RESEARCH ARTICLE

BioSANS: A software package for symbolic and numeric biological simulation

Erickson Fajiculay[1,2,3], **Chao-Ping Hsu**[1,4,5]*

1 Institute of Chemistry, Academia Sinica, Taipei, Taiwan, **2** Bioinformatics Program, Institute of Information Science, Taiwan International Graduate Program, Academia Sinica, Taipei, Taiwan, **3** Institute of Bioinformatics and Structure Biology, National Tsinghua University, Hsinchu, Taiwan, **4** Physics Division, National Center for Theoretical Sciences, Taipei, Hsinchu, Taiwan, **5** Genome and Systems Biology Degree program, National Taiwan University, Taipei, Taiwan

* cherri@sinica.edu.tw

Citation: Fajiculay E, Hsu C-P (2022) BioSANS: A software package for symbolic and numeric biological simulation. PLoS ONE 17(4): e0256409. https://doi.org/10.1371/journal.pone.0256409

Editor: Bashar Ibrahim, Gulf University for Science and Technology, GERMANY

Received: August 1, 2021

Accepted: March 15, 2022

Published: April 18, 2022

Data Availability Statement: All codes are available in https://github.com/efajiculay/SysBioSoft/tree/BioSANS_updated/BioSANS/src/BioSANS2020 All installer are available in https://github.com/efajiculay/BioSANS_installers, and https://github.com/efajiculay/SysBioSoft/tree/BioSANS_updated/BioSANS_installer All relevant data are within the paper and its Supporting information files. The software is also available in our gihub page at https://pypi.org/project/BioSANS2020/, pypi repository at https://pypi.org/project/BioSANS2020/#files, and test.pypi repository at https://test.pypi.org/project/BioSANS2020/#files.

Abstract

Modeling biochemical systems can provide insights into behaviors that are difficult to observe or understand. It requires software, programming, and understanding of the system to build a model and study it. Softwares exist for systems biology modeling, but most support only certain types of modeling tasks. Desirable features including ease in preparing input, symbolic or analytical computation, parameter estimation, graphical user interface, and systems biology markup language (SBML) support are not seen concurrently in one software package. In this study, we developed a python-based software that supports these features, with both deterministic and stochastic propagations. The software can be used by graphical user interface, command line, or as a python import. We also developed a semi-programmable and intuitively easy topology input method for the biochemical reactions. We tested the software with semantic and stochastic SBML test cases. Tests on symbolic solution and parameter estimation were also included. The software we developed is reliable, well performing, convenient to use, and compliant with most of the SBML tests. So far it is the only systems biology software that supports symbolic, deterministic, and stochastic modeling in one package that also features parameter estimation and SBML support. This work offers a comprehensive set of tools and allows for better availability and accessibility for studying kinetics and dynamics in biochemical systems.

1. Introduction

The complex nature of biological systems often hinders a full understanding of behavior, an area in which mathematical models and simulation can help [1]. Clues from experiments are limited to the details of the sub-systems considered and can be difficult to interpret. Computer simulation allows for formulating a working model that can help explain experimental observations [2,3]. It can provide links between observations and unknowns in terms of a mathematical expression or numerical values, offering qualitative or quantitative insights. Good models can give testable predictions that can be used to evaluate the applicability, scope, and

Funding: Both EF and CPH acknowledges the support from project AS-IA-106-M01 from Academia Sinica and project 109-2113-M-001-022-MY4 and 110-2123-M-001-005 from Ministry of Science and Technology of Taiwan. The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Competing interests: The authors have declared that no competing interests exist.

limitation of the model [4,5]. No model can fully account for every detail of a system, but some can provide important aspects of the system within its scope [6].

Interest in modeling biological processes has been increasing [7]. Models of processes that constitute a network of interacting molecules are of particular interest [8–12]. Algorithms and theories dedicated to modeling gene expressions have been developed [13–15]. Modeling and experimental studies related to gene expression, biological clock, and diseases are now common [16–20]. The study of infectious diseases and host pathogen interactions have also been heavily involved with various modeling approaches [21–23]. Recently, works on the application of a variety of mathematical modeling techniques for COVID-19 have also been reported in the literature [24–27].

Another popular approach of modeling biological system is known as rule-based models. In these models, the interaction of molecules of a system is represented by a pattern or graph, with rules that described molecular processes and events. The description of system is usually simplified with the rules, with great potential in studying the combinatorial complexity in many biological problems. Several works have been reported following this modeling regime [10,28–31].

Simulating a model is not difficult, but for a complicated model or for elaborated analyses and tests with models, scripting or programming skills are necessary. To simulate a model, high-level programming tools such as Matlab or Python are often used. Complicated models may require advance programming skills and deep understanding of the domain concepts involved. Software exists for modeling biological systems; most require basic scripting [32–39]. Some do not require coding for basic simulations but have limited power in the graphical user interface (GUI) [40]. For most software, the propensity expression needs to be manually typed by the user. This is sometimes time-consuming and a hindrance to usability. Most systems-biology software packages support only certain types of modeling tasks. As far as we know, no systems biology software provides symbolic computation capability without the need to declare variables and write ordinary differential equation (ODE) expressions. Most are not user friendly, do not allow for easy-to-prepare input, do not support parameter estimation and do not provide a GUI. Some do not provide systems biology markup language (SBML) support and others have limited or minimal SBML supported features. Therefore, a software package addressing the above limitations is highly desirable because it allows access to mathematical modeling by a much broader range of users.

In this study, we took advantage of Python libraries for both numeric and symbolic computation and developed a program that can make systems biology analysis available to non-domain experts. In doing so, we provide a platform that can support a wide variety of modeling tasks. Currently, the software supports model construction, symbolic computation, propagation, analysis, and parameter estimation and also supports SBML. A semi-programmable topological model input was developed, for easy construction and understanding. We also provide a GUI. The algorithms are available in Python import format for expert users and also via a command line using our novel structured simulation language (SSL), which is very similar to MySQL commands in terms of readability. We tested the software in simulating systems of various sizes, and it was found reliable, practical, and easy to use. We provide an invaluable research tool for model simulations by domain experts from systems biology or chemistry.

2. Modeling scheme supported in BioSANS

The process of a typical modeling task generally involves model construction, propagation and analysis. Fig 1 lists these processes with functionalities provided in the developed software, BioSANS. The first step is to build a topological model and to establish the set of differential equations based on the topology of the model. For the initial model, temporary parameter

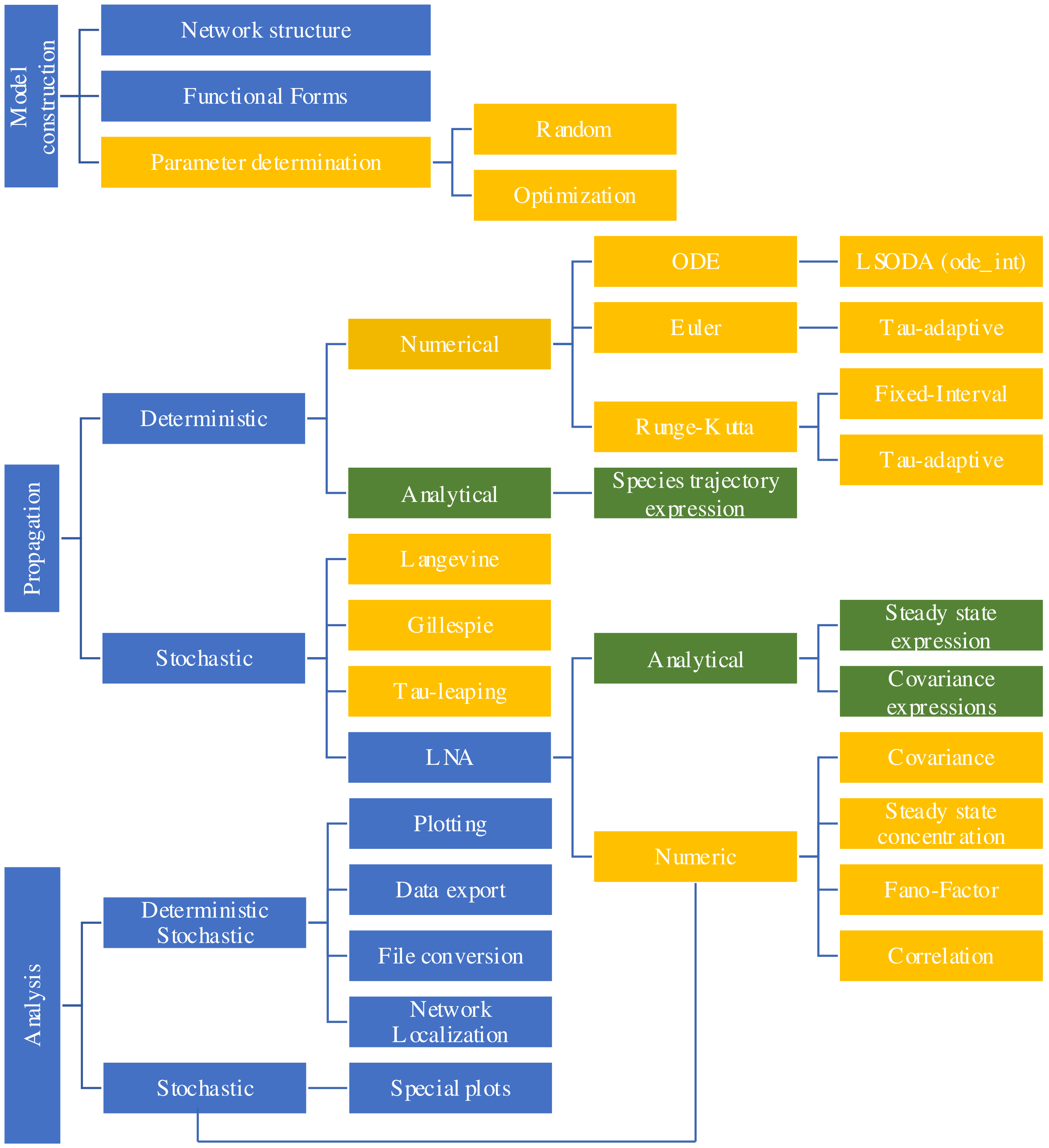


Fig 1. Schematic diagram showing some of the modeling tasks available in BioSANS. The basic modeling scheme includes model construction, propagation, and analysis. Except for blue colors, the same color in different branches indicates similar types of analysis under a common category. Green shaded boxes are for analytical expressions, and orange shaded boxes are for numerical results.

https://doi.org/10.1371/journal.pone.0256409.g001

values are required. These values can be based on databases, experimental design, physical intuition, and known ground truth. If sufficient data are available, parameter estimation may help tune the parameters. If the ODE is analytically solvable, species analytical expression as a function of time, initial conditions, and rate constants can be computed. If not, numerical integration techniques can be used to propagate the trajectories. Propagation can be deterministic or stochastic and compared to experimental data to gain some physical insights.

Linear noise approximation (LNA) is a convenient way to study noise in the system [41–43]. Derivation of analytical expression utilizing the LNA is supported in BioSANS and can be compared to the stochastic results. This allows for gaining physical insights into the noise effects in the system as a function of physical variables.

Once the simulation is performed and the analytical expressions are calculated, further analyses such as plotting, calculation of statistics, etc. can be performed to judge the result and to check consistencies with known ground truth. If the model has inconsistencies and/or fails to account for experimental observations, some of the hypotheses put forth may be proved wrong, and the model may need some modification. A cycle from gathering experimental data, model construction and simulation is needed to ensure consistency between the model and observations. A full list of modeling tasks and workflow to be followed in using BioSANS on a case-to-case basis is included in supporting information.

3. Description of algorithms and implementation

The codes in BioSANS are written in Python [44], which only needs a user-defined topology file that is intuitive to construct based on basic chemistry knowledge. The software algorithm grabs data from the topology file and calculates the stoichiometric matrix and propensity vector. It then sets up the ordinary differential equation for most typical types of simulations. This process is handled by building a dictionary of species, concentrations, reactants, products, rate constants, and lambda functions. Access to dictionary contents allows for automatic declaration of symbolic parameters for symbolic computation and easy establishment of inputs for numeric computations. Most numeric computations are handled by NumPy/SciPy [45,46] and symbolic computations by SymPy [47]. We have developed the necessary codes that prepare the inputs for solving ordinary differential equations for integration with SymPy and NumPy modules. We have also implemented functionalities such as tau-adaptive Euler, tau-adaptive Runge-Kutta (RK), fixed interval RK, stochastic simulation algorithm (SSA), tau-leaping algorithm, tau-adaptive chemical Langevin algorithm, fixed interval chemical Langevin algorithm, numerical and symbolic linear noise approximation, and Monte-Carlo expectation maximization, etc. that NumPy/SciPy and SymPy do not have built-in functions.

A relatively new method, network localization [48], allows for studying the qualitative effect of parameter perturbation from structural topology alone. This can be used to obtain physical insights even without propagating the ODE of the system. In BioSANS, both symbolic and numeric network localization is supported.

A list of the codes in BioSANS with a description of its role is provided in sections 7.1 to 7.2 of the supplementary material. BioSANS documentation, installer and the actual codes can be downloaded from the following GitHub repositories; https://github.com/efajiculay/BioSANS_installers, https://github.com/efajiculay/SysBioSoft/tree/BioSANS_updated/BioSANS, and the following GitHub pages; https://efajiculay.github.io/SysBioSoft/.

3.1 Model construction

To overcome the potential barriers in constructing a model, we have designed a new model input form, the topology file, which is a text file that follows how elementary chemical

```
#REACTIONS
A + B   =>  C       , 0.01
C      <=> Cs       , 2.4, 0.25
Cs      =>  A + B   , 1 ::::: lambda C : 0.01/(0.01 + C)

@CONCENTRATION
A , 100
B , 50
C , 0, lambda A, B, Cs : (A + B)/(A + Cs)
Cs, 0
```

Fig 2. An example of a topology file that can be used to perform simulations in BioSANS.

https://doi.org/10.1371/journal.pone.0256409.g002

equations [49] are written. With an example shown in Fig 2, such a file is intuitive to construct and contains only a list of reactions, rate constants, initial conditions, and additional parameter settings. The users do not necessarily need to provide expressions for fluxes or propensity, which is definitely needed when writing a regular code for simulation. BioSANS can automatically infer the propensity from the chemical reaction listed. Algebraic expressions for concentration and propensity modification are supported in case the needed expression differs from mass action kinetics, such as the commonly used Mechaelis-Mention kinetics or the Hill function. Time-dependent propensity and complicated conditional logic (i.e., events, events with delay, etc.) is supported in the algebraic expression. Models can also be written as differential equations in the topology file.

The BioSANS topology file has three main tags: Function_Definitions:, #REACTIONS, and @CONCENTRATION. Fig 2 shows the last 2 tags, which is the minimal requirement. The single-headed arrow represented as "=>" is used for forward reaction and the double headed arrow "<=>" is for reversible reaction. This setting allows a user to easily identify reactions, one way or reversible, in a topology file. The numbers after the comma in each row after the reactions are the rate constants. The number of rate constants listed should match the arrow used: 1 entry for one-way and 2 entries for reversible reactions. The initial concentration is defined after the comma in each corresponding species under the @CONCENTRATION tag. Propensity and concentration modifications can be written as a lambda expression in each row after some delimiter. If algebraic modifications are provided, the rate constant and or initial concentration will be ignored and the propensity expression is evaluated. The full details of how to construct a topology file including algebraic modifications is in section 4 of the supplementary material.

Most available models in the literature are provided as a Matlab or Python script or are available in SBML format. BioSANS can run a Python file directly, and ODE models written in Python will be easy to simulate in BioSANS. Shown in Fig 3 is an alternative input file called an ODE file. It can also be used to create models that require only the ODE expression, a set of initial conditions, and a rate constant.

An ODE file can be constructed with minor modification from an existing ODE model and is suitable to use if we have an idea of the underlying mathematics that the system obeys. The ODE_DECLARATIONS: tag is simply the set of species in the left-hand side and the corresponding differential expression in the right- hand side. This file needs to be converted to a topology file with BioSANS before we can propagate the model in the usual way. The converted file will contain the corresponding chemical reaction that satisfies the ODE expressions.

```
ODE_DECLARATIONS:
A = -ka*A/(10+B**2) + kb*B/(5+A**2)
B =  ka*A/(10+B**2) - kb*B/(5+A**2)

INI_CONCENTRATIONS:
A = 100
B = 0

RATE_CONSTANTS:
ka = 100
kb = 0.03
```

Fig 3. An example of an ODE file with 2 chemical species.

https://doi.org/10.1371/journal.pone.0256409.g003

Unlike the topology file, the ODE file does not support a lambda expression, and if a user wants to introduce events, such events have to be incorporated in the converted file. SBML files to topology file conversion is also supported in BioSANS for models that are available only in SBML format.

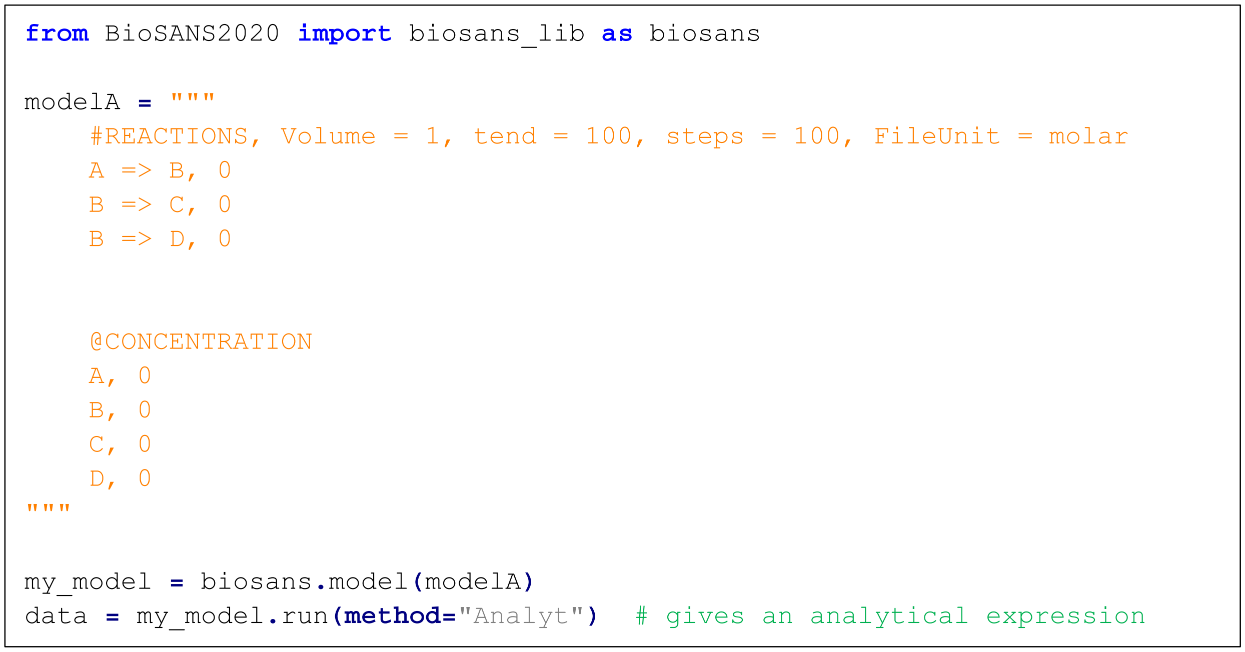
BioSANS also takes line commands with highly human readable formats, similar to SQL commands. Fig 4 shows a model using such commands, which we call SSL. This function allows for automations by saving many SSL scripts in one file and loading them via the BioSANS console. The model in the script is also converted by BioSANS to topology format and executes the remaining statements, which tell BioSANS how to process the model from propagation to analysis.

BioSANS can also be used as a python library which only requires minimal coding skills to perform complex simulation task. Fig 5 shows an example code that declares a topology model inside a string that is directly feed into BioSANS2020.biosans_lib model class. This minimal set of codes can be used to perform propagation of deterministic and stochastic trajectory, symbolic computation, and parameter estimation by just changing the method and modifying

```
propagate
    A => B, 0.2 &
    B => C, 0.3
where
    A=100 &
    B=0.2 &
    C=0
using CLE
with
    tn=50 &
    tlen=1000 &
    miter=30 &
    mult_proc=True &
    fout=Traj1;
```

Fig 4. An example of BioSANS structured simulation language (SSL) scripts.

https://doi.org/10.1371/journal.pone.0256409.g004



```
from BioSANS2020 import biosans_lib as biosans

modelA = """
    #REACTIONS, Volume = 1, tend = 100, steps = 100, FileUnit = molar
    A => B, 0
    B => C, 0
    B => D, 0


    @CONCENTRATION
    A, 0
    B, 0
    C, 0
    D, 0
"""

my_model = biosans.model(modelA)
data = my_model.run(method="Analyt")  # gives an analytical expression
```

Fig 5. An example of the use of BioSANS as a python import or library.

https://doi.org/10.1371/journal.pone.0256409.g005

few more parameters inside the model and run functions. A detailed list of methods key words that can be used inside the run function can be found in section 6.2.1 of the supplementary material.

As part of the general model construction, BioSANS can also perform parameter estimation for a given topology file and experimental data, a feature not available in many similar software packages. A tutorial is included in supplementary material for a detailed description of the different BioSANS input files (section 4). Topology files for parameter estimation with examples are discussed in sections 8.5 to 8.7.

3.2 Propagation

The trajectory of the state variables can be computed via deterministic and stochastic settings. For deterministic computation, symbolic analytical expression and numerical integration are supported. The stochastic calculation may be carried out with the SSA [50], chemical Langevin equation [51], tau-leaping algorithm [52], and LNA [41,43].

Symbolic computation is currently limited to mostly linear differential equations and a few nonlinear differential equations. The symbolic test performed in this work involves only up to 10 interacting chemical species; beyond that, the software may take a very long time for an answer. Topology files with a modified functional form of propensity with nonlinear functions are likely to fail in symbolic computation. Such capabilities can be further improved if the symbolic ODE solving is improved with SymPy in the future. Nevertheless, our work allows for a quick input in topology and a symbolic answer for solvable systems without the need to declare variables and write ODE expressions. To the best of our knowledge, this is a novel functionality among current systems-biology packages and a feature that is very convenient to use.

In propagating deterministic trajectories, we took advantage of the LSODA [53] algorithm in the SciPy module of python, which allows for fast and efficient numerical integration. We also provide 2 different Euler propagations [54], both with an adaptive time-step setting.

Runge-Kutta fourth (RK4) is also offered in 2 version, one is time adaptive and and the other is fixed time-interval [55].

Euler and RK4 were manually coded to support certain SBML features that do not fit the standard use of the LSODA library. The critical SBML features that require coding new integrators are events with delay, events triggered by time, events triggered by other events, use of infinity symbols, and use of SBML keywords such as csymbol delay, csymbol time, rateof, etc. For sophisticated events, it is necessary to keep track of the previous trajectories, so the standard built-in integrator in Python was inapplicable.

Euler propagation is simple and can provide a well-grounded side-by-side comparison with the Euler-Maruyama propagation for stochastic simulation. In our implementation, 2 different adaptive time-step Euler schemes are offered. The first scheme, labelled "Euler (tau-adaptive-1)," was inspired by the tau-leaping scheme [52], with the step size determined by limiting the largest change among the reactions. The second scheme is labelled "Euler (tau-adaptive-2)," which involved choosing a step size that maintains the error as compared with a second-order RK estimate.

RK4 is one of the most popular and widely used integrator. It is simple, accurate in most cases, and easy to implement. Further details about our Euler and RK4 implementation are provided in supplementary material, section 10.2.1.

Stochastic trajectories can be propagated by using our implementation of SSA, 2 different implementations of tau-leaping, and 2 different versions of chemical Langevin algorithms. The SSA, or the Gillespie's algorithm [50], is an exact realization of the chemical master equation, with probabilistic reactions taking place at the microscopic level. Tau-leaping and chemical Langevin algorithms are inexact stochastic algorithms that accelerate propagation by using a large time step. They allow for many incidents of chemical reactions by using a random variable to account for the number fluctuation.

Our first implementation of tau-leaping, labelled "tau-leaping-1," includes a regular Poisson random variable describing the number of times a reaction channel fires. In this implementation, the step size was determined by requiring a small change in the reaction propensity following the new tau-selection procedure [52]. Treatment of critical reactions, which are reactions that are close to exhausting some of its reactants after several fires, is not considered in "tau-leaping-1". We simply draw another random variable when the species concentration becomes negative because this is a very rare event under the new tau-selection procedure. The second version, labelled "tau-leaping-2," adopts the "modified Poisson tau-leaping," which separates the treatment of critical and non-critical reactions and also uses the new tau-selection procedure [52]. Critical reactions are monitored and treated as discussed in section C of [52].

The two different versions of Chemical Langevin equations (CLEs) [51] we implemented in BioSANS are the tau-adapted version, labelled "CLE-tau-adaptive," and a regular fixed-interval version, labelled "CLE-fixed-intvl". CLE-tau-adaptive employs a simple tau selection we developed, which is fast for non-stiff to moderately stiff problems. (A detailed description of this algorithm is in section 10.2.4 of supplementary material). CLE-fixed-intvl is a typical Euler-Maruyama [56] propagation with a fixed time step.

It is important to estimate the variation in stochastic simulations. In addition to simply analyzing variances and covariances via stochastic trajectories, LNA [41,43] is a convenient way to estimate these quantities of a system. For LNA, we provide both symbolic and numeric ways of solving the covariance matrix. Steady-state LNA and time-dependent LNA computation are supported. Propagation of covariance and Fano-factor is possible in the time-dependent LNA.

BioSANS can directly run python scripts. The script may contain symbolic and numeric models taking advantage of SymPy and NumPy libraries. All functions in BioSANS itself can be used inside a code for customized propagation of trajectories as needed.

3.3 Analysis

The analysis part of modeling involves analysis based on topology and post-processing of trajectories. The results of analysis can be used for interpretation and comparison with a known ground truth. BioSANS supports covariance, Fano-Factor, cross-correlation, overall trajectory density plot, density plot binned with time, histogram slice at selected time range, average of trajectory plot, phase portrait, and custom plots. Network localization [48] is also available as an analysis method and can be directly applied to a topology file. Export of a trajectory to a file and plotting is by default enabled after every simulation but can be disabled when needed. Customized analysis is also possible because BioSANS can run Python files.

3.4 Testing BioSANS algorithms

The BioSANS algorithms presented above were tested for systems of various complexities and sizes. Performance was qualitatively assessed based on ease of use and features supported. Here we also report 4 quantitative tests of various features of BioSANS: 1) the semantic test for correctly interpreting SBML, 2) the stochastic SBML test suite for the stochastic simulations, 3) tests for symbolic solutions, and 4) tests for parameter estimation.

Scripts for automated evaluation against all tests are used except for symbolic LNA, for which each test is performed manually. The details of the tests are discussed in the following subsection.

3.4.1 SBML sematic test. For the SBML sematic test, we used the fourth-order RK (RK4; implicit output) algorithm, and the test cases are as in [57], consisting of 1780 cases. Those test cases measure the ability of a software to interpret SBML files. We adopt the criteria provided with the sematic test as follows:

$$R|C_{i,j} - U_{i,j}| \leq T_a + T_r|C_{i,j}| \tag{1}$$

Where $\boldsymbol{C_{i,j}}$ and $\boldsymbol{U_{i,j}}$ are the expected and estimated value for observable $\boldsymbol{i}$, at time index $\boldsymbol{j}$, $\boldsymbol{T_\alpha}$ is the absolute tolerance for a test case, and $\boldsymbol{T_r}$ is the relative tolerance for a test case. The details of these tolerance values are included in each test case settings file together with the semantic test cases.

In the semantic test, we added a tau-scaler (step size modifier) in RK4 and reduce this scale sequentially from 0.1, 0.01, 0.001, and 0.0001. If it passes the test, the loop is halted and reported as passed.

3.4.2 The stochastic SBML test. For the stochastic SBML test, stochastic algorithms such as SSA, tau-leaping, and CLE are tested by using the SBML discrete stochastic test suite, containing 39 cases [58]. The suite measures the ability to perform stochastic simulation with less emphasis on interpreting SBML. We adopt the criteria for the mean and standard deviation provided with the test cases.

In the mean test, a scaled z-score is represented as follows:

$$Z_t = \sqrt{n}\left(\frac{\bar{X}_t - \mu_t}{\sigma_t}\right) \tag{2}$$

Where $\boldsymbol{Z_t}$ is the z-score at time $\boldsymbol{t}$, $\boldsymbol{\bar{X}_t}$ is the mean of trajectories at $\boldsymbol{t}$, $\boldsymbol{\mu_t}$ is the true or accepted mean at $\boldsymbol{t}$, $\boldsymbol{\sigma_t}$ is the true or accepted standard deviation at $\boldsymbol{t}$, and $\boldsymbol{n}$ is the number of trajectories. If $\boldsymbol{Z_t}$ is in the range (-3,3), the test passes at time $\boldsymbol{t}$; otherwise it fails.

The standard deviation test makes use of a scaled ratio of the variance, which can be summarized as follows:

$$Y_t = \sqrt{\frac{n}{2}}\left(\frac{S_t^{\,2}}{\sigma_t^{\,2}} - 1\right) \tag{3}$$

$$\hat{S}_t^{\,2} = \frac{1}{n}\sum_{i=1}^{n}\left(X_t^{(i)} - \mu_t\right)^2 \tag{4}$$

Where $\boldsymbol{Y_t}$ is the scaled ratio of the variance, $\hat{\boldsymbol{S}}_{\boldsymbol{t}}^{\,2}$ is the variance, and $\boldsymbol{X_t}^{(\boldsymbol{i})}$ is the value of species $\boldsymbol{X}$ at time $\boldsymbol{t}$ for trajectory $\boldsymbol{i}$. If $\boldsymbol{Y_t}$ is in the range (-5,5), then the test passes at time $\boldsymbol{t}$; otherwise it fails.

For the inexact tests, we adopt the mean ratio and standard deviation ratio as suggested in the SBML stochastic test suite for the inexact simulator. We used the 0.95 to 1.05 cutoff for the mean ratio and standard deviation ratio for an acceptable test.

In the stochastic (except CLE) tests, our script reruns the test at most 3 times and it needs to pass at least once to be considered passed. For CLE, we use up to 7 repeats with decreasing tau-scaler. This tau-scaler modifies $\boldsymbol{\tau}$ in both fixed and tau-adaptive CLE. If CLE passes one of the settings, it is considered passed. This is because a randomly chosen tau for CLE may not satisfy the requirement that $\boldsymbol{\tau}$ be small enough for no appreciable change in propensity to occur and large enough that the expected number of occurrences of each reaction channel in the interval is > 1.

3.4.3 Tests for symbolic solution. To test performance on symbolic computation, we created 40 analytical expression test cases and 20 symbolic LNA tests. We used relative absolute deviation (RAD) at each time point from the numerically propagated trajectory as a criterion, defined as follows:

$$RAD = \frac{T - E}{T} \tag{5}$$

where $\boldsymbol{T}$ is the true value of the observable, and $\boldsymbol{E}$ is the estimated value of the observable. $\boldsymbol{RAD}$, $\boldsymbol{T}$ and $\boldsymbol{E}$ can be time-dependent (i.e., if the observable is a species trajectory). If RAD is > 0.05, we consider that it failed the test; otherwise it passed.

For symbolic LNA (covariance) and steady-state concentration, we use numerical results as the true value and RAD to decide a passed or failed test.

3.4.4 Tests for parameter estimation. To test performance on parameter estimation, 45 parameter-estimation test cases were created. The first 40 cases are the same test cases from the symbolic test and the additional 5 cases are for extremely different orders of rate-constant magnitudes. The same RAD as defined in Eq (5) was used for comparing the rate constants to the true values.

For the parameter tests, failed test cases are manually rerun using different settings for at most 3 times before we finally label the performance on that test case.

4. Results and discussion

4.1 Ease of use

BioSANS can be used via a GUI, command line interface, and as a Python import, which we believe is a great improvement over other software because many software packages for systems biology require basic programming and commands [32–38]. Some can be used without coding but have limited functionality and computation power if only using the GUI [40].

In handling reactions with their mathematical expressions (as fluxes and propensity), unlike many previous programs, BioSANS takes an intuitive, simple-to-construct topology representation. In most, this topology file, as introduced in section 3.1, does not require declaration of variables and typing propensity expression. Once a topology file is prepared, most of the features available in BioSANS can be used. BioSANS also offers interconversions of topological files and SBML.

The input files for most existing systems biology software requires declaration of the full propensity expression. Some software packages require transforming reversible reactions into 2 forward reactions [32]. Others only accept SBML files as input, which is difficult to construct manually [59]. Software packages for preparing SBML files [60,61] do exist, but newcomers to the field will have to study several before starting to work on the problem they want to simulate.

4.2 Features supported as compared to some selected software

In Table 1, we list features supported by BioSANS and some selected software with similar purposes: Copasi [40], Cerena [39], Stochpy [32], Stochkit2 [38], Gillespy2 [62], and Pysb [63]. In this list, only BioSANS supports symbolic computations. As far as we know, no existing software supports symbolic computation without the need to declare variables and write ODE expressions. Currently, this new feature is available for time trajectories for the species, LNA covariance matrices, steady-state concentration, and network localization.

Table 1 also shows that BioSANS features a GUI, an easy input format; parameter estimation; network localization; and LNA. Some software packages provide limited SBML support (unquantified) and others do not claim support at all. BioSANS supports all features listed except for histogram distance, propensities, moments, and waiting times in the output. Those exceptions can be calculated from trajectory data, and because BioSANS can run Python script, an advance user can also calculate those unsupported features.

As an important implementation, the topological input of BioSANS supports coupled propagation of dependent and independent systems in one file. This can help mimic a mixture of unrelated chemical reactions that coexist. This feature works for both deterministic and stochastic settings except for SSA and tau-leaping2.

COPASI is a popular and powerful software in systems biology that supports many types of computations. It is widely used in various fields and constantly maintained by a group of developers. It is relatively easy to use with a GUI, console interface, and bindings to other scripting languages. As compared with BioSANS, it does not support symbolic or analytical expression. The GUI in COPASI does not support multiple trajectory sampling (ensemble of trajectory sampling) for stochastic simulations. Model creation in the GUI requires typing the rate law if it is not present in the list of functions or not previously saved. It supports models imported from SBML, but how much support it provides to the SBML stochastic test suite is not clear. The console interface and bindings it provides can be used together or called in other programming languages such as Python, c/c++, etc. but is usually critical with versions. Together with some non-trivial programming skills, COPASI is a powerful software for systems biology.

Stochpy is a Python-based software that is dedicated to stochastic simulation. It can make use of Python libraries, which can be very powerful to use. It can interface with other software like Cain, Stochkit2, etc. which gives it speed for problems that do not involved fancy SBML features. It is the only software that passed the SBML stochastic test suite when it was published. Model creation is easy and follows the PySCeS model description language, but it requires typing the rate law expression. It can import SBML files, but the SBML semantic test

Table 1. Feature comparison between BioSANS and selected software with a similar purpose.

Feature	Copasi	Stochkit2	Stochpy	Cerena	GillesPy2	pysb	BioSANS
Numerical analysis							
Exact SSA	*	*	*	*	*	*	*
Inexact SSA	*	*	*	*	*	*	*
LNA	*			*			*
Parameter estimation	*		b	*	b	b	*
Network localization							*
Symbolic analysis (Analytical expression derivation)							
Species analytical expression			b		b	b	*
Steady state concentration			b		b	b	*
LNA							*
Network localization							*
Simulation options							
SBML (stochastic)	*	?	*	*	?	?	*
SBML (semantic)	*		?	?	?	?	*
Easy to prepare input[a]	*		*				*
Actual interval output	*	?	*		*	*	*
Fixed-interval output	*	*	*		*	*	*
Output analysis							
Auto-correlation			*				*
Histogram distance		*					
Propensities	*		*				
Moments			*				
Waiting times			*				
Probability density with time							*
time-slice of densities							*
Average of trajectory							*
Bootstrapped covariance							*
Software characteristics							
Plotting	*	?	*		*	*	*
Data exportation	*	*	*		*	*	*
GUI	*						*
console	*	*	*		*	*	*
Flexible environment	*		*		*	*	*

[a]—do not necessarily need to type rate law expression, no coding.

[b]—can be supported with minor to complex coding since the software is based on python.

[?]—feature not fully supported, with limited capability.

SSA, stochastic simulation algorithm; SBML, systems biology markup language;

GUI, graphical user interface; LNA—linear noise approximation.

https://doi.org/10.1371/journal.pone.0256409.t001

results are not seen on the SBML website. It provides support for fixed-interval output and actual-time output. However, as a Python module, it can only be used via command line and as a library import. Moreover, as compared with BioSANS, Stochpy does not provide support for symbolic computation, parameter estimation, LNA, and deterministic calculation.

Stochkit2 is a software for discrete stochastic simulation, its main advantage being speed and parallelization. Models can be created in an xml-like format using certain tags and also allowing for defining arbitrary function. Its latest version released support events (conditional statements) as well. However, Stochkit2 does not quantify how much SBML support it provides. As compared with BioSANS, the rate law expression is not automatically inferred and requires typing under the rate tag. It does not provide support for GUI, parameter estimation, and symbolic computation nor theoretical computation such as LNA.

CERENA is a MATLAB-based software that provides many functions. It supports time-dependent propensity, parameter estimation, LNA, and other high-level mathematical analysis. Models can be created using a model definition file, which uses MATLAB-like declarations. It supports importing SBML files, which allows for producing the model definition file automatically. Nevertheless, CERENA does not provide support for symbolic computation, does not provide a GUI and can only be used by MATLAB programmers. The model definition file it provides is not as intuitive as chemical elementary equations.

Gillespy2 is a python-based software for model building and stochastic simulation. It also supports deterministic integration. Models can be build using an object-oriented approach and would only requires few lines of codes to build a model. Its main advantage is speed of computation due to its interface with Stockhit2. It also supports SBML but currently does not support events, function definitions, propensity modification, etc. Parameter estimation and symbolic computations are not specifically supported. It did not provide a graphical user interface and did not provide support to theory-based noise analysis such as LNA. Gillespy2 syntax is not as easy as that in using BioSANS as a library. In general, BioSANS is much easier to use and have more supported features.

Pysb is a general-purpose software for systems biology that supports a wide variety of features including deterministic and stochastic simulation. It provides interfaces for many other softwares including BioNetGen which provides its computing power in stochastic simulations. It has a similar model building strategy as that of Gillespy2 but with more details and clarity in model description. However, it also did not specifically provide support to symbolic computation and parameter estimation. The SBML support it provides is also not quantified and did not provide a graphical user interface. The use of BioSANS as a library is much simpler and easier compared to Pysb model building syntax.

BioSANS supports a wide variety of features as mentioned above and in the previous section. The GUI and console interface (structure simulation language) allow for multiple trajectory simulation and support parallel runs by taking advantage of a multiprocessing library [64] in Python. It supports time-dependent propensities and complex logic in the topology but requires some experience to properly set up complicated logic if necessary. Because it is Python-based, it can also make use of powerful libraries from Python, which will be very useful in automation (i.e., model selection if used as a Python library). It did not specifically provide an interface with fast solvers such as Stochkit2 but it won't be too much of a problem to make them interoperate in a python environment. Hence, BioSANS is a software that makes systems biology modeling available even to nonexperts and is well suited for newcomers in the field. It can be used for teaching systems biology and for performing simple to complex modeling tasks.

4.3 Performance and testing of the software

The following sub-section summarizes the performance of BioSANS against various tests. The detailed results in each test are provided in section 11 of the supplementary material.

4.3.1 Semantic test. In Table 2, we list the results of SBML sematic tests comparing BioSANS and other software that submit results to the SBML database website. BioSANS passed most of the test cases, with about 70% correct test results. This is already above the average level for SBML support among all the software packages that submit their results.

Table 2. Comparison of performance of several software packages for the SBML semantic test case.

Software[a]	Date submitted	Passed
BioUML 2018.2	18-Jun-18	100%
Morpheus 2.1	18-Jan-19	81%
Biostoch/BioSANS1.0	7-Dec-20	71%
BioUML 0.9.5	4-Jun-13	67%
COPASI Build 4.23.189	12-Jun-18	67%
iBioSim 2.4.2	4-Feb-13	67%
Simulation Core Library 1.2	28-Mar-13	63%
libRoadRunner 1.3	29-Dec-14	60%
RoadRunner 2.10.0	16-Jan-13	60%
WinBEST-KIT 2.0.0	20-Dec-18	60%
LibSBMLSim 1.1.0	31-Jan-13	55%
AMICI 0.11.8	1-Sep-19	20%
FluxBalance 1.9	21-Jul-14	1%

[a] -The software included here are those that submit their results to an SBML website.

https://doi.org/10.1371/journal.pone.0256409.t002

Currently only BioUML (Kolpakov *et al*., 2019) provides 100% support for all semantic standards set by SBML. BioUML is a general package of programs for systems biology that provides a lot of functionality (300+ different types of analysis if connected to Galaxy), including deterministic and stochastic simulation. Its major advantage is being a web-based software, so it is accessible to users who have a BioStore account. It supports analysis of biomedical data from omics experiments, parameter estimation, databased links, etc. It can generate codes in Java from models created via its diagram editor in a drag-and-drop manner. Conversely, BioUML does not provide support for symbolic computation, LNA, network localization and does not mention performance on stochastic SBML tests.

Morpheus is a modeling software that focuses on the study of multi-scale and multicellular systems using ordinary and partial differential equations [65]. Models can be created by describing them in the GUI instead of in a code and the inputs can be symbolic expression. It can plot cellular processes and dynamics in an image, depicting actual cells, tissues, etc. involved in the process. However, even with the GUI, a user still needs to define the global variables, the system of differential equation and other constraints. The creators did not claim to give output in a form of symbolic/analytical expression. The support for parameter estimation is limited to parameter sweeping.

Currently, the SBML support BioSANS offers has some limitations. Features such as overlapping delays, CSymbolDelay, use of infinity symbols, and latest features available only in level 3.1 and above are not fully supported. To interpret SBML files, BioSANS converts the SBML file to a topology file. If BioSANS fails to convert the file properly, there will be an error at run time and no output will be returned. Manual inspection and basic editing of the converted file may help correct the conversion error. Still, this will require understanding the system to properly correct the file or reading the SBML manually to check for inconsistencies.

4.3.2 Performance on stochastic test. BioSANS exact stochastic algorithms are tested by using the SBML discrete stochastic model test suite (SBML DSMTS). In the DSMTS test, a good number (10,000 in our case) of stochastic trajectories was collected for each of the 39 test models, with the standard deviation for the mean and the variance calculated at 50 time points. Table 3 lists the algorithms and the number of test cases with their corresponding percentage of time points passed in the DSMTS tests as columns. Following the suggested routine, owing

Table 3. Comparison between StochPy and BioSANS in passing tests in the SBML discrete stochastic model test suite (DSMTS).

Algorithm	Mean test				Standard deviation test			
	100%	95–99%	90–95%	<90%	100%	95–99%	90–95%	<90%
BioSANS								
SSA[a]	36	0	1	2	36	1	1	1
SSA	36	2	1	0	36	1	1	1
Tau-leaping-1	35	3	0	1	36	1	0	2
Tau-leaping-2	37	0	0	2	34	2[b]	2	1
CLE1	31	4	1	3	23	1	0	15
CLE2	31	5	0	3	21	0	0	18
Stochpy								
SSA[a]	37	0	0	2	36	1	0	2
SSA	36	1	0	2	37	0	0	2
Tau-leaping	37	0	0	2	34	0	1	4

[a]—Shown are number of test cases with their corresponding number of cases passed for a percentage range among 39 cases using the exact test. All others were tested by using the inexact test.

[b]—Very close to 100% and sometimes correct for more than 3 trials.

SSA, stochastic simulation algorithm.

https://doi.org/10.1371/journal.pone.0256409.t003

to the probabilistic nature of such simulations, 2 to 3 error cases in the mean test and 4 to 6 errors in the variance test are expected for a perfectly working stochastic algorithm under the exact SSA test [58]. The performance of our SSA implementation falls within the criteria, with few cases suffering considerable discrepancy mostly in the timed trigger cases, possibly because of the treatment of exact timing and the subsequent error from such an uncertainty. The time-triggered cases are still within the 90% to 99% window if tested under the inexact test. To the best of our knowledge, we cannot find software comparisons reported for stochastic test cases. We have also performed tests with StochPy, which is claimed to have passed SBML DSMTS tests. StochPy and BioSANS show comparable performance for SSA as listed in Table 3. Thus, BioSANS SSA implementation is reliable.

The approximate algorithms in BioSANS were also tested by using SBML DSMTS. For approximate simulators, SBML DSMTS suggested observing the ratio of the mean and variance obtained from reasonable sampling trajectories (10,000 in our case) with their corresponding standard values (provided in the test). The inexact tests in Table 3 show the number of cases for a certain percentage with estimated mean or variance within 0.95 to 1.05 of the standard value. Tau-leaping2 has 2 errors in the mean test and 5 errors in the standard deviation test. This is still within the allowed range of error suggested in SBML DSMTS. The data in Table 3 are the best outcome for 3 repeats of 10,000 trajectory simulations. The detailed results of the tests and explanation for the failed results are included in supporting information. A failed result does not necessarily mean it is too far from analytically derived trajectories, and we believe that it is still useful for qualitative purposes. From the inexact test, the BioSANS tau-leaping-2 is comparable to StochPy tau-leaping implementation. CLE1, CLE2, and tau-leaping-1 have slightly more errors than the allowable number, but they are generally faster algorithms. Therefore, these methods are suitable for systems that require a large amount of computation, but they should be used cautiously.

BioSANS SSA solver computing speed where also tested using SBML DSMTS and compared with solvers from other softwares such as; SSACSolver and NumPySSASolver from Gillespy2, SSA solver in Stochpy, and SSA solver from Pysb. We picked the test cases where most

Table 4. Performance based on simulation time at 10, 100, 1000, and 10000 trajectories.

cases[a]	SSACSolver				NumPySSASolver				BioSANS SSA				Stochpy SSA				Pysb SSA			
	10	**100**	**1000**	**10000**	**10**	**100**	**1000**	**10000**	**10**	**100**	**1000**	**10000**	**10**	**100**	**1000**	**10000**	**10**	**100**	**1000**	**10000**
Stiff problems																				
5	2.5	3.1	8.1	59.7	8.5	85.2	843	8902	4.6	28.7	276	2673	13.6	131	1292	13466	0.7	3.6	31.2	327.3
23	2.6	3.2	8.3	65.8	8.6	85.9	897	9173	5.0	31.2	299	2966	14.0	145	1357	14730	0.8	3.5	31.5	322.9
non-Stiff problems																				
1	2.5	2.5	2.8	9.7	0.1	1.0	8.7	90.4	0.1	0.4	2.9	27.3	0.2	1.5	15.1	157.4	0.5	2.0	17.4	187.5
2	2.5	2.5	2.8	7.6	0.1	0.9	8.6	88.9	0.1	0.4	3.0	27.3	0.2	1.7	15.1	159.0	0.5	2.1	17.5	187.4
7	2.5	2.6	2.8	7.9	0.1	0.9	9.1	95.9	0.1	0.4	2.9	27.9	0.2	1.8	16.5	167.3	0.5	2.1	17.7	188.0
8	2.5	2.5	2.8	7.6	0.1	0.9	8.8	90.9	0.1	0.4	3.0	27.1	0.2	1.6	15.0	159.1	0.5	2.1	17.5	185.9
9	2.5	2.5	2.8	7.6	0.1	0.9	8.4	90.0	0.1	0.4	2.9	27.4	0.2	1.6	15.3	159.9	0.5	2.0	17.3	187.2
10	2.5	2.5	2.8	7.8	0.1	0.8	8.6	90.6	0.1	0.4	2.9	27.4	0.2	1.6	15.4	159.2	0.5	2.0	17.5	187.5
12	2.6	2.6	2.8	7.5	0.1	0.9	9.0	95.4	0.1	0.4	2.9	27.8	0.2	1.6	14.5	162.5	0.5	1.9	17.4	188.6
13	2.5	2.5	2.8	7.6	0.1	0.9	8.6	92.0	0.1	0.4	2.9	27.6	0.2	1.6	15.0	170.7	0.5	2.0	17.5	187.0
14	2.5	2.5	2.8	7.8	0.1	0.9	9.0	95.3	0.1	0.4	2.9	27.6	0.2	1.6	14.8	161.3	0.5	2.1	18.1	193.7
15	2.5	2.5	2.8	7.9	0.1	0.9	9.3	94.9	0.1	0.5	3.0	27.6	0.2	1.6	15.0	161.6	0.6	2.1	18.3	195.2
16	2.5	2.5	2.8	7.5	0.1	0.9	8.7	92.7	0.1	0.4	2.9	27.6	0.2	1.6	14.7	160.5	0.5	2.1	17.8	188.6
17	2.5	2.5	2.7	7.7	0.1	1.1	8.8	92.5	0.1	0.4	3.3	27.1	0.2	1.6	14.9	159.6	0.5	2.1	17.5	186.9
31	2.5	2.6	2.8	7.9	0.1	0.7	6.4	68.2	0.1	0.3	2.2	20.0	0.2	1.4	13.5	131.3	0.6	2.3	19.3	207.8

[a]—DSMTS test cases with SBML syntax that gillespy2 fully supports out of 39 tests cases.

DSMTS—Discrete Stochastic Method Test Suite.

https://doi.org/10.1371/journal.pone.0256409.t004

of this software can parse the SBML correctly. Table 4 summarized the speed of each solver at 10, 100, 1000, and 10000 trajectories for stiff and non-stiff problems. The simulation was performed on machines equipped with 48 CPU cores based on Intel(R) Xeon(R) Silver 4214 @ 2.20GHz with x86_64 architectures, 211 GB of available memory, 37 GB RAM, and 7 GB swap memory.

When it comes to speed, in stiff problems SSACSolver is the fastest, followed by Pysb SSA, BioSANS SSA, NumPySSASolver, and by Stochpy SSA. For non-stiff cases, SSACSolver is still the fastest but now followed by BioSANS SSA, NumPySSASolver, Stochpy SSA and by Pysb SSA. BioSANS SSA solver is comparable to SSACSolver for non-stiff problems from 10 to 1000 trajectories but is slower at 10000 trajectories. Currently, BioSANS SSA have problems on speed for stiff problems since the stiffness of ODE cannot be parallelized. However, BioSANS can simulate multi-trajectory problems quite well with the help of multiprocessing specially for non-stiff problems.

From Table 4, we can see that as the number of trajectories increases, the computing time linearly follows for NumPySSASolver and Stochpy SSA. For SSACSolver, Pysb SSA, and BioSANS SSA, they have less computing time compared to what have been predicted linearly. However, at above 1000 trajectories, the computing time of BioSANS SSA, and Pysb SSA becomes linearly dependent to the number of trajectories. On the other hand, SSACSolver is still way faster than what we would expect from a massive number of trajectory increase.

4.3.3 Performance in symbolic tests. The following table summarizes the performance of BioSANS in symbolic computation. Typically, all species in symbolic solution should be a function of time $\boldsymbol{t}$. It would be desirable for general solutions, with symbolic dependence on the initial concentration $\boldsymbol{x_0}$, and parameters such as rate constant $\boldsymbol{k}$. In BioSANS, we provide a way to report analytical expressions in 4 different categories of functions such as $\boldsymbol{f(t)}$, $\boldsymbol{f(t,x_0)}$,

Table 5. Performance of BioSANS in generating analytical expression for species concentration with different symbolic computation modes.

Schemes	Variables	Correct expression	No expression
Pure symbolic	***t*, x_0, *k***	50%	50%
Semi-symbolic	***t*, *k***	50%	50%[a]
	***t*, x_0**	93%	7%
	t	97%	3%

[a]—Some of the expressions obtained were correct, but not all of the expressions needed were obtained.

https://doi.org/10.1371/journal.pone.0256409.t005

f(t,k), and ***f(t,k,x_0)***. This notation corresponds to functions of the remaining variables (after substitution) before symbolic computation is attempted. We refer to ***f(t,k,x_0)*** as a pure symbolic expression and others are called semi-analytical expressions. The semi-analytical expressions are easier to solve by using SymPy, which has the majority of test cases passed, as seen in Table 5. The pure symbolic expression is not easy to handle and often generates large analytical expressions. Sometimes it also takes a lot of time to finish computation.

For the analytical expression, BioSANS supports only linear differential equations and few nonlinear differential equations. There were some linear ODEs that BioSANS could not fully support, such as overlapping reversible reaction, overlapping cyclic and reversible reaction, cyclic structure inside cyclic structure, and overlapping cyclic structure. Some of these not-fully-supported linear ODEs can still work using the ***f(t)*** mode. Our test cases consist of reactions with fewer than 10 chemical species. It is possible to solve systems with greater than 10 species especially using ***f(t)*** and ***f(t,x_0),*** mode. Other modes will take longer, and the expression might be larger, so not usable for physical interpretation. The ***f(t,x_0)*** mode describes the effect of initial concentration on the species analytical expression and is easier to solve than the pure symbolic mode. Dependence on the rate constant can be inspected by using the ***f(t,k)*** mode, but it is almost as difficult to calculate as with a pure symbolic solution. The ***f(t)*** mode is useful if we only want to know the trajectory with respect to time and do not intend to discover the relevance of the initial concentration and rate constant in the final trajectory.

Symbolic expression with the steady-state LNA for the covariance matrices can be reported in the same 4 modes as well. The steady-state concentration expression is included when computing symbolic LNA. Table 6 summarizes the performance in symbolic computation for the covariance matrix analytical expression and steady-state analytical expression. The None entry in the variables column pertains to a numeric answer but is symbolically derived.

The symbolic computation functions in BioSANS are currently useful for small- and moderate-sized systems. We think it necessary to have symbolic capability because analytical expression can give more physical insights than can simulations. Whenever a system can be

Table 6. Performance of BioSANS in generating LNA and a steady-state analytical expression.

Variables	Correct expression		Takes too long
	Steady state	Cov/Var	
LNA-symbolic-*f*(x_0, *k*)	95%	55%	45%
LNA-symbolic-*f*(x_0)	95%	90%	10%
LNA-symbolic-*f*(*k*)	95%	55%	45%
None[a]	95%	90%	10%[b]

[a]—Numeric answer but symbolically derived.

[b]—Some returns wrong expression.

https://doi.org/10.1371/journal.pone.0256409.t006

Table 7. Performance of different algorithms in BioSANS for parameter estimation.

Algorithms	Speed[a]	Rate constant	Trajectory
		Percent passed	
Nelder-Mead (NM)	Fast	93%	98%
Monte Carlo EM	Slow	93%	93%
Differential evolution (DE)	Slow	93%	93%
NM-DE-NM combined	Slow	98%	98%
Powell	Fast	91%	91%
L-BFGS-B	Fast	80%	87%
Parameter slider[b]	N/A	44%	64%

[a]—A rough estimate of the time required. Usually "fast," within 5 min, with a typical desktop computer. (slow $\geq$ 30 min).

[b]—A graphical user interface element for sliding values of parameter.

https://doi.org/10.1371/journal.pone.0256409.t007

simplified to a size whereby we can generate an analytical expression, our BioSANS symbolic computation can help generate a reasonable expression.

Symbolic solving ODEs with programs has become more available in recent years. As the development of SymPy progresses, BioSANS symbolic solving could be further improved.

4.3.5 Performance in parameter estimation. The test for parameter estimation is assessed (using RAD) by the ability to reproduce the trajectory and rate constant values. Simulated trajectories are generated for 45 models with given rate constants. Most of the simulated trajectories contain 201 data points, except for test cases 15, 19, and 43, with 1000 data points. Test cases 15 and 19 include reversible reactions, and test case 43 is a stiff problem. The algorithms in Table 7 are used to estimate the rate constants given all species concentrations as a function of time. Even though this is an ideal case because most biological experiments allow for measuring only a few numbers of species and with limited time points, our test for BioSANS aimed to provide the minimal requirement for estimating parameters for mathematical modeling.

Table 7 summarizes the performance of BioSANS parameter-estimation algorithms given all the species trajectory. Most of the algorithms can capture the true rate constant and trajectory except for the Newton-based algorithm, L-BFGS-B. See section 11.5 of the supplementary material for a detailed classification of test results. There is always a chance of being insensitive to some of the parameters in the dynamics of a system, and parameter estimation based on the trajectories is subject to this natural constraint. Therefore, even if some rate constants are not close to the true value, the trajectories can still be reproduced.

Insights into parameter perturbation is easier to interpret visually. BioSANS offers a parameter slider, which is a GUI element that allows for visually inspecting perturbation effects. With this feature, a user can modify parameters and see the effect of the modification in the results shown as plots. Given some plotted experimental data, we can slide the parameter such that the results of the ODE are similar to those of the experimental data. This will allow for some insights into the parameter in a qualitative sense. However, this is only useful for small systems; as we can see in Table 7, it is the least performant.

4.4 Other functionalities

The law of localization in chemical reaction networks, known as network localization, mentioned in section 3.0, is a useful way to investigate perturbation effects of a network based on only topological information [48]. BioSANS supports symbolic and numeric computation of network localization in the form of network sensitivity. This is a qualitative alternative to symbolic solution of species concentration and covariance when SymPy fails to give an answer. From network localization, one can infer the effect of perturbation of a rate constant to species steady-state concentration and covariance. However, we did not provide a test case for these functionalities.

5. Conclusion

BioSANS is a software for both experts and nonexperts that supports most of the tasks needed in systems biology modeling including useful features left out by other software. The symbolic computation capability in BioSANS provides analytical expression of solvable cases without the need to type the ODE expression and declaring variables. From model creation, propagation, and analysis, BioSANS provides reliable algorithms that can facilitate the modeling process.

BioSANS can be extended to other fields that make use of ordinary differential equations. For example, ODE is heavily used in pharmacokinetics, pharmacodynamics, Epidemiology, environmental models, economic models etc. ODE is also used in studying reaction mechanisms in chemistry. As long as we can represent the state of the systems as symbols that follow ODE, BioSANS may be used to model such systems.

Supporting information

S1 File. BioSANS manual/tutorial/supplementary.
(PDF)

Acknowledgments

We express our gratitude to Mr. Yu-Chuan Chen and Dr. Yi-chen Chen for testing BioSANS and providing suggestions for improvements.

Author Contributions

Conceptualization: Erickson Fajiculay.

Funding acquisition: Chao-Ping Hsu.

Methodology: Erickson Fajiculay.

Project administration: Chao-Ping Hsu.

Software: Erickson Fajiculay.

Supervision: Chao-Ping Hsu.

Validation: Erickson Fajiculay, Chao-Ping Hsu.

Visualization: Erickson Fajiculay.

Writing – original draft: Erickson Fajiculay, Chao-Ping Hsu.

Writing – review & editing: Erickson Fajiculay, Chao-Ping Hsu.

References

1. Weng G, Bhalla US, Iyengar R. Complexity in Biological Signaling Systems. Science. 1999; 284: 92–96. https://doi.org/10.1126/science.284.5411.92 PMID: 10102825
2. Kaizer JS, Heller AK, Oberkampf WL. Scientific computer simulation review. Reliability Engineering & System Safety. 2015; 138: 210–218. https://doi.org/10.1016/j.ress.2015.01.020

3. Lumb JR. Computer simulation of biological systems. Mol Cell Biochem. 1987; 73: 91–98. https://doi.org/10.1007/BF00219423 PMID: 3561387
4. Richards D, Berry S, Howard M. Illustrations of Mathematical Modeling in Biology: Epigenetics, Meiosis, and an Outlook. Cold Spring Harbor Symposia on Quantitative Biology. 2012; 77: 175–181. https://doi.org/10.1101/sqb.2013.77.015941 PMID: 23339832
5. Servedio MR, Brandvain Y, Dhole S, Fitzpatrick CL, Goldberg EE, Stern CA, et al. Not Just a Theory—The Utility of Mathematical Models in Evolutionary Biology. PLOS Biology. 2014; 12: e1002017. https://doi.org/10.1371/journal.pbio.1002017 PMID: 25489940
6. Mead BE, Karp JM. All models are wrong, but some organoids may be useful. Genome Biology. 2019; 20: 66. https://doi.org/10.1186/s13059-019-1677-4 PMID: 30917855
7. Palaniappan SK, Yachie-Kinoshita A, Ghosh S. Computational Systems Biology. In: Ranganathan S, Gribskov M, Nakai K, Schönbach C, editors. Encyclopedia of Bioinformatics and Computational Biology. Oxford: Academic Press; 2019. pp. 789–795.
8. Rozenberg G, Bäck T, Kok JN, editors. Handbook of Natural Computing. Berlin Heidelberg: Springer-Verlag; 2012. https://www.springer.com/gp/book/9783540929093.
9. Bersani AM, Bersani E, Mastroeni L. Deterministic and stochastic models of enzymatic networks—applications to pharmaceutical research. Computers & Mathematics with Applications. 2008; 55: 879–888. https://doi.org/10.1016/j.camwa.2006.12.092
10. Faeder JR, Blinov ML, Goldstein B, Hlavacek WS. Rule-based modeling of biochemical networks. Complexity. 2005; 10: 22–41. https://doi.org/10.1002/cplx.20074
11. van Riel NAW. Dynamic modelling and analysis of biochemical networks: mechanism-based models and model-based experiments. Briefings in Bioinformatics. 2006; 7: 364–374. https://doi.org/10.1093/bib/bbl040 PMID: 17107967
12. Shen Yan-Yan W S-Q. Analyzing the Evolution of Biochemical Reaction System with a Complex Network Based Approach. Enz Eng. 2013; 02. https://doi.org/10.4172/2329-6674.1000113
13. Yan C-CS, Chepyala SR, Yen C-M, Hsu C-P. Efficient and flexible implementation of Langevin simulation for gene burst production. Scientific Reports. 2017; 7: 16851. https://doi.org/10.1038/s41598-017-16835-y PMID: 29203832
14. Sharan R, Shamir R. CLICK: a clustering algorithm with applications to gene expression analysis. Proc Int Conf Intell Syst Mol Biol. 2000; 8: 307–316. PMID: 10977092
15. Yan C-CS, Hsu C-P. The fluctuation-dissipation theorem for stochastic kinetics—Implications on genetic regulations. The Journal of Chemical Physics. 2013; 139: 224109. https://doi.org/10.1063/1.4837235 PMID: 24329058
16. Frontiers | Gene Expression Noise Produces Cell-to-Cell Heterogeneity in Eukaryotic Homologous Recombination Rate | Genetics. [cited 27 Jul 2020]. https://www.frontiersin.org/articles/10.3389/fgene.2019.00475/full.
17. Dar RD, Hosmane NN, Arkin MR, Siliciano RF, Weinberger LS. Screening for noise in gene expression identifies drug synergies. Science. 2014; 344: 1392–1396. https://doi.org/10.1126/science.1250220 PMID: 24903562
18. Locke JCW, Southern MM, Kozma-Bognár L, Hibberd V, Brown PE, Turner MS, et al. Extension of a genetic network model by iterative experimentation and mathematical analysis. Mol Syst Biol. 2005; 1: 2005.0013. https://doi.org/10.1038/msb4100018 PMID: 16729048
19. Sun X, Zhang J, Zhao Q, Chen X, Zhu W, Yan G, et al. Stochastic modeling suggests that noise reduces differentiation efficiency by inducing a heterogeneous drug response in glioma differentiation therapy. BMC Systems Biology. 2016; 10: 73. https://doi.org/10.1186/s12918-016-0316-x PMID: 27515956
20. Wu J-F, Tsai H-L, Joanito I, Wu Y-C, Chang C-W, Li Y-H, et al. LWD-TCP complex activates the morning gene CCA1 in Arabidopsis. Nat Commun. 2016; 7: 13181. https://doi.org/10.1038/ncomms13181 PMID: 27734958
21. Ewald J, Sieber P, Garde R, Lang SN, Schuster S, Ibrahim B. Trends in mathematical modeling of host–pathogen interactions. Cell Mol Life Sci. 2020; 77: 467–480. https://doi.org/10.1007/s00018-019-03382-0 PMID: 31776589
22. Barrila J, Crabbé A, Yang J, Franco K, Nydam SD, Forsyth RJ, et al. Modeling Host-Pathogen Interactions in the Context of the Microenvironment: Three-Dimensional Cell Culture Comes of Age. Infect Immun. 2018; 86: e00282–18. https://doi.org/10.1128/IAI.00282-18 PMID: 30181350
23. Casadevall A, Pirofski L. Host-Pathogen Interactions: Basic Concepts of Microbial Commensalism, Colonization, Infection, and Disease. Infect Immun. 2000; 68: 6511–6518. https://doi.org/10.1128/IAI.68.12.6511-6518.2000 PMID: 11083759

24. Hufsky F, Lamkiewicz K, Almeida A, Aouacheria A, Arighi C, Bateman A, et al. Computational strategies to combat COVID-19: useful tools to accelerate SARS-CoV-2 and coronavirus research. Briefings in Bioinformatics. 2021; 22: 642–663. https://doi.org/10.1093/bib/bbaa232 PMID: 33147627

25. Cava C, Bertoli G, Castiglioni I. In Silico Discovery of Candidate Drugs against Covid-19. Viruses. 2020; 12: 404. https://doi.org/10.3390/v12040404 PMID: 32268515

26. Nashiry A, Sumi SS, Islam S, Quinn JMW, Moni MA. Bioinformatics and system biology approach to identify the influences of COVID-19 on cardiovascular and hypertensive comorbidities.: 15.

27. Zoabi Y, Deri-Rozov S, Shomron N. Machine learning-based prediction of COVID-19 diagnosis based on symptoms. npj Digital Medicine. 2021; 4: 1–5. https://doi.org/10.1038/s41746-020-00373-5 PMID: 33398041

28. Maus C, Rybacki S, Uhrmacher AM. Rule-based multi-level modeling of cell biological systems. BMC Systems Biology. 2011; 5: 166. https://doi.org/10.1186/1752-0509-5-166 PMID: 22005019

29. Blinov ML, Faeder JR, Goldstein B, Hlavacek WS. BioNetGen: software for rule-based modeling of signal transduction based on the interactions of molecular domains. Bioinformatics. 2004; 20: 3289–3291. https://doi.org/10.1093/bioinformatics/bth378 PMID: 15217809

30. Hwang M, Garbey M, Berceli SA, Tran-Son-Tay R. Rule-Based Simulation of Multi-Cellular Biological Systems—A Review of Modeling Techniques. Cel Mol Bioeng. 2009; 2: 285–294. https://doi.org/10.1007/s12195-009-0078-2 PMID: 21369345

31. Gruenert G, Ibrahim B, Lenser T, Lohel M, Hinze T, Dittrich P. Rule-based spatial modeling with diffusing, geometrically constrained molecules. BMC Bioinformatics. 2010; 11: 307. https://doi.org/10.1186/1471-2105-11-307 PMID: 20529264

32. Maarleveld TR, Olivier BG, Bruggeman FJ. StochPy: A Comprehensive, User-Friendly Tool for Simulating Stochastic Biological Processes. PLOS ONE. 2013; 8: e79345. https://doi.org/10.1371/journal.pone.0079345 PMID: 24260203

33. Siso-Nadal F, Ollivier JF, Swain PS. Facile: a command-line network compiler for systems biology. BMC Systems Biology. 2007; 1: 36. https://doi.org/10.1186/1752-0509-1-36 PMID: 17683566

34. Landeros A, Stutz T, Keys K, Alekseyenko A, Sinsheimer JS, Lange K, et al. BioSimulator.jl: Stochastic simulation in Julia. Comput Methods Programs Biomed. 2018; 167: 23–35. https://doi.org/10.1016/j.cmpb.2018.09.009 PMID: 30501857

35. Erhard F, Friedel CC, Zimmer R. FERN–a Java framework for stochastic simulation and evaluation of reaction networks. BMC Bioinformatics. 2008; 9: 356. https://doi.org/10.1186/1471-2105-9-356 PMID: 18755046

36. Sheppard PW, Rathinam M, Khammash M. SPSens: a software package for stochastic parameter sensitivity analysis of biochemical reaction networks. Bioinformatics. 2013; 29: 140–142. https://doi.org/10.1093/bioinformatics/bts642 PMID: 23104889

37. Ostrenko O, Incardona P, Ramaswamy R, Brusch L, Sbalzarini IF. pSSAlib: The partial-propensity stochastic chemical network simulator. PLOS Computational Biology. 2017; 13: e1005865. https://doi.org/10.1371/journal.pcbi.1005865 PMID: 29206229

38. Sanft KR, Wu S, Roh M, Fu J, Lim RK, Petzold LR. StochKit2: software for discrete stochastic simulation of biochemical systems with events. Bioinformatics. 2011; 27: 2457–2458. https://doi.org/10.1093/bioinformatics/btr401 PMID: 21727139

39. Kazeroonian A, Fröhlich F, Raue A, Theis FJ, Hasenauer J. CERENA: ChEmical REaction Network Analyzer—A Toolbox for the Simulation and Analysis of Stochastic Chemical Kinetics. PLOS ONE. 2016; 11: e0146732. https://doi.org/10.1371/journal.pone.0146732 PMID: 26807911

40. Hoops S, Sahle S, Gauges R, Lee C, Pahle J, Simus N, et al. COPASI—a COmplex PAthway SImulator. Bioinformatics. 2006; 22: 3067–3074. https://doi.org/10.1093/bioinformatics/btl485 PMID: 17032683

41. Elf J, Ehrenberg M. Fast Evaluation of Fluctuations in Biochemical Networks With the Linear Noise Approximation. Genome Res. 2003; 13: 2475–2484. https://doi.org/10.1101/gr.1196503 PMID: 14597656

42. Stochastic Processes in Physics and Chemistry—3rd Edition. [cited 2 Jul 2021]. https://www.elsevier.com/books/stochastic-processes-in-physics-and-chemistry/van-kampen/978-0-444-52965-7.

43. Kampen NGV. Stochastic Processes in Physics and Chemistry. 3 edition. Amsterdam; Boston: North Holland; 2007.

44. Rossum GV, Drake FL. Python 3 Reference Manual: Hampton, NH: CreateSpace Independent Publishing Platform; 2009.

45. van der Walt S, Colbert SC, Varoquaux G. The NumPy Array: A Structure for Efficient Numerical Computation. Computing in Science Engineering. 2011; 13: 22–30. https://doi.org/10.1109/MCSE.2011.37

46. Virtanen P, Gommers R, Oliphant TE, Haberland M, Reddy T, Cournapeau D, et al. SciPy 1.0: fundamental algorithms for scientific computing in Python. Nature Methods. 2020; 17: 261–272. https://doi.org/10.1038/s41592-019-0686-2 PMID: 32015543

47. Meurer A, Smith CP, Paprocki M, Čertík O, Kirpichev SB, Rocklin M, et al. SymPy: symbolic computing in Python. PeerJ Computer Science. 2017; 3: e103. https://doi.org/10.7717/peerj-cs.103

48. Okada T, Mochizuki A. Law of Localization in Chemical Reaction Networks. Phys Rev Lett. 2016; 117: 048101. https://doi.org/10.1103/PhysRevLett.117.048101 PMID: 27494502

49. Chemistry (IUPAC) TIU of P and A. IUPAC—chemical reaction equation (C01034). [cited 9 Apr 2021].

50. Gillespie DT. Exact stochastic simulation of coupled chemical reactions. J Phys Chem. 1977; 81: 2340–2361. https://doi.org/10.1021/j100540a008

51. Gillespie DT. The chemical Langevin equation. The Journal of Chemical Physics. 2000; 113: 297–306. https://doi.org/10.1063/1.481811

52. Cao Y, Gillespie DT, Petzold LR. Efficient step size selection for the tau-leaping simulation method. The Journal of Chemical Physics. 2006; 124: 044109. https://doi.org/10.1063/1.2159468 PMID: 16460151

53. Petzold L. Automatic Selection of Methods for Solving Stiff and Nonstiff Systems of Ordinary Differential Equations. SIAM J Sci and Stat Comput. 1983; 4: 136–148. https://doi.org/10.1137/0904010

54. Ascher UM, Petzold LR. Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations. SIAM; 1998.

55. Press WH, Teukolsky SA. Adaptive Stepsize Runge-Kutta Integration. Comput Phys. 1992; 6: 188. https://doi.org/10.1063/1.4823060

56. Higham DJ. An Algorithmic Introduction to Numerical Simulation of Stochastic Differential Equations.: 22.

57. Hucka M, Smith L, Bergmann F, Keating SM. SBML Test Suite release 3.3.0. Zenodo; 2017.

58. Evans TW, Gillespie CS, Wilkinson DJ. The SBML discrete stochastic models test suite. Bioinformatics. 2008; 24: 285–286. https://doi.org/10.1093/bioinformatics/btm566 PMID: 18025005

59. Zi Z, Klipp E. SBML-PET: a Systems Biology Markup Language-based parameter estimation tool. Bioinformatics. 2006; 22: 2704–2705. https://doi.org/10.1093/bioinformatics/btl443 PMID: 16926221

60. Cannistra C, Medley K, Sauro HM. SimpleSBML: A Python package for creating and editing SBML models. Systems Biology; 2015 Oct. https://doi.org/10.1101/030312

61. Funahashi A, Matsuoka Y, Jouraku A, Morohashi M, Kikuchi N, Kitano H. CellDesigner 3.5: A Versatile Modeling Tool for Biochemical Networks. Proceedings of the IEEE. 2008; 96: 1254–1265. https://doi.org/10.1109/JPROC.2008.925458

62. Abel JH, Drawert B, Hellander A, Petzold LR. GillesPy: A Python Package for Stochastic Model Building and Simulation. IEEE Life Sciences Letters. 2016; 2: 35–38. https://doi.org/10.1109/LLS.2017.2652448 PMID: 28630888

63. Programming biological models in Python using PySB. Molecular Systems Biology. 2013; 9: 646. https://doi.org/10.1038/msb.2013.1 PMID: 23423320

64. Hunt J. Advanced Guide to Python 3 Programming. Springer International Publishing; 2019.

65. Starruß J, de Back W, Brusch L, Deutsch A. Morpheus: a user-friendly modeling environment for multiscale and multicellular systems biology. Bioinformatics. 2014; 30: 1331–1332. https://doi.org/10.1093/bioinformatics/btt772 PMID: 24443380