Journal of Computational Neuroscience (2020) 48:357–363
https://doi.org/10.1007/s10827-020-00748-3

Non-replicability circumstances in a neural network model with Hodgkin-Huxley-type neurons

Wilfredo Blanco[1,2] · Paulo H. Lopes[1] · Anderson Abner de S. Souza[1] · Michael Mascagni[3,4]

Received: 5 August 2019 / Revised: 30 March 2020 / Accepted: 22 April 2020 / Published online: 9 June 2020

Abstract

Building upon previous experiments can be used to accomplish new goals. In computing, it is imperative to reuse computer code to continue development on specific projects. Reproducibility is a fundamental building block in science, and experimental reproducibility issues have recently been of great concern. It may be surprising that reproducibility is also of concern in computational science. In this study, we used a previously published code to investigate neural network activity and we were unable to replicate our original results. This led us to investigate the code in question, and we found that several different aspects, attributable to floating-point arithmetic, were the cause of these replicability issues. Furthermore, we uncovered other manifestations of this lack of replicability in other parts of the computation with this model. The simulated model is a standard system of ordinary differential equations, very much like those commonly used in computational neuroscience. Thus, we believe that other researchers in the field should be vigilant when using such models and avoid drawing conclusions from calculations if their qualitative results can be substantially modified through non-reproducible circumstances.

Keywords Neural network · Numerical simulation · Replicability · Floating-point precision

1 Introduction

Science is built upon experimentation, theory, and more recently computer simulations. An issue which causing considerable concern, especially in the experimental community, is how to enforce and check whether experimental results are reproducible. This also affects results generated by computer simulations, despite their supposedly deterministic nature.

Computational simulation of real phenomena typically requires the use of floating-point arithmetic (FPA) to calculate numerical solutions. Mathematical models, typically described through equations, are implemented as source code *via* a high-level computer language. Code can be executed through computational simulations, which are meant to represent a real experiment. The simulation output values are analyzed to verify how well the model was able to provide data that resemble the results found in the real event (*i.e.* reproducibility); or if the model can precisely provide the same results - a replica - already found in real events (*i.e.* replicability). Replicating a real process or experiment by a computer simulation is not possible. On the other hand, replicating computer simulations seems to be achievable when considering that

Action Editor: David Terman

Electronic supplementary material The online version of this article (https://doi.org/10.1007/s10827-020-00748-3) contains supplementary material, which is available to authorized users.

✉ Wilfredo Blanco
wilfredoblanco@uern.br; wblancof@gmail.com

Paulo H. Lopes
paulo_lopes11@hotmail.com

Anderson Abner de S. Souza
and.abner@gmail.com

Michael Mascagni
mascagni@fsu.edu

[1] Computer Science Department, State University of Rio Grande do Norte, Natal, RN, Brazil

[2] Bioinformatics Department, Federal University of Rio Grande do Norte, Natal, RN, Brazil

[3] Computer Science Department, Florida State University, Tallahassee, FL, USA

[4] Applied and Computational Mathematics Division, National Institute of Standards and Technology, Gaithersburg, MD, USA

having the source code is enough to provide replicability (Mcdougal et al. 2016).

Computer simulations sometimes need to be reproduced or replicated to reuse previous results/models to achieve new goals (Mcdougal et al. 2016). Nevertheless, there are important aspects which need to be considered when replications are computed. A detailed description of the model is the starting point, which could be achievable following best-practices (Nordlie et al. 2009; Waltemath et al. 2011). However, the issues related to the limitations in representing real numbers in computers is far from being resolved. These limitations lead to rounding and truncation errors (Higham 2002), which are unavoidable consequences of working with finite precision arithmetic, *i.e.* the limited-digit number to represent floating-point numbers, which is an intrinsic fact in digital computers (Datta 2010). This is the reason why numerical computation executed on different hardware architectures (32 bits or 64 bits) produce different errors. This fact also compromises results which are based on replicability. Many groups do not pay attention to this issue, believing that these errors are infrequent; even worse, they are seen as a potentially wasteful distraction (Drummond 2009).

The purpose of this article is to discuss aspects of replicability in a numerical model. As an example, a typical mathematical model from the field of computational neuroscience will be used, specifically simulating the activity of neural networks.

1.1 Neural network case study

This study models a recurrent all-to-all neural network's activity during early development, in which inhibitory neurons still play an excitatory role (Ben-ari 2002; Ben-ari et al. 2007). Different proportions of excitatory and inhibitory neurons were previously simulated in this biological scenario (Blanco et al. 2017), and the excitatory and inhibitory activities of pre-synaptic neurons were integrated separately. Excitatory and inhibitory neurons are modeled using the same equations but changing their reversal potential value. This allowed us to label neurons as inhibitory but treat them as excitatory.

The model contains two population variables, the spontaneous network activity and the network synaptic efficacy, represented by <A> and <S> respectively. Both variable profiles are characterized by cyclic dynamics: <A>, a fast profile variable with high activity episodes separated by quiescent periods (inter-episode intervals), and <S>, a slow profile during quiescent periods of <A>, but fast depression during the high activity episodes (Blanco et al. 2017; Tabak et al. 2010). Reproducibility occurs when network activity <A> and <S> preserve the cyclic dynamics, but not necessarily with the same values over time between simulations. On the order hand, replicability is achieved when <A> and <S> values are identical at each time step between simulations.

2 Methods

A small neural network model composed of 100 Hodgkin-Huxley (HH) type neurons with all-to-all coupling was simulated. All neurons were identical, except for the value of their input currents (I_{app} *vector*), which were randomly chosen (the same for all simulations). The cells were modeled as a single compartment *via* an ordinary differential equation (ODE) system for the voltage. It is well known that network activity profile differs with the time step dt and the chosen numerical method (Hansel et al. 1998; Morrison et al. 2007; Shelley and Tao 2001), a fixed $dt = 0.01$ *ms* was used to replicate (Blanco et al. 2017) and to guarantee no significant difference in episode durations or inter-episode interval distributions as seen in other experiments (Tabak et al. 2010). The neural network model was originally taken from (Tabak et al. 2010), simulating only excitatory neurons.

The ODEs were solved by the Runge–Kutta fourth-order method (RK4) using the Boost C++ library v1.71.0 (Schling 2011). The equations and parameters were taken from (Blanco et al. 2017). The C/C++ source code files and documentation are available as freeware on GitHub *via* https://github.com/wblancof/neural-numerical-replicability.

2.1 Platform specifications

The same source files were compiled using GCC v9.2.0 and executed on Windows, Linux and Mac platforms. Hardware platforms and OS specifications are shown in Table 1. The command lines to execute the simulations are in the README file in the GitHub repository.

2.2 Simulations executed on windows

Two main scenarios were tested: (1) A neural network composed of 100 excitatory neurons; and (2) a neural network composed of 80/20 excitatory/inhibitory neurons, respectively. All implementations used ***double*** floating-point precision (size = 64bits) defined in the C/C++ language. In order to increase precision, code variables were re-defined using two floating-point precision types: C/C++ ***long double*** precision (size = 128bits), and a ***cpp_dec_float_100*** *type* floating-point with a hundred decimal digit precision (size = 640bits) implemented in Boost. A parameter was added into the compilation command line to redefine the variables (*via tydedef* specifier) to the specific floating-point type.

All decimal numeric values were saved in files with fourteen-digit accuracy because rounding and truncation errors are introduced (IEEE Standard for FPA 2019) when real numbers are represented with more than fifteen decimal digits (Datta 2010).

The absolute error equation $\epsilon(t) = |A_i(t) - A_j(t)|$ was used for calculating the discrepancies at each timestep of the spontaneous network activity <A> between simulations i and j.

Table 1 Hardware and software specification

Hardware/ Software	Windows	Mac	Linux - Ubuntu
CPUs	Intel i7-8550U 1.80GHz	Intel Core (TM) i5-4250U 1.30GHz	Intel Core i5 M460 2.53GHz
RAM	8GB, DDR3	4GB, DDR3	4GB, DDR3
OS	Windows 10	MacOS Mojave v10.14.6	Ubuntu 18.04.2 LTS

3 Results

This section will illustrate the activity derived from an all-to-all coupling neural network model during early development. Here, inhibitory neurons play an excitatory role; however, pre-synaptic activity integration used to calculate the post-synaptic neuron activity is still calculated separately: one for the excitatory neurons and another for the inhibitory neurons. The numerical simulations based on these ODEs was used to check computational replicability in this paper.

3.1 Simulations on several platforms caused different results

The same source code was compiled on three computational platforms across three different operating systems (OSs) (Table 1). Simulations with the same parameters were executed, and the results of the network activity <A> and synaptic efficacy <S> are shown in Fig. 1. Although there was a persistent attempt to control the software issues, different results caused by the diversity of OSs and hardware were expected

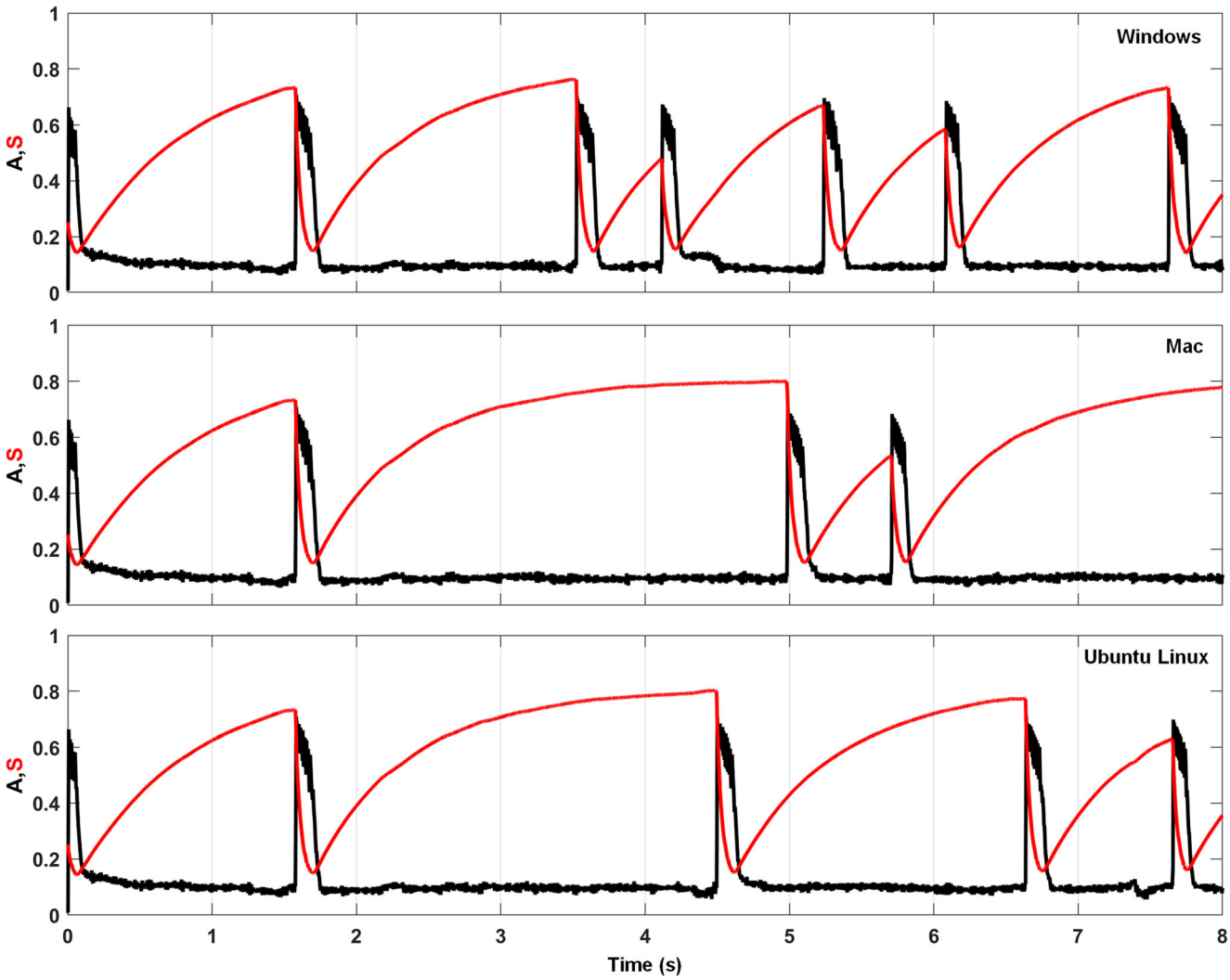

Fig. 1 Implementations across hardware platforms and OS induce "errors". The network activity (A, black curve) and synaptic efficacy (S, red curve) over time are shown under Windows (top panel), Mac (middle panel) and Ubuntu Linux (bottom panel). Check Table 1 for more hardware/software specifications

(Waltemath et al. 2011) since our system is sensitive to round-off and truncation errors. Moreover, these errors are mostly system dependent. However, the size of divergence occurring in a short period of time, visually perceptible after the first 3 s of the simulation (Fig. 1), was not expected.

Previous simulations were executed under a different compiler version (not shown). However, since we had less familiarity with the Mac OS compiler, we decided to see if we could account for the differences we observed when varying the version of Unix. The only mathematical functions used in this code were the *exp()* and *pow()* functions from <math.h > and they are housed in the standard library named glibc. In particular, this is the case in Mint and Gentoo Linux. In Cygwin, the C standard library is called newlib, and so we ran our code in Gentoo, but linked our code with newlib. This produced the behavior shown, and had the effect of running the code on Gentoo with the mathematical function definitions from Cygwin. This behavior begs the question of creating a standard set of elementary functions that are used by all vendors. We are currently exploring this avenue with our colleagues at National Institute of Standards and Technology (NIST).

3.2 Small difference in summation order on windows caused different results

The hardware and software were invariant on different Windows runs and the same source code was compiled and executed. However, the parameter related to the proportions of excitatory and inhibitory neurons was changed, splitting the summation of the network pre-synaptic activity in two blocks. Two scenarios were created: (1) the network had 100 excitatory neurons, thus the simulation only went through only the first inner-loop to calculate the summation of the pre-synaptic activity (*atotExc* variable) (Fig. 2a, light red graphics elements); (2) the last 20 neurons were labeled as inhibitory (keeping reversal potential $V_{exc} = V_{inh} = 70\ mV$). Hence, excitatory and inhibitory pre-synaptic contribution were calculated through the first and second inner-loops respectively (Fig. 2a, light red and blue graphics elements). The network activity (A, black curve) and synaptic efficacy (S, red curve) are shown for these two cases in Fig. 2b, c. Identical results were expected since the network topology is all-to-all connected. However, when $t > 3\ s$, the plots already show significant discrepancies, which are visually perceptible (even with a new activity episode) in the <A> and <S> profiles. Moreover, the absolute error is $> 10^{-6}$ for $t >= 1.69\ s$ (vertical black line), indicating that discrepancies in population activity take place earlier than the spike time differences (Fig. 2c, black diamond markers on raster plot). Although 10^{-6} is an arbitrary threshold, the idea is to identify that discrepancies in populational activity <A > are happening earlier, and are eventually causing further differences between spikes.

3.3 Using higher precision floating-point still produced different results

Since double precision seemed inadequate, the source code was adapted to use two higher precision floating point types: C/C++ long double precision and a floating-point with 100 decimal places precision implemented in Boost (Schling 2011). The simulations were once again executed for the two previously described computational scenarios. This resulted in new profiles for the activity <A> and the synaptic efficacy <S> of the network (Fig. 3).

The simulations using Long Double precision also showed discrepancies (Fig. 3, left column). The absolute error reached values $> 10^{-6}$ later, from t = *2.57 s* (Fig. 3, vertical black line), compared to previous simulations (Fig. 2b). Spike time differences (Fig. 3, black diamond markers on raster plot) were also observed later than activity differences. By only using the hundred-digit precision from Boost, the <A> and <S> values, as well as individual spike time are identical (for the fourteen-digits saved) during 8 s of the simulation (Fig. 2, right column). However, this does not mean that discrepancies will not appear later on. One important point to highlight is that these simulations took more than 4 days to simulate 8 s of biological time on Windows (see Methods).

4 Discussion

Results from this research suggest that computer precision may be a variable which affects computational replicability. To illustrate this assumption, we used a computational approach which simulated the activity in a neural network with all-to-all coupling during its early development.

The outcomes revealed that executing the same code with the same parameters on different platforms presented a similar activity profile (episodic bursts of intense activity separated by quiescent periods), but were far from identical. In other words, the results were reproduced, but not replicated. These results suggest that it is not sufficient to have exactly the same source code to replicate a computational study, and moreover, rounding and truncation errors are platform dependent.

The results in compiling and executing the same code on the same hardware and software platform but splitting the summation of pre-synaptic activity (excitatory and inhibitory) were surprisingly not replicated. High absolute error values at the very beginning of the simulation time were

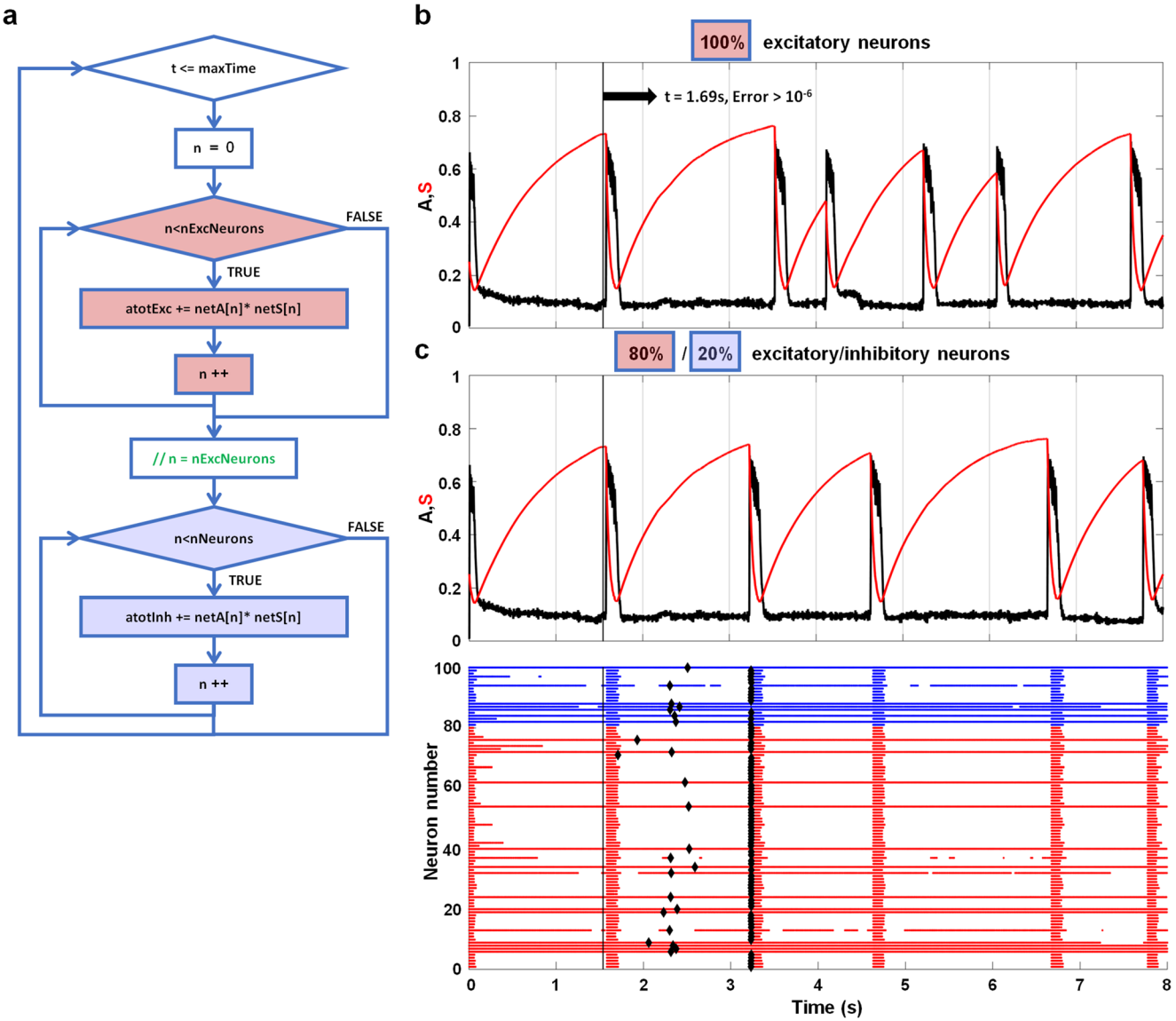

Fig. 2 Summation order generates different results. (**a**) Flowchart schema of the main loop simulation over time. There are two inner-loops (light red and blue colors highlighted) at each time step, and the simulations get them to calculate the pre-synaptic contribution for excitatory and inhibitory neurons, respectively. (**b**) The network activity (A, black curve) and synaptic efficacy (S, red curve) over time are shown when the network is composed of 100% of excitatory neurons. In this scenario, the simulation only went through the first loop (light red). (**c**) A new scenario in which 20% of neurons are inhibitory, but they still preserved the excitatory features; we executed the simulation with the main goal to slightly split the summation of the input, i.e. 80% calculated in the first loop (light red color, representing excitatory contribution) and 20% calculated in the second loop (light blue color, representing a fake pre-synaptic inhibitory loop). The blue traces in the raster plot correspond to inhibitory neurons, while the red traces correspond to excitatory neurons. Black diamond markers are located in the first spike time which differed for each neuron. In this simulation, the vertical line at $t=1.69\ s$ marks the time moment in which the error between the simulations (B, C panels) is greater than 10^{-6}.

observed. The precision of the variables in the last scenario was increased using higher precision types: *long double* and *cpp_dec_float_100* from Boost. Substantial differences appeared in a short period of time, making further results unreliable for analysis. We only replicated the numerical computational results during 8 s by using *cpp_dec_float_100*, in which the rounding and truncation errors only take place beyond 100 digits, but at a high computational cost.

Errors begin to appear if numbers are produced during computation which exceed the established precision range (Datta 2010). Although the IEEE Standard for floating-point attempts to minimize these errors for most processors, we just showed that the standard does not guarantee that operations, defined by libraries or programming languages, will have the same result on different OSs, or even different versions of the same function on the same OS. Furthermore, simple

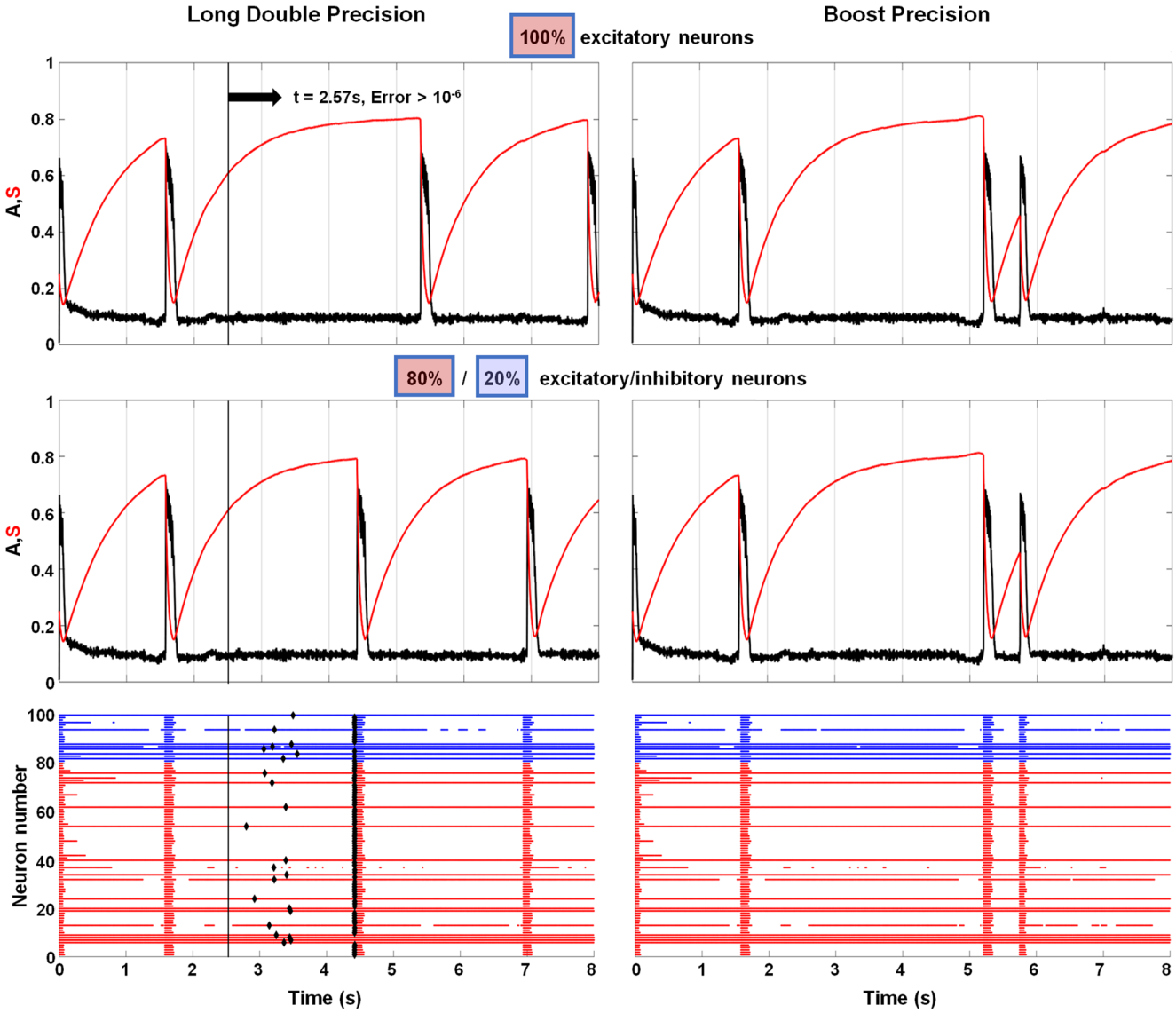

Fig. 3 Increasing double precision still generates different results. The network activity (A, black curve) and synaptic efficacy (S, red curve) over time are shown in the plots and they are under the same two scenarios, as shown in Fig. 1 (first row with 100% excitatory neurons and second row with 80/20% of excitatory/inhibitory neurons respectively). In the left column the two scenarios are using Long Double precision and the right column is using Boost precision. The A and S curves generated by using Long Double precision (left column) are still showing large discrepancies. The absolute error is already high ($>10^{-6}$) starting from t > 2.57 s (vertical black line). The corresponding raster plots are shown in the last row, in which blue traces correspond to inhibitory neurons, while the red traces correspond to excitatory neurons. Black diamond markers are located at first time value where spikes differ for each neuron. The A, S and individual spike time values are identical (for the fourteen-digits places saved) during the 8 s of simulation when only using 100 decimal place precision implemented by Boost library software.

summations are often affected due to the non-associativity of FPA (IEEE Standard for FPA 2019; Datta 2010). After all, it is important for the research community to investigate the behavior of complex biological systems to be aware that these issues are still causing serious non-replicability risks. This can be used to minimize the impact of these on conclusions drawn from computations.

Acknowledgements This research started in 2016 when WB was hosted by the Mathematics department and the Institute of Molecular Biophysics at Florida State University. WB was also supported by a scholarship (Process #202320/2015-4) from the Brazilian National Council for Scientific and Technological Development (*Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq*). PHL was supported by an undergraduate scientific research scholarship from *CNPq* in 2017 and from the State University of Rio Grande do Norte (*UERN*) in 2018.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interests.

Disclaimer Certain commercial equipment, instruments, or materials are identified in this paper in order to specify the experimental procedure adequately. Such identification is not intended to imply recommendation or endorsement by the National Institute of Standards and Technology (NIST), nor is it intended to imply that the materials or equipment identified are necessarily the best available for the purpose.

References

Ben-ari, Y. (2002). Excitatory actions of gaba during development: The nature of the nurture. *Nature Reviews Neuroscience, 3*(9), 728–739.

Ben-ari, Y., Gaiarsa, J.-L., Tyzio, R., & Khazipov, R. (2007). GABA: A pioneer transmitter that excites immature neurons and generates primitive oscillations. *Physiological Reviews, 87*(4), 1215–1284.

Blanco, W., Bertram, R., & Tabak, J. (2017). The effects of GABAergic polarity changes on episodic neural network activity in developing neural systems. *Frontiers in Computational Neuroscience, 11*, 88.

Datta, B. N. (2010) Numerical linear algebra and applications. Siam.

Drummond, C. (2009) Replicability is not reproducibility: nor is it good science.

Hansel, D., Mato, G., Meunier, C., & Neltner, L. (1998). On numerical simulations of integrate-and-fire neural networks. *Neural Computation, 10*(2), 467–483.

Higham, N. J. (2002) Accuracy and stability of numerical algorithms. Siam. 0898718023.

IEEE Standard for Floating-Point Arithmetic. (2019) IEEE Std 754-2019 (Revision of IEEE 754-2008), p. 1–84.

Mcdougal, R. A., Bulanova, A. S., & Lytton, W. W. (2016). Reproducibility in Computational Neuroscience Models and Simulations. *IEEE transactions on bio-medical engineering, 63*(10), 2021–2035, 03/08.

Morrison, A., Straube, S., Plesser, H. E., & Diesmann, M. (2007). Exact subthreshold integration with continuous spike times in discrete-time neural network simulations. *Neural Computation, 19*(1), 47–79.

Nordlie, E., Gewaltig, M.-O., & Plesser, H. E. (2009). Towards reproducible descriptions of neuronal network models. *PLoS computational biology, 5*(8), e1000456.

Schling, B. (2011) The Boost C++ Libraries. XML Press, 262 p 0982219199, 9780982219195.

Shelley, M. J., & Tao, L. (2001). Efficient and accurate time-stepping schemes for integrate-and-fire neuronal networks. *Journal of Computational Neuroscience, 11*(2), 111–119.

Tabak, J., Mascagni, M., & Bertram, R. (2010). Mechanism for the universal pattern of activity in developing neuronal networks. *Journal of Neurophysiology, 103*(4), 2208–2221.

Waltemath, D., Adams, R., Beard, D. A., Bergmann, F. T., et al. (2011). Minimum information about a simulation experiment (MIASE). *PLoS computational biology, 7*(4), e1001122.