLogo model source code to be printed into the Procedures Tab. Line breaks within a string can be represented as `\n`.
`append.model`	(optional) Determines whether existing code in the Procedures Tab (i.e., a loaded model) will be appended by the new code or will be replaced. By default, all existing code will be appended.
`nl.obj`	(optional) A string identifying a reference to a NetLogo instance created with `NLStart`.

Details

This function only works with NetLogo instances with GUI. It does not work in headless mode.

Value

No return value.

See Also

`NLReport`, `NLGetAgentSet`, `NLGetGraph`, `NLDfToList`

Examples

```
## Not run:
nl.path <- "C:/Program Files/NetLogo 5.0.4"
NLStart(nl.path)
setup <- "to setup\n ca\n crt 10\nend \n"
go <- "to go\n ask turtles [\n set xcor random-xcor\n set
   ycor random-ycor\n ]\nend \n"
reporter1 <- "to-report noturtles\n report count turtles\n
   end \n"
NLSourceFromString(setup,go,reporter1, append.model=FALSE)
NLCommand("setup")
NLCommand("go")
noturtles <- NLReport("noturtles")
print(noturtles)

## End(Not run)
```

`NLStart` *Creates an instance of NetLogo*

Description

`NLStart` creates a new instance of NetLogo in either headless (without the Graphical User Interface) or GUI mode.

Usage

```
NLStart(nl.path, gui=TRUE, nl.obj=NULL, nl.version=5,
   is3d=FALSE)
```

Arguments

`nl.path`	An absolute path to your NetLogo installation (the folder where the *NetLogo.jar* is) starting from the root. On Windows, for example, something like "*C:/Program Files/NetLogo 5.0.4*".
`gui`	(optional) A boolean value: if `TRUE`, NetLogo will be started with GUI (only one instance with GUI can be created currently!). `FALSE` will start NetLogo in headless mode.
`nl.obj`	(optional) A string which is used to identify the created NetLogo instance reference internally (in `.rnetlogo` environment). To refer to this instance just use the same name in the other functions of this package. If `nl.obj=NULL` (default), the internal name to the reference is `_nl.intern_` and is not needed to be submitted to the other functions of this package. After using `NLQuit`, the identical name can be used again for a new instance.
`nl.version`	(optional) An integer value specifying the (major) version of NetLogo being started. Do not try to start a NetLogo version 4.x with `nl.version=5` and vice versa. It is not possible to mix NetLogo versions in one R session. Please use different R sessions if you want to start RNetLogo with version 4 and version 5. The default is `nl.version=5` for NetLogo 5.0. For NetLogo 4.1.x use `nl.version=4`. For NetLogo 4.0.x use `nl.version=40`. Please note that the data transformation is much faster in NetLogo 5.0 than in NetLogo 4.x. Therefore, using NetLogo 5 is strongly recommended. See Appendix G for performance details.
`is3d`	(optional) A boolean value: if `TRUE`, NetLogo 3D will be started. `FALSE` will start the conventional 2D NetLogo. This functionality is experimental. All RNetLogo functions should work in NetLogo 3D as they do in conventional 2D NetLogo except NLSetPatches, which is currently not implemented to work in NetLogo 3D properly.

Details

You can start multiple instances of NetLogo in headless mode and store each in another variable (using `nl.obj`) but it is not possible to start multiple instances in GUI mode. (It would result in an crash of R since there is no way to detach the Java Virtual Machine via rJava.) But there is a trick to run RNetLogo in GUI mode multiple times described in Appendix H.

Note for Mac OS users: If you want to run RNetLogo in headless mode (without GUI, i.e., setting argument `gui=FALSE`) you have to disable AWT before loading the package. Just execute `Sys.setenv(NOAWT=1)` before executing `library(RNetLogo)`. If you want to run RNetLogo in GUI mode you have to start it from the JGR application (see http://cran.r-project.org/web/packages/JGR/index.html and the note at http://groups.yahoo.com/group/netlogo-users/message/14817). It can be necessary to run `Sys.setenv(NOAWT=1)` before loading the JGR package and run `Sys.unsetenv("NOAWT")` before starting JGR via `JGR()`.

Note for Linux users: If you want to run RNetLogo in GUI mode you should start RNetLogo in the JGR application (see http://cran.r-project.org/web/packages/JGR/index.html).

Note for Windows 32-bit users: Starting RNetLogo (in GUI mode) on 32-bit Windows (not 64-bit Windows running in 32-bit mode) may fail in R version 2.15.2 and 2.15.3 (see description here: https://stat.ethz.ch/pipermail/r-devel/2013-January/065576.html). The reason could be the increased C stack size in 2.15.2 and 2.15.3. If you execute `Cstack_info()` you can see how large the C stack size is. The problem seems to be resolved with 3.0.0. A workaround is to use R 2.15.1 or 3.x or to start RNetLogo from JGR (see http://cran.r-project.org/web/packages/JGR/index.html) or RStudio (see http://www.rstudio.com/).

Avoid manually changing the working directory of R, because NetLogo needs to have the working directory pointed to its installation path. As the R working directory and the Java working directory depend on each other, changing the R working directory can result in unexpected behaviour of NetLogo. Therefore, you should use absolute paths for I/O processes in R instead of submitting `setwd(...)`. Note that the RNetLogo package changes the working directory automatically when loading NetLogo and changes back to the former working directory closing the last active instance of NetLogo with `NLQuit`.

As mentioned in `NLQuit`, it is currently not possible to quit NetLogo completely.

If you want to specify options for the underlying Java Virtual Machine (JVM), like increasing the Java Heap Space for large models, execute `options(java.parameters="...")` before loading the RNetLogo package with `library(RNetLogo)` or `require(RNetLogo)`. For increasing the Java Heap Space it can be `options(java.parameters="-Xmx1024m")`, for example. Use a vector of strings for setting multiple options, for example `options(java.parameters=c("-server","-Xmx1300m"))`. See also http://ccl.northwestern.edu/netlogo/docs/faq.html#howbig and http://permalink.gmane.org/gmane.comp.lang.r.rosuda.devel/1284.

See the directory *examples* in the installation directory of the package for example codes to all RNetLogo functions.

See the paper in directory *tutorial* in the installation directory of the package for a step-by-step usage tutorial.

See Appendix G for performance notes.

See Appendix Hon how to run RNetLogo on multiple processors/clusters in parallel.

Value

No return value.

Warning

It is not possible to run multiple instances of NetLogo in GUI mode! Closing NetLogo from the NetLogo Window is blocked, because it would quit the whole R process. To close the NetLogo call `NLQuit`. If you use the headless mode you should first load a model with `NLLoadModel` before executing other commands or reporters. In GUI

mode you can execute commands and reporters already with the initial empty model without loading a specific one.

See Also

NLQuit

Examples

```
## Not run:
library(RNetLogo)
nl.path <- "C:/Program Files/NetLogo 5.0.4"
NLStart(nl.path)
NLCommand("create-turtles 10")
noturtles <- NLReport("count turtles")
print(noturtles)

# create a second NetLogo instance in headless mode (=
   without GUI)
# stored in a variable
nlheadless1 <- "nlheadless1"
NLStart(nl.path, gui=F, nl.obj=nlheadless1)
model.path <- "/models/Sample Models/Earth
   Science/Fire.nlogo"
NLLoadModel(paste(nl.path,model.path,sep=""),
   nl.obj=nlheadless1)
NLCommand("setup", nl.obj=nlheadless1)
burned1 <- NLDoReport(20, "go", c("ticks","burned-trees"),
   as.data.frame=TRUE, df.col.names=c("tick","burned"),
   nl.obj=nlheadless1)
print(burned1)

# create a third NetLogo instance in headless mode (=
   without GUI)
# with explicit name of stored object
nlheadless2 <- "nlheadless2"
NLStart(nl.path, gui=F, nl.obj=nlheadless2)
model.path <- "/models/Sample Models/Earth
   Science/Fire.nlogo"
NLLoadModel(paste(nl.path,model.path,sep=""),
   nl.obj=nlheadless2)
NLCommand("setup", nl.obj=nlheadless2)
burned2 <- NLDoReport(10, "go", c("ticks","burned-trees"),
   as.data.frame=TRUE, df.col.names=c("tick","burned"),
   nl.obj=nlheadless2)
print(burned2)

## End(Not run)
```

APPENDIX G

Performance Notes to the RNetLogo Package

G.1. Abstract

RNetLogo is a flexible interface for NetLogo to R, however, it was noticed that the performance is slow for large datasets in some circumstances. Here, I show that these problems are solved for newer versions of RNetLogo/NetLogo, and give some specific hints for using RNetLogo in situations where runtime is critical. Specifically, I present the results of an execution time measurement study of `NLGetAgentSet` using NetLogo 4.0.5, 4.1.3, and 5.0 as well as RNetLogo 0.9.2 and 0.9.3. The results show that using NetLogo 5.0 is highly recommended since its transformation times/list operations are substantially faster than in older versions of NetLogo. Some further speed improvements (for `NLGetAgentSet` or `NLGetPatches`) can be achieved by using RNetLogo >= 0.9.3.

G.2. Preliminary Note

All tests and measurements have been performed on Windows XP Professional SP3 (32-bit) with a DELL Latitude D630 notebook with an Intel(R) Core(TM)2 Duo T9500 chipset with 2.60GHz and used RAM of 3.49 GB.

The simulations have been run only once. This does not deliver reliable performance measures, because system background processes can cover the real execution times. Normally, one would run simulations multiple times and take only the minimum execution times. But to get a rough impression of the dimensions, running one simulation should be sufficient for the purpose here.

G.3. Motivation

Some of the users of the RNetLogo package recognized that the data transfer using functions like the `NLGetAgentSet` was very slow on large datasets/numbers of agents. Moreover, the processing time increased non-linearly with an increasing number of datasets/agents. To make it possible to work also with large datasets/number of agents I systematically analysed the problem and identified some reasons. This paper documents the problem analysis as well as shows how to resolve the bottleneck. It also shows what has changed in RNetLogo 0.9.3

in the functions `NLGetAgentSet` and `NLGetPatches` especially regarding the return data types.

G.4. Changes in NLGetAgentSet and NLGetPatches

G.4.1. Until RNetLogo 0.9.2

The results shown in the following are produced using the Fireflies model from NetLogo's Model Library. This model is initialized with different numbers of flies (i.e., turtles). Afterwards the `NLGetAgentSet` function from the RNetLogo package is used to get variables from all flies/turtles from NetLogo into R.

With RNetLogo 0.9.2 there are two data type variants for the output of the `NLGetAgentSet` function available: an R list (default) and an R data.frame.

The structure of the list is as follows: In case of multiple requested agent variables: for each agent there is one list element. Each of these elements contains the requested agent variables in a vector. In case of just one requested agent variable: only a single vector with the values of the different agents instead of a list is returned.

For example (using RNetLogo 0.9.2):

```
R> #RNetLogo version:
R> print(vers)
R> nl.path <- "C:/Program Files/NetLogo 5.0"
R> NLStart(nl.path, nl.version=5, gui=F)
R> model.path <- "/models/Sample
   Models/Biology/Fireflies.nlogo"
R> NLLoadModel(paste(nl.path, model.path, sep=""))
R> NLCommand("set number 10")
R> NLCommand("setup")
```

```
R> #RNetLogo version:
R> print(vers)
```

Output:

```
$Version
[1] "0.9.2"
```

```
R> # for only one agent variable it is just a vector
R> test_t5 <- NLGetAgentSet("who", "turtles", as.data.frame=F)
```

```
R> str(test_t5)
```

Output:

```
 num [1:10] 0 1 2 3 4 5 6 7 8 9
```

```
R> is.list(test_t5)
```

Output:

```
[1] FALSE
```

```
R> # for more than one agent variable it is a list
R> # with one list element for each agent and a
R> # vector in each list element containing
R> # the values of the requested agents variables
R> test_t6 <- NLGetAgentSet(c("who", "xcor", "ycor"),
+ "turtles", as.data.frame=F)
```

```
R> str(test_t6)
```

Output:

```
List of 10
 $ : num [1:3] 0 -35.45 9.24
 $ : num [1:3] 1 -14.1 31.1
 $ : num [1:3] 2 -14.34 6.19
 $ : num [1:3] 3 18.5 -31.1
 $ : num [1:3] 4 -31.5 -26.3
 $ : num [1:3] 5 9.68 -21.5
 $ : num [1:3] 6 -24.9 -24
 $ : num [1:3] 7 -31.4 -18.8
 $ : num [1:3] 8 -34.4 -21.1
 $ : num [1:3] 9 -10.2 -27.9
```

```
R> is.list(test_t6)
```

Output:

```
[1] TRUE
```

If we request a data.frame, we have to set the argument `as.data.frame` to `TRUE` and should submit the names for the data.frame columns in the argument `df.col.names`.

If we request only one agent variable, again, just a single vector is returned. But if we request more than one agent variable a data.frame is constructed with the agent variables in the columns and the values of each agent in rows.

For example:

```
R> # for only one agent variable it is just a vector
R> test_t1 <- NLGetAgentSet("who", "turtles", as.data.frame=T,
+ df.col.names=c("who"))
```

```
R> str(test_t1)
```

Output:

```
 num [1:10] 0 1 2 3 4 5 6 7 8 9
```

```
R> is.data.frame(test_t1)
```

Output:

```
[1] FALSE
```

```
R> # for more than one agent variable it is a data.frame
R> # with agent variables in columns and
R> # one row for each agent
R> test_t2 <- NLGetAgentSet(c("who", "xcor", "ycor"),
   "turtles",
+ as.data.frame=T, df.col.names=c("who",
+ "xcor", "ycor"))
```

```
R> str(test_t2)
```

Output:

```
'data.frame': 10 obs. of 3 variables:
 $ who : num 0 1 2 3 4 5 6 7 8 9
 $ xcor: num -35.4 -14.1 -14.3 18.5 -31.5 ...
 $ ycor: num 9.24 31.08 6.19 -31.14 -26.29 ...
```

```
R> is.data.frame(test_t2)
```

Output:

```
[1] TRUE
```

This pattern is the same for the `NLGetPatches` function, since its functioning is equivalent to `NLGetAgentSet`.

In the manual of RNetLogo 0.9.2 it was mentioned that the data.frame variant is much faster when requesting more than one agent variable.

This is one of the reasons why I decided to change the default return value of `NLGetAgentSet`/ `NLGetPatches` in RNetLogo 0.9.3.

G.4.2. Since RNetLogo 0.9.3

Since RNetLogo 0.9.3 the argument `df.col.names` is not available anymore because the names of the requested agent variables are used as column names of the data.frame. This prevents mistakes in the order of variables and their names. If you want to replace these names just use R's `names` function.

Because the data.frame is the default return type now, it means that the function argument `as.data.frame` is `TRUE` by default. This was done because the data.frame is the standard data type in R. But if we change `as.data.frame` to `FALSE` we are not getting the data structure as known from RNetLogo 0.9.2 but a list where the list elements are the agents variables (instead of the agents) and these list elements contain vectors with the values for the agents. Therefore, it is very similar to the data structure of the data.frame. The list

elements are also named with the names of the agent variables. This new list structure is created substantially faster than the old one.

If you want to produce the default list structure known from RNetLogo 0.9.2 instead, you have to set a further argument called `agents.by.row` to `TRUE` in combination with setting `as.data.frame` to `FALSE`.

Since RNetLogo 0.9.3 you will always get the expected data type independent from the number of agent variables requested. In RNetLogo 0.9.2 we got a vector when we requested only one agent variable. Now, since RNetLogo 0.9.3, we are getting either a list or data.frame depending on what was requested. This new behaviour will prevent unexpected return types for single agent requests. You will now consequently get what you asked for.

There is also the option to get a simple vector when requesting only one agent variable. Therefore, we have to set the function argument `as.vector` to `TRUE`. The result is equivalent to the output of RNetLogo 0.9.2 when requesting one agent variable independent from the requested output data type (see above).

Maybe it is more clear to see it with an example (using RNetLogo 0.9.3):

```
R> library(RNetLogo)
R> vers <- (packageDescription("RNetLogo")["Version"])
R> nl.path <- "C:/Program Files/NetLogo 5.0"
R> NLStart(nl.path, nl.version=5, gui=F)
R> model.path <- "/models/Sample
   Models/Biology/Fireflies.nlogo"
R> NLLoadModel(paste(nl.path, model.path, sep=""))
R> NLCommand("set number 10")
R> NLCommand("setup")
```

```
R> #RNetLogo version:
R> print(vers)
```

Output:

```
$Version
[1] "0.9.3"
```

```
R> # the equivalent to the RNetLogo 0.9.2 default:
R> # the list variant for only one agent variable
R> # (but now as list not as vector)
R> test_t5 <- NLGetAgentSet("who", "turtles", as.data.frame=F,
   agents.by.row=T)
```

```
R> str(test_t5)
```

Output:

```
List of 10
 $ : num 0
 $ : num 1
 $ : num 2
 $ : num 3
 $ : num 4
 $ : num 5
 $ : num 6
 $ : num 7
 $ : num 8
 $ : num 9
```

```
R> is.list(test_t5)
```

Output:

```
[1] TRUE
```

```
R> # for more than one agent variable
R> test_t6 <- NLGetAgentSet(c("who", "xcor", "ycor"),
   "turtles",
+ as.data.frame=F, agents.by.row=T)
```

```
R> str(test_t6)
```

Output:

```
List of 10
 $ : num [1:3] 0 13.1 -18
 $ : num [1:3] 1 -30.6 15.6
 $ : num [1:3] 2 16.7 29.8
 $ : num [1:3] 3 -15.5 31.6
 $ : num [1:3] 4 -16.5 -4.93
 $ : num [1:3] 5 -23.64 -7.98
 $ : num [1:3] 6 -21.6 33.6
 $ : num [1:3] 7 3.98 -30.96
 $ : num [1:3] 8 -1.09 13.02
 $ : num [1:3] 9 7.06 -30.02
```

```
R> is.list(test_t6)
```

Output:

```
[1] TRUE
```

```
R> # now the new default: the data.frame
R> # it is a data.frame independent from
R> # the number of agent variables requested
R> test_t1 <- NLGetAgentSet("who", "turtles")
```

```
R> str(test_t1)
```

Output:

```
'data.frame': 10 obs. of 1 variable:
 $ who: num 0 1 2 3 4 5 6 7 8 9
```

```
R> is.data.frame(test_t1)
```

Output:

```
[1] TRUE
```

```
R> # with three agent variables:
R> test_t2 <- NLGetAgentSet(c("who", "xcor", "ycor"),
   "turtles")
```

```
R> str(test_t2)
```

Output:

```
'data.frame': 10 obs. of 3 variables:
 $ who : num 0 1 2 3 4 5 6 7 8 9
 $ xcor: num 13.1 -30.6 16.7 -15.5 -16.5 ...
 $ ycor: num -18.02 15.59 29.83 31.61 -4.93 ...
```

```
R> is.data.frame(test_t2)
```

Output:

```
[1] TRUE
```

```
R> # Next, the new list style (similar to the data.frame):
R> test_t3 <- NLGetAgentSet("who", "turtles", as.data.frame=F)
```

```
R> str(test_t3)
```

Output:

```
List of 1
 $ who: num [1:10] 0 1 2 3 4 5 6 7 8 9
```

```
R> is.list(test_t3)
```

Output:

```
[1] TRUE
```

```
R> # for three agent variables:
R> test_t4 <- NLGetAgentSet(c("who", "xcor", "ycor"),
   "turtles", as.data.frame=F)
```

```
R> str(test_t4)
```

Output:

```
List of 3
 $ who : num [1:10] 0 1 2 3 4 5 6 7 8 9
 $ xcor: num [1:10] 13.1 -30.6 16.7 -15.5 -16.5 ...
 $ ycor: num [1:10] -18.02 15.59 29.83 31.61 -4.93 ...
```

```
R> is.list(test_t4)
```

Output:

```
[1] TRUE
```

```
R> # the old data structure for one agent variable
R> # (a simple vector):
R> test_t7 <- NLGetAgentSet("who", "turtles", as.vector=T)
```

```
R> str(test_t7)
```

Output:

```
 num [1:10] 0 1 2 3 4 5 6 7 8 9
```

```
R> is.list(test_t7)
```

Output:

```
[1] FALSE
```

```
R> is.vector(test_t7)
```

Output:

```
[1] TRUE
```

Later on, we will have a look on the performance of the `NLGetAgentSet` function depending on the different output data structures.

But first, we will look on the performance depending on the NetLogo version used in conjunction with RNetLogo.

G.5. NetLogo Dependent Performance

G.5.1. RNetLogo 0.9.2

NetLogo 4.1.3

In this section I will give you an impression of the performance of the `NLGetAgentSet` function in dependence of the NetLogo version used with RNetLogo.

From postings on the NetLogo mailing list like this one: http://groups.yahoo.com/group/netlogo-users/message/12919, we know that the performance of list operations can be expected to be much better in NetLogo 5.0 than in NetLogo 4.1.2. As the NetLogo primitive `sort` is used in the background of the `NLGetAgentSet` function (as well as the `NLGetPatches` function), list operations are very important for the performance of this function.

As support for NetLogo 4.0.x comes first with RNetLogo 0.9.3, we can only test NetLogo 4.1.x (here I will use 4.1.3) against NetLogo 5.0 with RNetLogo 0.9.2.

Let us start with defining a function which sets up the Fireflies model and requests all flies with different numbers of agent variables (`t1` & `t5`: one variable; `t2` & `t6`: three variables) and different returning data types (`t1` & `t2`: data.frame; `t5` & `t6`: list). Then, we call this function with different numbers of flies (i.e., turtles) starting with 100 and multiplying it with 2 until 409.600 turtles.

```
R> library(RNetLogo)
R> vers <- (packageDescription("RNetLogo")["Version"])
R> nl.path <- "C:/Program Files/NetLogo 4.1.3"
R> NLStart(nl.path, nl.version=4, gui=F)
R> model.path <- "/models/Sample
   Models/Biology/Fireflies.nlogo"
R> NLLoadModel(paste(nl.path, model.path, sep=""))
R> f_GetAgentSet <- function(x) {
+ NLCommand(paste("set number ",x,sep=""))
+ NLCommand("setup")
+
+ t1 <- system.time(df_a_1 <- NLGetAgentSet("who",
   "turtles",as.data.frame=T,
+ df.col.names=c("who"))
+ )[["user.self"]]
+ t2 <- system.time(df_a_3 <- NLGetAgentSet(c("who", "xcor",
   "ycor"),
+ "turtles", as.data.frame=T,
+ df.col.names=c("who", "xcor", "ycor"))
+ )[["user.self"]]
+
+ t5 <- system.time(li2_a_1 <- NLGetAgentSet("who", "turtles",
+ as.data.frame=F)
+ )[["user.self"]]
+ t6 <- system.time(li2_a_3 <- NLGetAgentSet(c("who", "xcor",
   "ycor"),
+ "turtles", as.data.frame=F)
```

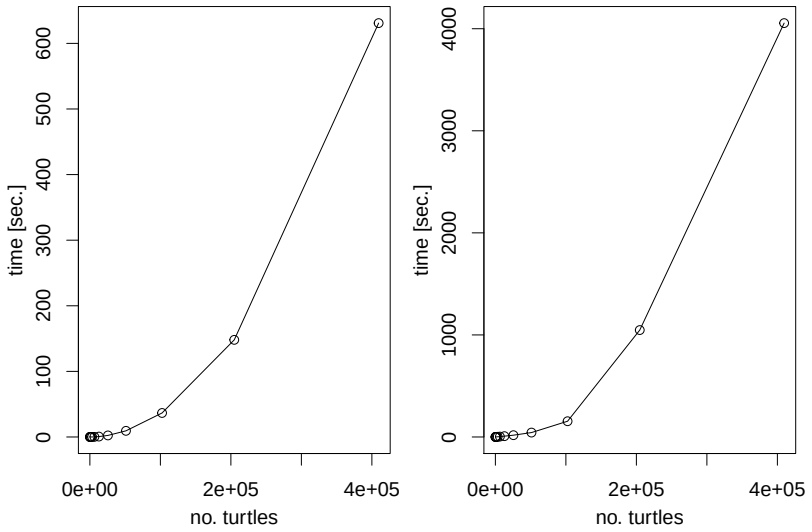


Figure G.1.: Execution time of `NLGetAgentSet` with list output for different numbers of turtles using RNetLogo 0.9.2 and NetLogo 4.1.3. Left: one agent variable, Right: three agent variables requested.

```
+ )[["user.self"]]
+ return(data.frame(t1,t2,t5,t6))
+ }
R> it <- c(100, 200, 400, 800, 1600, 3200, 6400, 12800, 25600,
   51200, 102400, 204800, 409600)
R> times_092_NL413 <- lapply(it, function(x)
   {f_GetAgentSet(x)})
R> #NLQuit()
```

The execution time for getting the agent variables within a plain list output (one list element for each agent) with one (`t5`) as well as three (`t6`) agent variables is shown in Figure G.1.

We see that the execution time increases more than linearly with an increasing number of turtles in both cases.

Now, we will have a look on the performance of the same procedure but with the data.frame as return type shown in Figure G.2.

We see that the pattern is the same as with the list output type. The data.frame output type with one requested agent variable (`t1`) is in all cases slightly slower than the list output type. The opposite is true when three agent variables are requested (`t2` vs. `t6`) with an exception for the last step (409.600 turtles).

NetLogo 5.0

Now, let us do the same with NetLogo 5.0.

The execution times for producing the plain list output (one list element for each agent) with one (`t5`) as well as three (`t6`) agent variables is given in Figure G.3. Maybe you wonder why the execution time can be higher for smaller agentsets. If you have some experiences

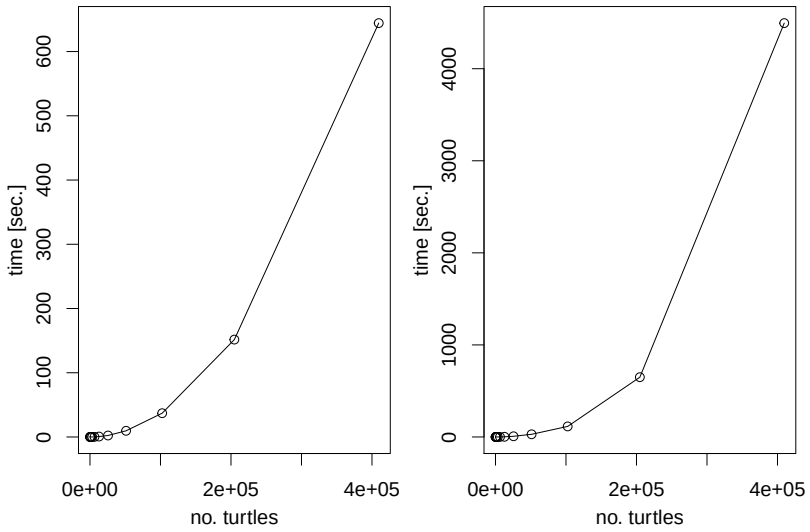


Figure G.2.: Execution time of `NLGetAgentSet` with data.frame output for different numbers of turtles using RNetLogo 0.9.2 and NetLogo 4.1.3. Left: one agent variable, Right: three agent variables requested.

with NetLogo you may have seen that execution time of NetLogo strongly varies (partly due to system background process but also due to NetLogo internals). Because the execution times are very small now, we can see these variations in the plots which have been covered before by the extremely long (and increasing) operation time.

Let us also have a look on the results for the data.frame output type shown in Figure G.4.

All in all, we see an extremely large reduction of the execution time by just switching from NetLogo 4.1.3 to NetLogo 5.0. For example, transforming 409.600 turtles with three agent variables into a data.frame, the execution time reduced from approximately 4495 seconds to only 8.46 seconds, which is a reduction of more than 530 times! This is the reason why NetLogo 5.0 is the default version since RNetLogo 0.9.3 (argument `nl.version` in `NLStart`).

But, even this performance can be improved as you can see when switching to RNetLogo 0.9.3.

G.5.2. RNetLogo 0.9.3

NetLogo 5.0

Let us start with NetLogo 5.0, as we know, this is the most interesting version. At the end, we will have a short look on the comparison with NetLogo 4.1.3 and NetLogo 4.0.5.

As mentioned above, RNetLogo 0.9.3 offers three output types. The classical list output where the list elements represent the agents (which was the default until RNetLogo 0.9.2), the new list style where each list element represents an agent variable (the fastest variant for requesting multiple agent variables), and the data.frame where each agent variable is represented in a column and each agent is represented by a row (the default in NetLogo 0.9.3).

Here is the definition of the function to iterate through different numbers of turtles:

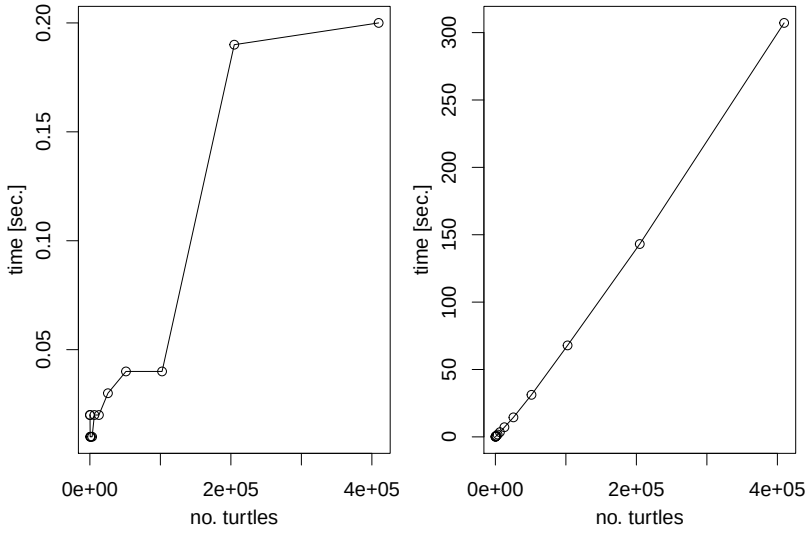


Figure G.3.: Execution time of `NLGetAgentSet` with list output for different numbers of turtles using RNetLogo 0.9.2 and NetLogo 5.0. Left: one agent variable, Right: three agent variables requested.

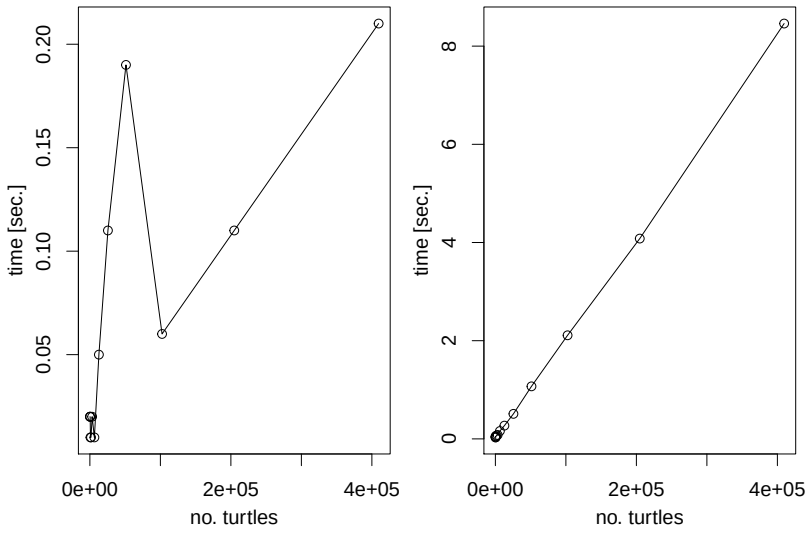


Figure G.4.: Execution time of `NLGetAgentSet` with data.frame output for different numbers of turtles using RNetLogo 0.9.2 and NetLogo 5.0. Left: one agent variable, Right: three agent variables requested.

```
R> library(RNetLogo)
R> vers <- (packageDescription("RNetLogo")["Version"])
R> nl.path <- "C:/Program Files/NetLogo 5.0"
R> NLStart(nl.path, nl.version=5, gui=F)
R> model.path <- "/models/Sample
   Models/Biology/Fireflies.nlogo"
R> NLLoadModel(paste(nl.path, model.path, sep=""))
R> f_GetAgentSet <- function(x) {
+ NLCommand(paste("set number ",x,sep=""))
+ NLCommand("setup")
+
+ t1 <- system.time(df_a_1 <- NLGetAgentSet("who", "turtles")
+ )[["user.self"]]
+ t2 <- system.time(df_a_3 <- NLGetAgentSet(c("who", "xcor",
   "ycor"),
+ "turtles")
+ )[["user.self"]]
+
+ t3 <- system.time(li_a_1 <- NLGetAgentSet("who", "turtles",
+ as.data.frame=F)
+ )[["user.self"]]
+ t4 <- system.time(li_a_3 <- NLGetAgentSet(c("who", "xcor",
   "ycor"), "turtles",
+ as.data.frame=F)
+ )[["user.self"]]
+
+ t5 <- system.time(li2_a_1 <- NLGetAgentSet("who", "turtles",
   as.data.frame=F,
+ agents.by.row=T)
+ )[["user.self"]]
+ t6 <- system.time(li2_a_3 <- NLGetAgentSet(c("who", "xcor",
   "ycor"), "turtles",
+ as.data.frame=F,
+ agents.by.row=T)
+ )[["user.self"]]
+
+ t7 <- system.time(ve_a_1 <- NLGetAgentSet(c("who", "xcor",
   "ycor"), "turtles",
+ as.vector=T)
+ )[["user.self"]]
+ return(data.frame(t1,t2,t3,t4,t5,t6,t7))
+ }
R> it <- c(100, 200, 400, 800, 1600, 3200, 6400, 12800, 25600,
   51200, 102400, 204800, 409600)
R> times_093_NL5 <- lapply(it, function(x) {f_GetAgentSet(x)})
R>
R> #NLQuit()
```

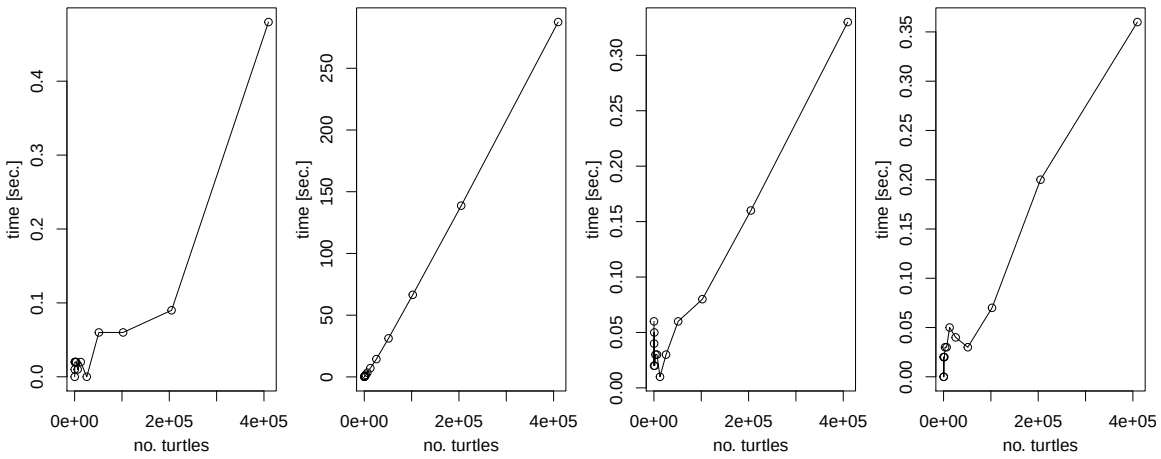


Figure G.5.: Execution time of `NLGetAgentSet` with one requested agent variable for different numbers of turtles using RNetLogo 0.9.3 and NetLogo 5.0. Left: simple vector, Second from left: old list style, Second from right: data.frame, Right: new list style.

Let us start with a comparison of the performance of the three different output types for requesting one agent variable as shown in Figure G.5.

We see that the execution times for the old list style (which was the default in RNetLogo <= 0.9.2) are larger than in RNetLogo 0.9.2. The explanation for this is very easy: the old RNetLogo version returned just a vector when the user requested only one agent variable while the new version constructs the requested list which is more logical but takes more time. The same is true for the data.frame construction. Furthermore, we see that the new list style is not faster than the data.frame construction when requesting only one agent variable.

But this conclusion changes when we look on the execution times for three agent variables (Figure G.6).

The time for requesting three agent variables in the old list style takes approximately the same time with RNetLogo 0.9.3 as with the old version. But the performance of the data.frame is much better than with the old version when requesting more than one agent variable. For 409.600 turtles with three variables it took approximately 8.46 seconds with RNetLogo 0.9.2 and now, with RNetLogo 0.9.3, it takes approximately 1.26 seconds, which is a reduction of more than six times. Furthermore, we see that the new list style is even slightly faster than the data.frame with only 0.7 seconds for 409.600 turtles with three variables (i.e., the transformation of 1.228.800 values!).

NetLogo 5.0 vs. 4.1.3 vs. 4.0.5

Just for interest, we will compare also the performance of NetLogo 4.1.3 with the one of NetLogo 4.0.5, which is possible since RNetLogo 0.9.3, as well as with NetLogo 5.0.

The results for only one agent variable are shown in Figure G.7 and the results for requesting three agent variables are given in Figure G.8.

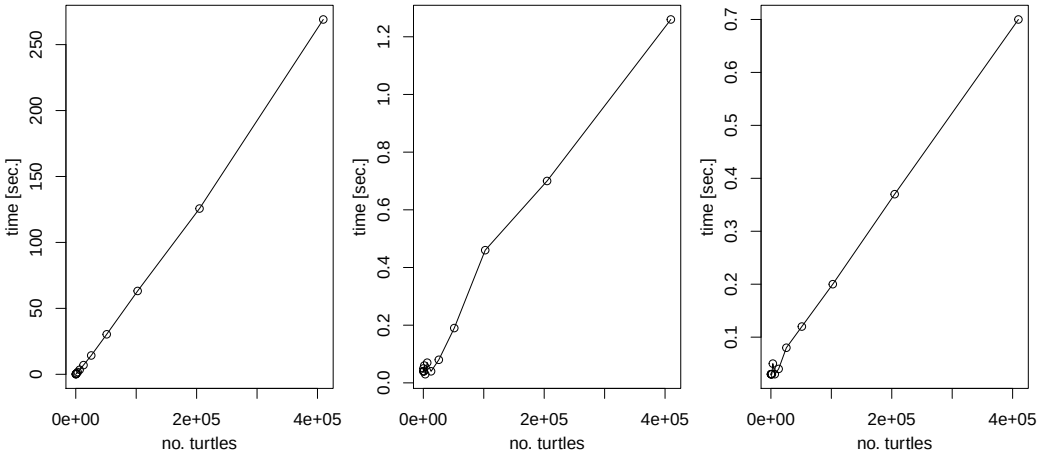


Figure G.6.: Execution time of `NLGetAgentSet` with three requested agent variables for different numbers of turtles using RNetLogo 0.9.3 and NetLogo 5.0. Left: old list style, Middle: data.frame, Right: new list style.

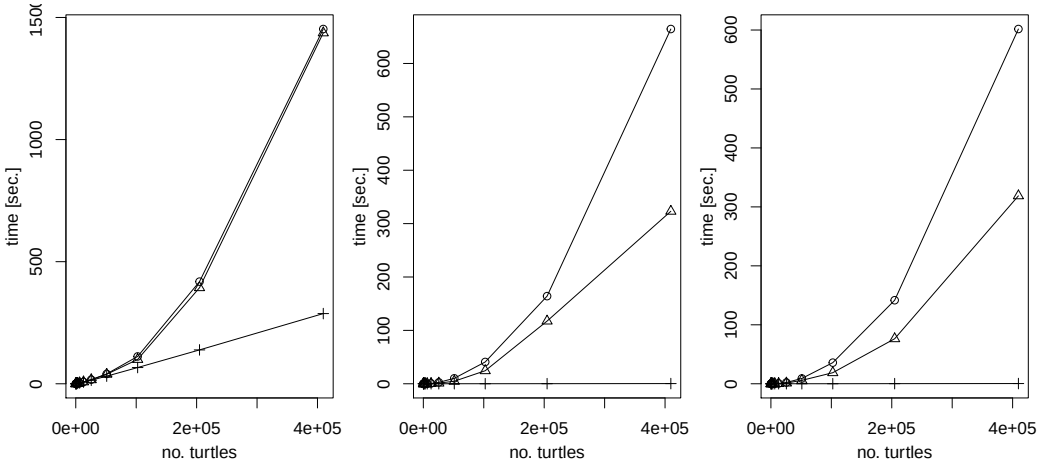


Figure G.7.: Execution time of `NLGetAgentSet` with one requested agent variable for different numbers of turtles using RNetLogo 0.9.3 and NetLogo 4.0.5 (dots), 4.1.3 (triangle), and 5.0 (plus). Left: old list style, Middle: data.frame, Right: new list style.

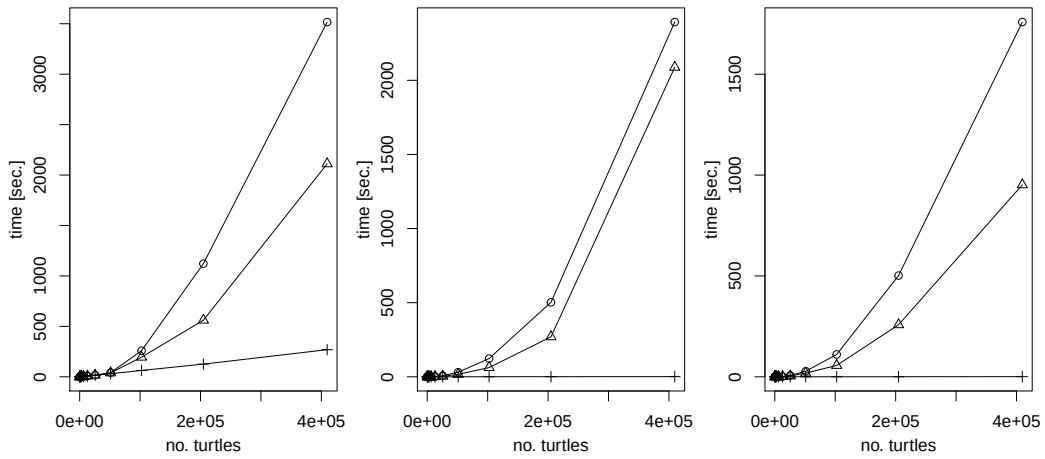


Figure G.8.: Execution time of `NLGetAgentSet` with three requested agent variables for different numbers of turtles using RNetLogo 0.9.3 and NetLogo 4.0.5 (dots), 4.1.3 (triangle), and 5.0 (plus). Left: old list style, Middle: data.frame, Right: new list style.

In (nearly) all cases, NetLogo 4.0.5 performs better than NetLogo 4.1.3 but it is also far behind NetLogo 5.0.

G.6. Conclusion

We have seen that, whenever possible, NetLogo 5.0 should be preferred over NetLogo 4.x. The performance of `NLGetAgentSet` is substantially better with NetLogo 5.0. These results are representative for other operations/functions. Furthermore, RNetLogo 0.9.3 performs better than older versions when requesting more than one agent variable with the `NLGetAgentSet` or `NLGetPatches` functions. The old list style should be avoided. The data.frame is now the default return type and performs well. But if it is sufficient to proceed with a list, the new list style is even faster than the data.frame and should be preferred.

With NetLogo 5.0 and RNetLogo 0.9.3, in most cases the data transformation from NetLogo to R should be fast enough even for requesting large numbers of turtles or patches. To get three agent variables for 409.600 turtles (i.e., 1.228.800 values) from NetLogo into R took in the example only 1.26 seconds for a data.frame and 0.7 seconds for the new list style. The `setup` procedure in the Fireflies model in pure NetLogo (without using RNetLogo and switching off the visual update) with 409.600 turtles took for example approximately 61 seconds and the execution of one simulation step (`go` procedure) with this number of turtles took approximately 80 seconds. In conclusion, I think you will find RNetLogo is not a bottleneck.

Acknowledgement

Thanks go to Florian Hartig (UFZ - Helmholtz Centre for Environmental Research, Leipzig) and Thomas Petzold (University of Dresden) who found it took very long to transform large agentsets and motivated me to look closer on this topic. Furthermore, Florian made some valuable notes on this manuscript.

APPENDIX H

Parallel Processing with the RNetLogo Package

H.1. Abstract

RNetLogo is a flexible interface for NetLogo to R. It opens various possibilities to connect agent-based models with advanced statistics. It opens the possibility to use R as the starting point to design systematic experiments with agent-based models and perform parameter fittings and sensitivity analysis. Therefore, it can be necessary to perform repeated (independent) simulations which can be parallelized. Here, I present how such a parallelization can be done for the RNetLogo package. The techniques presented here can be used to run multiple simulations in parallel on a single computer with multiple processors or to spread multiple simulations to several processors in computer clusters/grids. Using the parallel package has a positive side effect: It enables you to start more than one NetLogo instance with GUI in parallel, which is not possible without parallelization.

H.2. Motivation

Since modern computers mostly have more than one processor and agent-based simulations are often complex and time consuming it is desirable to spread repeated simulations, for example for parameter fitting or sensitivity analysis, to multiple processors in parallel. Here, I will present one way of how it is possible to spread multiple NetLogo simulations controlled from R via the RNetLogo package to multiple processors.

H.3. Parallelization in R

R itself is not able to make use of multiple processors of a computer. But there are several R packages available, which enable the user to spread repeated processes to multiple processors. There is a CRAN Task View called "High-Performance and Parallel Computing with R" at http://cran.r-project.org/web/views/HighPerformanceComputing.html. Since R version 2.14.0 there is the parallel package included in every standard R installation. In the following I will present how to use this parallel package in conjunction with RNetLogo. Therefore, to follow the examples it requires that you have an R version >= 2.14.0 installed. There is a pdf file coming with the parallel packing giving a short introduction into the usage of

the package and the platform specific differences. You should always start by reading this document. A last note, before we start: The commands presented in the following have been tested on Windows and Linux operation systems only. If you have experiences with Mac OS please let me know.

H.4. Parallelize a Simple Process

The basic concept of the parallel package is to parallelize an `apply` (or `lapply`, `sapply` etc.) operation. This means that the process you want to parallelize has to be a process which is applied to an array, matrix, list, etc.

Let us start with a simple example without using RNetLogo. First, we define a simple function which calculates the square of an input number.

```
R> testfun1 <- function(x) {
+ return(x*x)
+ }
```

We can apply this simple function to a vector of values using `sapply` like this:

```
R> my.v1 <- 1:10
R> print(my.v1)
```

Output:

```
 [1] 1 2 3 4 5 6 7 8 9 10
```

```
R> my.v1.quad <- sapply(my.v1, testfun1)
R> print(my.v1.quad)
```

Output:

```
 [1] 1 4 9 16 25 36 49 64 81 100
```

The result is a vector with the squared values of the input vector, i.e., the function was applied sequentially to each element of the input vector.

One way to use the parallel package is to run the parallel version of the `sapply` function which is called `parSapply`.

Before we can use this function, we have to make/register a cluster, as you know from the manual of the parallel package. Therefore, we could, for example, detect the number of cores of our local computer and start a local cluster with this number of processors, as shown here:

```
R> # load the parallel package
R> library(parallel)
R> # detect the number of cores available
R> processors <- detectCores()
```

```
R> # create a cluster
R> cl <- makeCluster(processors)
```

Then, we can run our simple function on this cluster. At the end, we should always stop the cluster.

```
R> # call parallel sapply
R> my.v1.quad.par <- parSapply(cl, my.v1, testfun1)
R> print(my.v1.quad.par)
```

Output:

```
 [1] 1 4 9 16 25 36 49 64 81 100
```

```
R> # stop cluster
R> stopCluster(cl)
```

H.5. Parallelize RNetLogo

As you know from the RNetLogo manual, it requires an initialization using the `NLStart` and (maybe) `NLLoadModel` function. To parallelize RNetLogo we need this initialization to be done for every processor, because they are independent from each other (which is a very important property, because, for example, random processes in parallel simulations should not being influenced by each other).

Therefore, it makes sense to put the initialization, the simulation, and the quitting process into separate functions. These functions can look like the following (if you want to test these, do not forget to adapt the paths appropriately):

```
R> # the initialization function
R> prepro <- function(dummy, gui, nl.path, model.path) {
+ library(RNetLogo)
+ NLStart(nl.path, nl.version=5, gui=gui)
+ NLLoadModel(model.path)
+ }
R> # the simulation function
R> simfun <- function(x) {
+ NLCommand("print ",x)
+ NLCommand("set density", x)
+ NLCommand("setup")
+ NLCommand("go")
+ NLCommand("print count turtles")
+ ret <- data.frame(x, NLReport("count turtles"))
+ names(ret) <- c("x","turtles")
+ return(ret)
+ }
R> # the quit function
R> postpro <- function(x) {
+ NLQuit()
+ }
```

H.5.1. With Graphical User Interface (GUI)

Now, we have to start the cluster, run the initialization function in each processor, which will open as many NetLogo windows as we have processors.

Note that this is also a nice way to run multiple NetLogo GUI instances in parallel, which is not possible within one R session without this parallelization.

```
R> # load the parallel package, if not already done
R> require(parallel)
R> # detect the number of cores available
R> processors <- detectCores()
R> # create cluster
R> cl <- makeCluster(processors)
R> # set variables for the start up process
R> # adapt path appropriately (or set an environment variable
   NETLOGO_PATH)
R> gui <- TRUE
R> nl.path <- Sys.getenv("NETLOGO_PATH", "C:/Program
   Files/NetLogo 5.0.4")
R> model.path <- "models/Sample Models/Earth
   Science/Fire.nlogo"
R> # load NetLogo in each processor/core
R> invisible(parLapply(cl, 1:processors, prepro, gui=gui,
+ nl.path=nl.path, model.path=model.path))
```

After the initialization is done in all processors, we can run the simulation. Here, we will use the Fire model from NetLogo's Model Library. We will vary the density value from 1 to 20, i.e., we will run 20 independent simulations each with a different density value.

```
R> # create a vector with 20 density values
R> density <- 1:20
R> print(density)
```

Output:

```
 [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

```
R> # run a simulation for each density value
R> # by calling parallel sapply
R> result.par <- parSapply(cl, density, simfun)
```

```
R> print(data.frame(t(result.par)))
```

Output:

```
   x turtles
1 1 252
2 2 261
```

```
3 3 261
4 4 264
5 5 261
6 6 265
7 7 269
8 8 269
9 9 277
10 10 276
11 11 273
12 12 284
13 13 279
14 14 289
15 15 282
16 16 289
17 17 302
18 18 296
19 19 300
20 20 294
```

At the end, we should stop all NetLogo instances and the cluster.

```
R> # Quit NetLogo in each processor/core
R> invisible(parLapply(cl, 1:processors, postpro))
R> # stop cluster
R> stopCluster(cl)
```

H.5.2. Headless

The same is possible with the headless mode, i.e., without the GUI. We just have to set the variable `gui` to `FALSE`.

It can look like this:

```
R> # run in headless mode
R> gui <- FALSE
R> # create cluster
R> cl <- makeCluster(processors)
R> # load NetLogo in each processor/core
R> invisible(parLapply(cl, 1:processors, prepro, gui=gui,
+ nl.path=nl.path, model.path=model.path))
```

```
R> # create a vector with 20 density values
R> density <- 1:20
R> print(density)
```

Output:

```
 [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

```
R> # run a simulation for each density value
R> # by calling parallel sapply
R> result.par <- parSapply(cl, density, simfun)
```

```
R> print(data.frame(t(result.par)))
```

Output:

```
   x turtles
1  1     253
2  2     256
3  3     259
4  4     258
5  5     268
6  6     267
7  7     272
8  8     272
9  9     278
10 10    278
11 11    280
12 12    281
13 13    276
14 14    284
15 15    290
16 16    295
17 17    296
18 18    295
19 19    300
20 20    289
```

```
R> # Quit NetLogo in each processor/core
R> invisible(parLapply(cl, 1:processors, postpro))
R> # stop cluster
R> stopCluster(cl)
```

H.6. Conclusion

We have seen one way of how it is possible to spread repeated and independent simulations to multiple processors using the parallel package. Therefore, RNetLogo can be efficiently used to perform parameter fittings and sensitivity analysis where large numbers of repeated simulations are required.

APPENDIX I

ODD Model Description to Kerr et al. [2002]

The model description follows the ODD (Overview, Design concepts, Details) protocol for describing individual- and agent-based models [Grimm et al. 2006, 2010].

I.1. Purpose

The model analyses the effect of local interactions and dispersal on coexistence of three competing species in the context of theoretical analysis of the mechanisms that maintain biodiversity.

I.2. Entities, State Variables, and Scales

The model consists of 250 x 250 equal-sized grid cells. Each grid cell is described by its x- and y-coordinates and a state variable "species" defining by which of the three species C (colicinogenic cells), S (sensitive cells), R (resistant cells), or none it is occupied. The boundaries of the world are wrapped around. A time step is called an "epoch" meaning that all grid cells are updated once. A run for coexistence check comprises 10,000 epochs.

I.3. Process Overview and Scheduling

In one time step (epoch) all grid cells are updated asynchronously in a random order. Updating means that, first, the relative share of neighbouring grid cells occupied by each species are determined. If the current cell is not settled by a species it is recolonized depending on the states of the neighbours. Otherwise, mortality takes place - as natural background mortality and, in case of species S, by poisoning by species C from neighbouring grid cells.

```
to go
  ; iterate all patches (in random order)
  ask patches [
    ; get relative share of species among neighbours
    let f_c 0.0
    let f_r 0.0
    let f_s 0.0
```

```
    ; local interaction alternative
    set f_c count neighbors with [species = 0] / 8.0
    set f_r count neighbors with [species = 1] / 8.0
    set f_s count neighbors with [species = 2] / 8.0

    ; update species state
    ifelse (species = 3)
    [ recolonization f_c f_r f_s ]
    [ mortality f_c ]
  ]

  ; increase time
  tick
end
```

I.4. Design Concepts

Basic principles. The hypothesis underlying the model is that coexistence and in turn maintenance of biodiversity depends on the locality of interactions and dispersal of competing species. The model is constructed on stochastic neighbourhood interaction rules only. No further aspects, like species-specific habitat suitability is taken into account. The interaction between the species follows the principles of the children's game rock-paper-scissors, where rock crushes scissors, scissors cuts paper, and paper covers rock.

Emergence. Running the model over multiple epochs coexistence or extinction of species emerge from the neighbourhood interactions.

Adaptation. No adaptive traits.

Objectives. No objectives.

Learning. No learning.

Prediction. No prediction.

Sensing. No sensing.

Interaction. Neighbouring grid cells are able to occupy empty grid cells and cells occupied by species C increase mortality probability on neighbouring grid cells settled by species S.

Stochasticity. All processes are random, namely the order of grid cells update, the recolonization of empty grid cells and mortality.

Collectives. No collectives.

Observation. The number of grid cells occupied by the different species are counted in each time step. After all time steps this list is checked if all species are present, i.e., coexistence has taken place.

I.5. Initialization

At the beginning of a simulation the state variable "species" of all 250 x 250 grid cells is initialized randomly. The three different species as well as no species (i.e., empty grid cell) are assigned with equal probability.

I.6. Input Data

No input data.

I.7. Submodels

I.7.1. Re-Colonization

Re-colonization takes place when a cell is not occupied by a species. The re-colonization is a stochastic process with probabilities depending on the species shares of the neighbouring grid cells.

```
to recolonization [f_c f_r f_s]
  let rand random-float 1.0
  ifelse ((rand < f_c) and f_c > 0.0)
  [ set species 'C' ]
  [
    ifelse (rand < (f_c + f_r) and f_r > 0.0)
    [ set species 'R' ]
    [
      if (rand < (f_c + f_r + f_s) and f_s > 0.0)
      [ set species 'S' ]
    ]
  ]
end
```

I.7.2. Mortality

If a grid cell is occupied by species S or R, mortality takes place. Mortality is a stochastic process with mortality probability of species S depending on the share of species C among the neighbouring grid cells. If mortality takes place the grid cell is assigned as empty.

```
to mortality [f_c]
  let mort_prob 0.0
  ifelse (species = 'C')
  [ set mort_prob delta_c ]
  [
    ifelse (species = 'R')
    [ set mort_prob delta_r ]
```

```
      [ set mort_prob delta_s0 + tau * f_c ]
    ]
    let rand random-float 1.0
    if (rand < mort_prob)
    [
      set species 'empty'
    ]
  end
```

with parameter values:

- *delta_c*: fixed at $1/3$
- *delta_r*: varied between $1/4$ and $1/3$ (base value: $10/32$)
- *delta_s0*: fixed at $1/4$
- *tau*: varied between $1/8$ and $3/4$ (base value: $3/4$)