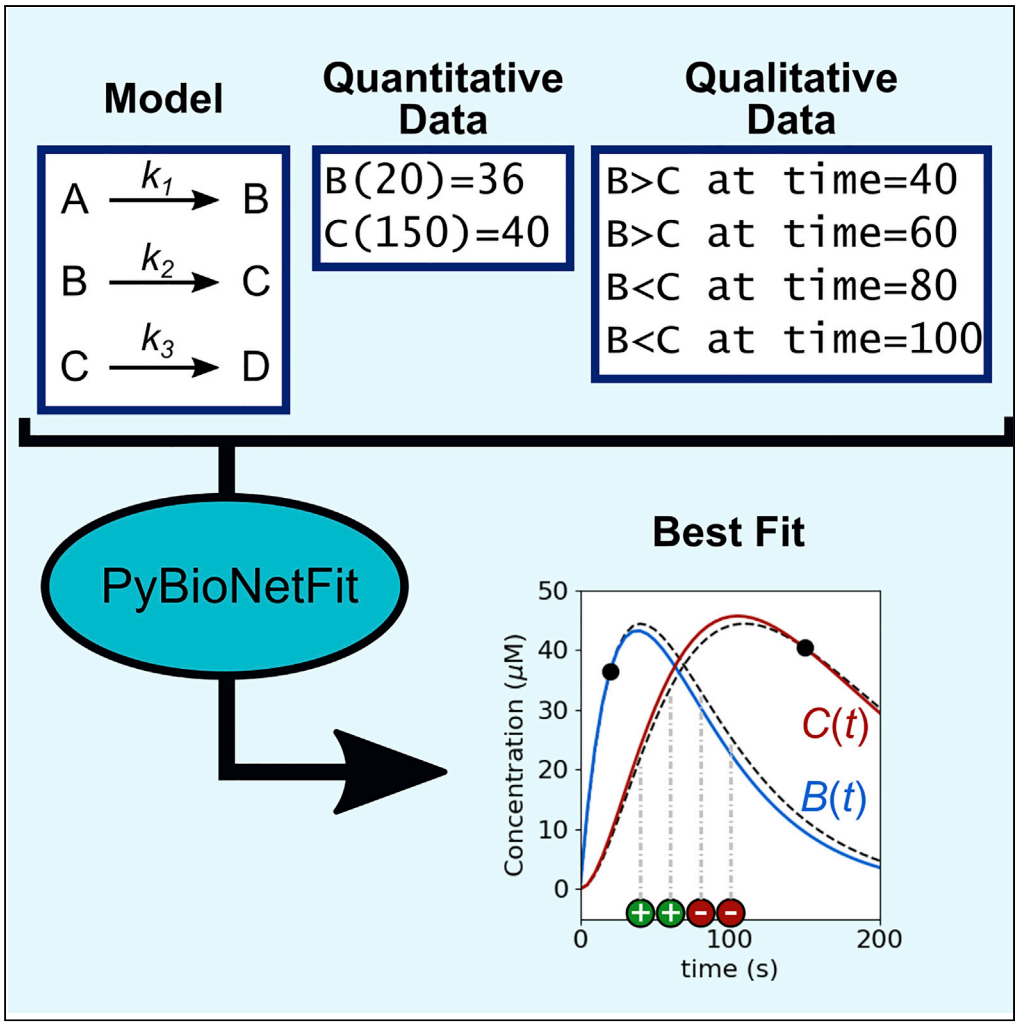
iScience

CellPress

Article

PyBioNetFit and the Biological Property Specification Language



Eshan D. Mitra, Ryan Suderman, Joshua Colvin, ..., Herbert M. Sauro, Richard G. Posner, William S. Hlavacek

wish@lanl.gov

HIGHLIGHTS

PyBioNetFit is a software tool for parameterizing systems biology models

PyBioNetFit has support for uncertainty quantification, model checking, and design

BPSL enables formulation of qualitative system properties to use in fitting

Example problems are demonstrated on single workstations and on computer clusters

Mitra et al., iScience 19, 1012–1036
September 27, 2019 © 2019 The Author(s).
https://doi.org/10.1016/j.isci.2019.08.045

iScience

CellPress

Article

PyBioNetFit and the Biological Property Specification Language

Eshan D. Mitra,[1] Ryan Suderman,[1,4] Joshua Colvin,[2] Alexander Ionkov,[1,5] Andrew Hu,[3] Herbert M. Sauro,[3] Richard G. Posner,[2] and William S. Hlavacek[1,6,*]

SUMMARY

In systems biology modeling, important steps include model parameterization, uncertainty quantification, and evaluation of agreement with experimental observations. To help modelers perform these steps, we developed the software PyBioNetFit, which in addition supports checking models against known system properties and solving design problems. PyBioNetFit introduces Biological Property Specification Language (BPSL) for the formal declaration of system properties. BPSL allows qualitative data to be used alone or in combination with quantitative data. PyBioNetFit performs parameterization with parallelized metaheuristic optimization algorithms that work directly with existing model definition standards: BioNetGen Language (BNGL) and Systems Biology Markup Language (SBML). We demonstrate PyBioNetFit's capabilities by solving various example problems, including the challenging problem of parameterizing a 153-parameter model of cell cycle control in yeast based on both quantitative and qualitative data. We demonstrate the model checking and design applications of PyBioNetFit and BPSL by analyzing a model of targeted drug interventions in autophagy signaling.

INTRODUCTION

An important step in the development of a mathematical model for a biological system is using experimental data to identify model parameters. In a conventional approach, the experimental data of most utility are quantitative time courses and/or dose-response curves. Parameters are adjusted to minimize the difference between the model outputs and the experimental data (as measured, for example, by a residual sum-of-squares function).

In some cases, there are straightforward solutions for parameter identification. For example, software tools such as Data2Dynamics (Raue et al., 2015) and COPASI (Hoops et al., 2006) implement practical parameterization methods for biological applications. These programs can, for example, use gradient-based optimization to solve the benchmark problems of Raue et al. (2013) and Hass et al. (2019). These problems feature ODE models, which typically consist of tens of equations. One contains 500 equations. As powerful and practical as Data2Dynamics and COPASI are, not all biological models fall into a category that can be solved with these tools. When current software tools are inadequate, modelers must resort to either problem-specific code or manual adjustment of parameters. Both these approaches are tedious from the perspective of the modeler and also present challenges for reproducibility of the modeling work (Medley et al., 2016; Waltemath and Wolkenhauer, 2016). Therefore, there is strong motivation to expand the scope of problems that can be solved using general-purpose software compatible with standard model definition formats.

We developed the software PyBioNetFit to solve three major classes of parameterization problems for which current software solutions are limited. (1) Problems with larger than usual numbers of ODEs. The size of an ODE-fitting problem depends primarily on two considerations: the number of differential equations and the number of free parameters. Parameterization cost typically has a dependence on both these quantities, but the relative importance depends on the method used for parameterization. Large problem size in terms of equation count often arises when using rule-based modeling. Rule-based modeling is the preferred approach for processes in which a combinatorial explosion in the number of possible chemical species makes it challenging to enumerate every possible chemical reaction (Chylek et al., 2013; Faeder et al., 2005). In a rule-based model, a concise set of rules can be expanded to generate a much larger system of ODEs (hundreds to thousands of equations from a model with tens of rules). Although the number of equations grows large, the number of parameters remains proportional to the number of rules, which is typically much smaller than the number of rule-implied reactions. In this way, rule-derived ODE models

[1]Theoretical Biology and Biophysics Group, Theoretical Division, Los Alamos National Laboratory, Los Alamos, NM, USA

[2]Department of Biological Sciences, Northern Arizona University, Flagstaff, AZ, USA

[3]Department of Bioengineering, University of Washington, Seattle, WA, USA

[4]Present address: Immunetrics, Pittsburgh, PA, USA

[5]Present address: University of Wisconsin–Madison, Madison, WI, USA

[6]Lead Contact

*Correspondence: wish@lanl.gov

https://doi.org/10.1016/j.isci.2019.08.045

 © 2019 The Author(s).
This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

differ from manually formulated ODE models, which typically have a parameter count proportional to the number of reactions. For ODE systems at the scale typically found in rule-based modeling, gradient-based methods using finite differences or forward sensitivity analysis are computationally expensive. Adjoint sensitivity analysis enables more scalable gradient computation (Cao et al., 2002), but current software is limited in the supported workflows. (2) Problems featuring models that are simulated stochastically. This class of problems includes rule-based models in which the implied ODE system is so large that it cannot be derived from rules or numerically integrated efficiently (Sneddon et al., 2011; Suderman et al., 2019). In such cases, the objective function is not differentiable, so standard gradient-based methods cannot be used. (3) Problems including unconventional experimental data, in particular non-numerical *qualitative data*. Such datasets are often collected by experimentalists and have the potential to inform model parameterization (Mitra et al., 2018), but currently are rarely used in practice. Notable exceptions are works of Tyson and coworkers (Chen et al., 2000, 2004; Csikász-Nagy et al., 2006; Kraikivski et al., 2015; Oguz et al., 2013) and Pargett and coworkers (Pargett and Umulis, 2013; Pargett et al., 2014).

We address problems (1) and (2) by using parallelized metaheuristic optimization algorithms in place of gradient-based algorithms. Metaheuristics are a well-established class of optimization algorithms that do not rely on gradient information. If gradient information is available, metaheuristics can benefit by working in combination with gradient-based methods (Villaverde et al., 2019), as in memetic algorithms (Neri et al., 2012). Metaheuristics carry no guarantee for convergence to a global optimum but are found to be effective in many use cases (Gandomi et al., 2013). Examples of metaheuristics include differential evolution (Storn and Price, 1997), particle swarm optimization (Eberhart and Kennedy, 1995), and scatter search (Glover et al., 2000). Such algorithms often include some type of iterative randomized selection of candidate parameter sets, followed by evaluation of the selected parameter sets, which is used to direct the selection of parameters in future iterations to more favorable regions of parameter space. Many modern descriptions of metaheuristics allow for parallelized evaluation of parameter sets (Moraes et al., 2015; Penas et al., 2015, 2017), which is valuable when each model simulation is computationally expensive. Although these algorithms are well-established, software designed for biological applications is limited. COPASI (Hoops et al., 2006) and Data2Dynamics (Raue et al., 2015) both include metaheuristic algorithms, but these algorithms are not parallelized, which limits their performance with computationally intensive models. The software BioNetFit (Thomas et al., 2016) (called BioNetFit 1 in this report to distinguish it from the newly developed software) was an early effort to use a parallelized evolutionary algorithm to parameterize rule-based biological models. However, the BioNetFit 1 algorithm is inefficient in many cases, and in general, optimization algorithm performance is problem dependent, and so a toolbox of methods is needed to enable a wide range of problems to be solved efficiently. PyBioNetFit was inspired by BioNetFit 1 but is an entirely new code base that includes multiple, robust metaheuristic algorithms.

We address Problem 3 by following the approach of Mitra et al. (2018) for parameterizing models using both qualitative and quantitative data. In this approach, properties of interest are represented as one or more inequality constraints on the outputs of a model, enforced during some portion of a simulation. In some cases, a single qualitative observation, such as the viability of a particular mutant, implies several system properties (inequalities). For example, in a model of the yeast cell cycle (Laomettachit et al., 2016), if a yeast strain is viable, three variables representing bud formation, origin activation, and spindle assembly must each exceed a specified threshold. After defining inequalities, we cast each inequality as a static penalty function (Smith and Coit, 1997), added to the objective function to be minimized. The result is a scalar-valued objective function with contributions from both qualitative and quantitative data; this function is minimized during fitting. This approach is fairly straightforward, and it has been demonstrated to be effective for parameterization of biological models using qualitative data (Mitra et al., 2018). An important feature (in contrast to other constrained optimization methods) is allowance for the possibility that some of the inequality constraints may not be satisfied (because they arise from uncertain experimental data).

To extend this approach for use in general-purpose software, we require a language to express arbitrary system properties of interest. In systems biology, there is no established means for formalizing system properties, although attempts have been made to do so with temporal logic (Clarke et al., 2008; David et al., 2012; Heath et al., 2008; Kwiatkowska et al., 2008), sometimes as part of model parameterization (Hussain et al., 2015; Khalid and Jha, 2018; Liu and Faeder, 2016). There is a lack of software tools tailored for biological modeling that support property specification languages—most studies that incorporate temporal logic do so with problem-specific code. In addition, there are few demonstrations of how the formalism

of temporal logic, originally developed for computer science applications (Clarke et al., 1986), can be applied to describe biologically interesting properties such as case-control comparisons. To address these deficiencies, we developed the Biological Property Specification Language (BPSL) as part of PyBioNetFit. BPSL is a domain-specific language for declaration of biological system properties and allows such properties to be used as part of parameterization.

To complement its parameterization features, PyBioNetFit includes methods for uncertainty quantification of parameter estimates. Bayesian uncertainty quantification can be performed using Markov chain Monte Carlo (MCMC) with the Metropolis-Hastings (MH) algorithm (reviewed by Chib and Greenberg, 1995) or parallel tempering (reviewed by Earl and Deem, 2005). These methods start with an assumed prior probability distribution for each parameter, and a likelihood function, and aim to sample the multidimensional posterior probability distribution of the parameters given the data. Simulations can be performed using sampled parameter sets to quantify the uncertainty of model predictions. PyBioNetFit also supports bootstrapping, which performs uncertainty quantification by resampling data (Efron and Tibshirani, 1993; Press et al., 2007).

Although PyBioNetFit and BPSL were designed primarily for model parameterization, BPSL also enables formalized approaches to model checking, somewhat as in computer science (Clarke et al., 1999), and design, somewhat as in optimal control. For our application, we define model checking as performing verification of whether a model reproduces a set of specified properties. Applications of formal model checking to biological processes have been considered in earlier work, including for stochastic models (Clarke et al., 2008; Heath et al., 2008; Kwiatkowska et al., 2008). Much more often, model checking in biology is done informally as part of building a model. However, as models become more detailed, with an increasing number of known properties, a more formal and systematic system of model checking is useful: it can help in communicating what knowledge went into building the model and for comparing the predictions of different models. Design represents a related application, analogous to the classical use of constrained optimization techniques. In a design problem in PyBioNetFit, we seek an intervention (a perturbation of a parameterized model) that brings about a desired set of BPSL-defined system behaviors, for example, choosing drug doses to up- or down-regulate the activity of a target pathway.

All the above-mentioned features of PyBioNetFit are designed to be used in conjunction with existing model definition standards, avoiding the need for problem-specific code. PyBioNetFit natively supports models defined in BioNetGen Language (BNGL) (Faeder et al., 2009), a language for rule-based models, and core SBML (Hucka et al., 2003), a language for more conventional models. For BNGL models, PyBioNetFit supports the simulators available in BioNetGen (Harris et al., 2016; Faeder et al., 2009; Blinov et al., 2004; Sneddon et al., 2011). For SBML models, PyBioNetFit uses the simulator libRoadRunner (Somogyi et al., 2015). PyBioNetFit has a modular design that makes it possible to add support for additional model standards and simulators in the future. Currently, other model standards are indirectly supported by converting to BNGL or SBML. For example, rule-based models defined in the Kappa language (Danos and Laneve, 2004; Sorokina et al., 2013) can be converted to BNGL using the software tool TRuML (Suderman and Hlavacek, 2017).

To demonstrate the capabilities of PyBioNetFit, we solved a series of example optimization problems. We solved a total of 31 problems, 25 of which featured published, biologically relevant models (Blinov et al., 2006; Boehm et al., 2014; Brännmark et al., 2010; Chylek et al., 2014; Dunster et al., 2014; Erickson et al., 2019; Faeder et al., 2003; Fey et al., 2015; Harmon et al., 2017; Hlavacek et al., 2018; Kocieniewski et al., 2012; Kozer et al., 2013; Kühn and Hillmann, 2016; Lee et al., 2003; Mitra et al., 2018; Monine et al., 2010; Mukhopadhyay et al., 2013; Oguz et al., 2013; Romano et al., 2014; Shirin et al., 2019; Suderman and Deeds, 2013; Webb et al., 2011; Zheng et al., 2012). With four of these problems, we performed extensive benchmarking using different algorithms and different levels of parallelization. Not surprisingly, we find that the optimal algorithm depends on the fitting problem, which demonstrates the value of having a toolbox of several algorithms available. We then focus on a particularly challenging example problem: parameterizing the model of Tyson and co-workers for cell cycle control in yeast (Chen et al., 2000, 2004; Csikász-Nagy et al., 2006; Kraikivski et al., 2015; Oguz et al., 2013). This model was originally parameterized by hand-tuning (Chen et al., 2000, 2004) and later by automated optimization with problem-specific code (Mitra et al., 2018; Oguz et al., 2013). Here we consider our most recent description of the problem (Mitra et al., 2018), which has a 153-dimensional parameter space. We define the problem

using BPSL and solve it using the general-purpose functionality of PyBioNetFit. Thus we demonstrate that PyBioNetFit can solve this general class of problem, that of using both qualitative and quantitative data to parameterize a biological model.

Finally, we considered a model describing drug intervention in autophagy signaling (Shirin et al., 2019) to demonstrate the capabilities of PyBioNetFit and BPSL beyond model parameterization. We show that BPSL can be used to define a set of system properties, which can then be used in model checking. We also demonstrate how BPSL can be used to configure a design problem, finding a combination of drug doses to achieve a desired level of autophagy regulation.

RESULTS

Workflow Enabled by PyBioNetFit

The steps involved in using PyBioNetFit are illustrated in Figure 1. PyBioNetFit is configured with a set of plain-text input files (Figure 1A). The input files must have particular filename extensions: .conf, .bngl, .xml, .exp, and .prop. We will refer to these as CONF files, BNGL files, etc. The files may be prepared in any standard text editor.

Figure 2 shows an example set of PyBioNetFit input files for a simple problem. The problem is to parameterize a model for the chemical kinetics of three reactions (Figure 2A) using synthetic quantitative and qualitative data (Figure 2B).

A model file (Figure 2C; filename extension .xml for SBML models or .bngl for BNGL models) defines the model to be fit. In the case of BNGL models, the model file also defines simulation protocols. BNGL allows for sophisticated simulation protocols such as equilibration to a basal steady state. For SBML models, simulation protocols must be defined in the CONF file (see below). BNGL files may be prepared with a standard text editor, or with the text editor available within RuleBender (Xu et al., 2011), an integrated development environment for BioNetGen. SBML files are not human readable and should be prepared using SBML-compatible software such as COPASI (Hoops et al., 2006) or Tellurium (Medley et al., 2018; Choi et al., 2018). Model files must conform to certain conventions for compatibility with PyBioNetFit. For BNGL files, each free parameter to be fit must be assigned a name ending in `__FREE` (Figure 2C lines 3–5) and each model output to be compared with measurements must be introduced as a BNGL observable or function. For SBML files, each free parameter must be an SBML parameter or the initial concentration of a species, and each model output to be compared with measurements must be an SBML species concentration or population. In addition, each simulation command must have an associated string identifier, called a *suffix*. If the simulation is defined in a BNGL file, the suffix is specified using the `suffix` argument of the `simulate` or `param_scan` action. If the simulation is defined in the CONF file (see description of the CONF file below), the suffix is specified as part of the `time_course` or `parameter_scan` declaration. The suffix must match the name of the corresponding experimental data file (e.g., "d1" in Figure 2). In the simplest case, a fitting job has one model file. However, PyBioNetFit supports jobs with multiple model files, such as, the problem considered in the section Application: Fitting a Model of Yeast Cell Cycle Control Using Both Qualitative and Quantitative Data. This feature is useful when two or more models have parameters in common, such as two models that represent the same process in wild-type and mutant cells.

Experimental measurements are supplied in EXP (Figure 2D) and PROP (Figure 2E) files. EXP files contain tabular quantitative data, such as time courses or dose-response curves. These files are specified in a space-delimited format in which the first column corresponds to the independent variable and other columns correspond to dependent variables (the same format as is used in GDAT files output by BioNetGen). PROP files contain statements written using BPSL, which is described in the next section. PROP files are used for qualitative data.

A configuration file, or CONF file (Figure 2F), provides the settings for running a fitting job. These settings include which model and data files to use (line 3), which parameters will be free to vary in fitting (lines 15–17), which fitting algorithm to use (line 9), which objective function to use (line 10), and settings specific to the selected fitting algorithm (lines 11–12), such as the mutation rate in the case of differential evolution. Additional configuration keys are available to define simulation protocols (time courses or parameter scans), and to declare free parameters that vary in logarithmic rather than linear space. A complete listing of the available configuration keys is provided in the PyBioNetFit user manual (Mitra and Suderman, 2019).

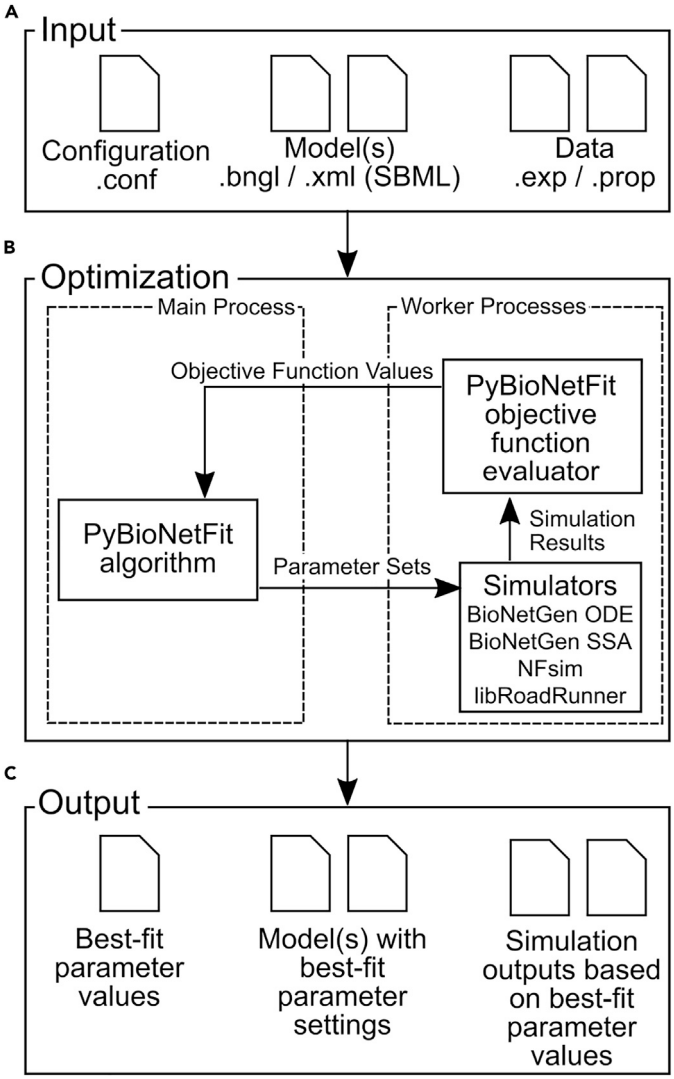


Figure 1. Inputs, Outputs, and Operations of PyBioNetFit

(A) PyBioNetFit input files are a set of plain-text files: a CONF file specifying program settings, one or more model files in BNGL and/or SBML format, and one or more data files containing experimental data. EXP files contain quantitative data, and PROP files contain qualitative data. Examples of these files are shown in Figure 2.

(B) When running PyBioNetFit, the user-selected optimization algorithm generates candidate parameter sets, which are passed to the appropriate simulator (for SBML models, libRoadRunner; for BNGL models, the simulator selected in the BNGL file). PyBioNetFit calculates the value of a user-selected objective function from the simulation results obtained for each trial parameter set, which is then used to inform future iterations of the algorithm. Each simulation and objective function evaluation is started as a separate worker process, which is run on a separate core of a multicore workstation or cluster if available.

(C) PyBioNetFit output files include a text file reporting the best-fit parameter values, model files with the best-fit parameter settings, and output files resulting from simulating the models using the best-fit parameter values.

After generating all the required files, a user can run PyBioNetFit from the command line, as described in the user manual (Mitra and Suderman, 2019). Figure 1B illustrates the internal operations of PyBioNetFit. PyBioNetFit iteratively passes proposed parameter sets to the appropriate simulator, reads the simulation results, and calculates the value of the user-selected objective function. The objective function values are fed back into the optimization algorithm and affect which parameter sets are proposed in future iterations. Upon termination of the algorithm, PyBioNetFit outputs the best-fit parameter values, new model files that include those parameter settings, and (optionally) simulation results generated from those model files (Figure 1C).

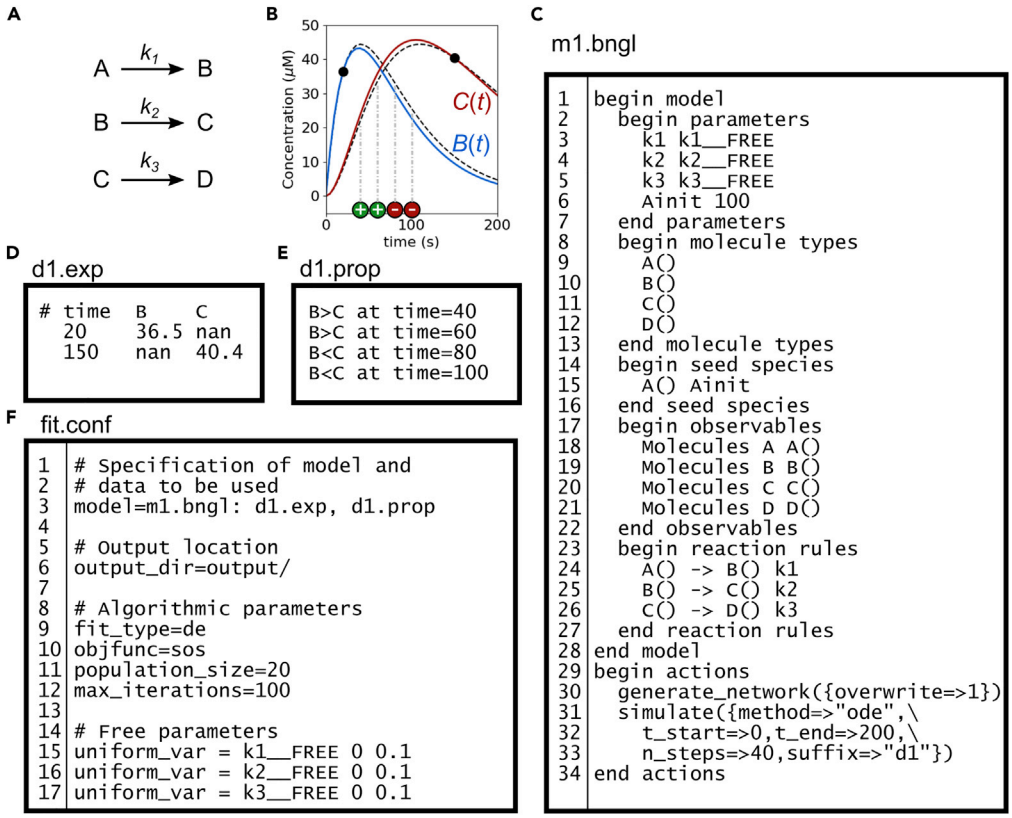


Figure 2. A Fitting Problem Configured to Run in PyBioNetFit

(A–C) (A) Reaction scheme of the model to be parameterized. The model is a coupled system of ODEs for the mass-action kinetics of the reactions shown here. The problem is to estimate values for the rate constants k_1, k_2, and k_3. (B) Time courses of concentrations of species B and C. Black broken curves give the ground truth. For fitting, two quantitative data points (black points) and four qualitative data points (colored circles) are available. The qualitative data indicate whether the concentration of species B or C is larger at a particular time. A plus sign indicates $B > C$ and a minus sign indicates $B < C$. Colored curves show the quality of fit after running PyBioNetFit on the input files listed in (C–F). (C) Implementation in BNGL of a model for the reaction scheme in (A). Note that the parameters to be tuned by PyBioNetFit are named with the suffix `__FREE` (lines 3–5). The parameter `Ainit`, which is to be held fixed, is not named with `__FREE` (line 6).

(D) EXP file containing the quantitative data points shown in (B). The keyword `nan` is used to indicate missing data.

(E) PROP file encoding the qualitative data points shown in (B).

(F) CONF file used to configure PyBioNetFit. As described in the main text, the CONF file specifies the paths to the other files, the algorithm to be used, and the free parameters to be adjusted. The files pictured here are available in Data S1 (Problem 5).

Property Specification with BPSL

To allow fitting to qualitative data, we implemented the approach described by Mitra et al. (2018) in PyBioNetFit. For this feature, we developed BPSL, a novel property specification language. BPSL is designed for writing system properties of cellular regulatory networks. In BPSL, system properties are expressed as inequalities involving the dependent variables of an experiment or model. We refer to such dependent variables in this section as simply "variables." Typically, BPSL statements are written for the purpose of parameterizing a particular model, in which case the names of the variables should match the names of outputs of that model, similar to column headings of an EXP file. However, we note that like EXP files, BPSL statements primarily encode (experimental) data, and the same data could be considered in conjunction with any model for the system of interest (possibly only after changing the variable names to match the output names of the new model). Variables in BPSL are flexible: in addition to what is supported in EXP files—quantities corresponding to BNGL or SBML model outputs—it is possible to compare variables/readouts between different models/systems. One application of this feature would be case-control comparisons, such as comparing a mutant to wild-type.

Keyword	Meaning
`always`	At all times
`once`	At one or more time points
`at` ⟨*condition*⟩	At the first time point where ⟨*condition*⟩ is true
`between` ⟨*condition1*⟩, ⟨*condition2*⟩	Over the range of time points starting with the first point where ⟨*condition1*⟩ is true and ending with the first subsequent time point where ⟨*condition2*⟩ is true

Table 1. Keywords Used to Define Enforcement Conditions in BPSL

Definitions assume that the independent variable is time, but any arbitrary independent variable may be considered, as when considering a steady-state dose-response curve instead of a time course.

Each inequality declared in BPSL is enforced at a particular value or range of values of the independent experimental variable (e.g., time). For example, an inequality might be enforced at one specific time, or at all times in a time course. As described below, BPSL syntax provides a means to define inequalities, where they are enforced, and how much they contribute to the objective function during optimization.

A BPSL statement consists of three parts: an inequality, followed by an enforcement condition, followed by a weight. The inequality establishes a relationship ($<$, $>$, $\leq$, or $\geq$) between a variable and a constant or between two variables. The enforcement condition specifies where in a time course or dose-response curve the constraint is in effect. Enforcement conditions are defined using the keywords `always`, `once`, `at`, and `between`, as summarized in Table 1. The weight (declared with the `weight` keyword) specifies the static penalty coefficient to be used during optimization when the inequality is not satisfied. Specifically, if the constraint $g(\hat{\mathbf{y}})<0$ is not satisfied for the model outputs $\hat{\mathbf{y}}$, the objective function adds a penalty equal to $C \cdot g(\hat{\mathbf{y}})$ where C is the weight of the constraint. Note that, in this formulation, the penalty decreases as we move closer to satisfying the constraint. This feature of the objective function serves to guide an optimization algorithm toward constraint satisfaction.

We illustrate the use of BPSL with the following examples, assuming time course outputs $X(t)$ and $Y(t)$. The BPSL statement

```
X > 5 always weight 2
```

defines a constraint requiring $X(t)$ to be greater than 5 at all times. If the constraint is violated, a penalty of $2 \cdot (5 - \min(X(t)))$ is added to the objective function. The BPSL statement

```
X < 1 between time = 8, Y = 5 weight 3
```

defines a constraint requiring $X(t)$ to be less than 1 over a specified time range. The start point of this time range is specified directly: `time = 8`. The endpoint is specified indirectly based on the value of $Y(t)$; it is the first time point after $t = 8$ where $Y(t) = 5$. More precisely, to avoid numerical error, PyBioNetFit checks when $Y(t)$ *crosses* 5, i.e., finds, after $t = 8$, the first two consecutive output times t_1 and t_2 such that $Y(t_1)<5 \leq Y(t_2)$ or $Y(t_1)>5 \geq Y(t_2)$ and sets t_2 as the endpoint. If the constraint is violated at any point in the above time range, the penalty is $3 \cdot (\max(X(t)) - 1)$, where $\max(X(t))$ is evaluated over the time range.

Metaheuristic Fitting Algorithms

PyBioNetFit features four recommended parallelized metaheuristic fitting algorithms, which we will refer to as differential evolution (DE), asynchronous differential evolution (aDE), particle swarm optimization (PSO), and scatter search (SS). The details of each algorithm's implementation and configuration options are provided in the PyBioNetFit user manual (Mitra and Suderman, 2019). Note that aDE and PSO are implemented as asynchronous algorithms, which address load-balancing issues by submitting a new simulation job whenever one is completed. Such an implementation prevents CPU cores from remaining idle, but requires new trial parameter sets to be proposed with limited new information. In contrast, our synchronous DE and SS algorithms require all simulations performed within an iteration to complete before moving on to the next iteration.

#	Key Model component(s)	Data	Sim.	Pars.	Rxns.	Eqs.	Pts.	Sims.	Algs.
1	Histones	E	RR	46	60	30	48	1	D,A,P,S
2	EGFR, Grb2, Sos	S	B-ode	37	3,749	356	40	1	D,A,P,S
3	IgE receptor	S	B-ssa	20	58,276	3,744	66	3	D,A,P,S
4	EGFR	E	B-nf	9	–	–	24	12	D,A,P,S
5	Simple reactions	S	B-ode	3	3	3	6*	1	D,A,P,S
6	Degranulation	E	B-ode	16	86	23	6	11	D,A,S
7	Egg-shaped curve	S	B-ode	10	1	2	362	1	D,A,P,S
8	Yeast cell cycle regulators	E	RR	153	39	26	2352*	122	S
9	mTORC, ULK1, AMPK	D	RR	6	5	5	2*	1	S
10	EGFR	E	B-ode	9	11,918	923	24	6	D,A,P,S
11	Trivalent ligand	E	B-nf	3	–	–	12	36	D,A,P,S
12	TCR	E	B-nf	34	–	–	68	1	D,A,P,S
13	Ligand/receptor	S	B-ode	6	54	15	26	1	D,A,P,S
14	Ligand/receptor	S	B-nf	6	–	–	26	2	D,A,P,S
15	IGF1R	E	B-ode	7	96	27	38	38	D,A,P,S
16	Raf, MST, ERK	S	RR	63	31	21	60	1	D,A,P,S
17	EGFR, Grb2, Sos	S	B-ssa	37	3,749	356	40	3	D,A
18	MAPK	S	B-ode	13	487	85	28	7	D,A,P,S
19	Raf inhibitor	S	B-ode	2	12	6	28*	13	D,A,P,S
20	Raf inhibitor	S	B-ode	4	12	6	28*	13	D,A,P,S
21	Immune cells	E	RR	7	10	7	21	1	D,A,P,S
22	STAT	E	RR	6	9	8	48	1	D,A,S
23	Insulin receptor	E	RR	22	11	9	43	9	D,S
24	Jnk	E	B-ode	12	330	66	59	22	D,A,P,S
25	Cells expressing Fas or FasL	S	RR	11	17	7	64	16	D,A,P,S
26	TCR	E	B-nf	10	–	–	9	450	P,S
27	Wnt, Axin, APC	S	RR	25	17	15	68	1	D,A,P
28	MAPK	E	B-nf	25	–	–	96	2	A

Table 2. Summary of the 31 Example Problems Provided in Data S1

(Continued on next page)

To demonstrate the breadth of problems that can be solved using PyBioNetFit, we ran these algorithms on a total of 31 example problems, listed in Table 2. The problems are described in the following references: 1, Hass et al. (2019); Zheng et al. (2012); 2, Blinov et al. (2006); Gupta and Mendes (2018); 3, Faeder et al. (2003); Sneddon et al. (2011); Gupta and Mendes (2018); 4, Kozer et al. (2013); Thomas et al. (2016); 5, none; 6, Harmon et al. (2017); 7, Hlavacek et al. (2018); 8, Laomettachit (2011); Oguz et al. (2013); Mitra et al. (2018); 9, Shirin et al. (2019); 10, Kozer et al. (2013); Thomas et al. (2016); 11, Monine et al. (2010); Posner et al. (2007); Thomas et al. (2016); 12, Chylek et al. (2014); Thomas et al. (2016); 13, Thomas et al. (2016); 14, Thomas et al. (2016); 15, Erickson et al. (2019); Kiselyov et al. (2009); 16, Romano et al. (2014); 17, Blinov et al. (2006); Gupta and Mendes (2018); 18, Kocieniewski et al. (2012); 19, Mitra et al. (2018); 20, Mitra et al. (2018); 21, Dunster et al. (2014); Xue and Del Bigio (2000); 22, Boehm et al. (2014); Hass et al. (2019); 23, Brännmark et al. (2010);

#	Key Model component(s)	Data	Sim.	Pars.	Rxns.	Eqs.	Pts.	Sims.	Algs.
29	Schwefel function	S	RR	2	0	1	1	1	D,P,S
30	Job market	S	B-nf	6	–	–	330	3	D,A,P,S
31	Elephant-shaped curve	S	B-ode	82	1	2	930	1	D,A,P,S

Table 2. Continued

Table columns are summarized as follows. "Key model component(s)" lists some components of the model (but is not intended as a complete description of the model). "Data" gives the type of data used in fitting: E, experimental; S, synthetic; D, specification of desired system properties for a design problem. "Sim." gives the simulator used: RR, libRoadRunner, B-ode, BioNetGen ODE; B-ssa, BioNetGen SSA; B-nf, NFsim. Note that models using libRoadRunner are implemented in SBML and models using the other three simulators are implemented in BNGL. "Pars." gives the number of free parameters. "Rxns." gives the number of chemical reactions in the model. "Eqs." gives the number of differential equations in the model. Reaction and equation counts are not given for models simulated with NFsim because the simulation is run without enumerating all reactions and equations. "Pts." gives the number of data points. When indicated (*), this total includes qualitative data points (i.e., inequality constraints). "Sims." gives the number of individual time course simulations required for one evaluation of the objective function. "Algs." lists the algorithms used to solve the problem: D, DE; A, aDE; P, PSO; S, SS.

Hass et al. (2019); 24, Fey et al. (2015); 25, Webb et al. (2011); 26, Mukhopadhyay et al. (2013); Manz et al. (2011); 27, Lee et al. (2003); 28, Suderman and Deeds (2013); Yi et al. (2003); Yu et al. (2008); Leeuw et al. (1998); 29, none; 30, Kühn and Hillmann (2016); 31, Hlavacek et al. (2018). See also Table S1. Input files, descriptions, and results for each of these problems are provided in Data S1, a ZIP archive containing 31 numbered folders, one for each example problem. We will refer to the problems by these numbers. For example, we will refer to the folder associated with Problem 1 in Table 2 as Data S1 (Problem 1). In some cases, we fit models to published experimental data. In other cases where no appropriate experimental dataset was available, we generated synthetic data by simulating the model with an assumed ground truth parameter set. The synthetic data included noise; depending on the problem, this was added as Gaussian white noise, uniformly distributed noise, or noise inherent to performance of a single stochastic simulation. In total, the example problems included 19 rule-based models defined in BNGL, nine of which were fit to experimental data; nine manually formulated ODE models defined in SBML, 6 of which were fit to experimental data; and 3 problems using closed-form functions. All the problems could be solved with an acceptable fit (defined as reaching a target objective function value, which is specified in Data S1 for each individual problem) with at least one of the available algorithms using the default algorithmic parameters. Most could be solved with all four algorithms tested, albeit with different efficiencies. We do not perform a comprehensive analysis of every model (which would entail varying algorithmic parameters, performing additional replicates of fitting, etc.), but with the fitting runs we performed, we illustrate that PyBioNetFit can be used to analyze a variety of SBML- and BNGL-formatted models.

To demonstrate additional specific features of PyBioNetFit, which were not feasible or applicable to run on all example problems, we selected specific problems from Table 2 to use for illustration, as indicated in Table 3. We will describe results for these illustrative problems in the sections that follow.

To evaluate which algorithms are most effective in typical use cases, we performed timed benchmarking. We used the default algorithmic parameters for each algorithm. Because of the stochastic nature of the algorithms, many replicates of the same fitting job were necessary to make conclusions about the typical run time of each algorithm. As benchmark problems, we chose Problems 1–4 in Table 2, which have fitting run times on the order of hours. Such problems are not trivial, but it is still feasible to run many fitting replicates on a cluster.

To examine the full scope of PyBioNetFit functionality, our selected benchmarks include one problem using each of the four key simulators supported in PyBioNetFit, which we refer to as libRoadRunner, BioNetGen ODE, BioNetGen SSA, and NFsim. These simulators are described briefly as follows. (1) libRoadRunner is an SBML simulator. By default, libRoadRunner interfaces with CVODE (Hindmarsh et al., 2005) to perform numerical integration. (2) BioNetGen ODE refers to the numerical integration capability of BioNetGen, accessed with the action `simulate(method=>"ode")`. Like libRoadRunner, this functionality interfaces with CVODE (Hindmarsh et al., 2005). (3) BioNetGen SSA refers to an efficient

Problem	Feature
1–4	Timed benchmarking of performance
5	Demonstration of configuration
6	Bayesian uncertainty quantification
7	Bootstrapping
8	Real-world problem using qualitative data
9	Model checking and design

Table 3. Example Problems Selected for Demonstrations of Additional Features of PyBioNetFit, Presented Throughout the Results Section

variation of Gillespie's stochastic simulation algorithm (Gillespie, 2006) implemented in BioNetGen, accessed with the action `simulate(method=>"ssa")`. (4) NFsim refers to the component of BioNetGen accessed with the action `simulate(method=>"nf")` that performs agent-based stochastic simulations without generation of a reaction network (Sneddon et al., 2011). For the stochastic simulators BioNetGen SSA and NFsim, PyBioNetFit performs smoothing by averaging a user-specified number of replicate runs before comparing the results to experimental data.

For each of the benchmark problems, we chose a target objective function value (described for each problem in Data S1) and measured the run times required for each algorithm to reach the target value. We evaluated the run times of the four algorithms and also measured how the run times scaled with an increasing number of available cores on a cluster. As described in Transparent Methods, we adjusted the population size of each algorithm based on the core count, such that each iteration used all available cores. The resulting distributions of run times are shown in Figure 3. We found that in most cases, the algorithms show good capacity for taking advantage of parallelization, in that the median run time decreases as the number of available cores increases. The best algorithm varies by problem, and also varies by the number of cores available. Notably, with a large number of available cores (288), PSO (an asynchronous algorithm) is most effective for the benchmarks using stochastic simulators (BioNetGen SSA and NFsim) (Figures 3C and 3D). According to the Mann-Whitney U statistical test, for Problem 3, PSO is faster than aDE with $p = 3.3 \times 10^{-5}$ and faster than SS with $p = 7.7 \times 10^{-3}$. For Problem 4, PSO is faster than SS with $p = 1.4 \times 10^{-4}$. However, PSO is outperformed by aDE and SS for the other benchmarks (Figures 3A and 3B). For Problems 1 and 2, DE and aDE encountered convergence failures with small core counts because the corresponding population sizes were too small to effectively explore the parameter space.

Variability in algorithm performance is expected when considering a broad range of problems. In the end, the best algorithm and level of parallelization are problem specific, and must be selected through trial and error. PyBioNetFit helps users in this regard by providing robust implementations of several algorithms, allowing for easy testing of different approaches.

Two additional metaheuristic algorithms are implemented in PyBioNetFit but not rigorously benchmarked: simulated annealing (SA) and the parallelized island-based differential evolution (iDE) algorithm of Penas et al. (2015). These algorithms were challenging to include in benchmarking because of the need to tune problem-specific parameters (temperature and step size in the case of SA and trade-offs between island size and number of islands versus the number of available cores in the case of iDE). Still, we include the implementations in PyBioNetFit with the hope that users will find them useful for specific problems.

Local Optimization

PyBioNetFit includes a parallelized implementation (Lee and Wiswall, 2007) of the simplex algorithm (Nelder and Mead, 1965), a gradient-free local search algorithm. The simplex algorithm may be used on its own or to refine the best fit obtained from any of the other algorithms. For rugged parameter landscapes, which we expect to be common for problems considered in PyBioNetFit, a gradient-free local search algorithm is unlikely to find the global minimum on its own. Therefore, our recommended use of the simplex algorithm is for refinement of an existing best fit.

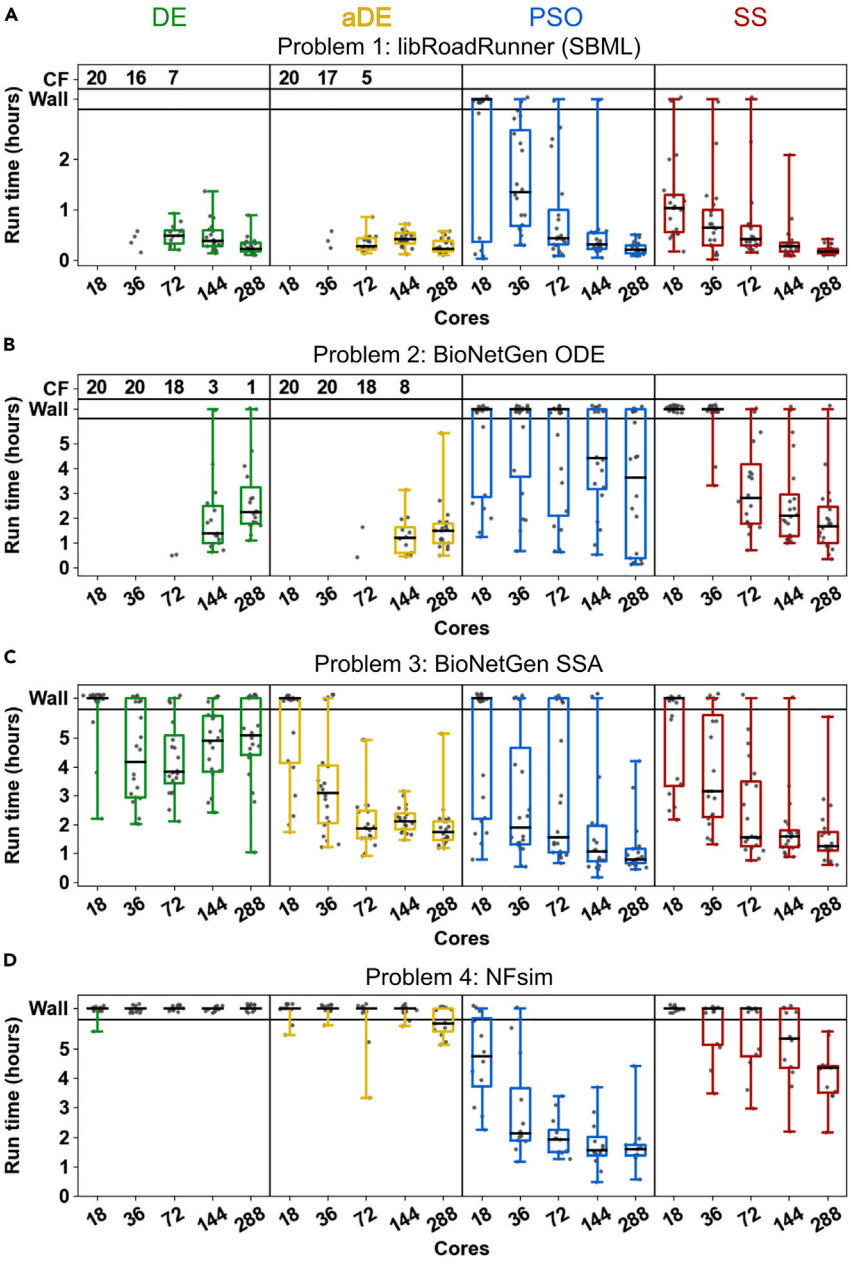


Figure 3. Results from Timed Benchmarking of PyBioNetFit

(A–D) Run times required to reach a target objective value are shown for our four selected benchmark problems (Table 2, Problems 1–4), for the DE, aDE, PSO, and SS algorithms implemented in PyBioNetFit. Box plots indicate the distribution of 20 replicates in (A–C) and 12 replicates in (D). Gray points represent results from individual replicates. Replicates that ran for the maximum wall time (3 h in A, 6 h in B–D) without reaching the target value are plotted in the "Wall" band. When calculating percentiles for box plots, "Wall" replicates were taken to be larger than any successful replicate. "CF" (convergence failure) gives the number of replicates, out of 20 total, that failed because the population converged to a single point that was worse than the target value. Box plot statistics do not include convergence failures. Box plots are not shown for settings in which more than half the replicates were convergence failures. Each pair of whiskers indicates the minimum and maximum. Each box indicates the quartiles. Each horizontal line indicates the median.

Comparison to a Gradient-Based Optimization Method

Although we did not rigorously benchmark PyBioNetFit against other parameterization tools, we tested the gradient-based method of Data2Dynamics (Raue et al., 2015) on Problems 1 and 2 (Table 2) to obtain a

rough view of how the performance of this tool compares to that of PyBioNetFit. Results are provided in Data S1 (Problems 1 and 2) and considered further in Discussion. We note that the results are provided only with the intention of concretely demonstrating discussion points.

Uncertainty Quantification

For Bayesian uncertainty quantification, PyBioNetFit offers two MCMC methods: the conventional MH algorithm and parallel tempering (PT). These methods are used by setting `fit_type = mh` or `fit_type = pt` in the CONF file, similar to how `de` is selected in Figure 2F, line 9. To validate the accuracy of PyBioNetFit, we used MH and PT with the model of mast cell signaling described by Harmon et al. (2017) (Table 2, Problem 6). Harmon et al. (2017) observed differences in mast cell degranulation as a function of the time delay between two pulses of antigen stimulation of IgE receptor activity. The model describes the activities of Syk and Ship1 during this two-stage stimulation protocol. The original study included Bayesian uncertainty quantification of model parameters and predictions using problem-specific code that implemented MH. We ran MH and PT in PyBioNetFit using input files provided in Data S1 (Problem 6). We found that both PyBioNetFit algorithms achieved good agreement with the published results for parameter uncertainty (Figures 4A–4F) and prediction uncertainty (Figures 4G–4L). For this problem, the MH and PT algorithms converged to the correct distribution at roughly the same rate. Convergence was checked by dividing the samples into two independent sets (sampled by different Markov chains) and confirming by inspection that the two sets of samples had similar distributions.

Note that to calculate the posterior probability distribution, the objective function is assumed to correspond to a negative log likelihood. This assumption is valid for the chi-square objective function, which was used in this example. Bayesian MCMC algorithms will not produce statistically meaningful results if used with PyBioNetFit's other available objective functions or when penalty terms arising from qualitative data are added to the objective function.

When using Bayesian MCMC methods, it is important to choose algorithmic parameters such that the posterior distribution is sampled accurately. In particular, some number of unsampled "burn-in" iterations should be used to allow the Markov chains to reach a starting point in a region of high probability density. In addition, an adequately large number of iterations must be sampled for the Markov chains to fully explore the posterior distribution. The Gelman-Rubin statistic (Gelman and Rubin, 1992) is a popular quantitative test for convergence of sampling. Exploring the target distribution may be especially challenging when the distribution is multimodal, and it is a rare event for a Markov chain to move between modes. In these situations, PT is expected to outperform MH by providing a faster means of escape.

Run times of Bayesian MCMC algorithms are expected to be dominated by the run times of the large number of simulations required to adequately sample probability distributions. We therefore do not expect a noticeable difference in performance between different implementations of the same MCMC algorithm run with the same settings, aside from differences in simulator efficiency. PyBioNetFit is a convenient tool for running MCMC because it supports both BNGL and SBML models without the need for custom code. In addition, MCMC in PyBioNetFit takes advantage of parallelization. In MH, individual Markov chains are not parallelizable, but PyBioNetFit can run multiple independent Markov chains in parallel and pool the results to create a larger sample of a probability distribution. In PT, the algorithm requires the simultaneous propagation of several Markov chains, and these chains are run in parallel. Efficiency of MH and PT is known to decline for high-dimensional parameter spaces. Problem 7 has a 16-dimensional parameter space, which is the largest for which we have used these methods.

MH and PT are widely used algorithms, but they are not suitable for every problem. More sophisticated MCMC algorithms described elsewhere include (among many others) Hamiltonian Monte Carlo (Betancourt, 2017), a gradient-based method implemented in other tools such as the statistical software package Stan (Carpenter et al., 2017), and the differential evolution Markov Chain family of algorithms (ter Braak and Vrugt, 2008). These more advanced algorithms are not included in the initial PyBioNetFit release, but the extensibility of PyBioNetFit (described in the Section Continued Development of PyBioNetFit) may allow them to be added in future development.

PyBioNetFit offers bootstrapping (Efron and Tibshirani, 1993; Press et al., 2007) as another uncertainty quantification method. Bootstrapping relies on the assumption that the experimental data points are

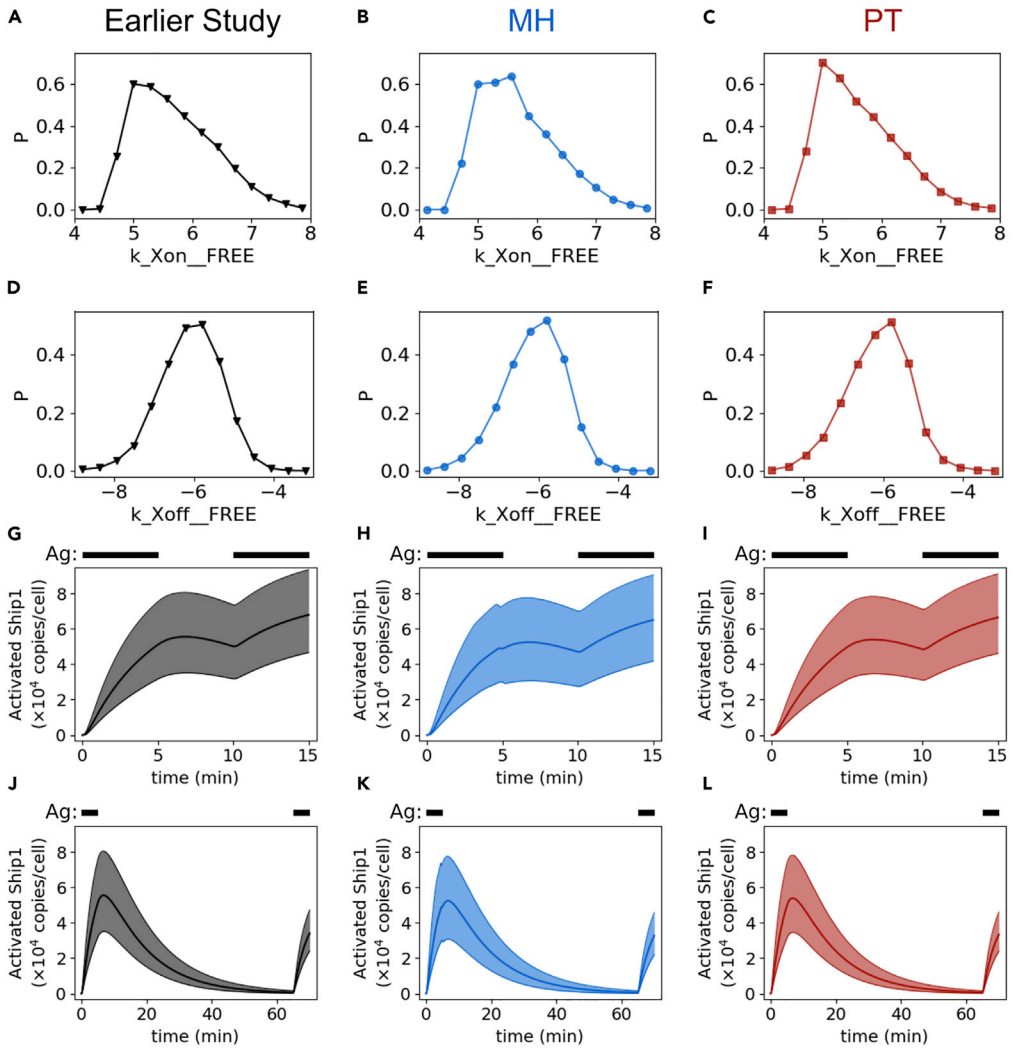


Figure 4. Bayesian Uncertainty Quantification in PyBioNetFit

(A–L) Results from PyBioNetFit's MH (B, E, H, and K) and PT (C, F, I, and L) algorithms are compared with the problem-specific code of Harmon et al. (2017) (A, D, G, and J). The data plotted in (A, D, G, and J) originally appeared in Harmon et al. (2017). (A–F) Marginal posterior probability distributions for selected parameters of the model of Harmon et al. (2017). Two examples of the 16 model parameters are shown. (G–L) Prediction uncertainty quantification for time courses of activated Ship1, one of the model outputs. Two antigen stimulation protocols are shown: one in (G–I) and the other in (J–L). Black bars above graphs indicate times when multivalent antigen was present. Solid curves indicate the median, and shaded areas indicate the 68% credible interval.

drawn from some (unknown) probability distribution and that drawing a sample from the data available is a good approximation of drawing a sample from the distribution.

Results of bootstrapping are typically reported as a "confidence interval" for the value of each parameter. Three important caveats must be kept in mind when interpreting this confidence interval. First, the interval refers specifically to the location of the *best-fit* parameter set. For this reason, bootstrapped intervals tend to be narrower than those obtained from a likelihood-based Bayesian approach (Fröhlich et al., 2014). In addition, if a parameter is unidentifiable, bootstrapping can yield a misleadingly narrow interval. Second, a bootstrapped confidence interval includes both uncertainty arising from experimental data and uncertainty introduced by imperfect performance of the fitting algorithm used (unless the algorithm has perfect performance with respect to finding the global minimum). Thus, when we obtain a "90% confidence interval" from bootstrapping, it means that if the experiment was repeated, and the fitting was repeated using

the new data, then the best-fit parameter is expected to fall within the interval with 90% confidence. Third, bootstrapping relies on the assumption that a resampled dataset is a good approximation of repeating an experiment. This assumption may not be valid when the size of the original experimental dataset is small. Interested readers can find further discussion of the advantages and limitations of bootstrapping in Chernick and LaBudde (2011).

To illustrate how bootstrapping can be used to measure uncertainty arising from different fitting algorithms, we consider a fitting problem consisting of an egg-shaped curve (Table 2, Problem 7), originally presented by Hlavacek et al. (2018). This toy problem is simple enough for PyBioNetFit's SS algorithm to find the global minimum, but the BioNetFit 1 algorithm is less effective. We performed bootstrapping on this problem with PyBioNetFit and found that the best fit for each parameter was identified to precision of order 10^{-4} with 90% confidence (Data S1, Problem 7). This high level of precision is unsurprising, given that the input data consist of densely sampled points on the target curve with minimal noise. In contrast, 90% confidence intervals reported using BioNetFit 1 span large ranges, of order 1 in some cases (Hlavacek et al., 2018). We conclude that the uncertainty reported with BioNetFit 1 arises mainly from limitations of the fitting algorithm, rather than from limitations in the amount or quality of data for fitting. Again, results from bootstrapping would be independent of the optimizer if the optimizer is always able to find a unique global minimum, but this is not a realistic expectation for many problems.

In summary, the uncertainty quantification methods in PyBioNetFit provide different and complementary functionalities. The Bayesian MH and PT algorithms estimate a multidimensional probability distribution showing the most probable parameter values (treated as random variables) based on the data. Different Bayesian algorithms with the same input data are expected to produce the same results, as long as algorithmic settings allow for sufficient sampling of the posterior probability distribution. Bootstrapping evaluates the uncertainty given a fitting algorithm in combination with a particular dataset. The resulting bootstrap confidence interval represents the confidence in the best-fit parameter values if both the experiment and the fitting were to be repeated.

Application: Fitting a Model of Yeast Cell Cycle Control Using Both Qualitative and Quantitative Data

To demonstrate the capabilities of PyBioNetFit to parameterize models using both qualitative and quantitative data, we used PyBioNetFit to re-solve a challenging, published parameterization problem involving a model of yeast cell cycle control developed by Tyson and co-workers (Chen et al., 2000, 2004; Csikász-Nagy et al., 2006; Kraikivski et al., 2015; Oguz et al., 2013). Early versions of this model were parameterized by hand-tuning (Chen et al., 2000, 2004), and later by problem-specific code with a search space informed by previous hand-tuned results (Oguz et al., 2013). In our previous work, we used problem-specific code to parameterize the model *ab initio* (Mitra et al., 2018). Our problem formulation used the model described by Oguz et al. (2013) and Laomettachit (2011), incorporating the qualitative data tabulated by Laomettachit et al. (2016) and the quantitative data of Spellman et al. (1998) (Table 2, Problem 8). Our goal in this work was to use PyBioNetFit to obtain a similar quality of fit to previous work. Because this example serves primarily as an illustration of PyBioNetFit functionality, we configured the problem to be identical to the previous study (Mitra et al., 2018) in terms of models, datasets, and objective function.

PyBioNetFit contains all the features needed to repeat the fitting job of Mitra et al. (2018). The input files to run the fitting job are provided as Data S1 (Problem 8). Like in the original study, we performed optimization using scatter search, as described in Transparent Methods.

We ran the fitting job in PyBioNetFit, and the resulting fit was of similar quality to that of previous work. We present a subset of the results in Figure 5 and the parameterized model in Data S1 (Problem 8). Our reported fit is the best result from 40 independent replicates. Convergence plots for all 40 replicates are shown in Figure S1. We achieved a minimum objective function value of 80, compared with 70 in Mitra et al. (2018). A difference is not surprising given the stochastic nature of the SS algorithm (or any metaheuristic). For comparison, our best objective function value from a starting sample of 500 random parameter sets was 5,493. Our fit is not identical to the previously published fit, which is expected because some model parameters were shown not to be identifiable (Mitra et al., 2018). However, like the published fit, the fit generated by PyBioNetFit shows reasonable consistency with the qualitative data (Figures 5A–5F). In five of the six example panels shown, the parameterized model is consistent with the constraint indicated by the horizontal lines (as described in the

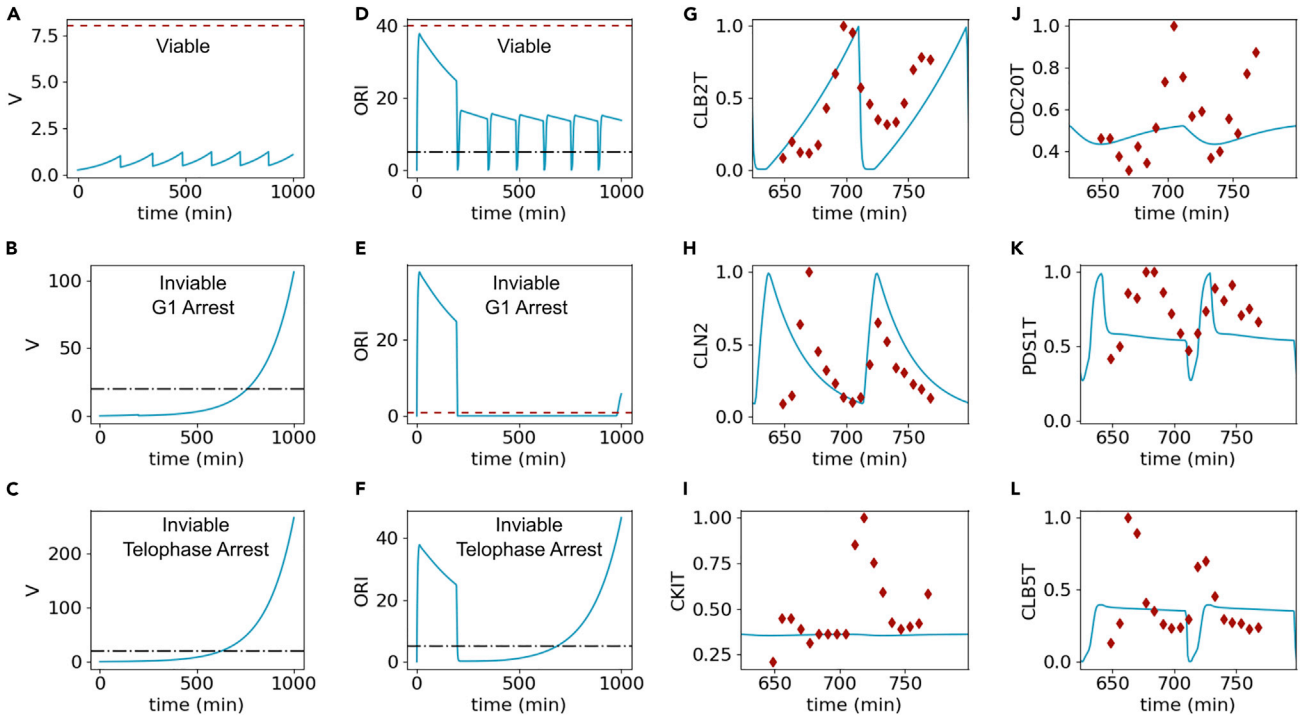


Figure 5. Example Outputs of the Model for Yeast Cell Cycle Control Parameterized with PyBioNetFit

(A–F) Selected output showing agreement with qualitative data. Two output variables are shown: *V* (A–C), representing cell volume, and *ORI* (D–F), a flag that indicates origin activation is completed when its value reaches 1. Results for three selected yeast strains are shown: wild-type (A and D), which is viable; a mutant (*cln3*Δ *bck2*Δ) (B and E), which has a G1 arrest phenotype; and another mutant (*cdc14-ts*) (C and F), which has a telophase arrest phenotype. Horizontal lines indicate qualitative constraints: time courses should exceed black dash-dot lines and should not exceed red dashed lines.

(G–L) Selected output showing agreement with quantitative data of Spellman et al. (1998) (red diamonds). These plots were shown in Mitra et al. (2018) with the best-fit results obtained in that study. Gene expression levels are shown for CLB2T (G), CLN2 (H), CKIT (I), CDC20T (J), PDS1T (K),and CLB5T (L).

See also Figure S1.

figure caption). In one panel (Figure 5E), the time course is inconsistent with the constraint, illustrating that although most constraints are satisfied by our best fit, not all are satisfied. The fit found by PyBioNetFit also captures certain features of the quantitative data (Figures 5G–L), such as, for example, the location of the peaks in (G) and (H). A more rigorous analysis of the misfit to quantitative data would require information about experimental measurement error, which was not available for this dataset.

Applications beyond Fitting: Model Checking

Although PyBioNetFit was designed for model parameterization, the property specification language of PyBioNetFit has additional applications in the analysis of parameterized models, namely, model checking and design. To demonstrate these applications, we consider the model of Shirin et al. (2019) (Figure 6A). The model describes the interactions between four kinases involved in the regulation of autophagy, a cellular recycling process. The model also describes the effects of six types of drugs in modulating these interactions and the level of autophagy. In the original study, this model was used to investigate the capabilities of the six drugs (labeled D_1 through D_6) to control the number of autophagic vesicles (AVs) in a cell. For our analysis, we assume that the published parameterization of the model, which was shown to be consistent with certain experimental data in the original study, is acceptable.

Model checking, as defined here, consists of evaluating whether a particular model satisfies a set of specified properties. To illustrate model checking, we considered eight hypothetical alternatives to the model of Shirin et al. (2019), each obtained by removing one of the labeled interactions from the network illustrated in Figure 6A. These changes are arbitrary for demonstration of the model checking workflow, but represent a scenario that could arise in practice: often, many models of the same biological process are developed by different research groups for different purposes, and a particular interaction might be

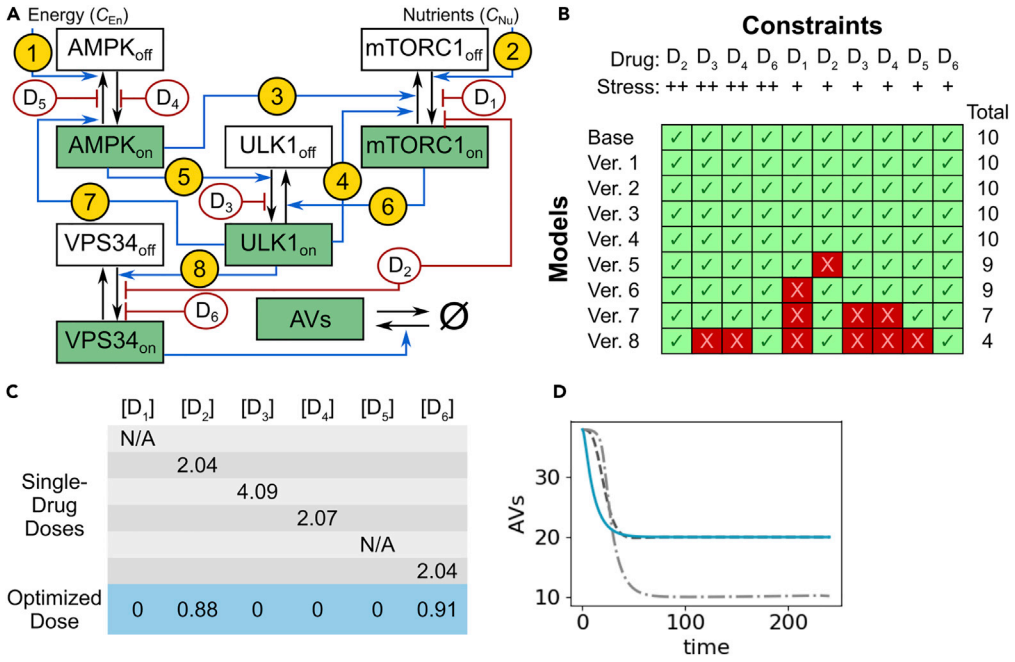


Figure 6. Applications of PyBioNetFit in Analysis of a Parameterized Model

(A) Schematic of the model to be analyzed, adapted from Shirin et al. (2019). Six drugs labeled D_1 through D_6 are capable of modulating various processes in the network shown. Interactions among kinases considered in the model are numbered 1–8.

(B) Model checking performed by PyBioNetFit of hypothetical alternatives to the model shown in (A). Each alternative model version (numbered 1–8) was obtained by removing one interaction from (A), corresponding to the version number. Each model version was checked against 10 qualitative behaviors characterized by Shirin et al. (2019): the change (increase or decrease) in AV count in response to a particular drug at a particular stress level, high (++) or medium (+).

(C) Optimizing drug dosing to achieve a desired system behavior. This table shows the minimal constant drug concentration to reduce AV count to 20 per cell or below, in a cell under high stress. Gray rows show optimized doses for each drug individually. The blue row shows the optimized dose by simultaneously tuning all six drug concentrations. Although all six drug doses were allowed to vary, the optimal solution had only two drugs with nonzero dose.

(D) Time course of AV counts under the treatments shown in (C). The gray broken line shows the response to treatment with D_3 only, the gray dash-dot line shows the response to treatment with D_4 only, and the blue solid line shows the response to the optimized six-drug dose. Responses to D_2 or D_6 only are indistinguishable from the response to the optimized dose (but require more total drug than with the optimized drug combination).

present in one model but absent in another. In such a scenario, it is reasonable to ask whether the interaction is important to the model's ability to reproduce certain system properties. As system properties to be checked in our demonstration, we use the characterization of the system's response to drug treatment by Shirin et al. (2019), which we take to be the established truth. Specifically, for the six drug treatments allowed in the model and two levels of cellular stress (determined by the energy and nutrient parameters of the model, C_{En} and C_{Nu}), Shirin et al. (2019) characterized the change in the number of AVs relative to control. Ten of these 12 model settings resulted in an increase or decrease in AV count. For our model checking exercise, we determined whether each of our hypothetical alternative models is able to reproduce these 10 qualitative behaviors.

To perform model checking in PyBioNetFit, we must express each property of interest in BPSL. For this model, properties can be written as inequalities between the AV count for the untreated case and AV counts for the drug-treated cases. In PyBioNetFit, this type of case-control comparison is configured by performing simulations corresponding to multiple versions of the model—here, one version for each stress/drug combination considered plus one version at each stress level with no drug. As described in the section Workflow Enabled by PyBioNetFit, PyBioNetFit requires each of these simulations to have a unique suffix (a string defined in the BNGL or CONF file). These suffixes can be used in the PROP file to refer to the outputs of specific simulations. For example, suppose that the simulation of wild-type has the suffix `data`, the simulation with drug D_2 has suffix `data_D2`, and after the system has equilibrated, the AV count

should be lower in the presence of D_2. We assume that the system is equilibrated in the time window of 120–240 min. Then the constraint would be written as

```
data.AV > data_D2.AV between 120,240
```

The full implementation of the model checking problem in PyBioNetFit is provided as Data S1 (Problem 9).

The results of model checking are shown in Figure 6B. Four of the variant models (versions 1–4) remain consistent with all 10 system properties, whereas the other four (versions 5–8) no longer satisfy one or more of the properties. In the context of this model, these results suggest that interactions 1–4 in Figure 6A are not essential to the qualitative properties that we considered. More generally, this example demonstrates the ability of PyBioNetFit's model checking utility to help distinguish between models.

Applications beyond Fitting: Design

In a design problem, we seek perturbations of a system to achieve a set of desired properties defined in BPSL. To illustrate a design problem in PyBioNetFit, we consider a problem similar to the original study of Shirin et al. (2019). Namely, we want to choose the concentrations of drugs D_1 through D_6 so as to drive the AV count below a desired threshold, while minimizing the total quantity of drug used. We arbitrarily choose a threshold of 20 AVs and set $C_{En} = C_{Nu} = 0.1$ (on a scale of 0–1), corresponding to a high level of cellular stress. In the original study, arbitrary time courses of drug dosing were permitted and simultaneous dosing of up to two drugs at a time (out of the six drugs in the model) was considered. Here, we solve a different problem in which we limit ourselves to constant drug concentrations, but allow for simultaneous dosing of up to six drugs.

We configure this problem as a fitting problem in PyBioNetFit, in which the free parameters to be estimated represent the unknown concentrations of each of the six drugs. The desired system property of reducing AV count below 20 is implemented as an inequality constraint, and the goal to minimize drug concentration is implemented as a quantitative data point (i.e., minimizing the difference between the actual total drug dose and 0). The full configuration of this problem is provided as Data S1 (Problem 9).

The optimization results are shown in Figure 6C (bottom row). For comparison, we also performed optimizations in which only one of the drug concentrations was allowed to vary (Figure 6C). The results are consistent with those reported by Shirin et al. (2019). Note that the optimized combined dose allowing all six drugs uses less total drug than any of the single-drug doses. The optimized dosing schemes for both single-drug and combination treatments achieve the desired property of driving AV count below 20 (Figure 6D).

This example illustrates an additional, important class of problems that can be addressed with PyBioNetFit: the design of perturbations to a biological system to achieve specified behavior. More specifically, the example illustrates optimization of targeted drug treatments, which has been a long-standing goal in systems biology (Fitzgerald et al., 2006). The automated design of perturbations, with formal definition of target behavior, is systematic and less likely to miss effective perturbations than an *ad hoc* approach to model analysis.

DISCUSSION

Comparison to Related Tools

Some features of PyBioNetFit are unique and novel, whereas other features have some overlap with other available optimization tools. Here we analyze and discuss the strengths and weaknesses of PyBioNetFit when compared with other published tools.

As PyBioNetFit was designed for parameterization of rule-based models written in BNGL, our primary comparison is with PyBioNetFit's predecessor, BioNetFit 1 (Thomas et al., 2016), which was previously state of the art for this application domain. In particular, no other tools to our knowledge support parameterization of models simulated with BioNetGen's stochastic algorithms (BioNetGen SSA, and NFsim). PyBioNetFit makes major improvements over BioNetFit 1 in terms of new functionality, as well as improved implementation of BioNetFit 1 functionality.

In our experience, PyBioNetFit far outperforms BioNetFit 1. As one example comparison, we ran Problem 2 in BioNetFit 1 with population size 144 (Data S1, Problem 2). We considered using the cluster-computing

capabilities of BioNetFit 1 but found that the fitting ran faster on a single node (due to overhead in communicating with the cluster manager). BioNetFit 1 was unable to reach the target objective value within 10 h in any of five fitting replicates. For comparison, the fastest PyBioNetFit algorithm at a parallel count of 144 on a cluster had a median run time of 1.9 h (Figure 3B), which is a significantly better performance by the Mann-Whitney U test ($p = 1.1 \times 10^{-3}$).

A larger set of software is available for parameterization of ODE models defined in SBML. For smaller ODE models, we recommend gradient-based methods implemented in other tools as a starting point, as these algorithms tend to be more efficient than metaheuristics for problems where they are feasible (Raue et al., 2013). Data2Dynamics (Raue et al., 2015) uses forward sensitivity analysis (Leis and Kramer, 1988) to calculate the gradient of the objective function. Its default optimizer, which the developers recommend for most applications (Raue et al., 2013), is MATLAB's lsqnonlin function (which implements a trust region-reflective algorithm, MathWorks, 2018). Gradient-based methods are also supported in COPASI (Hoops et al., 2006), which calculates gradients by the finite difference approximation. We chose not to include gradient-based methods in PyBioNetFit at this time because existing tools already provide acceptable solutions.

In rugged parameter landscapes, gradient-based methods are susceptible to becoming trapped in local minima and slowed near saddle points. This issue can be addressed by performing multiple optimization runs at different start points but can become limiting if the parameter space has too many local minima. Metaheuristic algorithms can also become trapped in local minima, but experience suggests that they are more capable of escape than gradient-based methods. Various factors likely contribute to this capability, including uphill moves, random behavior, and exchange of information between multiple searchers.

We expect forward sensitivity analysis to perform well for ODE problems on the typical scale of the problems benchmarked by Hass et al. (2019). This method has been shown to scale roughly linearly with respect to the number of free parameters (Kapfer et al., 2019). The cost also depends on the number of equations. Scaling with respect to number of equations is of particular interest for rule-derived ODE models because such models often result in many more equations than typically arise in manually formulated ODE models. Even fairly simple rule-based models (in terms of number of parameters and rules defined) can imply hundreds to thousands of differential equations.

To illustrate the scaling behavior (with respect to number of ODEs) of forward sensitivity analysis as implemented in Data2Dynamics, we measured the run time of optimization on the ODE models of Problems 1 and 2, which were also used to benchmark PyBioNetFit. We note that Data2Dynamics can run multiple independent optimization runs in parallel to improve the probability of finding a good solution, but, in contrast to the parallelization of metaheuristic algorithms, this parallelization cannot improve the wall time of an individual run. On Problem 1 (a conventional ODE model with 30 equations and 46 parameters), Data2Dynamics completed optimization in 4 min, compared with median run times ranging from 11 to 14 min on 288 cores for the four metaheuristic algorithms of PyBioNetFit. Multiple runs of Data2Dynamics on this problem suggested that there is not large variability in run times between runs. On Problem 2 (a rule-derived ODE model with 356 equations and 37 parameters), Data2Dynamics required 8 h to complete one optimization run, compared with median run times ranging from 1.5 to 3.6 h on 288 cores for the four algorithms of PyBioNetFit. In this case, Data2Dynamics used a significant amount of run time simply for setup of the forward sensitivity equations. Of course, one cannot draw broad conclusions based on the results of only two problems. However, these results are consistent with what we would expect given that integration of ODE systems with many equations is costly, and forward sensitivity analysis requires more expensive integration (to calculate sensitivities with respect to each parameter) than is needed for simple objective function evaluation. The illustrated behavior is also what we would expect for gradient-based optimization in COPASI.

Recent work has demonstrated that adjoint sensitivity analysis can be effective for gradient computation for larger ODE models when forward sensitivity analysis is inefficient (Fröhlich et al., 2018). This approach has been used to solve a parameterization problem with 1,200 equations and 4,100 parameters (Fröhlich et al., 2018), which is a larger scale than we have considered with PyBioNetFit. To the best of our knowledge, adjoint methods have yet to be demonstrated for rule-derived ODE systems (or any system with many more equations than free parameters), but the good scaling properties of adjoint methods suggest such an approach would be feasible. The package AMICI (Fröhlich et al., 2017) supports adjoint sensitivity

analysis for biological applications but offers only limited workflows. (Adjoint sensitivity analysis is also available in some general-purpose ODE solvers, Rackauckas et al., 2018.) AMICI is designed for use with time-series data with a known initial condition. Model parameterization can be performed by writing code to use AMICI in combination with the optimization toolbox PESTO (Stapor et al., 2018). We recommend that AMICI/PESTO be used for parameterizing large ODE models (hundreds of differential equations or larger) if the available workflows support the problem of interest.

A unique feature of PyBioNetFit is its support for a domain-specific property specification language (BPSL). To our best knowledge, no other biological modeling tool has a comparable functionality for specification of qualitative properties. Previous work on biological property specification (Clarke et al., 2008; David et al., 2012; Heath et al., 2008; Hussain et al., 2015; Khalid and Jha, 2018; Kwiatkowska et al., 2008; Liu and Faeder, 2016) relied on bespoke software, whereas BPSL can be used with the general-purpose functionality of PyBioNetFit. BPSL is also designed to be more human readable than conventional linear temporal logic (LTL), for instance. We expect a BPSL statement (but not an LTL expression) to be understandable to anyone with a background in biological modeling. For example, consider the following BPSL statement:

```
A<1 between B = 2, B = 3
```

This statement is equivalent to the following LTL expression:

$$\mathbf{F}(B = 2) \Rightarrow ((\neg(B = 2))\mathbf{U}(B = 2\hat{}(A < 1\mathbf{W}B = 3)))$$

where **F** is the "future" operator, **U** is the "until" operator, and **W** is the "weak until" operator. A drawback of BPSL relative to LTL is that the available enforcement keywords (Table 1) enable only a subset of what is possible with LTL. However, the current BPSL grammar is sufficient to support all constraints formulated in Mitra et al. (2018) to fit the yeast cell cycle model of Oguz et al. (2013) and Laomettachit (2011). PyBioNetFit was written with extensibility in mind, such that it is possible to add to the BPSL grammar as needs arise in other modeling problems. We also note that although PyBioNetFit is the first tool to support BPSL, information represented in BPSL need not be tied to one model or software tool. In the future, it will be possible for us or others to develop additional tools compatible with BPSL.

Figure 7 summarizes the niche filled by PyBioNetFit in relation to other software supporting complete fitting workflows for biological models. PyBioNetFit is unique in its support for qualitative data (including model checking and design applications) and for its built-in, well-engineered support for cluster computing. PyBioNetFit is also notable for its multiple algorithm options that provide algorithm-level parallelization. BioNetFit 1 provides only one such algorithm, and other tools support only parallelization of independent runs. PyBioNetFit has the largest overlap in functionality with its predecessor BioNetFit 1, but as described above, PyBioNetFit far outperforms BioNetFit 1 in head-to-head comparisons. PyBioNetFit is recommended over BioNetFit 1 for all overlapping features, including parameterization of stochastic models. Data2Dynamics and COPASI tend to have use cases distinct from PyBioNetFit, such as for ODE problems that benefit from gradient-based optimization using forward sensitivity analysis or the finite difference approximation.

Comparison to Problem-Specific Coding

PyBioNetFit joins Data2Dynamics and COPASI in the class of software supporting standardized biological model-definition formats and complete workflows for model parameterization and has strengths that are complementary to these existing tools. These free-standing applications contrast with the approach of using problem-specific code written in a high-level programming language such as Python, R, or MATLAB. We acknowledge that problem-specific code is a good choice in some use cases, such as when the model of interest is already implemented in one of these languages, or when analyzing a model with an unusual feature that is not supported in SBML or BNGL. Many packages are available that can help streamline model parameterization in high-level programming languages. For coding a model, one could use standard differential equation packages, or PySB (Lopez et al., 2013), a package for building biological models in Python. Gradient-based algorithms with forward sensitivity analysis are available in dMod (Kaschek et al., 2019). Other packages implement metaheuristic optimization algorithms (Egea et al., 2014; Garrett, 2012; Fortin et al., 2012) and Bayesian uncertainty quantification algorithms (Eydgahi et al., 2013; Gupta et al., 2018; Shockley et al., 2018). AMIGO (Balsa-Canto et al., 2016) is a notable

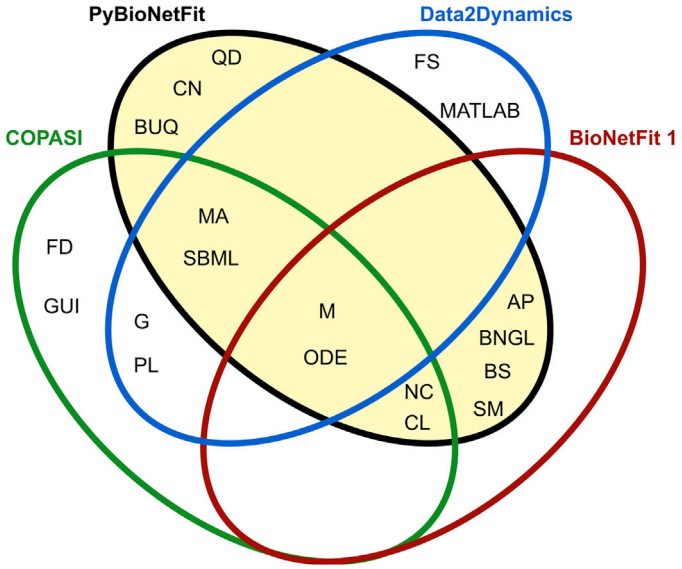


Figure 7. Venn Diagram Comparing the Functionality Provided in PyBioNetFit with that of Three Other Programs Supporting Parameterization of Biological Models

The abbreviations in the diagram stand for the following features: AP, algorithm-level parallelization: each algorithm step runs multiple objective function evaluations in parallel; BNGL, support for BNGL models; BS, bootstrapping; BUQ, Bayesian uncertainty quantification; CL, command-line interface; CN, native support for cluster computing. (Although any program can be run on a cluster with sufficient configuration by the user, PyBioNetFit was designed for this purpose. Its documentation includes instructions for how to run the program on multiple cluster nodes, and we have demonstrated this use case with up to 8 nodes, 288 cores.) FD, finite difference approximation; FS, forward sensitivity analysis; G, gradient-based algorithms; GUI, GUI for configuring and running fitting; M, metaheuristic algorithms; MA, multiple algorithm options available; MATLAB, MATLAB interface; NC, free with no commercial dependencies; ODE, support for ODE models; PL, profile likelihood; QD, fitting with qualitative data (including model checking and design applications); SBML, support for SBML models; SM, support for stochastic models.

MATLAB optimization toolbox. Packages such as dask.distributed (Rocklin, 2015) are available to help with parallelization on clusters.

Even with the sophistication of these tools, some amount of custom code is necessary to use these tools to solve a given problem of interest. We argue that in cases where writing BNGL or SBML models is feasible, the functionality of PyBioNetFit is preferable to problem-specific code. PyBioNetFit combines all the functionality required for model parameterization into a single package. It removes the need for debugging at the level of the programming language, which reduces the propensity for errors in the modeling work. PyBioNetFit allows a modeler to instead focus on designing models and choosing appropriate algorithms for parameterization and analysis. BPSL facilitates consideration of qualitative data, improving on our published approach using problem-specific code (Mitra et al., 2018).

A second advantage of using PyBioNetFit is in the reproducibility of results (Medley et al., 2016; Waltemath and Wolkenhauer, 2016). Although it is possible to create well-documented, reproducible problem-specific code, using IPython or R notebooks, for example, such good practices are not always followed. Often problem-specific code is developed to be run on a specific machine, without portability in mind. Concerns of expedience dominate the coding effort. In contrast, PyBioNetFit achieves a separation of concerns, in which a job can be documented by providing the set of input files used, along with the version number of the code, and there is no need to disentangle this information from the implementation of any algorithm.

Continued Development of PyBioNetFit

PyBioNetFit is released open source on GitHub (https://github.com/lanl/PyBNF) with the hope that we and others will continue to improve PyBioNetFit. The GitHub page includes an active issue tracker that facilitates reporting of bugs and feature requests.

We welcome contributions to PyBioNetFit from the community. We designed PyBioNetFit such that it should be straightforward to implement additional optimization and MCMC algorithms, as we are aware that many such algorithms are described in the literature. The PyBioNetFit documentation (Mitra and Suderman, 2019) includes instructions for contributing new algorithms to the PyBioNetFit code base.

Conclusion

PyBioNetFit offers a versatile set of tools, which we expect to be useful in parameterization of new biological models. PyBioNetFit is best in class for BNGL-formatted models, and notable for its support for stochastic biological models. PyBioNetFit supports several workflows, including fitting to time-series data, dose-response data, and qualitative data. We provide the first available implementation of our recent approach (Mitra et al., 2018) for leveraging both quantitative and qualitative data in a single parameterization problem. This approach is enabled by BPSL, which can also be used for model checking and design. The workflows supported in PyBioNetFit can be used for parameterizing standard ODE models, although for this application, gradient-based tools may be more efficient.

Our hope is that PyBioNetFit lowers the technical barrier to parameter fitting, by enabling fitting without problem-specific coding. PyBioNetFit will promote reproducible modeling by encouraging the use of existing model standards (BNGL and SBML).

We have shown that parameter identification can be challenging, and the best choice of fitting algorithm is not always obvious. By providing robust implementations of several algorithms, we encourage experimentation with different algorithms and settings to find the best choice for a problem of interest.

Limitations of the Study

PyBioNetFit can solve a wide variety of biological modeling problems, but is not the best solution for every problem. As described in the main text, many ODE models are more effectively parameterized using gradient-based algorithms. In addition, PyBioNetFit's metaheuristic algorithms can find fits that appear reasonable, but cannot guarantee that a global optimum has been reached. Parameterization using qualitative data, as implemented in BPSL, has the limitation that the objective function lacks a statistical interpretation, and so cannot be used in Bayesian uncertainty quantification algorithms.

METHODS

All methods can be found in the accompanying Transparent Methods supplemental file.

DATA AND CODE AVAILABILITY

The most recent version of PyBioNetFit is v1.0.1, available online at https://github.com/lanl/PyBNF. The repository includes a user manual, `Documentation_PyBioNetFit.pdf`. The same user manual is available online as a standalone website (Mitra and Suderman, 2019). General information about PyBioNetFit is available at http://bionetfit.nau.edu/.

PyBioNetFit can be installed on any current Linux, macOS, or Windows computer, as well as on Linux clusters. Installation of Python 3 is required if it is not already included with the operating system. Root access is not usually required, allowing for PyBioNetFit to be readily installed on shared clusters. PyBioNetFit can be installed from source by downloading the code at the above GitHub link, or can be installed directly using the pip package manager with the command

```
python3 -m pip install pybnf
```

Data associated with the example fitting problems (Table 2) are provided as Data S1 and are also available online at https://github.com/RuleWorld/RuleHub/tree/2019Aug21/Published/Mitra2019. MCMC samples associated with Figure 4 are available in the BioStudies database (http://www.ebi.ac.uk/biostudies) under accession number S-BSST240.

SUPPLEMENTAL INFORMATION

Supplemental Information can be found online at https://doi.org/10.1016/j.isci.2019.08.045.

ACKNOWLEDGMENTS

This work was supported by grant R01GM111510 from the National Institute of General Medical Sciences (NIGMS) of the National Institutes of Health (NIH). W.S.H. acknowledges support from the Joint Design of Advanced Computing Solutions for Cancer (JDACS4C) program established by the U.S. Department of Energy (DOE) and the National Cancer Institute (NCI) of NIH. R.S. and A.I. acknowledge support from the Center for Nonlinear Studies at Los Alamos National Laboratory (LANL), which is operated for the National Nuclear Security Administration (NNSA) of the DOE under contract 89233218CNA000001. H.M.S. acknowledges the support of grant R01GM123032 from NIGMS/NIH and grant P41EB023912 from the National Institute of Biomedical Imaging and Bioengineering (NIBIB) of NIH. We thank J. Kyle Medley and Kiri Choi for assistance with libRoadRunner development. We thank Adrian Hauber for assistance with Data2Dynamics. Computational resources used in this study included the following: the Darwin cluster at LANL, which is supported by the Computational Systems and Software Environment (CSSE) subprogram of the Advanced Simulation and Computing (ASC) program at LANL, which is funded by NNSA/DOE; resources were provided by the LANL Institutional Computing program, which is funded by NNSA/DOE, and Northern Arizona University's Monsoon computer cluster, which is funded by Arizona's Technology and Research Initiative Fund.

AUTHOR CONTRIBUTIONS

W.S.H. and R.G.P. designed the study. E.D.M. and R.S. wrote the software. A.I. performed alpha testing. E.D.M. and J.C. performed benchmarking. A.H. and H.S. upgraded libRoadRunner to enable integration into PyBioNetFit. E.D.M. and W.S.H. wrote the manuscript with input from the other authors. All authors read and approved the final manuscript.

DECLARATION OF INTERESTS

The authors declare no competing interests.

Received: April 1, 2019
Revised: June 21, 2019
Accepted: August 22, 2019
Published: September 27, 2019

REFERENCES

Balsa-Canto, E., Henriques, D., Gábor, A., and Banga, J.R. (2016). AMIGO2, a toolbox for dynamic modeling, optimization and control in systems biology. Bioinformatics *32*, 3357–3359.

Betancourt, M. (2017). A conceptual introduction to Hamiltonian Monte Carlo. arXiv, https://arxiv.org/abs/1701.02434.

Blinov, M.L., Faeder, J.R., Goldstein, B., and Hlavacek, W.S. (2004). BioNetGen: software for rule-based modeling of signal transduction based on the interactions of molecular domains. Bioinformatics *20*, 3289–3291.

Blinov, M.L., Faeder, J.R., Goldstein, B., and Hlavacek, W.S. (2006). A network model of early events in epidermal growth factor receptor signaling that accounts for combinatorial complexity. BioSystems *83*, 136–151.

Boehm, M.E., Adlung, L., Schilling, M., Roth, S., Klingmüller, U., and Lehmann, W.D. (2014). Identification of isoform-specific dynamics in phosphorylation-dependent STAT5 dimerization by quantitative mass spectrometry and mathematical modeling. J. Proteome Res. *13*, 5685–5694.

Suderman, R., Mitra, E.D., Lin, Y.T., Erickson, K.E., Feng, S., and Hlavacek, W.S. (2019). Generalizing Gillespie's direct method to enable network-free simulations. Bull. Math. Biol. *81*, 2822–2848.

Brännmark, C., Palmér, R., Glad, S.T., Cedersund, G., and Strålfors, P. (2010). Mass and information feedbacks through receptor endocytosis govern insulin signaling as revealed using a parameter-free modeling framework. J. Biol. Chem. *285*, 20171–20179.

Cao, Y., Li, S., and Petzold, L. (2002). Adjoint sensitivity analysis for differential-algebraic equations: algorithms and software. J. Comput. Appl. Math. *149*, 171–191.

Carpenter, B., Gelman, A., Hoffman, M.D., Lee, D., Goodrich, B., Betancourt, M., Brubaker, M., Guo, J., Li, P., and Riddell, A. (2017). Stan: a probabilistic programming language. J. Stat. Softw. *76*, 1–32.

Chen, K.C., Csikasz-Nagy, A., Gyorffy, B., Val, J., Novak, B., and Tyson, J.J. (2000). Kinetic analysis of a molecular model of the budding yeast cell cycle. Mol. Biol. Cell *11*, 369–391.

Chen, K.C., Calzone, L., Csikasz-Nagy, A., Cross, F.R., Novak, B., and Tyson, J.J. (2004). Integrative analysis of cell cycle control in budding yeast. Mol. Biol. Cell *15*, 3841–3862.

Chernick, M.R., and LaBudde, R.A. (2011). An Introduction to Bootstrap Methods with Applications to R (John Wiley & Sons).

Chib, S., and Greenberg, E. (1995). Understanding the Metropolis-Hastings algorithm. Am. Stat. *49*, 327–335.

Choi, K., Medley, J.K., König, M., Stocking, K., Smith, L., Gu, S., and Sauro, H.M. (2018). Tellurium: an extensible python-based modeling environment for systems and synthetic biology. BioSystems *171*, 74–79.

Chylek, L.A., Harris, L.A., Tung, C.-S., Faeder, J.R., Lopez, C.F., and Hlavacek, W.S. (2013). Rule-based modeling: a computational approach for studying biomolecular site dynamics in cell signaling systems. Wiley Interdiscip. Rev. Syst. Biol. Med. *6*, 13–36.

Chylek, L.A., Akimov, V., Dengjel, J., Rigbolt, K.T.G., Hu, B., Hlavacek, W.S., and Blagoev, B. (2014). Phosphorylation site dynamics of early T-cell receptor signaling. PLoS One *9*, e104240.

Clarke, E.M., Emerson, E.A., and Sistla, A.P. (1986). Automatic verification of finite state concurrent system using temporal logic specifications. ACM Lett. Program Lang. Syst. *8*, 244–263.

Clarke, E.M., Grumberg, O., Peled, D., and Belta, P.C. (1999). Model Checking (Cambridge: MIT Press).

Clarke, E.M., Faeder, J.R., Langmead, C.J., Harris, L.A., Jha, S.K., and Legay, A. (2008). Statistical model checking in BioLab: applications to the automated analysis of T-cell receptor signaling pathway. In Computational Methods in Systems Biology, M. Heiner and A.M. Uhrmacher, eds. (Springer), pp. 231–250.

Csikász-Nagy, A., Battogtokh, D., Chen, K.C., Novák, B., and Tyson, J.J. (2006). Analysis of a generic model of eukaryotic cell-cycle regulation. Biophys. J. *90*, 4361–4379.

Danos, V., and Laneve, C. (2004). Formal molecular biology. Theor. Comput. Sci. *325*, 69–110.

David, A., Larsen, K.G., Legay, A., Mikučionis, M., Poulsen, D.B., and Sedwards, S. (2012). Runtime verification of biological systems. In Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change, T. Margaria and B. Steffen, eds. (Springer), pp. 388–404.

Dunster, J.L., Byrne, H.M., and King, J.R. (2014). The resolution of inflammation: a mathematical model of neutrophil and macrophage interactions. Bull. Math. Biol. *76*, 1953–1980.

Earl, D.J., and Deem, M.W. (2005). Parallel tempering: theory, applications, and new perspectives. Phys. Chem. Chem. Phys. *7*, 3910.

Eberhart, R. and Kennedy, J.. (1995), A new optimizer using particle swarm theory, in MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science, IEEE, pp. 39–43.

Efron, B., and Tibshirani, R.J. (1993). An Introduction to the Bootstrap (Chapman and Hall).

Egea, J.A., Henriques, D., Cokelaer, T., Villaverde, A.F., MacNamara, A., Danciu, D.-P., Banga, J.R., and Saez-Rodriguez, J. (2014). MEIGO: an open-source software suite based on metaheuristics for global optimization in systems biology and bioinformatics. BMC Bioinformatics *15*, 136.

Erickson, K.E., Rukhlenko, O.S., Shahinuzzaman, M., Slavkova, K.P., Lin, Y.T., Suderman, R., Stites, E.C., Anghel, M., Posner, R.G., Barua, D., et al. (2019). Modeling cell line-specific recruitment of signaling proteins to the insulin-like growth factor 1 receptor. PLoS Comput. Biol. *15*, e1006706.

Eydgahi, H., Chen, W.W., Muhlich, J.L., Vitkup, D., Tsitsiklis, J.N., and Sorger, P.K. (2013). Properties of cell death models calibrated and compared using Bayesian approaches. Mol. Syst. Biol. *9*, 644.

Faeder, J.R., Hlavacek, W.S., Reischl, I., Blinov, M.L., Metzger, H., Redondo, A., Wofsy, C., and Goldstein, B. (2003). Investigation of early events in FcεRI-mediated signaling using a detailed mathematical model. J. Immunol. *170*, 3769–3781.

Faeder, J.R., Blinov, M.L., Goldstein, B., and Hlavacek, W.S. (2005). Rule-based modeling of biochemical networks. Complexity *10*, 22–41.

Faeder, J.R., Blinov, M.L., and Hlavacek, W.S. (2009). Rule-based modeling of biochemical systems with BioNetGen. Methods Mol. Biol. *500*, 113–167.

Fey, D., Halasz, M., Dreidax, D., Kennedy, S.P., Hastings, J.F., Rauch, N., Munoz, A.G., Pilkington, R., Fischer, M., Westermann, F., et al. (2015). Signaling pathway models as biomarkers: patient-specific simulations of JNK activity predict the survival of neuroblastoma patients. Sci. Signal. *8*, 1–16.

Fitzgerald, J.B., Schoeberl, B., Nielsen, U.B., and Sorger, P.K. (2006). Systems biology and combination therapy in the quest for clinical efficacy. Nat. Chem. Biol. *2*, 458–466.

Fortin, F.-A., De Rainville, F.-M., Gardner, M.-A., Parizeau, M., and Gagné, C. (2012). DEAP: evolutionary algorithms made easy. J. Mach. Learn. Res. *13*, 2171–2175.

Fröhlich, F., Theis, F.J., and Hasenauer, J. (2014). Uncertainty analysis for non-identifiable dynamical systems: profile likelihoods, bootstrapping and more. In Computational Methods in Systems Biology, P. Mendes, J.O. Dada, and K. Smallbone, eds. (Springer International Publishing), pp. 61–72.

Fröhlich, F., Kaltenbacher, B., Theis, F.J., and Hasenauer, J. (2017). Scalable parameter estimation for genome-scale biochemical reaction networks. PLoS Comput. Biol. *13*, e1005331.

Fröhlich, F., Kessler, T., Weindl, D., Shadrin, A., Schmiester, L., Hache, H., Muradyan, A., Schütte, M., Lim, J.-H., Heinig, M., et al. (2018). Efficient parameter estimation enables the prediction of drug response using a mechanistic pan-cancer pathway model. Cell Syst. *7*, 567–579.e6.

Gandomi, A.H., Yang, X.-S., Talatahari, S., and Alavi, A.H. (2013). Metaheuristic algorithms in modeling and optimization. In Metaheuristic Applications in Structures and Infrastructures, A.H. Gandomi, X.-S. Yang, S. Talatahari, and A.H. Alavi, eds. (Elsevier), pp. 1–24.

Garrett, A. (2012). Inspyred: a framework for creating bio-inspired computational intelligence algorithms in Python. https://github.com/aarongarrett/inspyred.

Gelman, A., and Rubin, D.B. (1992). Inference from iterative simulation using multiple sequences. Stat. Sci. *7*, 457–511.

Gillespie, D.T. (2006). Stochastic simulation of chemical kinetics. Annu. Rev. Phys. Chem. *58*, 35–55.

Glover, F., Laguna, M., and Martí, R. (2000). Fundamentals of scatter search and path relinking. Control Cybernetics *29*, 652–684.

Gupta, A., and Mendes, P. (2018). An overview of network-based and -free approaches for stochastic simulation of biochemical systems. Computation *6*, 9.

Gupta, S., Hainsworth, L., Hogg, J.S., Lee, R.E.C. and Faeder, J.R.. (2018), Evaluation of parallel tempering to accelerate Bayesian parameter estimation in systems biology, in 2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP), pp. 690–697.

Harmon, B., Chylek, L.A., Liu, Y., Mitra, E.D., Mahajan, A., Saada, E.A., Schudel, B.R., Holowka, D.A., Baird, B.A., Wilson, B.S., et al. (2017). Timescale separation of positive and negative signaling creates history-dependent responses to IgE receptor stimulation. Sci. Rep. *7*, 15586.

Harris, L.A., Hogg, J.S., Tapia, J.-J., Sekar, J.A.P., Gupta, S., Korsunsky, I., Arora, A., Barua, D., Sheehan, R.P., and Faeder, J.R. (2016). BioNetGen 2.2: advances in rule-based modeling. Bioinformatics *32*, 3366–3368.

Hass, H., Loos, C., Alvarez, E.R., Timmer, J., Hasenauer, J., and Kreutz, C. (2019). Benchmark problems for dynamic modeling of intracellular processes. Bioinformatics. https://doi.org/10.1093/bioinformatics/btz020. Epub ahead of print.

Heath, J., Kwiatkowska, M., Norman, G., Parker, D., and Tymchyshyn, O. (2008). Probabilistic model checking of complex biological pathways. Theor. Comput. Sci. *391*, 239–257.

Hindmarsh, A.C., Brown, P.N., Grant, K.E., Lee, S.L., Serban, R., Shumaker, D.E., and Woodward, C.S. (2005). SUNDIALS: suite of nonlinear and differential/algebraic equation solvers. ACM Trans. Math. Softw. *31*, 363–396.

Hlavacek, W.S., Csicsery-Ronay, J., Baker, L.R., Ramos Álamo, M.D.C., Ionkov, A., Mitra, E.D., Suderman, R., Erickson, K.E., Dias, R., Colvin, J., et al. (2018). A step-by-step guide to using BioNetFit. In Modeling Biomolecular Site Dynamics, *1945*, W.S. Hlavacek, ed. Methods in Molecular Biology (Humana Press), pp. 391–419.

Hoops, S., Gauges, R., Lee, C., Pahle, J., Simus, N., Singhal, M., Xu, L., Mendes, P., and Kummer, U. (2006). Copasi - a complex pathway simulator. Bioinformatics *22*, 3067–3074.

Hucka, M., Finney, A., Sauro, H.M., Bolouri, H., Doyle, J.C., Kitano, H., Arkin, A.P., Bornstein, B.J., Bray, D., Cornish-Bowden, A., et al. (2003). The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. Bioinformatics *19*, 524–531.

Hussain, F., Langmead, C.J., Mi, Q., Dutta-Moscato, J., Vodovotz, Y., and Jha, S.K. (2015). Automated parameter estimation for biological models using Bayesian statistical model checking. BMC Bioinformatics *16*, S8.

Kapfer, E.-M., Stapor, P., and Hasenauer, J. (2019). Challenges in the calibration of large-scale ordinary differential equation models. bioRxiv, 690222, https://www.biorxiv.org/content/10.1101/690222v1.

Khalid, A. and Jha, S.K.. (2018), Calibration of rule-based stochastic biochemical models using statistical model checking, in 2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), IEEE, pp. 179–184.

Kaschek, D., Mader, W., Fehling-Kaschek, M., Rosenblatt, M., and Timmer, J. (2019). Dynamic modeling, parameter estimation, and uncertainty analysis in R. J. Stat. Softw. *88*, https://doi.org/10.18637/jss.v088.i10.

Kiselyov, V.V., Versteyhe, S., Gauguin, L., and De Meyts, P. (2009). Harmonic oscillator model of the insulin and IGF1 receptors' allosteric binding and activation. Mol. Syst. Biol. *5*, 243.

Kocieniewski, P., Faeder, J.R., and Lipniacki, T. (2012). The interplay of double phosphorylation and scaffolding in MAPK pathways. J. Theor. Biol. *295*, 116–124.

Kozer, N., Barua, D., Orchard, S., Nice, E.C., Burgess, A.W., Hlavacek, W.S., and Clayton, A.H.A. (2013). Exploring higher-order EGFR oligomerisation and phosphorylation–a combined experimental and theoretical approach. Mol. Biosyst. *9*, 1849–1863.

Kraikivski, P., Chen, K.C., Laomettachit, T., Murali, T.M., and Tyson, J.J. (2015). From START to FINISH: computational analysis of cell cycle control in budding yeast. NPJ Syst. Biol. Appl. *1*, 15016.

Kühn, C., and Hillmann, K. (2016). Rule-based modeling of labor market dynamics: an introduction. J. Econ. Interact. Coord. *11*, 57–76.

Kwiatkowska, M., Norman, G., and Parker, D. (2008). Using probabilistic model checking in systems biology. ACM SIGMETRICS Perform. Eval. Rev. *35*, 14.

Laomettachit, T. (2011). Mathematical Modeling Approaches for Dynamical Analysis of Protein Regulatory Networks with Applications to the Budding Yeast Cell Cycle and the Circadian Rhythm in Cyanobacteria, PhD thesis (Virginia Polytechnic Institute and State University).

Laomettachit, T., Chen, K.C., Baumann, W.T., and Tyson, J.J. (2016). A model of yeast cell-cycle regulation based on a standard component modeling strategy for protein regulatory networks. PLoS One *11*, e0153738.

Lee, D., and Wiswall, M. (2007). A parallel implementation of the simplex function minimization routine. Computat. Econ. *30*, 171–187.

Lee, E., Salic, A., Krüger, R., Heinrich, R., and Kirschner, M.W. (2003). The roles of APC and axin derived from experimental and theoretical analysis of the Wnt pathway. PLoS Biol. *1*, 116–132.

Leeuw, T., Wu, C., Schrag, J.D., Whiteway, M., Thomas, D.Y., and Leberer, E. (1998). Interaction of a G-protein β-subunit with a conserved sequence in Ste20/PAK family protein kinases. Nature *391*, 191–195.

Leis, J.R., and Kramer, M.A. (1988). The simultaneous solution and sensitivity analysis of systems described by ordinary differential equations. ACM Trans. Math. Softw. *14*, 45–60.

Liu, B. and Faeder, J.R.. (2016), Parameter estimation of rule-based models using statistical model checking, in 2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), IEEE, pp. 1453–1459.

Lopez, C.F., Muhlich, J.L., Bachman, J.A., and Sorger, P.K. (2013). Programming biological models in Python using PySB. Mol. Syst. Biol. *9*, 646.

Manz, B.N., Jackson, B.L., Petit, R.S., Dustin, M.L., and Groves, J. (2011). T-cell triggering thresholds are modulated by the number of antigen within individual T-cell receptor clusters. Proc. Natl. Acad. Sci. U S A *108*, 9089–9094.

MathWorks. (2018). Least-squares (model fitting) algorithms. https://www.mathworks.com/help/optim/ug/least-squares-model-fitting-algorithms.html.

Medley, J.K., Goldberg, A.P., and Karr, J.R. (2016). Guidelines for reproducibly building and simulating systems biology models. IEEE Trans. Biomed. Eng. *63*, 2015–2020.

Medley, J.K., Choi, K., König, M., Smith, L., Gu, S., Hellerstein, J., Sealfon, S.C., and Sauro, H.M. (2018). Tellurium notebooks - an environment for reproducibile dynamical modeling in systems biology. PLoS Comput. Biol. *14*, e1006220.

Mitra, E., and Suderman, R. (2019). PyBioNetFit. https://pybnf.readthedocs.io/en/latest/.

Mitra, E.D., Dias, R., Posner, R.G., and Hlavacek, W.S. (2018). Using both qualitative and quantitative data in parameter identification for systems biology models. Nat. Commun. *9*, 3901.

Monine, M.I., Posner, R.G., Savage, P.B., Faeder, J.R., and Hlavacek, W.S. (2010). Modeling multivalent ligand-receptor interactions with steric constraints on configurations of cell-surface receptor aggregates. Biophys. J. *98*, 48–56.

Moraes, A.O.S., Mitre, J.F., Lage, P.L.C., and Secchi, A.R. (2015). A robust parallel algorithm of the particle swarm optimization method for large dimensional engineering problems. Appl. Math. Model. *39*, 4223–4241.

Mukhopadhyay, H., Cordoba, S.-P., Maini, P.K., van der Merwe, P.A., and Dushek, O. (2013). Systems model of T cell receptor proximal signaling reveals emergent ultrasensitivity. PLoS Comput. Biol. *9*, e1003004.

Nelder, J.A., and Mead, R. (1965). A simplex method for function minimization,. Computer J. *7*, 308–313.

Neri, F., Cotta, C., and Moscato, P. (2012). Handbook of Memetic Algorithms, *Vol. 379* (Springer).

Oguz, C., Laomettachit, T., Chen, K.C., Watson, L.T., Baumann, W.T., and Tyson, J.J. (2013). Optimization and model reduction in the high dimensional parameter space of a budding yeast cell cycle model. BMC Syst. Biol. *7*, 53.

Pargett, M., and Umulis, D.M. (2013). Quantitative model analysis with diverse biological data: applications in developmental pattern formation. Methods *62*, 56–67.

Pargett, M., Rundell, A.E., Buzzard, G.T., and Umulis, D.M. (2014). Model-based analysis for qualitative data: an application in Drosophila germline stem cell regulation. PLoS Comput. Biol. *10*, e1003498.

Penas, D.R., Banga, J.R., González, P., and Doallo, R. (2015). Enhanced parallel differential evolution algorithm for problems in computational systems biology. Appl. Soft Comput. *33*, 86–99.

Penas, D.R., González, P., Egea, J.A., Doallo, R., and Banga, J.R. (2017). Parameter estimation in large-scale systems biology models: a parallel and self-adaptive cooperative strategy. BMC Bioinformatics *18*, 52.

Posner, R.G., Geng, D., Haymore, S., Bogert, J., Pecht, I., Licht, A., and Savage, P.B. (2007). Trivalent antigens for degranulation of mast cells. Organ. Lett. *9*, 3551–3554.

Press, W.H., Teukolsky, S.A., Vetterling, W.T., and Flannery, B.P. (2007). Numerical Recipes 3rd Edition: The Art of Scientific Computing (Cambridge University Press).

Rackauckas, C., Ma, Y., Dixit, V., Guo, X., Innes, M., Revels, J., Nyberg, J., and Ivaturi, V. (2018). A comparison of automatic differentiation and continuous sensitivity analysis for derivatives of differential equation solutions. arXiv, https://arxiv.org/abs/1812.01892.

Raue, A., Schilling, M., Bachmann, J., Matteson, A., Schelke, M., Kaschek, D., Hug, S., Kreutz, C., Harms, B.D., Theis, F.J., et al. (2013). Lessons learned from quantitative dynamical modeling in systems biology. PLoS One *8*, e74335.

Raue, A., Steiert, B., Schelker, M., Kreutz, C., Maiwald, T., Hass, H., Vanlier, J., Tönsing, C., Adlung, L., Engesser, R., et al. (2015). Data2Dynamics: a modeling environment tailored to parameter estimation in dynamical systems,. Bioinformatics *31*, 3558–3560.

Rocklin, M.. (2015), Dask: Parallel computation with blocked algorithms and task scheduling, in Proceedings of the 14th Python in Science Conference, pp. 130–136.

Romano, D., Nguyen, L.K., Matallanas, D., Halasz, M., Doherty, C., Kholodenko, B.N., and Kolch, W. (2014). Protein interaction switches coordinate Raf-1 and MST2/Hippo signalling. Nat. Cell Biol. *16*, 673–684.

Shirin, A., Klickstein, I.S., Feng, S., Lin, Y.T., Hlavacek, W.S., and Sorrentino, F. (2019). Prediction of optimal drug schedules for controlling autophagy. Sci. Rep. *9*, 1428.

Shockley, E.M., Vrugt, J.A., and Lopez, C.F. (2018). PyDREAM: high-dimensional parameter inference for biological models in python. Bioinformatics *34*, 695–697.

Smith, A.E., and Coit, D.W. (1997). Penalty functions. In Handbook of Evolutionary Computation, T. Baeck, D. Fogel, and Z. Michalewicz, eds. (Oxford University Press), pp. C5.2:1–C5.2:6, chapter C5.2.

Sneddon, M.W., Faeder, J.R., and Emonet, T. (2011). Efficient modeling, simulation and coarse-graining of biological complexity with NFsim. Nat. Methods *8*, 177–183.

Somogyi, E.T., Bouteiller, J.-M., Glazier, J.A., König, M., Medley, J.K., Swat, M.H., and Sauro, H.M. (2015). LibRoadRunner: a high performance SBML simulation and analysis library. Bioinformatics *31*, 3315–3321.

Sorokina, O., Sorokin, A., Armstrong, J.D., and Danos, V. (2013). A simulator for spatially extended kappa models. Bioinformatics *29*, 3105–3106.

Spellman, P.T., Sherlock, G., Zhang, M.Q., Iyer, V.R., Anders, K., Eisen, M.B., Brown, P.O., Botstein, D., and Futcher, B. (1998). Comprehensive identification of cell cycle-regulated genes of the yeast Saccharomyces cerevisiae by microarray hybridization. Mol. Biol. Cell *9*, 3273–3297.

Stapor, P., Weindl, D., Ballnus, B., Hug, S., Loos, C., Fiedler, A., Krause, S., Hroß, S., Fröhlich, F., and Hasenauer, J. (2018). PESTO: parameter EStimation TOolbox. Bioinformatics *34*, 705–707.

Storn, R., and Price, K. (1997). Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. J. Glob. Optim. *11*, 341–359.

Suderman, R., and Deeds, E.J. (2013). Machines vs. ensembles: effective MAPK signaling through heterogeneous sets of protein complexes. PLoS Comput. Biol. *9*, e1003278.

Suderman, R. and Hlavacek, W.S.. (2017), TRuML: A translator for rule-based modeling languages, in Proceedings of the 8th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics, Vol. 1, ACM Press, New York, New York, USA, pp. 372–377.

ter Braak, C.J.F., and Vrugt, J.A. (2008). Differential Evolution Markov chain with snooker updater and fewer chains. Stat. Comput. *18*, 435–446.

Thomas, B.R., Chylek, L.A., Colvin, J., Sirimulla, S., Clayton, A.H., Hlavacek, W.S., and Posner, R.G. (2016). BioNetFit: a fitting tool compatible with BioNetGen, NFsim and distributed computing environments. Bioinformatics *32*, 798–800.

Villaverde, A.F., Frölich, F., Weindl, D., Hasenauer, J., and Banga, J.R. (2019). Benchmarking optimization methods for parameter estimation in large kinetic models. Bioinformatics *35*, 830–838.

Waltemath, D., and Wolkenhauer, O. (2016). How modeling standards, software, and initiatives support reproducibility in systems biology and systems medicine. IEEE Trans. Biomed. Eng. *63*, 1999–2006.

Webb, S.D., Sherratt, J.A., and Fish, R.G. (2011). Cells behaving badly: a theoretical model for the Fas/FasL system in tumour immunology. Math. Biosci. *179*, 113–129.

Xu, W., Smith, A.M., Faeder, J.R., and Marai, G.E. (2011). RuleBender: a visual interface for rule-based modeling. Bioinformatics *27*, 1721–1722.

Xue, M., and Del Bigio, M.R. (2000). Intracerebral injection of autologous whole blood in rats: time course of inflammation and cell death. Neurosci. Lett. *283*, 230–232.

Yi, T.-M., Kitano, H., and Simon, M.I. (2003). A quantitative characterization of the yeast heterotrimeric G protein cycle. Proc. Natl. Acad. Sci. U S A *100*, 10764–10769.

Yu, R.C., Pesce, C.G., Colman-Lerner, A., Lok, L., Pincus, D., Serra, E., Holl, M., Benjamin, K., Gordon, A., and Brent, R. (2008). Negative feedback that improves information transmission in yeast signalling. Nature *456*, 755–761.

Zheng, Y., Sweet, S.M.M., Popovic, R., Martinez-Garcia, E., Tipton, J.D., Thomas, P.M., Licht, J.D., and Kelleher, N.L. (2012). Total kinetic analysis reveals how combinatorial methylation patterns are established on lysines 27 and 36 of histone H3. Proc. Natl. Acad. Sci. U S A *109*, 13549–13554.

ISCI, Volume *19*

Supplemental Information

PyBioNetFit and the Biological Property Specification Language

Eshan D. Mitra, Ryan Suderman, Joshua Colvin, Alexander Ionkov, Andrew Hu, Herbert M. Sauro, Richard G. Posner, and William S. Hlavacek

Transparent Methods

Implementation details

PyBioNetFit is written in Python 3.6. The PyBioNetFit package includes novel code, and functionality that is provided through installation of dependencies. The following features are implemented by novel code: all supported optimization and MCMC algorithms, parsing of CONF, EXP, and PROP files, and processing of simulation results, including evaluating a user-selected objective function. Full documentation of the PyBioNetFit code and its features is provided in the user manual (Mitra and Suderman, 2019).

Implementations of algorithms are based on published descriptions. Myriad variants of the algorithms have been described in the literature; here we cite the specific descriptions that we referred to when implementing the algorithms in PyBioNetFit. The iDE algorithm is described in Penas et al. (2015), and our DE and aDE algorithms are based on the simpler algorithm described in the same reference. Our implementation of PSO is based on Moraes et al. (2015). Our implementation of SS follows the outline presented in the introduction of Penas et al. (2017), and uses the recombination method of Egea et al. (2009). Our implementation of MH MCMC is described in Kozer et al. (2013). Our implementation of PT is described in Gupta et al. (2018). Our implementation of SA is analogous to MH, but with a temperature parameter that decreases over the course of the run. Our parallelized implementation of the simplex algorithm is described in Lee and Wiswall (2007). All algorithms are described in full detail in the PyBioNetFit user manual (Mitra and Suderman, 2019).

PyBioNetFit includes four choices for objective functions using quantitative data, which are described in the user manual (Mitra and Suderman, 2019). Briefly, the objective functions all take the form $\sum_i (y_i - \hat{y}_i)^2 / w_i$, where y_i are experimental data points, $\hat{y}_i$ are model outputs, and w_i depends on the choice of objective function (specified in the CONF file). The qualitative objective function uses a static penalty formulation as follows. Each BPSL statement is converted to an inequality $g_i(\hat{\boldsymbol{y}})$ where g_i is a function of the model outputs $\hat{\boldsymbol{y}}$. For example, for the BPSL statement `A<4 always`, the inequality would be $\max(A) - 4 < 0$. The corresponding term in the objective function is $C_i \cdot \max(0, g_i(\hat{\boldsymbol{y}}))$, where C_i is the constraint `weight` specified in BPSL. If both quantitative and qualitative data are used, the contributions from the two datasets are added together, with the choices of C_i determining the relative weighting. A recommended heuristic for choosing the weights is discussed in Mitra et al. (2018). Briefly, in the absence of additional information, one assumes that all constraints should have roughly equal influence on the objective value. In this case the C_i should be chosen to offset differences in the expected scale of constraint violation. For example, a constraint $A > 0.01$ could be assigned a weight 100 times that of a constraint $B > 1$. Additionally, all C_i should be scaled such that both the quantitative and the qualitative data make reasonable contributions to the objective function (i.e., roughly equal contributions if the two datasets are taken to be equally trusted and equally important).

PyBioNetFit interfaces with the dask.distributed (Rocklin, 2018; 2015) package to provide parallelization on multi-core workstations or computer clusters. PyBioNetFit submits simulation jobs to dask.distributed, and dask.distributed assigns those jobs to the available

workers as efficiently as possible. Dask.distributed is automatically installed during installation of PyBioNetFit. The user does not typically need to interact directly with dask.distributed, except when using certain unusual cluster environments (e.g., clusters in which SSH access between nodes is only possible with host-based authentication).

PyBioNetFit uses third-party software to run simulations of models. libRoadRunner (Somogyi et al., 2015) is a Python package used to run SBML models. libRoadRunner is automatically installed during installation of PyBioNetFit and does not require manual configuration. BioNetGen (Harris et al., 2016) is used to run BNGL models; it provides the simulation methods described in Results as BioNetGen ODE, BioNetGen SSA, and NFsim. BioNetGen must be installed manually as a dependency of PyBioNetFit. The path to BioNetGen must be provided to PyBioNetFit using the `bng_command` key in the CONF file or by setting the environment variable `BNGPATH`. Complete installation instructions are provided in the user manual (Mitra and Suderman, 2019). Simulation settings (e.g., integration tolerance) are determined by the simulator used; we used the simulator defaults for all demonstrations. In the case of BioNetGen, these settings can be configured in a BNGL file.

Running benchmark problems

Example fitting problems were run in PyBioNetFit using the model, data and configuration files provided in Supplemental Data, using a variety of computing resources as described in the README files in Supplemental Data. The configuration files include all technical details of the fitting runs such as parameter bounds and algorithmic settings.

Timed benchmark problems (Figure 3) were run on a homogeneous computer cluster, consisting of Intel E5-2695_v4 nodes. Each node had 36 cores, 125 GB RAM, and a base clock rate of 2.10 GHz. The nodes supported multithreading (2 threads per core), but PyBioNetFit was configured such that only one worker process per core was created. To run benchmarks with a specified number of cores, PyBioNetFit v0.2.2 was run with the appropriate number of nodes allocated (benchmarks using 36, 72, 144, and 288 cores were run on allocations of 1, 2, 4, and 8 nodes respectively). For 18-core benchmarks, two PyBioNetFit processes with 18 worker processes each were simultaneously run on the same 36-core node.

Input files required to run the four benchmarks are provided in Supplemental Data (Problems 1-4). The `parallel_count` and `population_size` settings in the provided CONF files were edited to equal the number of available cores for the run. For SS, because the number of parallel simulations for population size n is $n(n-1)$, `population_size` was instead set to 4, 6, 9, 11, and 17 for core counts of 18, 36, 72, 144, and 288 respectively. Note that in general it is possible to set `population_size` to any value regardless of core count. We chose to have `population_size` follow core count in these benchmarks because this is how we would expect a typical user to configure a fitting problem.

For Bayesian uncertainty quantification (Figure 4), we used PyBioNetFit to sample 270,000 parameter sets by MH and 54,000 parameter sets by PT (configuration provided in Supplemental Data (Problem 6)). Note that the PT run used the same total number of simulations, but obtained fewer samples because only the replicas at the lowest temperature are sampled. To compare with the results of Harmon et al. (2017), we used the raw list of

270,000 samples generated in the original study. Raw data from all three runs are available in the BioStudies database (http://www.ebi.ac.uk/biostudies) under accession number S-BSST240. We generated histograms (Figure 4A-F) directly from the lists of sampled parameters. To generate prediction uncertainty plots (Figure 4G-L), we reran simulations using each sampled parameter set, and plotted the median, and the 16th and 84th percentile values, at each time point. We note that the 16th and 84th percentile curves were plotted erroneously in the original study, and are accurate in the present work.

Fitting of the yeast cell cycle control model (Figure 5) consisted of 40 fitting replicates, run on a heterogeneous set of nodes on multiple computer clusters. Each replicate used 2 nodes each with 28 to 44 cores, and completed in under 48 hours. Model, data, and configuration files required to run this problem are provided in Supplemental Data (Problem 8). We configured the fitting job using SS, and kept the same algorithmic settings as the original study with a few exceptions. Whereas the original study performed a single fitting job for 70,000 iterations, we ran an ensemble of 40 replicates of the job for 5,000 iterations each. We also added a short, 500-iteration refinement of the best fit using the simplex local search algorithm. The choice of 40 fitting replicates was arbitrary, but we note that our total number of iterations (200,000 across all replicates) is considerably larger than in the original study. Convergence plots for these runs (Figure S1) show that individual replicates do not make steady improvements after iteration 1000, which suggests that our protocol of using multiple replicates is a reasonable alternative to a single, long fitting run. We also note that taking the best result from among multiple replicates is a standard technique for handling the poorly characterized convergence properties of metaheuristic algorithms.The modified protocol is better able to take advantage of parallelism: with sufficient parallel resources, all replicates can be run in parallel, and the entire fitting job can be completed in about 30 hours, compared to about 10 days in the original study.

To analyze the autophagy model (Figure 6), we implemented the model of Shirin et al. (2019) in SBML format using COPASI (Hoops et al., 2006). We ran PyBioNetFit on the input files provided in Supplemental Data (Problem 9), which includes the SBML model file.

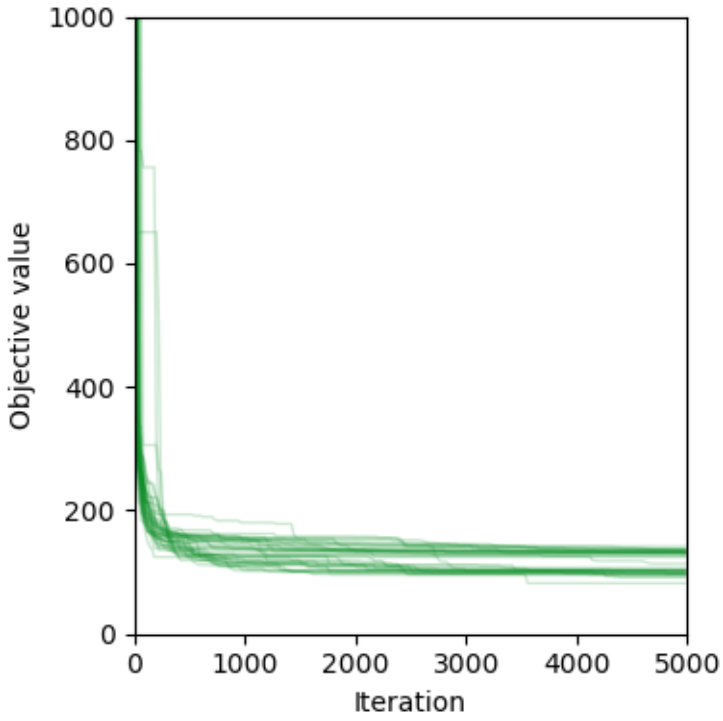


Figure S1. Convergence of a PyBioNetFit job to parameterize a model of yeast cell cycle control, Related to Figure 5. Forty independent replicates were performed of the fitting job provided in Supplemental Data (Problem 8). The minimum objective function value reached is plotted with respect to iteration number for each of the 40 replicates. The overall best fit obtained is shown in Figure 5.

Supplemental References

Egea, J.A., Balsa-Canto, E., García, M.-S.G., Banga, J.R., 2009. Dynamic optimization of nonlinear processes with an enhanced scatter search method. Industrial & Engineering Chemistry Research 48, 4388–4401. https://doi.org/10.1021/ie801717t

Gupta, S., Hainsworth, L., Hogg, J.S., Lee, R.E.C., Faeder, J.R., 2018. Evaluation of parallel tempering to accelerate Bayesian parameter estimation in systems biology, in: 2018 26th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP). pp. 690–697. https://doi.org/10.1109/PDP2018.2018.00114

Harmon, B., Chylek, L.A., Liu, Y., Mitra, E.D., Mahajan, A., Saada, E.A., Schudel, B.R., Holowka, D.A., Baird, B.A., Wilson, B.S., Hlavacek, W.S., Singh, A.K., 2017. Timescale separation of positive and negative signaling creates history-dependent responses to IgE receptor stimulation. Scientific Reports 7, 15586. https://doi.org/10.1038/s41598-017-15568-2

Harris, L.A., Hogg, J.S., Tapia, J.-J., Sekar, J.A.P., Gupta, S., Korsunsky, I., Arora, A., Barua, D., Sheehan, R.P., Faeder, J.R., 2016. BioNetGen 2.2: Advances in rule-based modeling. Bioinformatics 32, 3366–3368. https://doi.org/10.1093/bioinformatics/btw469

Hoops, S., Gauges, R., Lee, C., Pahle, J., Simus, N., Singhal, M., Xu, L., Mendes, P., Kummer, U., 2006. COPASI - A COmplex PAthway SImulator. Bioinformatics 22, 3067–3074. https://doi.org/10.1093/bioinformatics/btl485

Kozer, N., Barua, D., Orchard, S., Nice, E.C., Burgess, A.W., Hlavacek, W.S., Clayton, A.H.A., 2013. Exploring higher-order EGFR oligomerisation and phosphorylation—a combined experimental and theoretical approach. Molecular BioSystems 9, 1849–1863. https://doi.org/10.1039/c3mb70073a

Lee, D., Wiswall, M., 2007. A parallel implementation of the simplex function minimization routine. Computational Economics 30, 171–187. https://doi.org/10.1007/s10614-007-9094-2

Mitra, E., Suderman, R., 2019. PyBioNetFit [WWW Document]. URL https://pybnf.readthedocs.io/en/latest/ (accessed 3.8.19).

Mitra, E.D., Dias, R., Posner, R.G., Hlavacek, W.S., 2018. Using both qualitative and quantitative data in parameter identification for systems biology models. Nature Communications 9, 3901. https://doi.org/10.1038/s41467-018-06439-z

Moraes, A.O.S., Mitre, J.F., Lage, P.L.C., Secchi, A.R., 2015. A robust parallel algorithm of the particle swarm optimization method for large dimensional engineering problems. Applied Mathematical Modelling 39, 4223–4241. https://doi.org/10.1016/j.apm.2014.12.034

Penas, D.R., Banga, J.R., González, P., Doallo, R., 2015. Enhanced parallel differential evolution algorithm for problems in computational systems biology. Applied Soft Computing 33, 86–99. https://doi.org/10.1016/j.asoc.2015.04.025

Penas, D.R., González, P., Egea, J.A., Doallo, R., Banga, J.R., 2017. Parameter estimation in large-scale systems biology models: a parallel and self-adaptive cooperative strategy. BMC Bioinformatics 18, 52. https://doi.org/10.1186/s12859-016-1452-4

Rocklin, M., 2018. Dask.distributed [WWW Document]. URL http://distributed.dask.org/en/latest/ (accessed 11.29.18).

Rocklin, M., 2015. Dask: Parallel computation with blocked algorithms and task scheduling, in: Proceedings of the 14th Python in Science Conference. pp. 130–136.

Shirin, A., Klickstein, I.S., Feng, S., Lin, Y.T., Hlavacek, W.S., Sorrentino, F., 2019. Prediction of optimal drug schedules for controlling autophagy. Scientific Reports 9, 1428. https://doi.org/10.1038/s41598-019-38763-9

Somogyi, E.T., Bouteiller, J.-M., Glazier, J.A., König, M., Medley, J.K., Swat, M.H., Sauro, H.M., 2015. LibRoadRunner: A high performance SBML simulation and analysis library. Bioinformatics 31, 3315–3321. https://doi.org/10.1093/bioinformatics/btv363