Chapter 4
Learning from the Past: Approaches for Reproducibility in Computational Neuroscience

Sharon M. Crook, Andrew P. Davison, and Hans E. Plesser

Abstract Reproducible experiments are the cornerstone of science: only observations that can be independently confirmed enter the body of scientific knowledge. Computational science should excel in reproducibility, as simulations on digital computers avoid many of the small variations that are beyond the control of the experimental biologist or physicist. However, in reality, computational science has its own challenges for reproducibility: many computational scientists find it difficult to reproduce results published in the literature, and many authors have met problems replicating even the figures in their own papers. We present a distinction between different levels of replicability and reproducibility of findings in computational neuroscience. We also demonstrate that simulations of neural models can be highly sensitive to numerical details, and conclude that often it is futile to expect exact replicability of simulation results across simulator software packages. Thus, the computational neuroscience community needs to discuss how to define successful reproduction of simulation studies. Any investigation of failures to reproduce published results will benefit significantly from the ability to track the provenance of the original results. We present tools and best practices developed over the past 2 decades that facilitate provenance tracking and model sharing.

S.M. Crook, Ph.D. (✉)
School of Mathematical and Statistical Sciences and School of Life Sciences, Center for Adaptive Neural Systems, Arizona State University, Physical Sciences Building A Room 216, 901 S. Palm Walk, Tempe, AZ 85287-1804, USA
e-mail: sharon.crook@asu.edu

A.P. Davison
Unité de Neuroscience Information et Complexité (UNIC), CNRS, Gif-sur-Yvette 91198, France

H.E. Plesser
Department of Mathematical Sciences and Technology, Norwegian University of Life Sciences, Aas, Norway

Introduction

Reproducible experimental results have been the cornerstone of science since the time of Galileo (Hund 1996, p. 103): Alice's exciting finding will not become part of established scientific knowledge unless Bob and Charlie can reproduce her results independently. A related concept is provenance—the ability to track a given scientific result, such as a figure in an article, back through all analysis steps to the original raw data and the experimental protocol used to obtain it.

The quest for reproducibility raises the question of what it means to reproduce a result independently. In the experimental sciences, data will contain measurement error. Proper evaluation and judgement of such errors requires a sufficient understanding of the processes giving rise to fluctuations in measurements, as well as of the measurement process itself. An interesting historical example is the controversy surrounding Millikan's oil drop experiments for the measurement of the elementary charge, which led Schrödinger to investigate the first-passage time problem in stochastic processes (Schrödinger 1915). As experimental error can never be eliminated entirely, disciplines depending on quantitative reproducibility of results, such as analytical chemistry, have developed elaborate schemes for ascertaining the level of reproducibility that can be obtained and for detecting deviations (Funk et al. 2006). Such schemes include round robin tests in which one out of a group of laboratories prepares a test sample, which all others in the group then analyze. Results are compared across the group, and these tests are repeated regularly, with laboratories taking turns at preparing the test sample.

In computational, simulation-based science, the reproduction of previous experiments and the establishment of the provenance of results should be easy, given that computers are deterministic and do not suffer from the problems of inter-subject and trial-to-trial variability of biological experiments. However, in reality, computational science has its own challenges for reproducibility.

As early as 1992, Claerbout and Karrenbach addressed the necessity of provenance in computational science and suggested the use of electronic documentation tools as part of the scientific workflow. Some of the first computational tools that included a complete documentation of provenance were developed in the signal processing community (Donoho et al. 2009; Vandewalle et al. 2009) and other fields followed (Quirk 2005; Mesirov 2010). Generally, these important efforts are discussed as examples of *reproducible research*. However, following Drummond (2009), we find it important to distinguish between the *reproduction* of an experiment by an independent researcher and the *replication* of an experiment using the same code perhaps some months or years later.

Independent reproducibility is the gold standard of science; however, replicability is also important and provides the means to determine whether the failure of others to reproduce a result is due to errors in the original code. Replication ought to be simple—it is certainly easier than independent reproduction—but in practice replicability is often not trivial to achieve and is not without controversy. Drummond (2009) has argued that the pursuit of replicability detracts from the promotion of independent reproducibility, since the two may be confused and due to the burden that ensuring replicability places on the researcher. Indeed, when the journal *Biostatistics* recently introduced a scheme for certifying papers as replicable after a review of code and data by the associate editor for reproducibility (Peng 2009), this led to a lively debate about the relative importance of replicability of data processing steps in the context of complex scientific projects (Keiding 2010a, b; Breslow 2010; Cox and Donnelly 2010; DeAngelis and Fontanarosa 2010; Donoho 2010; Goodman 2010b; Groves 2010; Peng 2010). Generally, the risk of confusing replicability with reproducibility is an argument for education and discussion, not for neglecting replicability, and the extra workload of carefully tracking full provenance information may be alleviated or eliminated by appropriate tools. Discussions of reproducibility generally include both of these concepts, and here, we find it useful to make further distinctions as follows.

Internal replicability: The original author or someone in their group can re-create the results in a publication, essentially by rerunning the simulation software. For complete replicability within a group by someone other than the original author, especially if simulations are performed months or years later, the author must use proper bookkeeping of simulation details using version control and electronic lab journals.

External replicability: A reader is able to re-create the results of a publication using the same tools as the original author. As with internal replicability, all implicit knowledge about the simulation details must be entered into a permanent record and shared by the author. This approach also relies on code sharing, and readers should be aware that external replicability may be sensitive to the use of different hardware, operating systems, compilers, and libraries.

Cross-replicability: The use of "cross" here refers to simulating the same model with different software. This may be achieved by re-implementing a model using a different simulation platform or programming language based on the original code, or by executing a model described in a simulator-independent format on different simulation platforms. Simulator-independent formats can be divided into declarative and procedural approaches. Assuming that all simulators are free of bugs, this would expose the dependence of simulation results on simulator details, but leads to questions about how to compare results.

Reproducibility: Bob reads Alice's paper, takes note of all model properties, and then implements the model himself using a simulator of his choice. He does not download Alice's scripts. Bob's implementation thus constitutes an independent implementation of the scientific ideas in Alice's paper based on a textual description. The boundary line between cross-replicability and reproducibility is not always clear. In particular, a declarative description of a model is a structured, formalized version of what should appear in a publication, so an implementation by Charlie based on a declarative format might be considered to be just as independent as that by Bob based on reading the article.

In our terminology, the *reproducible research* approach propagated by Donoho (2010) ensures internal replicability and to quite a degree external replicability as well. As de Leeuw (2001) has pointed out, though, Donoho's specific approach depends on the commercial Matlab software together with a number of toolboxes, and thus does not aim to ensure cross-replicability and independent reproducibility as defined above. So how have approaches for replicability and reproducibility evolved in the field of computational neuroscience? As early as 1992, one group of simulation software developers realized the need for computational benchmarks as a first step toward cross-replicability (Bhalla et al. 1992); however, these benchmarks were not broadly adopted by the simulator development community. There were also early efforts to encourage simulator-independent model descriptions for complex neural models. Building on general-purpose structures proposed by Gardner et al. (2001), the first declarative description tools for models were described in 2001 by Goddard et al. Around the same time, the NMODL language developed by Michael Hines for describing biophysical mechanisms in the NEURON simulator was extended by Hines and Upinder Bhalla to work with GENESIS (GMODL; Wilson et al. 1989), making it perhaps the first programmatic simulator-independent model description language in computational neuroscience (Hines and Carnevale 2000). More recently, the activities of organizations, such as the *Organization for Computational Neuroscience* (http://www.cnsorg.org) and the *International Neuroinformatics Coordinating Facility* (http://www.incf.org), focused journals such as *Neuroinformatics* and *Frontiers in Neuroinformatics* and dedicated workshops (Cannon et al. 2007; Djurfeldt and Lansner 2007) have provided fora for an ongoing discussion of the methodological issues our field is facing in developing an infrastructure for replicability and reproducibility. The first comprehensive review of neuronal network simulation software (Brette et al. 2007) provides an example of the gains of this process.

However, there are still many improvements needed in support of reproducibility. Nordlie et al. (2009) painted a rather bleak picture of the quality of research reporting in our field. Currently, there are no established best practices for the description of models, especially neuronal network models, in scientific publications, and few papers provide all necessary information to successfully reproduce the simulation results shown. Replicability suffers from the complexity of our code and our computing environments, and the difficulty of capturing every essential piece of information about a computational experiment. These difficulties will become even more important to address as the ambition of computational neuroscience and the scrutiny placed upon science in general grow (Ioannidis 2005; Lehrer 2010).

In what follows, we will discuss further details of replicability and reproducibility in the context of computational neuroscience. In section "The Limits of Reproducibility," we examine the limits of reproducibility in the computational sciences with examples from computational neuroscience. Section "Practical Approaches to Replicability" deals with practical approaches to replicability such as code sharing, tracking the details of simulation experiments, and programmatic or procedural descriptions of complex neural models that aid in cross-replicability. In section "Structured, Declarative Descriptions of Models and Simulations," we introduce a number of efforts to formalize declarative descriptions of models and simulations and the software infrastructure to support this. Finally, in section

"Improving Research Reporting," we discuss more general efforts to improve research reporting that go beyond software development.

The Limits of Reproducibility

Independent reproduction of experimental results will necessarily entail deviations from the original experiment, as not all conditions can be fully controlled. Whether a result is considered to have been reproduced successfully thus requires careful scientific judgement, differentiating between core claims and mere detail in the original report. However, solving the same equation twice yields precisely the same result from a mathematical point of view. Consequently, it might appear that any computational study which is described in sufficient detail should be *exactly* reproducible: by solving the equations in Alice's paper using suitable software, Bob should be able to obtain figures identical to those in the paper. This implies that Alice's results are also perfectly replicable. It is obviously a prerequisite for such exact reproducibility that results can be replicated internally, externally, and across suitable software applications. In this section, we discuss a number of obstacles to external and cross-replicability of computational experiments which indicate that it is futile to expect perfect reproducibility of computational results. Rather, computational scientists need to apply learned judgement to the same degree as experimentalists in evaluating successful reproduction.

Faulty computer hardware is the principal—though not the most frequent—obstacle: digital computers are electronic devices and as such are subject to failure, which often may go undetected. For example, memory may be corrupted by radiation effects (Heijmen 2011), and as we are rapidly approaching whole-brain simulations on peta-scale and soon exa-scale computers, component failure will become a routine issue. Consider a computer with one million cores. Even if each core has a mean time between failure of a million hours (roughly 115 years), one would expect on average one core failure per hour of operation. It seems questionable whether all such errors will be detected reliably—a certain amount of undetected data corruption appears unavoidable.

Even if hardware performs flawlessly, computer simulations are not necessarily deterministic. In parallel simulations, performance of the individual parallel processes will generally depend on other activity on the computer, so that the order in which processes reach synchronization points is unpredictable. This type of unpredictability should not affect the results of correct programs, but subtle mistakes may introduce nondeterministic errors that are hard to detect. Even if we limit ourselves to serial programs running on perfect hardware, a number of pitfalls await those trying to reproduce neuronal network modeling results from the literature, which fall into the following categories:

1 Insufficient, ambiguous, or inaccurate descriptions of the model in the original publication.

2 Models that are mathematically well-defined, but numerically sensitive to implementation details.
3 Model specifications that are unambiguous and complete from a neuroscience point of view, but underspecified from a computational point of view.
4 Dependencies on the computing environment.

First, consider the first and last points. An insufficient model specification in a publication can be a major stumbling block when trying to replicate a model. Thus, ambiguous model descriptions provide a significant impediment to science that can only be avoided if authors, referees, and publishers adhere to strict standards for model specification (Nordlie et al. 2009) or rigorous, resource-intensive curation efforts (Lloyd et al. 2008); we will return to this point in section "Improving Research Reporting." Dependencies on the computing environment, such as the versions of compilers and external libraries used, will be discussed in section "Is Code Sharing Enough to Ensure Replicability?" Here, we consider model descriptions that are mathematically ambiguous or sensitive to the implementation details before discussing the consequences for computational neuroscience in section "Defining Successful Reproduction."

Ambiguous Model Numerics

The model equation for the subthreshold membrane potential V of a leaky integrate-and-fire neuron with constant input is a simple, linear first-order ordinary differential equation (Lapicque 1907)

$$\frac{dV}{dt} = -\frac{V}{\tau} + \frac{I_E}{C}. \tag{4.1}$$

Here, membrane potential V (in mV) is defined relative to the resting potential of the neuron, τ is the membrane time constant (in ms), C the membrane capacitance (in pF), and $I(t)$ the input current to the neuron (in pA). As far as differential equations go, this equation is about as simple as possible, and solutions are well-defined and well-behaved. For the initial condition $V(t=0)=V_0$, (4.1) has the analytical solution

$$V(t) = V_0 e^{-t/\tau} + \frac{I_E \tau}{C}(1 - e^{-(t/\tau)}). \tag{4.2}$$

Abbreviating $V_k = V(kh)$ and $a = I_E\tau/C$ gives the following iteration rule for time step h:

$$V_{k+1} = V_k e^{-(h/\tau)} + a(1 - e^{-(h/\tau)}). \tag{4.3}$$

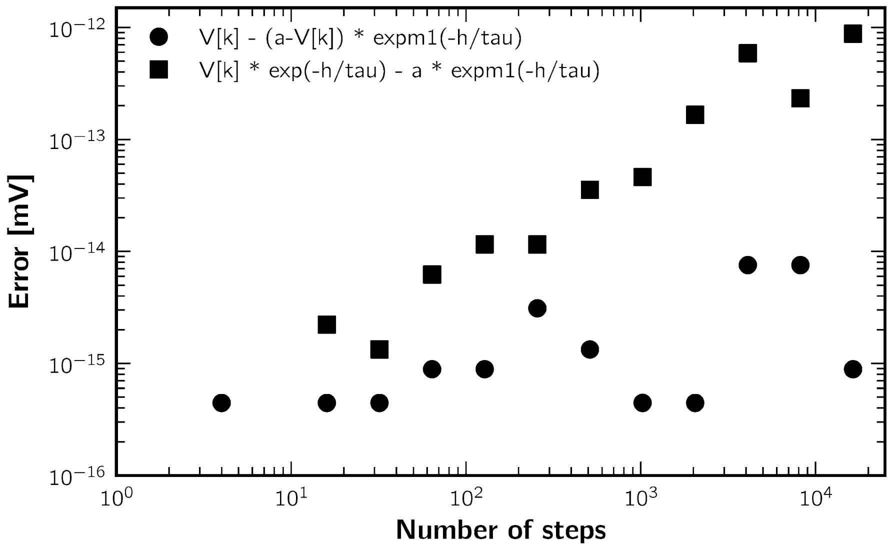

Fig. 4.1 Error of the membrane potential $V(T)$ for $T=1$ ms computed with two different implementations of (4.3) using step sizes from $h=2^{-1}$ to 2^{-14} ms, corresponding to 2 to 16384 steps. *Circles* show errors for the implementation given in (4.4), *squares* for the implementation given in (4.5); see text for details

This updating rule is mathematically exact, and similar rules can be found for any system of ordinary linear differential equations (Rotter and Diesmann 1999).

Now consider the following two implementations of (4.3)[1]:

$$V[k+1]=V[k]-(a-V[k])*\mathrm{expm1}(-h/\tau) \tag{4.4}$$

$$V[k+1]=V[k]*\exp(-h/\tau)-a*\mathrm{expm1}(-h/\tau). \tag{4.5}$$

Both implementations are mathematically equivalent, but differ significantly numerically, as can be seen by computing the evolution of the membrane potential for $T=1$ ms using different step sizes, starting from $V(t=0)=0$ mV. We obtain the reference solution $V^*=V(T)$ directly from (4.2) as a*expm1(T/τ) with the following parameter values: $I_E=1{,}000$ pA, $C=250$ pF, $\tau=10$ ms, so that $a=40$ mV. We then compute $V(T)$ using update steps from $h=2^{-1}$ ms down to $h=2^{-14}$ ms using both implementations and compute the difference from the reference solution; using step sizes that are powers of 2 avoids any unnecessary round-off error (Morrison et al. 2007). Results obtained with the update rule provided by (4.5) are several orders of magnitude larger than those obtained with the rule in (4.4), as shown in Fig. 4.1. Data were obtained with a custom C++ program compiled with the g++ compiler version 4.5.2 and default compiler settings on a Apple Mac Book Pro with an Intel

[1]expm1(x) is a library function computing exp(x)-1 with high precision for small x.

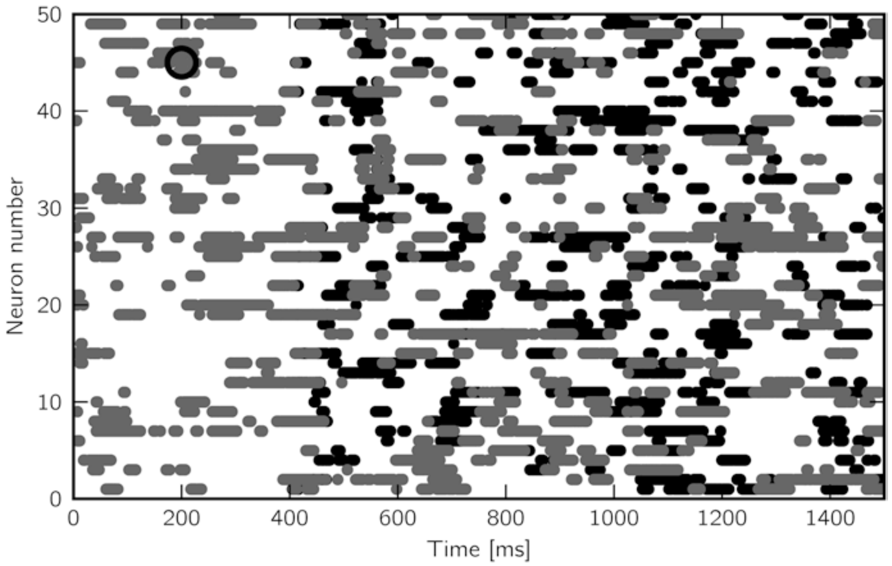

Fig. 4.2 Raster plots of spike trains of 50 excitatory neurons in a balanced network of 1,250 neurons exhibiting self-sustained irregular activity after Poisson stimulation during the first 50 ms and no external input thereafter. The network is based on Brunel (2000), but with significantly stronger synapses (Gewaltig and Koerner 2008). The first simulation (*black raster*) runs for 1,500 ms. The simulation is then repeated with identical initial conditions (*gray raster*), but after 200 ms, the membrane potential of a single neuron (*black circle*) is increased by 10^{-13} mV. From roughly 400 ms onwards, spike trains in both simulations differ significantly. Simulations were performed with NEST 2.0.0-RC2 using the iaf_psc_alpha_canon model neuron with precise spike timing (Morrison et al. 2007)

i5 CPU under Mac OSX 10.6.6. Source code for this and the other examples in this chapter is available from http://www.nest-initiative.org.

This result implies that even a model containing only equations as simple as (4.1) is not cross-replicable even if the (exact) method of iteration of (4.3) is specified as part of the model definition. Precise results depend on which of several mathematically equivalent numerical implementations is used. Clearly, different simulators should be allowed to implement exact solvers for (4.1) in different ways, and even if the solver of (4.3) were prescribed by the model, its implementation should not be. Indeed, when using modern simulators based on automatic code generation (Goodman 2010a), the computational scientist may not have any control over the precise implementation. Thus, even the simplest models of neural dynamics are numerically ambiguous, and it would be futile to expect exact model replicability across different simulation software even if detailed model specifications are provided.

One may raise the question, though, whether the errors illustrated in Fig. 4.1 are so minuscule that they may safely be ignored. Generally, the answer is no: many neuronal network models are exquisitely sensitive to extremely small errors, as illustrated in Fig. 4.2. This figure shows results from two simulations of 1,500 ms of activity in a balanced network of 1,250 neurons (80 % excitatory) based on Brunel

(2000), but with significantly stronger synapses (Gewaltig and Koerner 2008); see Fig. 4.4 for a summary overview of the model. In the second simulation run, the membrane potential of a single neuron is changed by 10^{-13} mV after 200 ms. Soon after, the spike activity in the two simulation runs has diverged entirely. In practice, this means that a scientist trying to replicate a model may obtain very different spike train results than those in the original publication. This holds both for cross-replication using different simulator software and for external replication in a different computational environment; the latter may even affect internal replication, cf. section "Is Code Sharing Enough to Ensure Replicability?" Gleeson et al. (2010) recently provided an example of the difficulties of cross-replication. They defined a network of 56 multicompartment neurons with 8,100 synapses using the descriptive NeuroML language and then simulated the model using the NEURON (Hines 1989; Hines and Carnevale 1997; Carnevale and Hines 2006), GENESIS (Bower and Beeman 1997), and MOOSE (http://moose.ncbs.res.in) software packages. These simulators generated different numbers of spikes in spite of the use of a very small time step (0.001 ms). Gleeson et al. concluded that "[t]hese results show that the way models are implemented on different simulators can have a significant impact on their behavior."

Computationally Underspecified Models

In the previous section we saw that even if the mathematics of a model are fully specified, numerical differences between implementations can lead to deviating model behavior. We shall now turn to models which may appear to be fully specified, but in fact leave important aspects to the simulation software. We refer to these models as *computationally underspecified*, as their specifications can be considered complete from a neuroscience point of view. We shall consider two cases in particular, spike timing and connection generation.

Most publications based on integrate-and-fire neurons contain a statement such as the following: "A spike is recorded when the membrane potential crosses the threshold, then the potential is reset to the reset potential." In many publications, though, it is not further specified at precisely which time the spike is recorded and the potential reset. As many network simulations are simulated on a fixed time grid, one can only assume that both events happen at the end of the time step during which the membrane potential crossed the threshold. Hansel et al. (1998) were the first to point out that tying membrane potential resets to a grid introduces a spurious regularity into network simulations that may, for example, lead to synchronization of firing activity. This observation spurred a quest for efficient methods for simulating networks with precisely timed spikes and resets (Hansel et al. 1998; Shelley and Tao 2001; Brette 2006, 2007; Morrison et al. 2007; Hanuschkin et al. 2010; D'Haene 2010).

Let us now consider how connections within neuronal network models are specified. Nordlie et al. (2009) demonstrated that connectivity information is often poorly defined in the literature, but even seemingly complete connectivity descriptions, such as given by Brunel (2000), will not ensure replicability across simulators. Specifically, Brunel states that each excitatory neuron receives input from 10 % of the excitatory neurons, chosen at random from a uniform distribution. Although different simulators may establish the correct number of connections using uniformly chosen sources, generally no two simulators will create the same network since the process of randomly drawing sources is implemented differently across simulators. Even if one prescribes which random number generators and seed values to use, little is gained unless the simulators use the random numbers in identical ways.

The only way to ensure that different software will create identical networks is to specify how the simulator iterates across nodes while creating connections. For example, the PyNN package (Davison et al. 2009) ensures that identical networks are generated in different simulators by iterating across nodes within PyNN and issuing explicit connection commands to the simulators, which will in general be slower than allowing the simulators to use their own internal routines. One might argue that such detail is beyond the scope of models created to explain brain function—after all, there is no biological counterpart to the arbitrary neuron enumeration schemes found in simulators, which naturally leads to a discussion of how one should define successful reproduction of modeling results.

Defining Successful Reproduction

Using proper documentation tools (see section "Practical Approaches to Replicability"), we can in principle achieve internal and external replicability in the short and long term. But this guarantees no more than that the same script on the same simulator generates the same results. As we have seen, in many cases we cannot take a simulation from one simulator to another and hope to obtain identical spike trains or voltage traces. Thus, there is no easy way to test the correctness of our simulations.

For many physical systems, a scientist can rely on a conservation law to provide checks and balances in simulation studies. As an example, Fig. 4.3 shows the movements of three point masses according to Newton's gravitational law where energy should be conserved in the system. Integrating the equations of motion using a forward Euler method yields incorrect results; when simulating with forward Euler, the total energy jumps to a much higher level when two planets pass close by each other. Fortunately, it is straightforward to compute the energy of the three-body system at any time, and a simulation using the LSODA algorithm (Petzold 1983) shows only a brief glitch in energy, demonstrating a better choice of numerical method. In the same manner, Ferrenberg et al. (1992) discovered important weaknesses in random number generators that were thought to be reliable when they observed implausible values for the specific heat of crystals in simulation experiments. The specific heat, a macroscopic quantity, was independently known from

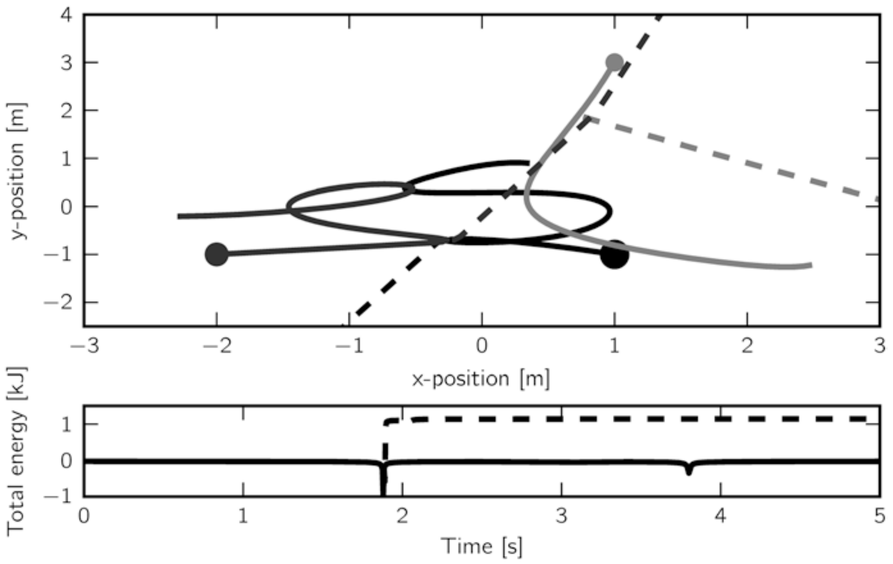

Fig. 4.3 Numerical solution of the Pythagorean planar three-body problem (Gruntz and Waldvogel 2004): Three point masses of 3 kg (*light gray*), 4 kg (*dark gray*), and 5 kg (*black*) are placed at rest at the locations marked by *circles* and move under the influence of their mutual gravitational attraction with gravitational constant $G = 1\ \mathrm{m}^3\ \mathrm{kg}^{-1}\ \mathrm{s}^{-2}$. *Top*: *Solid lines* show trajectories for the three bodies up to $T = 5$ s obtained with the LSODA algorithm (Petzold 1983) provided by the SciPy Python package (Jones et al. 2001), using a step size of 0.005 s; these agree with the trajectories depicted in Gruntz and Waldvogel (2004). *Dashed lines* show trajectories obtained using a custom forward Euler algorithm using the same step size. The trajectories coincide initially but diverge entirely as the black and dark gray planets pass each other closely. *Bottom*: Total energy for the solutions obtained using LSODA (*solid*) and forward Euler (*dashed*). While total energy remains constant in the LSODA solution except for short glitches around near encounters, the forward Euler solution "creates" a large amount of energy at the first close encounter, when trajectories begin to diverge

thermodynamical theory, thus providing a method for testing the simulations. Unfortunately, there are no known macroscopic laws governing neuronal dynamics (e.g., for the number of spikes in a network, for the resonance frequencies).

Twenty years ago Grebogi et al. (1990) posed a question for simulations of chaotic systems: "For a physical system which exhibits chaos, in what sense does a numerical study reflect the true dynamics of the actual system?" An interesting point in this respect is that the model which is based most closely on the physical system may not yield the best solution. Early models of atmospheric circulations were beset with numerical instabilities, severely limiting the time horizon of simulations. This changed only when Arakawa introduced some nonphysical aspects in atmospheric models which ensured long-term stability and produced results in keeping with meteorological observations (Küppers and Lenhard 2005). Similarly, the computational neuroscience community should embark on a careful discussion of criteria for evaluating the results of neuronal network simulations. Perhaps the same criteria for evaluating whether a model network provides a good model of neural activity should be used to compare the results of two different neuronal network simulations. Regardless, computational scientists must be able to distinguish between numerical or programming errors and differences in simulator implementations.

Practical Approaches to Replicability

As noted in the "Introduction," there are some intermediate steps between pure internal replication of a result (the original author rerunning the original code) and fully independent reproduction, namely external replication (someone else rerunning the original code) and cross-replication (running cross-platform code on a different simulation platform or re-implementing a model with knowledge of the original code). Both of these intermediate steps rely on sharing code, the simplest form of model sharing (Morse 2007). Code sharing allows other researchers to rerun simulations and easily extend models, facilitating a modular, incremental approach to computational science.

Approaches to Code Sharing

There are a number of possible methods for sharing code: by e-mail, on request; as supplementary material on a publisher's web-site; on a personal web-site; on a public source-code repository such as SourceForge (http://sourceforge.net), GitHub (https://github.com), BitBucket (https://bitbucket.org), or Launchpad (https://launchpad.net); or in a curated database such as ModelDB (Peterson et al. 1996; Davison et al. 2002; Migliore et al. 2003; Hines et al. 2004, http://senselab.med.yale.edu/modeldb), the Visiome platform (Usui 2003, http://visiome.neuroinf.jp), the BioModels database (Le Novere et al. 2006, http://www.biomodels.net), or the CellML Model Repository (http://models.cellml.org).

Sharing on request is the simplest option for a model author at the time of publication, but does not provide a public reference to which any extensions or derived models can be compared, and has the risk that contacting an author may not be straightforward if his/her e-mail address changes, he/she leaves science, etc. This option can also lead to future problems for the author, if the code cannot be found when requested, or "suddenly" produces different results.

Many journals offer the possibility of making model code available as supplementary material attached to a journal article, which makes it easy to find the model associated with a particular paper. The main disadvantages of this option are (1) lack of standardization in the format of the code archive or the associated metadata; (2) difficulty in updating the code archive and lack of versioning information if bugs are found, improvements are made, or contact details are changed; and (3) quality control—article referees may check that the code runs and is correct, but this is not universal. Also, not all journals offer the option of supplementary material; and some are moving away from offering it (Maunsell 2010).

A personal or institutional web-site allows the authors to more easily make updates to the code and to maintain a list of versions. The same lack of standardization of archive formats and metadata as for publisher sites exists. The major disadvantage is discoverability: it may not always be easy to find the author's web-site in the first place, for example, in case of a change of affiliation when hosting on an institutional site, expiration of domain name registrations for personal sites, or internal site reorganizations that break links. A further disadvantage is even less quality control than for supplementary material. An attempt to address the discoverability and metadata standardization problems was made by Cannon et al. (2002), who developed an infrastructure for distributed databases of models and data in which each group maintains its own local database and a central catalogue server federates them into a single virtual resource. This idea did not take off at the time, but is similar to the approach now being taken by the Neuroscience Information Framework (Gardner et al. 2008; Marenco et al. 2010).

Public general-purpose source-code repositories have many nice features for code sharing: versioning using mainstream version control tools, standard archive formats for downloading, some standardization of metadata (e.g., authors, version, programming language), issue tracking, wikis for providing documentation, and the stability of URLs. The disadvantages are possible problems with discoverability (SourceForge, for example, hosts over 260,000 projects. Finding which if any of these is a neuroscience model could be challenging), lack of standardization of neuroscience-specific metadata (which cell types, brain regions, etc.), and lack of external quality control.

Curated model repositories, in which a curator verifies that the code reproduces one or more figures from the published article, and which often have standardized metadata which make it easier to find models of a certain type (e.g., models of cortical pyramidal cells), address the issues of quality control, discoverability, and standardization (Lloyd et al. 2008). They perhaps lack some of the features available with public general-purpose code repositories, such as easy version tracking, issue tracking, and documentation editing, although the CellML Model Repository has fully integrated the Mercurial version control system (http://mercurial.selenic.com) into its site through the concept of a workspace for each model, and ModelDB has begun experimenting with integration of Mercurial.

Currently, the best solution for an author who wishes to share the code for their published model is probably to maintain the code in a public general-purpose repository such as GitHub or BitBucket (for the very latest version of the model and to maintain a record of previous versions) and also to have an entry for the model in a curated database such as ModelDB (for the latest version to have been tested by the curators, together with neuroscience-specific metadata). This recommendation may change as curated repositories become more feature-rich over time.

When sharing code, intellectual property issues should be considered. Presently, most researchers in computational neuroscience do not provide an explicit license when sharing their code, perhaps assuming that they are placing it in the public domain or that general scientific principles of attribution will apply when others reuse their code. For much more information on legal issues related to reproducible research see Stodden (2009a, b).

Steps from Replicability to Reproducibility

One approach to cross-replicability is the use of simulator-independent formats for describing models. These may be divided into declarative and programmatic—although here again the distinction is not always clear cut, since programming languages can be used in a declarative way, and not all declarative descriptions are really simulator-independent. Declarative model and simulation experiment specification is discussed in the next section. Here we consider programmatic simulator-independent formats.

A simulator-independent/simulator-agnostic programming interface allows the code for a simulation to be written once and then run on different simulator engines. Unlike declarative specifications, such a description is immediately executable without an intermediate translation step, which gives a more direct link between description and results. The use of a programming language also provides the full power of such a language, with loops, conditionals, subroutines, and other programming constructs. The great flexibility and extensibility this gives can be a strong advantage, especially in an exploratory phase of model building. It may also be a disadvantage if misused, leading to unnecessary complexity, bugs, and difficulty in understanding the essential components of the model, which are less common with declarative specifications.

In neuroscience, we are aware of only one such simulator-independent interface, PyNN (Davison et al. 2009), which provides an API in the Python programming language, and supports computational studies using the software simulators NEURON (Hines 1989; Hines and Carnevale 1997; Carnevale and Hines 2006), NEST (Gewaltig and Diesmann 2007; Eppler et al. 2008), PCSIM (Pecevski et al. 2009), and Brian (Goodman and Brette 2008), as well as a number of neuromorphic hardware systems (Brüderle et al. 2009; Galluppi et al. 2010). This is not a recent idea: as mentioned in the "Introduction," at one time the NMODL language developed by Michael Hines for describing biophysical mechanisms in the NEURON simulator was extended by Hines and Upinder Bhalla to work with GENESIS (GMODL; Wilson et al. 1989), making it perhaps the first simulator-independent description in computational neuroscience (Hines and Carnevale 2000). To the best of our knowledge, however, GMODL no longer exists and it is certainly not compatible with the most recent evolutions of NMODL.

Is Code Sharing Enough to Ensure Replicability?

Sharing code is not a panacea for ensuring replicability. It is often the case that a given result from a published paper cannot be re-created with code that has been made available. The reasons for this may include: differences in the version of the simulator, the compiler, or of shared libraries that are used by either the simulator or the code; differences in the computing platform (e.g., 32-bit vs. 64-bit systems, changes in the operating system); or simply poor record-keeping on the part of the researcher. It is our experience that often the set of parameters used to obtain a particular figure is different from that stated in the published article, sometimes due to typographical errors. Finally, for older publications, the model may have been run originally on a platform which is no longer available.

A more systematic approach to record-keeping is essential for improving the replicability of simulation studies. An important first step is the use of version control systems so that handling the problem of tracking which version of a model and which parameters are used to produce a particular result or figure is as simple as making a note of the version number. Making the version number part of a filename or embedded comments is even better.

The problem of changing versions of simulators and their dependencies, and of a changing computing environment, may be addressed by, first, making note of the software version(s) used to produce a particular result (including compiler versions and options and library versions, when the software has been compiled locally). It may be possible to automate this process to a certain extent (see below). A second step may be to capture a snapshot of the computing environment, for example, using virtual machines. For models that were originally simulated on now-obsolete hardware, software emulators (see, for example, http://www.pdp11.org) are a possible solution. Another is for the original authors or curators to port the code to a newer system or to a declarative description when the original system nears the end of its life.

A further step would be to automate the record-keeping process as much as possible, using, for example, an electronic lab notebook to automatically record the version of all software components and dependencies, and automatically check that all code changes have indeed been committed to a version control system prior to running a simulation. One of the authors (APD) has recently initiated an open-source project to develop such an automated lab notebook for computational experiments. Sumatra (http://neuralensemble.org/sumatra) consists of a core library implemented in Python, together with a command-line interface and a web-interface that builds on the library; a desktop graphical interface is planned. Each of these interfaces enables (1) launching simulations with automated recording of provenance information (versions, parameters, dependencies, input data files, and output files) and (2) managing a simulation project (browsing, viewing, annotating, and deleting simulations). The command-line and web-interface are independent of a particular simulator, although some information (e.g., model code dependencies) can only be tracked if a plug-in is written for the simulation language of interest; plug-ins for NEURON, GENESIS, and Python are currently available.

A number of tools exist for enabling reproducible data analysis and modeling workflows in a visual environment, for example, Kepler (Ludäscher et al. 2006), Taverna (Oinn et al. 2006), and VisTrails (Silva et al. 2007). VisTrails is of particular interest since it focuses on tracking changes to workflows over time in a way that is particularly well suited to the often exploratory process of neuronal modeling. The main disadvantage of using such a workflow framework is the extra development burden of wrapping simulation software to fit within the framework.

Structured, Declarative Descriptions of Models and Simulations

As described above, the intermediate steps between replicability and reproducibility for computational models include the expression of the model in a simulator-independent format that can be used on a different computational platform from the original model. One approach is to use a declarative description of the model that is simulator-independent. Software and database developers in many fields, including neuroscience, have enthusiastically adopted EXtensible Markup Language (XML) technology (Bray et al. 1998) as an ideal representation for complex structures such as models and data, due to its flexibility and its relation to the HTML standard for web pages. Like HTML, XML is composed of text and tags that explicitly describe the structure and semantics of the content of the document. Unlike HTML, developers are free to define the tags and develop a specific XML-based markup language that is appropriate for their application. A major advantage of XML is that it provides a machine-readable language that is independent of any particular programming language or software encoding, which is ideal for a structured, declarative description that can provide a standard for the entire community.

A representation of a model in a specific markup language is essentially a text document that consists of XML descriptions of the components of the model. Usually, the structure of a valid XML document is defined using a number of XML Schema Definition (XSD) files. Using these, standard XML handling libraries can be used to check the validity of an XML document against the language elements. Once an XML file is known to be valid, the contents of the file can be transformed into other formats in a number of different ways. For example, an application can read the XML natively using one of the commonly used parsing frameworks such as SAX (Simple API for XML, http://sax.sourceforge.net) or DOM (Document Object Model, http://en.wikipedia.org/wiki/Document_Object_Model). An alternative approach is to transform the XML description into another text format that can be natively read by an application, which can be done using Extensible Stylesheet Language (XSL) files. For more details regarding the use of XML technology for declarative model descriptions, see Crook and Howell (2007).

A number of ongoing projects focus on the development of these self-documenting markup languages that are extensible and can form the basis for specific implementations covering a wide range of modeling scales in neuroscience. The Systems Biology Markup Language, SBML (Hucka et al. 2003), and CellML (Hedley et al. 2000; Lloyd et al. 2004) are two popular languages for describing systems of interacting biomolecules that comprise models often used in systems biology, and both languages can be used for describing more generic dynamical models, including neural models. NeuroML (Goddard et al. 2001; Crook et al. 2007; Gleeson et al. 2010) differs from these languages in that it is a domain-specific model description language, and neuroscience concepts such as cells, ion channels, and synaptic connections are an integral part of the language. The International Neuroinformatics Coordinating Facility aims to facilitate the development of markup language standards for model descriptions in neuroscience, and is providing support for the development of NineML (Network Interchange format for NEuroscience, http://nineml.org), which focuses on descriptions of spiking networks. Additionally, the Simulation Experiment Description Markup Language (SED-ML) (Köhn and Le Novère 2008) is a language for encoding the details of simulation experiments, which follows the requirements defined in the MIASE (Minimal Information about Simulation Experiments) guidelines (http://biomodels.net/miase). These markup languages are complementary and, taken together, they cover the scales for the majority of neuroscience models. The use of namespaces allows for unambiguous mixing of several XML languages; thus, it is possible to use multiple languages for describing different modules of a multiscale model.

Here we provide more details about these languages and their contexts for declarative descriptions of models and simulations. We also provide an introduction to how these markup languages can provide an infrastructure for model sharing, tool development and interoperability, and reproducibility.

SBML and CellML

The main focus of SBML is the encoding of models consisting of biochemical entities, or species, and the reactions among these species that form biochemical networks. In particular, models described in SBML are decomposed into their explicitly labeled constituent elements, where the SBML document resembles a verbose rendition of chemical reaction equations. The representation deliberately avoids providing a set of differential equations or other specific mathematical frameworks for the model, which makes it easier for different software tools to interpret the model and translate the SBML document into the internal representation used by that tool.

In contrast, CellML is built around an approach of constructing systems of equations by linking together the variables in those equations. This equation-based approach is augmented by features for declaring biochemical reactions explicitly, as well as grouping components into modules. The component-based architecture facilitates the reuse of models, parts of models, and their mathematical descriptions.

Note that SBML provides constructs that are more similar to the internal data objects used in many software packages for simulating and analyzing biochemical networks, but SBML and CellML have much in common and represent different approaches for solving the same general problems. Although they were initially developed independently, the developers of the two languages are engaged in exchanges of ideas and are seeking ways of making the languages more interoperable (Finney et al. 2006).

Both of these model description efforts are associated with model repositories that allow authors to share simulator-independent model descriptions. Currently, the BioModels database (Le Novere et al. 2006, http://www.biomodels.net) contains several hundred curated models, and even more non-curated models, that are available as SBML documents as well as in other formats. The CellML Model Repository (Lloyd et al. 2008, http://models.cellml.org) also contains hundreds of models that are available as CellML documents. In addition, both efforts have associated simulation tools and modeling environments, tools that validate XML documents for models against the language specifications, and translation utilities that are described in detail on their web-sites.

NeuroML

The declarative approach of the NeuroML standards project focuses on the key objects that need to be exchanged among existing applications with some anticipation of the future needs of the community. These objects include descriptions of neuronal morphologies, voltage-gated ion channels, synaptic mechanisms, and network structure. The descriptions are arranged into levels that are related to different biological scales, with higher levels adding extra concepts. This modular, object-oriented structure makes it easier to add additional concepts and reuse parts of models. As models of single neurons are at the core of most of the systems being described, neuroanatomical information about the structure of individual cells forms the core of Level 1, which also includes the specification for metadata. The focus of Level 2 is the electrical properties of these neurons which allows for descriptions of cell models with realistic channel and synaptic mechanisms distributed on their membranes. Level 3 describes networks of these cells in three dimensions including cell locations and synaptic connectivity. Networks can be described with an explicit list of instances of cell positions and connections, or with an algorithmic template for describing how the instances should be generated.

While there is overlap in the types of models that NeuroML and SBML/CellML can describe, such as a single compartment conductance-based model, NeuroML provides a concise format for neuronal model elements that can be readily understood by software applications that use the same concepts. NeuroML version 2, which is under development, will have greater interaction with SBML and CellML, with SBML being an initial focus of the work. This will allow, for example, complex signaling pathways to be expressed in one of these formats with the rest of the cell and network model specified in NeuroML. Since NeuroML is completely compatible with the structure of the user layer of NineML (see below), the goal is to be able to represent a multiscale neuroscience model that includes processes from the molecular to the network levels with a combination of SBML, NeuroML, and NineML.

The NeuroML project (http://neuroml.org) provides a validator for NeuroML documents. The project also provides XSL files for mapping NeuroML documents to a HTML format that provides detailed user-friendly documentation of the model details and for mapping NeuroML documents to a number of simulator scripting formats, including NEURON (Hines 1989; Hines and Carnevale 1997; Carnevale and Hines 2006), GENESIS (Bower and Beeman 1997), PSICS (http://www.psics.org), and additional simulators through PyNN (Davison et al. 2009). This approach has the advantage that in the short term, applications need not be extended to natively support NeuroML, but can still have access to NeuroML models.

NineML

With an increasing number of studies related to large-scale neuronal network modeling, there is a need for a common standardized description language for spiking network models. The Network Interchange for Neuroscience Modeling Language (NineML) is designed to describe large networks of spiking neurons using a layered approach. The abstraction layer provides the core concepts, mathematics, and syntax for explicitly describing model variables and state update rules, and the user layer provides a syntax to specify the instantiation and parameterization of the network model in biological terms. In particular, the abstraction layer is built around a block diagram notation for continuous and discrete variables, their evolution according to a set of rules such as a system of ordinary differential equations, and the conditions that induce a regime change, such as a transition from subthreshold mode to spiking and refractory modes. In addition, the abstraction layer provides the notation for describing a variety of topographical arrangements of neurons and populations of neurons (Raikov and The INCF Multiscale Modeling Taskforce 2010). In contrast, the user layer provides the syntax for specifying the model and the parameters for instantiating the network, which includes descriptions of individual elements such as cells, synapses, and synaptic inputs, as well as the constructs for describing the grouping of these entities into networks (Gorchetchnikov and The INCF Multiscale Modeling Taskforce 2010). Like NeuroML, the user layer of NineML defines the syntax for specifying a large range of connectivity patterns. One goal of NineML is to be self-consistent and flexible, allowing addition of new models and mathematical descriptions without modification of the previous structure and organization of the language. To achieve this, the language is being iteratively designed using several representative models with various levels of complexity as test cases (Raikov and The INCF Multiscale Modeling Taskforce 2010).

SED-ML

A simulation experiment description using SED-ML (Köhn and Le Novère 2008) is independent of the model encoding that is used in the computational experiment; the model itself is only referenced using an unambiguous identifier. Then SED-ML is used to describe the algorithm used for the execution of the simulation and the simulation settings such as step size and duration. SED-ML also can be used to describe changes to the model, such as changes to the value of an observable or general changes to any XML element of the model representation. The simulation result sometimes does not correspond to the desired output of the simulation. For this reason, SED-ML also includes descriptions of postsimulation processing that should be applied to the simulation result before output such as normalization, mean-value calculations, or any other new expression that can be specified using MathML (Miner 2005). SED-ML further allows for descriptions of the form of the output such as a 2D-plot, 3D-plot, or data table. So far, SED-ML has been used in conjunction with several different model description languages including SBML, CellML, and NeuroML, and the BioModels database supports the sharing of simulation descriptions in SED-ML.

Other Tools Based on Declarative Descriptions

neuroConstruct is an example of a successful software application that uses declarative descriptions to its advantage (Gleeson et al. 2007). This software facilitates the creation, visualization, and analysis of networks of multicompartmental neurons in 3D space, where a graphical user interface allows model generation and modification without programming. Models within neuroConstruct are based on the simulator-independent NeuroML standards, allowing automatic generation of code for multiple simulators. This has facilitated the testing of neuroConstruct and the verification of its simulator independence, through a process where published models were re-implemented using neuroConstruct and run on multiple simulators as described in section "Ambiguous Model Numerics" (Gleeson et al. 2010).

The ConnPlotter package (Nordlie and Plesser 2010) allows modelers to visualize connectivity patterns in large networks in a compact fashion. It thus aids in communicating model structures, but is also a useful debugging tool. Unfortunately, it is at present tightly bound to the NEST Topology Library (Plesser and Austvoll 2009).

Improving Research Reporting

The past 2 decades have brought a significant growth in the number of specialized journals and conferences, sustaining an ever growing volume of scientific communication. Search engines such as Google Scholar (http://scholar.google.com) and

Thomson Reuters Web of Knowledge (http://wokinfo.com) have revolutionized literature search, while the internet has accelerated the access to even arcane publications from weeks to seconds. Electronic publication permits authors to complement terse papers with comprehensive supplementary material, and some journals even encourage authors to post video clips in which they walk their audience through the key points of the paper.

But have these developments improved the communication of scientific ideas between researchers? Recently, Nordlie et al. (2009) surveyed neuronal network model descriptions in the literature and concluded that current practice in this area is diverse and inadequate. Many computational neuroscientists have experienced difficulties in reproducing results from the literature due to insufficient model descriptions. Donoho et al. (2009) propose as a cure that all scientists in a field should use the same software, where the software is carefully crafted to cover the complete modeling process from simulation to publication figure. While this approach successfully addresses software quality and replicability issues, it falls short of contributing to the independent reproduction of results, which by definition requires re-implementation of a model based on the underlying concepts, preferably using a different simulator.

To facilitate independent reproduction of neural modeling studies, a systematic approach is needed for reporting models, akin to the ARRIVE Guidelines for Reporting Animal Research (Kilkenny et al. 2010). Such guidelines can serve as checklists for authors as well as for referees during manuscript review. For neuronal network models, Nordlie et al. (2009) have proposed a good model description practice, recommending that publications on computational modeling studies should provide:

- Hypothesis: a concrete description of the question or problem that the model addresses.
- Model derivation: a presentation of experimental data that support the hypothesis, model, or both.
- Model description: a description of the model, its inputs (stimuli) and its outputs (measured quantities), and all free parameters.
- Implementation: a concise description of the methods used to implement and simulate the model (e.g., details of spike threshold detection, assignment of spike times, time resolution), as well as a description of all third party tools used, such as simulation software or mathematical packages.
- Model analysis: a description of all analytical and numerical experiments performed on the model, and the results obtained.
- Model justification: a presentation of all empirical or theoretical results from the literature that support the results obtained from the model and that were not used to derive the model.

Nordlie et al. also provide a checklist for model descriptions, requiring information on the following aspects of a model: (1) model composition, (2) coordinate systems and topology, (3) connectivity, (4) neurons, synapses, and channels, (5) model input, output, and free parameters, (6) model validation, and (7) model

A	Model Summary
Populations	Three: excitatory, inhibitory, external input
Topology	—
Connectivity	Random convergent connections
Neuron model	Leaky integrate-and-fire, fixed voltage threshold, fixed absolute refractory time (voltage clamp)
Channel models	—
Synapse model	α-shaped current inputs
Plasticity	—
Input	Independent fixed-rate Poisson spike trains to all neurons (during initial stimulation period)
Measurements	Spike activity

B		Populations
Name	**Elements**	**Size**
E	Iaf neuron	$N_{\mathrm{E}} = 4N_{\mathrm{I}}$
I	Iaf neuron	N_{I}
$\mathrm{E}_{\mathrm{ext}}$	Poisson generator	$C_E(N_{\mathrm{E}} + N_{\mathrm{I}})$

C			Connectivity
Name	**Source**	**Target**	**Pattern**
EE	E	E	Random convergent $C_{\mathrm{E}} \to 1$, weight J, delay D
IE	E	I	Random convergent $C_{\mathrm{E}} \to 1$, weight J, delay D
EI	I	E	Random convergent $C_{\mathrm{I}} \to 1$, weight $-gJ$, delay D
II	I	I	Random convergent $C_{\mathrm{I}} \to 1$, weight $-gJ$, delay D
Ext	$\mathrm{E}_{\mathrm{ext}}$	$\mathrm{E} \cup \mathrm{I}$	Non-overlapping $C_{\mathrm{E}} \to 1$, weight J, delay D

D	Neuron and Synapse Model
Name	Iaf neuron
Type	Leaky integrate-and-fire, α-current input
Subthreshold dynamics	$\tau\dot{V}(t) = -V(t) + RI(t)$ if $t > t^* + \tau_{\mathrm{rp}}$ $V(t) = V_{\mathrm{r}}$ else $I(t) = \frac{\tau}{R}\sum_{\tilde{t}} w\alpha(t - (\tilde{t} + \Delta))\Theta(t - (\tilde{t} + \Delta))$
Spiking	If $V(t-) < \theta \wedge V(t+) \geq \theta$ 1. set $t^* = t$ 2. emit spike with time-stamp t^*

E	Input
Type	**Description**
Poisson generators	Fixed rate ν_{ext}, C_{E} generators per neuron, each generator projects to one neuron; active only during initial stimulation period

F Measurements
Spike activity as raster plots for subset of excitatory neurons

Fig. 4.4 Concise tabular presentation of the network model introduced in section "Ambiguous Model Numerics" with spike trains shown in Fig. 4.2, using the template proposed by Nordlie et al. (2009)

implementation. They further propose a concise tabular format for summarizing network models in publications; Fig. 4.4 provides an example. These guidelines and tables present information about a model that referees can check for completeness and consistency, and also allow referees to judge whether the employed simulation methods appear adequate for the model, as well as the plausibility of the results obtained.

Publication standards such as those discussed in Nordlie et al. ensure that all possible, relevant model details are provided. However, it is also important to note what such guidelines and tables cannot provide: sufficient detail to allow exact replication of the simulation and of the figures presented in the publication for the reasons discussed in the sections above. Since it is futile to strive for details that would lead to exact replication of scientific publications, and since referees cannot confirm correctness without replicating all of the work in a manuscript, some have advocated that authors should focus only on main concepts that can be evaluated by referees, neglecting model details. Note that the Journal of Neuroscience recently ceased to publish supplementary material for the precise reason that it provides unreviewed (and essentially unreviewable) detail (Maunsell 2010).

Where do all of these issues leave reproducibility? It seems that improved reproducibility requires two important measures for reporting results. First, on the technical side, replicability of simulation studies should be ensured by requiring that authors use proper code sharing techniques and automated practices for recording provenance as described in section "Practical Approaches to Replicability." If code is not reviewed, then this is best done through a curated database rather than in the supplementary material. This model deposition provides a reference for readers if they encounter difficulties in reproducing the results of a publication. Note that the increasing use of a limited set of simulator software packages (Brette et al. 2007) facilitates this type of model archeology due to the widespread expertise with these packages in the computational neuroscience community—no need to decipher Fortran code left behind by the Ph.D.-student of yesteryear.

Second, publications in computational neuroscience should provide much more information about why a particular model formulation was chosen and how model parameters were selected. Neural models commonly require significant parameter tuning to demonstrate robust, stable, and interesting dynamics. In some cases, the selection of models for synapses and excitable membranes may be shaped by neurophysiological evidence, while in others, they are selected based on ease of implementation or mathematical analysis. Such choices should be clearly articulated, and much would be gained by details about (1) which aspects of the model proved essential to obtaining "good" simulation results and (2) which quantities and properties constrained parameter tuning. In discussing these aspects, it might be helpful to consider recent advances in the theory of science with regard to both the role of simulations in the scientific process (Humphreys 2004) and the role of explanatory models that are common in neuroscience (Craver 2007).

Discussion

In this chapter we have exposed the distinction between replicability and reproducibility, and explored the continuum between the two. We also have seen that there are scientific benefits to promoting each point on the continuum. Replication based on reuse of code enables more rapid progress of computational studies by promoting modularization, improved code quality, and incremental development, while independent reproduction of an important result (whether manually, through reading an article's Methods section, or (semi-)automatically, using a declarative, machine-readable version of the methods) remains as the gold standard and foundation of science.

In our terminology, replication of results implies reusing the original code in some way, either by rerunning it directly or by studying it when developing a new implementation of a model. If this replication is done by someone other than the original developers, the code must therefore be shared with others. This raises many issues including licensing, version tracking, discoverability, documentation, software/hardware configuration tracking, and obsolescence.

Independent reproduction of a computational experiment without using the original code is necessary to confirm that a model's results are general and do not depend on a particular implementation. Traditionally, this has been done by reading the published article(s) describing the models, and most often corresponding with the authors to clarify incomplete descriptions. In this chapter we have discussed in some depth the difficulties usually encountered in this process, and ways in which published model descriptions can be improved. We summarize these recommendations below. More recently, several efforts have been made to produce structured, declarative, and machine-readable model descriptions, mostly based on XML. Such structured descriptions allow the completeness and internal consistency of a description to be verified, and allow for automated reproduction of simulation experiments in different simulation environments.

In considering how to improve the reproducibility of computational neuroscience experiments, it is important to be aware of the limits of reproducibility, due to component failure, environmental influences on hardware, floating point numerics, and the amplification of small errors by sensitive model systems. The question of how best to determine whether differences between two simulations are due to unavoidable computational effects or whether they reflect either errors in the code or important algorithmic differences has not been satisfactorily answered in computational neuroscience.

Based on the issues identified and discussed in this chapter, we propose a number of steps that can be taken to improve replication and reproduction of computational neuroscience experiments:

Modelers

- Use version control tools.
- Keep very careful records of the computational environment, including details of hardware, operating system versions, versions of key tools, and software libraries. Use automated tools where available.
- Use best practices in model publications; see Nordlie et al. (2009).
- Plan to release your code from the beginning of development to aid in code sharing.
- Make code available through ModelDB, BioModels, or other appropriate curated databases.

- Make models available using simulator-independent model descriptions if possible.
- Evaluate your career by downloads and users in addition to citations.

Tool developers

- Incorporate version control tools and tools for automated environment tracking into your software.
- Collaborate with model description language efforts.

Reviewers and editors

- Demand clear model descriptions following Nordlie et al. (2009).
- Demand and verify code and/or model availability.
- Make sure publications include details of model choices and behavior.

Computational neuroscience has made enormous progress in the past 20 years. Can the same be said for the reproducibility of our models and our results? The range and quality of the available tools for ensuring replicability and reproducibility has certainly improved, from better version control systems to structured, declarative model description languages and model databases. At the same time, the typical complexity of our models has also increased, as our experimental colleagues reveal more and more biological detail and Moore's Law continues to put more and more computing power in our laboratories. It is this complexity which is perhaps the major barrier to reproducibility. As the importance of computational science in scientific discovery and public policy continues to grow, demonstrable reproducibility will become increasingly important. Therefore, it is critical to continue the development of tools and best practices for managing model complexity and facilitating reproducibility and replicability. We must also attempt to change the culture of our computational community so that more researchers consider whether their reported results can be reproduced and understand what tools are available to aid in reproducibility. These changes are needed so that reproducibility can be front and center in the thinking of modelers, reviewers, and editors throughout the computational neuroscience community.

Acknowledgments S.C. acknowledges support from the National Institute of Mental Health under grant R01MH061905. A.P.D. acknowledges support from the Centre National de la Recherche Scientifique and from the European Union under grant FP7-ICT-FET-269921 (BrainScaleS). H.E.P. acknowledges support from the Research Council of Norway under grant 178892/V30 eNeuro.

References

Bower JM, Beeman D (1997) The book of GENESIS: exploring realistic neural models with the GEneral NEural SImulation System. Springer, New York

Bhalla U, Bilitch DH, Bower J (1992) Rallpacks: a set of benchmarks for neural simulators. Trends Neurosci 15:453–458

Bray T, Paoli J, Sperberg-McQueen C (1998) Extensible markup language (XML) 1.0. http://www.w3.org/TR/REC-xml

Breslow NE (2010) Commentary. Biostatistics 11(3):379–380. doi:10.1093/biostatistics/kxq025

Brette R (2006) Exact simulation of integrate-and-fire models with synaptic conductances. Neural Comput 18:2004–2027

Brette R (2007) Exact simulation of integrate-and-fire models with exponential currents. Neural Comput 19:2604–2609

Brette R, Rudolph M, Carnevale T, Hines M, Beeman D, Bower JM, Diesmann M, Morrison A, Goodman PH, Harris FC Jr, Zirpe M, Natschläger T, Pecevski D, Ermentrout B, Djurfeldt M, Lansner A, Rochel O, Vieville T, Muller E, Davison AP, Boustani SE, Destexhe A (2007) Simulation of networks of spiking neurons: a review of tools and strategies. J Comput Neurosci 23:349–398

Brüderle D, Muller E, Davison A, Muller E, Schemmel J, Meier K (2009) Establishing a novel modeling tool: a Python-based interface for a neuromorphic hardware system. Front Neuroinform 3:17. doi:10.3389/neuro.11.017.2009

Brunel N (2000) Dynamics of sparsely connected networks of excitatory and inhibitory spiking neurons. J Comput Neurosci 8:183–208

Cannon R, Howell F, Goddard N, De Schutter E (2002) Non-curated distributed databases for experimental data and models in neuroscience. Network 13:415–428

Cannon RC, Gewaltig MO, Gleeson P, Bhalla US, Cornelis H, Hines ML, Howell FW, Muller E, Stiles JR, Wils S, Schutter ED (2007) Interoperability of neuroscience modeling software: current status and future directions. Neuroinformatics 5:127–138

Carnevale N, Hines M (2006) The NEURON book. Cambridge University Press, Cambridge, UK

Claerbout JF, Karrenbach M (1992) Electronic documents give reproducible research a new meaning. In: SEG expanded abstracts, Society of Exploration Geophysicists, vol 11, pp 601–604

Cox DR, Donnelly C (2010) Commentary. Biostatistics 11(3):381–382. doi:10.1093/biostatistics/kxq026

Craver CF (2007) Explaining the brain: mechanisms and the Mosaic Unity of Neuroscience. Oxford University Press, New York

Crook S, Howell F (2007) XML for data representation and model specification. In: Crasto C (ed) Methods in molecular biology book series: neuroinformatics. Humana Press, Totowa, NJ, pp 53–66

Crook S, Gleeson P, Howell F, Svitak J, Silver R (2007) MorphML: Level 1 of the NeuroML standards for neuronal morphology data and model specification. Neuroinformatics 5:96–104

D'Haene M (2010) Efficient simulation strategies for spiking neural networks. PhD thesis, Universiteit Gent

Davison A, Morse T, Migliore M, Marenco L, Shepherd G, Hines M (2002) ModelDB: a resource for neuronal and network modeling. In: Kötter R (ed) Neuroscience databases: a practical guide. Kluwer Academic, Norwell, MA, pp 99–122

Davison A, Brüderle D, Eppler J, Kremkow J, Muller E, Pecevski D, Perrinet L, Yger P (2009) PyNN: a common interface for neuronal network simulators. Front Neuroinform 2:11. doi:10.3389/neuro.11.011.2008

de Leeuw J (2001) Reproducible research: the bottom line. Tech. Rep., Department of Statistics, UCLA, UC Los Angeles. http://escholarship.org/uc/item/9050x4r4

DeAngelis CD, Fontanarosa PB (2010) The importance of independent academic statistical analysis. Biostatistics 11(3):383–384. doi:10.1093/biostatistics/kxq027

Djurfeldt M, Lansner A (2007) Workshop report: 1st INCF Workshop on Large-scale Modeling of the Nervous System. Available from Nature Precedings http://dx.doi.org/10.1038/npre.2007.262.1

Donoho DL (2010) An invitation to reproducible computational research. Biostatistics 11(3):385–388. doi:10.1093/biostatistics/kxq028

Donoho DL, Maleki A, Rahman IU, Shahram M, Stodden V (2009) 15 years of reproducible research in computational harmonic analysis. Comput Sci Eng 11:8–18. doi:10.1109/MCSE.2009.15

Drummond C (2009) Replicability is not reproducibility: nor is it good science. In: Proceedings of the evaluation methods for machine learning workshop at the 26th ICML, Montreal, CA

Eppler JM, Helias M, Muller E, Diesmann M, Gewaltig MO (2008) PyNEST: a convenient interface to the NEST simulator. Front Neuroinform 2:12. doi:10.3389/neuro.11.012.2008

Ferrenberg AM, Landau DP, Wong YJ (1992) Monte Carlo simulations: hidden errors from "good" random number generators. Phys Rev Lett 69(23):3382–3384. doi:10.1103/PhysRevLett.69.3382

Finney A, Hucka M, Bornstein B, Keating S, Shapiro B, Matthews J, Kovitz B, Schilstra M, Funahashi A, Doyle J, Kitano H (2006) Software infrastructure for effective communication and reuse of computational models. In: Szallasi Z, Stelling J, Periwal V (eds) Systems modeling in cell biology: from concepts to nuts and bolts. MIT Press, Boston, pp 369–378

Funk W, Dammann V, Donnevert G (2006) Quality assurance in analytical chemistry, 2nd edn. Wiley-VCH, Weinheim

Galluppi F, Rast A, Davies S, Furber S (2010) A general-purpose model translation system for a universal neural chip. In: Wong K, Mendis B, Bouzerdoum A (eds) Neural information processing. Theory and algorithms, Lecture notes in computer science, vol 6443. Springer, Berlin, pp 58–65

Gardner D, Knuth KH, Abato M, Erde SM, White T, DeBellis R, Gardner EP (2001) Common data model for neuroscience data and data model exchange. J Am Med Inform Assoc 8:17–33

Gardner D, Akil H, Ascoli G, Bowden D, Bug W, Donohue D, Goldberg D, Grafstein B, Grethe J, Gupta A, Halavi M, Kennedy D, Marenco L, Martone M, Miller P, Müller H, Robert A, Shepherd G, Sternberg P, Van Essen D, Williams R (2008) The neuroscience information framework: a data and knowledge environment for neuroscience. Neuroinformatics 6(3):149–160

Gewaltig MO, Diesmann M (2007) NEST (NEural Simulation Tool). Scholarpedia 2(4):1430

Gewaltig MO, Koerner EE (2008) Self-sustained dynamics of sparsely connected networks without external noise. In: 2008 Neuroscience meeting planner, Society for Neuroscience, Washington, DC, program No. 220.1

Gleeson P, Steuber V, Silver RA (2007) neuroConstruct: a tool for modeling networks in 3D space. Neuron 54:219–235

Gleeson P, Crook S, Cannon RC, Hines ML, Billings GO, Farinella M, Morse TM, Davison AP, Ray S, Bhalla US, Barnes SR, Dimitrova YD, Silver RA (2010) NeuroML: a language for describing data driven models of neurons and networks with a high degree of biological detail. PLoS Comput Biol 6(6):e1000815. doi:10.1371/journal.pcbi.1000815

Goddard N, Hucka M, Howell F, Cornelis H, Shankar K, Beeman D (2001) NeuroML: model description methods for collaborative modelling in neuroscience. Philos Trans R Soc Lond B Biol Sci 356:1209–1228

Goodman D (2010a) Code generation: a strategy for neural network simulators. Neuroinformatics 8:183–196. doi:10.1007/s12021-010-9082-x

Goodman SN (2010b) Commentary. Biostatistics 11(3):389–390. doi:10.1093/biostatistics/kxq030

Goodman D, Brette R (2008) Brian: a simulator for spiking neural networks in Python. Front Neuroinform 2:5. doi:10.3389/neuro.11.005.2008

Gorchetchnikov A, The INCF Multiscale Modeling Taskforce (2010) NineML—a description language for spiking neuron network modeling: the user layer. BMC Neurosci 11(suppl 1):P71

Grebogi C, Hammel SM, Yorke JA, Sauer T (1990) Shadowing of physical trajectories in chaotic dynamics: containment and refinement. Phys Rev Lett 65(13):1527–1530. doi:10.1103/PhysRevLett.65.1527

Groves T (2010) The wider concept of data sharing: view from the BMJ. Biostatistics 11(3):391–392. doi:10.1093/biostatistics/kxq031

Gruntz D, Waldvogel J (2004) Orbits in the planar three-body problem. In: Gander W, Hřebíček J (eds) Solving problems in scientific computing using Maple and MATLAB, 4th edn. Springer, Berlin, pp 51–72

Hansel D, Mato G, Meunier C, Neltner L (1998) On numerical simulations of integrate-and-fire neural networks. Neural Comput 10:467–483

Hanuschkin A, Kunkel S, Helias M, Morrison A, Diesmann M (2010) A general and efficient method for incorporating exact spike times in globally time-driven simulations. Front Neuroinform 4:113. doi:10.3389/fninf.2010.00113

Hedley W, Nelson M, Nielsen P, Bullivant D, Hunter P (2000) XML languages for describing biological models. In: Proceedings of the Physiological Society of New Zealand, vol 19

Heijmen T (2011) Soft errors from space to ground: historical overview, empirical evidence, and future trends (chap 1). In: Nicolaidis M (ed) Soft errors in modern electronic systems, Frontiers in electronic testing, vol 41. Springer, New York, pp 1–25

Hines M (1989) A program for simulation of nerve equations with branching geometries. Int J Biomed Comput 24:55–68

Hines ML, Carnevale NT (1997) The NEURON simulation environment. Neural Comput 9(6):1179–1209

Hines ML, Carnevale NT (2000) Expanding NEURON's repertoire of mechanisms with NMODL. Neural Comput 12:995–1007

Hines ML, Morse T, Migliore M, Carnevale NT, Shepherd GM (2004) ModelDB: a database to support computational neuroscience. J Comput Neurosci 17(1):7–11. doi:10.1023/B:JCNS.0000023869.22017.2e

Hucka M, Finney A, Sauro H, Bolouri H, Doyle J, Kitano H, Arkin A (2003) The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. Bioinformatics 19:524–531

Humphreys P (2004) Extending ourselves: computational science, empiricism, and scientific method. Oxford University Press, Oxford

Hund F (1996) Geschichte der physikalischen Begriffe. Spektrum Akademischer Verlag, Heidelberg

Ioannidis JPA (2005) Why most published research findings are false. PLoS Med 2:e124. doi:10.1371/journal.pmed.0020124

Jones E, Oliphant T, Peterson P et al (2001) SciPy: open source scientific tools for Python. http://www.scipy.org/

Keiding N (2010a) Reproducible research and the substantive context. Biostatistics 11(3):376–378. doi:10.1093/biostatistics/kxq033

Keiding N (2010b) Reproducible research and the substantive context: response to comments. Biostatistics 11(3):395–396. doi:10.1093/biostatistics/kxq034

Kilkenny C, Browne WJ, Cuthill IC, Emerson M, Altman DG (2010) Improving bioscience research reporting: the ARRIVE guidelines for reporting animal research. PLoS Biol 8(6):e1000412. doi:10.1371/journal.pbio.1000412

Köhn D, Le Novère N (2008) SED-ML—an XML format for the implementation of the MIASE guidelines. In: Heiner M, Uhrmacher A (eds) Computational methods in systems biology, Lecture notes in computer science, vol 5307. Springer, Berlin, pp 176–190

Küppers G, Lenhard J (2005) Validation of simulation: patterns in the social and natural sciences. J Artif Soc Soc Simul 8(4):3

Lapicque L (1907) Recherches quantitatives sur l'excitation électrique des nerfs traitée comme une polarization. J Physiol Pathol Gen 9:620–635

Le Novere N, Bornstein B, Broicher A, Courtot M, Donizelli M, Dharuri H, Li L, Sauro H, Schilstra M, Shapiro B, Snoep J, Hucka M (2006) BioModels database: a free, centralized database of curated, published, quantitative kinetic models of biochemical and cellular systems. Nucleic Acids Res 34(Database issue):D689–D691

Lehrer J (2010) The truth wears off. The New Yorker, 13 Dec 2010:52

Lloyd C, Halstead M, Nielsen P (2004) CellML: its future, present and past. Prog Biophys Mol Biol 85:433–450

Lloyd CM, Lawson JR, Hunter PJ, Nielsen PF (2008) The CellML model repository. Bioinformatics 24(18):2122–2123. doi:10.1093/bioinformatics/btn390

Ludäscher B, Altintas I, Berkley C, Higgins D, Jaeger-Frank E, Jones M, Lee E, Tao J, Zhao Y (2006) Scientific workflow management and the Kepler system. Concurrency Comput Pract Exp 18(10):1039–1065

Marenco L, Wang R, Shepherd G, Miller P (2010) The NIF DISCO framework: facilitating automated integration of neuroscience content on the web. Neuroinformatics 8(2):101–112

Maunsell J (2010) Announcement regarding supplemental material. J Neurosci 30:10599–10600

Mesirov JP (2010) Accessible reproducible research. Science 327(5964):415–416. doi:10.1126/science.1179653

Migliore M, Morse TM, Davison AP, Marenco L, Shepherd GM, Hines ML (2003) ModelDB: making models publicly accessible to support computational neuroscience. Neuroinformatics 1(1):135–139

Miner R (2005) The importance of MathML to mathematical communication. Notices AMS 52:532–538

Morrison A, Straube S, Plesser HE, Diesmann M (2007) Exact subthreshold integration with continuous spike times in discrete time neural network simulations. Neural Comput 19:47–79

Morse TM (2007) Model sharing in computational neuroscience. Scholarpedia 2(4):3036

Nordlie E, Plesser HE (2010) Visualizing neuronal network connectivity with connectivity pattern tables. Front Neuroinform 3:39. doi:10.3389/neuro.11.039.2009

Nordlie E, Gewaltig MO, Plesser HE (2009) Towards reproducible descriptions of neuronal network models. PLoS Comput Biol 5(8):e1000456. doi:10.1371/journal.pcbi.1000456

Oinn T, Greenwood M, Addis M, Alpdemir M, Ferris J, Glover K, Goble C, Goderis A, Hull D, Marvin D, Li P, Lord P, Pocock MR, Senger M, Stevens R, Wipat A, Wroe C (2006) Taverna: lessons in creating a workflow environment for the life sciences. Concurr Comput Pract Exp 18(10):1067–1100. doi:10.1002/cpe.993

Pecevski D, Natschläger T, Schuch K (2009) PCSIM: a parallel simulation environment for neural circuits fully integrated with Python. Front Neuroinform 3:11. doi:10.3389/neuro.11.011.2009

Peng RD (2009) Reproducible research and biostatistics. Biostatistics 10(3):405–408. doi:10.1093/biostatistics/kxp014

Peng RD (2010) Discussion of Keiding. Biostatistics 11(3):393–394. doi:10.1093/biostatistics/kxq032

Peterson B, Healy M, Nadkarni P, Miller P, Shepherd G (1996) ModelDB: an environment for running and storing computational models and their results applied to neuroscience. J Am Med Inform Assoc 3:389–398

Petzold L (1983) Automatic selection of methods for solving stiff and nonstiff systems of ordinary differential equations. SIAM J Sci Comput 4:136–148. doi:10.1137/0904010

Plesser HE, Austvoll K (2009) Specification and generation of structured neuronal network models with the NEST topology module. BMC Neurosci 10(suppl 1):P56. doi:10.1186/1471-2202-10-S1-P56

Quirk JJ (2005) Computational science: "same old silence, same old mistakes; something more is needed". In: Plewa T, Linde T, Weirs VG (eds) Adaptive mesh refinement—theory and applications. Springer, Berlin, pp 3–28

Raikov I, The INCF Multiscale Modeling Taskforce (2010) NineML—a description language for spiking neuron network modeling: the abstraction layer. BMC Neurosci 11(suppl 1):P66

Rotter S, Diesmann M (1999) Exact digital simulation of time-invariant linear systems with applications to neuronal modeling. Biol Cybern 81:381–402

Schrödinger E (1915) Zur Theorie der Fall-und Steigversuche an Teilchen mit Brownscher Bewegung. Phys Zeit 16:289–295

Shelley MJ, Tao L (2001) Efficient and accurate time-stepping schemes for integrate-and-fire neuronal networks. J Comput Neurosci 11:111–119

Silva CT, Freire J, Callahan S (2007) Provenance for visualizations: reproducibility and beyond. Comput Sci Eng 9(5):82–90

Stodden V (2009a) Enabling reproducible research: open licensing for scientific innovation. Int J Commun Law Policy 13:2–25, http://www.ijclp.net/files/ijclp_web-doc_1-13-2009.pdf

Stodden V (2009b) The legal framework for reproducible research in the sciences: licensing and copyright. IEEE Comput Sci Eng 11(1):35–40

Usui S (2003) Visiome: neuroinformatics research in vision project. Neural Netw 16:1293–1300

Vandewalle P, Kovačević J, Vetterli M (2009) Reproducible research in signal processing. IEEE Signal Proc Mag 26(3):37–47. doi:10.1109/MSP.2009.932122

Wilson MA, Bhalla US, Uhley JD, Bower JM (1989) GENESIS: a system for simulating neural networks. In: Touretzky D (ed) Advances in neural information processing systems. Morgan Kaufmann, San Mateo, CA, pp 485–492