The Stochastic System Identification Toolkit (SSIT) to model, fit, predict, and design experiments

Alex N Popinga[1*], Jack Forman[1], Dmitri Svetlov[1], Huy Vo[1], and Brian Munsky[1*],

1 School of Biomedical and Chemical Engineering, Colorado State University, Fort Collins, Colorado, U.S.A.

* alexpopinga@gmail.com, brian.munsky@colostate.edu

Abstract

Biological data is prone to both intrinsic and extrinsic noise and variability between experimental replicas. That same stochasticity and heterogeneity can carry information about underlying biochemical mechanisms but, if not incorporated in modeling and probabilistic inference, can also bias parameter estimates and misguide predictions and, subsequently, experiment design. Mechanistic inference typically requires lengthy simulations (e.g., the Stochastic Simulation Algorithm (SSA)); approximations to chemical master equation (CME) solutions that lack rigorous error tracking; or deterministic averaging that lacks the complexity necessary to reflect the data. We introduce the Stochastic System Identification Toolkit (SSIT) - a fast, flexible, and open-source software package available on GitHub that makes use of MATLAB's efficient and diverse computational architecture. The SSIT is designed for building, simulating, and solving chemical reaction models using ODEs, moments, SSA, Finite State Projection truncations of the CME, or hybrid methods; sensitivity analysis and Fisher information quantification; parameter fitting using likelihood- or Bayesian-based methods; handling of experimental noise and measurement errors using probabilistic distortion operators; and sequential experiment design that empowers users to save time and resources while gaining the most information possible out of their data. The SSIT also offers advanced modeling tools, including model reduction methods for increased efficiency and joint fitting of models and datasets with overlapping reactions/parameters. To facilitate the ease and speed of use, the SSIT provides a graphical user interface and ready-made, adaptable pipelines that can be run in the background from commandline or high-performance computing clusters. We demonstrate features of the SSIT on two experimental datasets: the first consists of published mRNA count data that reflect *Saccharomyces cerevisiae* yeast cell response to osmotic shock using single-cell single-molecule fluorescence in situ hybridization; the second consists of single-cell RNA sequencing measurements of 151 activating genes in breast cancer cells following treatment with dexamethasone.

Author summary

We present the Stochastic System Identification Toolkit (SSIT) to model, fit, and predict any data that can be interpreted as changing populations or counts through time, including but not limited to single-cell experiments, economics, epidemiology, ecology, sociology, agriculture, and biotechnology. The SSIT was constructed particularly for stochastic modeling, which is important for systems whose states may

experience significant fluctuations from mean behavior, thus affecting the inference of the underlying rate parameters and predictions of subsequent behavior. The SSIT provides statistical inference tools for parameter estimation; sensitivity analysis and information calculation; handling of distortions to probability distributions caused by experimental and/or measurement processes (e.g., dropout in single-cell RNA sequence data and total fluorescence intensities versus spot counting/puncta analysis); and quantitative experimental design. The SSIT also offers a variety of complex modeling tools, including model reduction methods and fitting of combined models/datasets that share some behavior but remain distinct (e.g., different genes responding a single stimulus). The SSIT generates pipelines for easy, efficient analyses to run in the MATLAB environment, in the background on commandline, or on high-performance computing clusters, thus facilitating users to make informed, time- and cost-effective decisions about their next set of experiments.

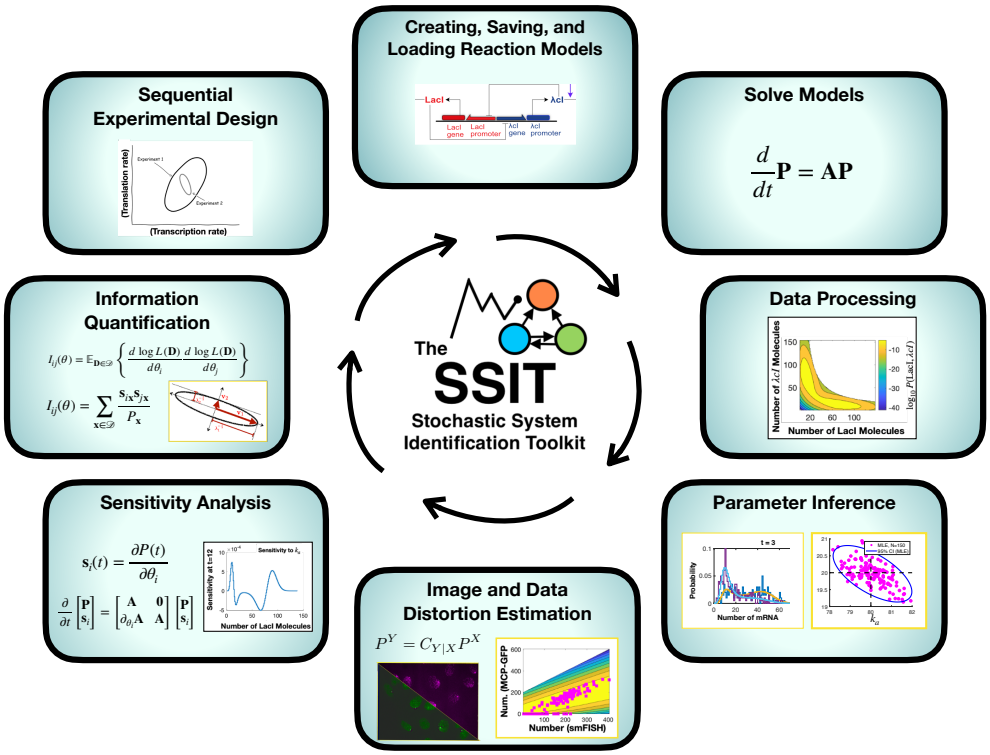

Fig 1. Key features of the Stochastic System Identification Toolkit (SSIT) include (top left and preceding clockwise) creating, saving and loading stochastic models; multiple solution schemes; data processing and parameter inference; image and data distortion estimation; sensitivity analysis; information quantification; sequential experiment design.

1 Introduction

1.1 Noisy, variable data

The Stochastic System Identification Toolkit (SSIT) addresses the need for a comprehensive software toolkit to analyze and predict noisy and heterogeneous data. System that feature small effective population sizes; population structures that may be multi-level or compartmentalized; slow or varying rates of change; or any other characteristic that can cause substantially divergent individual behavior may require a

probabilistic approach to understand and predict its changes through time. Biological systems are especially prone to these effects, particularly at the cellular and molecular levels: gene expression can be intrinsically stochastic, with additional variability arising from cell-to-cell fluctuations in shared factors (often framed as intrinsic vs. extrinsic noise) [1–4]. In contexts such as gene expression, intracellular signaling, and virus–host interactions, the copy numbers of key molecular species are often low (or frequently zero), and the timing of events such as transcription, translation, or degradation depends on factors hidden to the observer [1, 3]. Such variability can be informative about the processes that define the system and ignoring it can lead to biased inference results and predictions. Additionally, when noise is introduced by particular experimental, measurement, or data processing decisions, special care should be applied to understand and contend with any misleading effects on analyses.

1.2 Chemical reaction networks

Any system that can be described by populations that change in size (i.e., counts) in discrete steps that occur at random times can be modeled as a chemical reaction network, which naturally defines a continuous-time Markov jump process on the nonnegative integer lattice. The Chemical Master Equation (CME) is the corresponding forward equation that governs the time evolution of the probability of each count state [5], (see an illustration of the CME in Fig. 2).

Unfortunately, solving the exact CME is intractable for most interesting systems, certainly complex biological ones. The Stochastic Simulation Algorithm (SSA) [6, 7] simulates the same Markov jump process and uses these simulated trajectories to estimate the CME probability distribution by Monte Carlo sampling. However, the SSA is expensive for likelihood-based parameter estimation. Meanwhile, deterministic ODE and moment-based solutions focus on mean concentrations and assume large, well-mixed populations without accounting for random fluctuations/heterogeneity [8].

1.3 The Chemical Master Equation

To compute direct CME solutions, we can truncate the infinite space of all possible states into a subset of states that obtain most of the probability. There are different ways to handle the occurrence of a reaction near the boundary of the truncated state space that would move probability from states within the subset to those outside of the boundary. In some approaches, the boundary of the truncated space *reflects* [9] or *buffers* [10, 11] probability back into the domain when it would otherwise escape, and some approaches such as "slack reactants" *absorb* escaping probability. "Slack reactants" augment the reaction network with "slack" species that have a certain *capacity*; when molecule counts would otherwise exceed a chosen boundary, their "slack" is consumed, thus imposing a conservation measure which forbids the Markov chain from leaving the finite state space [12].

Among the many CME projection schemes, the *sink* state method employed by the Finite State Projection (FSP) algorithm [13, 14] remains one of the most powerful direct CME approximation. The main advantage of the FSP approach is that it offers a principled way to approximate the CME by projecting the infinite-dimensional state space onto a finite subset, enabling computation of the full probability distribution up to an *exact and computable error bound* [13]. In turn, the FSP Algorithm [13, 15] uses multiple absorbing sinks to automatically choose an appropriate subspace according to adjustable error thresholds, keeping track of the total absolute error from the loss of probability to states outside of the subspace (shown in Fig. 2).

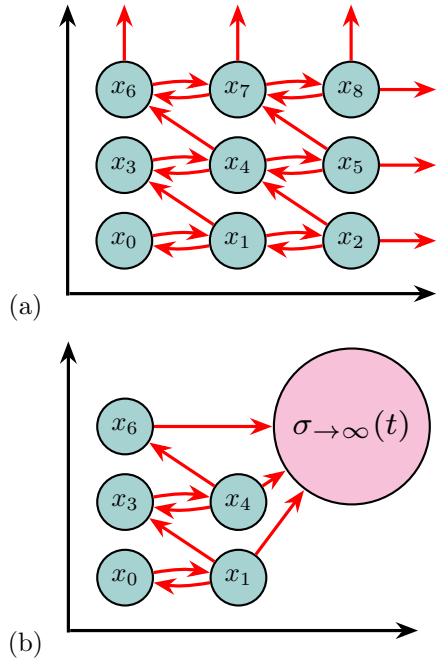

Fig 2. (a) A CME system where x_i are the possible states and each state represents a particular count or counts; (b) an FSP truncation of the CME, i.e., a subset of the full CME for which the probabilities of the system existing in these particular states at specific points in time are explicitly tracked. The probabilities of states outside of the truncated subset are absorbed into a monotonically increasing error sink state $\sigma_{\to\infty}(t)$. The total probability absorbed into $\sigma_{\to\infty}(t)$ is tracked but not the probabilities of individual states.

1.4 Finite State Projection

The efficiency of the FSP and other CME truncations can be further improved by several modifications. For example, *sliding window* procedures avoid the necessity for a global boundary condition by evolving the CME on a moving subset [16, 17]. Subsequent projections based on timescale separations [14], Krylov expansions [18], coarse grained state spaces [19], or principle orthogonal decompositions [20] can also reduce the order of the CME. Tensor train and quantized tensor train methods *compress* both states and operators by representing CME probability vectors and generators as products of "cores", allowing speedier matrix-vector operations [21, 22].

The accuracy and efficiency of the FSP has proven highly useful among computational experts to compare stochastic models to experimental data and make predictions [15, 23–29], yet the approach is often viewed as complicated and has not found wide adoption within the broader community. The practical implementation of stochastic modeling remains a significant barrier for many researchers in computational biology [30–32]. Tools often require specialized knowledge in mathematics or programming, and simulation can be prohibitively slow without optimization or parallelization [33]. Although there are many codes available to conduct ODE and SSA analyses of stochastic reaction networks [33–54], it can be tricky to write computationally efficient implementations of FSP algorithms, and there are far fewer established, readily available packages that yield efficient FSP solutions. Additionally, many existing software packages lack flexibility, transparency, or the ability to scale to complex biological networks [32, 36, 39]. Therefore, there is a growing demand for efficient, accessible, and extensible software frameworks that make stochastic modeling

more approachable, particularly for the rapidly expanding fields of bioengineering, biotechnology, and synthetic biology [32, 55]. Such tools should support model construction, simulation, inference, and visualization in an integrated and user-friendly environment. They should also enable reproducibility and interoperability with existing standards in systems biology (e.g., SBML [56, 57]).

1.5 The SSIT

The SSIT is designed with these considerations in mind and provides fast, transparent solutions for mechanistic models with precise error tracking [13], model fitting to experimental or simulated data with parameter estimation by full statistical inference with uncertainty assessment, parameter sensitivity analysis and Fisher information quantification [27], extrinsic noise handling by probabilistic distortion operators [29], and sequential experimental design recommendations for stochastic systems [58]. For efficient FSP solutions, the SSIT utilizes sparse tensors provided by the Tensor Toolbox [59] and Expokit for the solution of time-invariant master equations [60], along with optional other methods including deterministic ordinary differential equations (ODEs); moment calculations; and SSA [6, 61].

Model solutions can then be used to fit a model or models to experimental data. The SSIT provides tools for full statistical inference, including FSP-based likelihood sweeps, maximum likelihood estimation, and Bayesian uncertainty sampling using Metropolis-Hastings [20, 27]. Approximate Bayesian Computing (ABC) inference schemes [62, 63] are provided to make use of ODE, SSA and moments-based solution schemes. Model selection can be readily computed using Bayes factors [64], and models can be evaluated for how well they predict held-out data using automated cross-validation [65] using the SSIT.

Additionally, the SSIT supplies the machinery for computing the sensitivities of the probabilistic model to small changes in its parameters and Fisher Information Matrices (FIM), which quantifies the amount of information accessible about the values of the model parameters. The SSIT thus allows for ready use of an FSP-FIM [27] pipeline for informed sequential design of experiments [58].

Finally, the SSIT also offers various more complex modeling options. The SSIT provides probabilistic distortion operators (PDOs) to address measurement errors that results from inaccurate experimental approaches [29]; a multimodel function that combines multiple models and/or datasets to jointly estimate parameters from different experimental conditions or different types of data; and various model reduction approaches to increase efficiency, such as hybrid models that describe a subset of species deterministically and projections schemes based on proper orthogonal decomposition [20], eigenvalue decomposition, the quasi-steady-state assumption [14], and finite-element based coarse graining [19, 66].

The SSIT can be run from command line, within the MATLAB environment, and across high performance computing nodes. A graphical user interface is provided for increased accessibility and ease for novice programmers. The SSIT is open source and available at `https://github.com/MunskyGroup/SSIT`, built with future modifications in mind, and supports the rapid construction of complex modeling pipelines that can be launched on modern computational research clusters to quickly and simultaneously fit hundreds of combinations of models and datasets.

The remainder of this article is organized as follows. First, in Section 2.1, we discuss how to create, save, and load models in the SSIT. In Section 2.2, we discuss various methods to solve models (Sec. 2.2.1-2.2.6), how the SSIT can be used to perform sensitivity analysis on model parameters (Sec. 2.2.7), how to compute Fisher information (Sec. 2.2.8), and how to use Fisher information to design subsequent experiments (Sec. 2.2.9). In Section 2.3, we discuss different techniques for fitting

models to data and estimating parameters - including maximum likelihood estimation (Sec. 2.3.3), Bayesian inference (Sec. 2.3.4), Approximate Bayesian Computation (Sec. 2.3.5), and cross-validation (Sec. 2.3.6) - with demonstrations of each feature using a single-cell, single-molecule fluorescence in-situ hybridization (smFISH) experiment in *S. cerevisiae*. Section 2.4 details complex modeling approaches in the SSIT, including model reductions (Sec. 2.4.1), hybrid deterministic-stochastic models (Sec. 2.4.2); probabilistic distortion operators to account for experimental and measurement noise (Sec. 2.4.3); and fitting parameters that are shared across multiple models or datasets (Sec. 2.4.4). The complex modeling methods are illustrated using single-cell RNA sequencing (scRNA-seq) data for 151 genes that are activated in breast cancer cells following the application of a corticosteroid, dexamethasone. Section 2.5 discusses how to execute complex modeling pipelines on a research cluster or local machine. Each of these features is demonstrated through examples and scripts exploring various analyses and parameter estimation for either a four-state model of Mitogen Activated Protein Kinase (MAPK)-induced transcription in yeast or a set of coupled two-state models of glucorticoid receptor (GR)- induced transcription in cancer cells. The key steps taken to create and solve these models are shown in Example Boxes 1 - 23. Detailed example scripts are referred to throughout the text as `example_#_...`) and provided in `https://github.com/MunskyGroup/SSIT/tree/main/Examples`. Finally, in Section 3, we summarize and discuss the contributions, advantages, and limitations of the SSIT in the context of previous methods and software packages.

2 Results

2.1 The SSIT Provides Simple Tools to Create, Save, and Load Discrete Stochastic Models

Models of reaction systems can be used to understand how count data changes through time and to gain insight into the mechanisms and rates that drive those changes. A generalized reaction system contains population types, or count types, referred to as **species** and associated with defined **initial conditions** or initial probability distributions; a set of rules that apply changes to the count data referred to as **reactions**; model **parameters** and **input functions** that define various potentially time-varying rates of the process; a set of vectors encoding the updates to the number of each species for each possible reaction called **stoichiometry vectors**; and equations for stochastic reaction rates to define the probabilities that each reaction will occur in the next infinitesimal time step, called **propensity functions**. As a simple example, Fig. 3 illustrates the states, reactions, stoichiometry vectors and propensity functions for the common "Bursting Gene Model", which consists of a single gene that switches between ON and OFF states to capture the temporal and cell-to-cell heterogeneity often seen in the expression of mRNA or protein molecules in single-cells [67, 68].

A more complex model, along with real experimental data, will be used to demonstrate the utility of the SSIT. We will use the STL1 yeast (*Saccharomyces cerevisiae*) dataset and a model of four distinct gene states (which we will refer to as the "4-state STL1 model") from Neuert *et al.* [24], described and illustrated in Box 1. The data consist of mRNA counts taken at 15 different time points ranging from $t_0 = 0$ to $t_N = 55$(min) following exposure to osmotic shock from the introduction of NaCl.

In the course of this paper, we demonstrate a simple pipeline that loads the dataset into the SSIT; filters the data to focus on desired experimental conditions; associates the filtered dataset with the models we described in Section 2.1 (shown in Fig. 3 and Box 1); performs sensitivity analyses and computation of the Fisher Information; fits the model parameters using maximum likelihood and Metropolis-Hastings; transforms

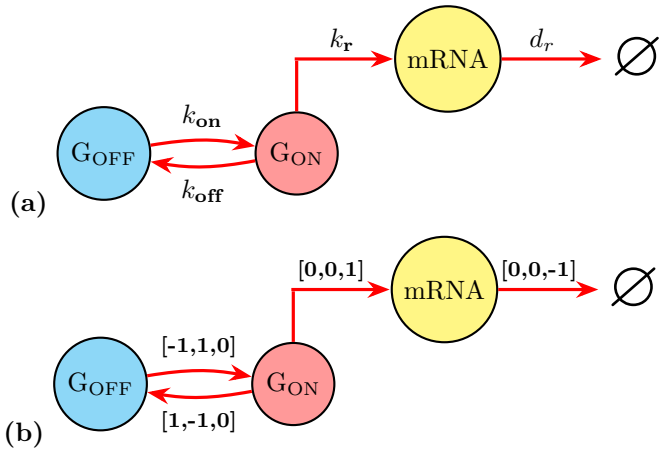

Fig 3. A simple Bursting Gene chemical reaction model. **(a)** There are three **species**: inactive genes (G_{OFF}), active genes (G_{ON}), and messenger RNA ($mRNA$). The set containing the number of each species at a given point in time is the state of the system. There are four **reactions**: G_{OFF} turns to G_{ON}, G_{ON} turns to G_{OFF}, G_{ON} transcribes $mRNA$, and $mRNA$ is degraded (its removal from the system is denoted by the symbol $\emptyset$); and each reaction has a rate **parameter**: k_{on}, k_{off}, k_{r}, g. Which reaction occurs in the next interval of time is determined by the reaction **propensities**: $w_1 \equiv k_{\mathrm{on}}[G_{\mathrm{OFF}}]$, $w_2 \equiv k_{\mathrm{off}}[G_{\mathrm{ON}}]$, $w_3 \equiv k_{\mathrm{r}}[G_{\mathrm{ON}}]$, $w_4 \equiv d_r[mRNA]$. **(b)** Above each reaction arrow are the **stochiometry** vectors for each reaction representing updates to each species count as a result of the reaction occurring.

probability distributions of the model parameters that have been distorted by particular experimental design choices; and finally, selecting the next best set of experiments that retain information, reduce uncertainty, and minimize cost.

Example Box 1: Schematic of 4-state STL1 model

In this paper, we use mRNA count data from smFISH experiments on the time-varying osmotic stress response of *Saccharomyces cerevisiae* collected by Neuert *et al.* (2013) [24] to demonstrate how to define a model; compute ODE, SSA, and FSP solutions; load and associate experimental data to the model; and fit the model to the data. To present these tools, we use a published 4-state model for the osmotic stress response yeast gene STL1 [24] as shown below:

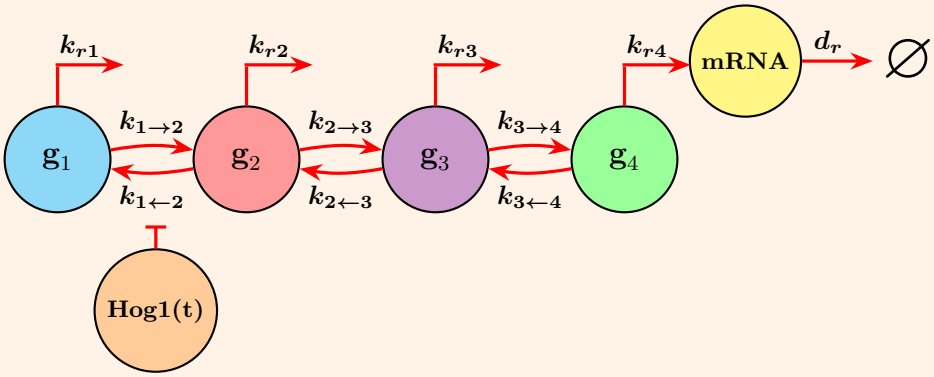

Figure B1. In this expansion of the simple two-state model (Fig. 3), additional states allow for more complex gene activation mechanisms. Transcription occurs in each of the four gene states $\mathbf{g}_1 ... \mathbf{g}_4$ but each transcription reaction has its own unique transcription rate $\mathbf{k}_{r1} ... \mathbf{k}_{r4}$. Upon activation by osmotic stress caused by a step increase in the concentration of NaCl, the transition $\mathbf{g}_2 \to \mathbf{g}_1$ is repressed by the presence of the osmotic shock kinase Hog1, which changes in time according to the function **Hog1(t)**. By preventing the system from returning to state $\mathbf{g}_1$, the kinase Hog1 allows the system to transition toward increasingly active states.

To generate a model in the SSIT, first ensure the source code (found in the folder 'src') is added to the MATLAB path. This can be done by running the command "install" in MATLAB when in the main SSIT directory. Then, one can either load a pre-built model that has been saved in the SSIT (e.g., 'BirthDeath', 'BurstingGene', 'ToggleSwitch', 'Repressilator'), or create a model from scratch. In this paper, we will focus on creating the 4-state STL1 model (Box 1) from scratch (Box 2), but both options are shown in `example_1_CreateSSITModels` at `https://github.com/MunskyGroup/SSIT/tree/main/Examples`.

Once an instance of the SSIT has been created and the model has been named, the following subsections may completed in any order, with their accompanying example boxes: 2.1.1 Define Model Species, 2.1.2 Define a time-varying input signal, 2.1.3 'Define Reactions', and 2.1.4 'Define Parameters (Reaction Rates)'. The subsections 2.1.5 'Print Model Summary' and 2.1.6 'Save and Load a Model' are also useful, for, respectively, checking that a model has been implemented as expected and for ensuring that model data is accessible later and/or on another machine.

Example Box 2: Add Path and Create Model

By running the installation function `install.m` after cloning the SSIT, the search path should be accurate. Alternatively, ensure that the SSIT source codes are on the search path:

```
addpath(genpath('../src'));
```

Then, instantiate the SSIT and name the model. Following the 4-state STL1 model described in Box 1, we will name our model 'STL1_4state':

```
STL1_4state = SSIT;
```

2.1.1 Define Model Species

Specify a name for each model species. The model 'species' are the different population types in the system, e.g., different molecules. Refer to Box 3 for an example.

Example Box 3: Define Species

The species of our 4-state STL1 model are: $\boldsymbol{x(t)} = [\mathbf{g_1} \quad \mathbf{g_2} \quad \mathbf{g_3} \quad \mathbf{g_4} \quad \mathbf{mRNA}]^\intercal$, which in the SSIT can be described by specifying:

```
STL1_4state.species = {'g1'; 'g2'; 'g3'; 'g4'; 'mRNA'};
```

2.1.2 Define a time-varying input signal

The SSIT has an optional property called 'inputExpressions' for cases where the system is affected by a time-varying input of some sort (e.g., delivery of a drug, introduction of a chemical via an osmotic source, etc.) An input expression contains rate parameters and relates inputs to particular time points - it may represent a chemical decay process, etc. The 4-state STL1 model has a complex, experimentally-derived signal which is shown in Box 4.

Example Box 4: Specify Complex Input Expressions

The complex input expression that best fits the STL1 mRNA count data in our 4-state model (Box 1) is:

$$\text{Hog1}(t) = A\left(\frac{\left(1-e^{-r_1(t-t_0)}\right)e^{-r_2(t-t_0)}}{1+\dfrac{\left(1-e^{-r_1(t-t_0)}\right)e^{-r_2(t-t_0)}}{M}}\right)^n, t > t_0. \tag{1}$$

This expression was determined experimentally by Neuert *et al.* [24] and includes an inferrable lag in the parameter t_0. In the SSIT, it is written as:

```
STL1_4state.inputExpressions = {'Hog1',['A*(((1-(exp(1)^(-r1*(t-t0))))*',...
        'exp(1)^(-r2*(t-t0)))/(1+((1-(exp(1)^(-r1*(t-t0))))*',...
        'exp(1)^(-r2*(t-t0)))/M))^n*(t>t0)']};
```

2.1.3 Define Reactions

In the SSIT, reactions are interpreted from defined propensity functions and the corresponding stoichiometry matrix. Reactions can either be added one at a time or in batches as a paired set of propensity vector and stoichiometry matrix. The order of the listed propensity functions matters, as it corresponds to the order of the *columns* of the stoichiometry matrix. Likewise, the order of model species listed (2.1.1) corresponds to the order of the *rows* of the stoichiometry matrix. Stoichiometry matrices define updates to species population vectors that are applied when a reaction occurs.

Box 5 shows an example for defining the reactions of the 4-state STL1 model from Box 1.

Example Box 5: Define Propensity Functions and Stoichiometry Matrix

Propensity Functions

As there are 11 reactions for our 4-state STL1 model (Box 1), there must be eight propensity functions given in the SSIT. The propensities for our model are:

```
STL1_4state.propensityFunctions = {'k12*g1';...
    '(max(0,k21o*(1-k21i*Hog1)))*g2'; 'k23*g2'; 'k32*g3'; 'k34*g3';...
    'k43*g4'; 'kr1*g1'; 'kr2*g2'; 'kr3*g3'; 'kr4*g4'; 'dr*mRNA'};
```

Note: Propensity functions are compiled into symbolic expressions and saved in separate files, which the SSIT calls when needed. The function called `formPropensitiesGeneral` allows the user to specify the prefix used for saving these propensity function files, which can be very useful for telling the SSIT which files to call when solving a particular model and avoiding errors arising from the SSIT trying to use files that contain propensity functions from an incompatible model. An example of `formPropensitiesGeneral` for our 4-state STL1 model is shown below:

```
STL1_4state = STL1_4state.formPropensitiesGeneral('STL1_4state');
```

Stoichiometry Matrix

There are five species and 11 reactions for our 4-state STL1 model (Box 1). (Note: 'Hog1' is not considered a model species but an input expression, Box 4.) The stoichiometry matrix in the SSIT assumes model species by row and reactions by column. The stoichiometry matrix for our model can be entered into the SSIT as:

```
STL1_4state.stoichiometry = [-1,1,0,0,0,0,0,0,0,0,0;...   % gene state 1
                              1,-1,-1,1,0,0,0,0,0,0,0;... % gene state 2
                              0,0,1,-1,-1,1,0,0,0,0,0;... % gene state 3
                              0,0,0,0,1,-1,0,0,0,0,0;...  % gene state 4
                              0,0,0,0,0,0,1,1,1,1,-1]     % mRNA
                  % Reactions: 1,2,3,4,5,6,7,8,9,10,11
```

Another way to add reactions to the SSIT is by using the 'addReaction' method. If we wanted to add another 'mRNA' degradation reaction, we could do it by:

```
newReaction.propensity = 'dr1*mRNA';
newReaction.stoichiometry = {'mRNA',-1};
newReaction.parameters = {'dr1',1};
STL1_4state = STL1_4state.addReaction(newReaction);
```

2.1.4 Define Parameters (Reaction Rates)

The reaction rates are the parameters of the model that drive the dynamics of the system and are incorporated in the propensity functions (2.1.3). Box 6 illustrates how to set model parameters, by supplying both a name for each parameter and its value (the units of which are inferred). Parameter values may be later updated by statistical inference during model fitting (Section 2.3). Note: The model parameters do not need to be listed in a particular order since each parameter is given a unique name.

Example Box 6: Specify Model Parameters

There are 18 parameters for our 4-state STL1 model, one parameter per each of the 11 reactions in Box 1, with the exception of the reaction $\mathbf{g_2} \rightarrow \mathbf{g_1}$. The transition from state $\mathbf{g_2}$ to $\mathbf{g_1}$ has a more complex propensity function '(max(0,k21o*(1-k21i*Hog1)))*$\mathbf{g_2}$' that splits the simple '$k_{2\rightarrow 1}$' parameter from Box 1 into two parameters, 'k_{21o}' and 'k_{21i}'. Additionally, the input 'Hog1' decays as a function of time according to our input expression, which requires five additional parameters. (These five parameters - 'r1', 'r2', 'A', 'M', and 'n' - are experimentally determined and will thus be fixed for model fitting, i.e., parameter estimation.) The 4-state STL1 parameters can thus be specified:

```
STL1_4state.parameters = ({'t0',3.17; 'k12',78; 'k21o',1.92e+05;...
    'k21i',3200; 'k23',0.402; 'k34',7.8; 'k32',1.62; 'k43',2.28;...
    'dr',0.294; 'kr1',4.68e-02; 'kr2',0.72; 'kr3', 59.4; 'kr4', 3.24;...
    'r1',4.14e-03; 'r2',0.426; 'A',9.3e+09; 'M',6.4e-04; 'n',3.1});
```

To change the value of a parameter that has already been specified, one can simply refer to it by its saved parameter name using 'changeParameter', such as:

```
STL1_4state = STL1_4state.changeParameter({'t0',10; 'dr',0.1})
```

2.1.5 Print Model Summary

It can be helpful to print a summary of an SSIT model to the screen to check for errors before proceeding to analysis. The function 'Model.summarizeModel' (see Box 7) will print model species, reactions, input signals, and model parameters to screen.

Example Box 7: Print Model Summary

The following command will print a model summary for our 4-state STL1 model (Box 1), which is shown in the Supporting Information:

```
STL1_4state.summarizeModel
```

2.1.6 Save and Load a Model

To save a model, along with any associated data and/or solutions, use the 'save' function with a named .mat file and the name of the model. To load a previously saved model, use the 'load' function with the name of the .mat file and the name of the desired model. Both functions are shown in Box 8.

Example Box 8: Save and Load Models

Below, we save the model 'STL1_4state' to a file called `example_1_CreateSSITModels`, which will automatically assume the file extension '.mat', and then load it by name.

```
save('example_1_CreateSSITModels','STL1_4state')
load('example_1_CreateSSITModels.mat')
```

2.1.7 Compatibility with SBML and SimBiology

The SSIT has been designed for compatability with the Systems Biology Markup Language (SBML) as well as SimBiology. SBML is an open and collaborative XML-based data format for implementing and sharing computational biology systems. SBML offers a standardized structure for the representation of biological models, especially biochemical reaction networks such as gene expression, metabolic pathways, cell signaling, and epidemics. As with the SSIT, SBML models are defined by their model elements, most commonly the model species, reactions, compartments, parameters, and reaction rules that are defined by stoichiometries and propensities. [56] [57] SimBiology is a MATLAB-based MathWorks toolbox for building, simulating, and analyzing quantitative dynamic models, primarily for applications in pharmacokinetics / pharmacodynamics (PK/PD) systems, by providing for modeling processes such as drug metabolism, efficacy, and safety; signaling pathways; gene expression; and optimization of dosing schedules using ODEs, stochastic simulations, global and local parameter sensitivity analysis, and compartmental modeling. SimBiology is also compatible with SBML, utilizing the standardized format to facilitate collaboration and integration with other modeling platforms. [57] [50]

The Supporting Information provides an example for the simple Bursting Gene model example shown in Fig. 3 and Section 2.1: 'Creating, Saving, and Loading Models in the SSIT', in SBML format. Detailed examples for loading SBML and SimBiology models into the SSIT to make use of the SSIT machinery for fast FSP solutions, sequential experiment design, etc., are provided in the Supporting Information.

2.2 The SSIT provides numerous tools to find and and visualize master equation solutions

Once the model reactions have been specified, one can then use a variety of methods to solve for population sizes over time. These solutions can then be used to estimate reaction rate parameters from experimental data.

The SSIT allows users to compute deterministic ordinary differential equations (ODEs); closed moment solutions; Stochastic Simulation Algorithm (SSA) trajectories; efficient solutions to the Finite State Projection (FSP) approximation of the full chemical master equation (CME); and hybrid deterministic-stochastic solutions.

Box 9 shows two arguments for setting up for any type of model solution.

Example Box 9: Set up for Solutions

Solutions will be evaluated at specific time intervals. The SSIT has default time points for computing solutions given by 'linspace(0,10,21)', but the property 'tSpan' can be used to set alternative times:

```
STL1_4state.tSpan = linspace(0,50,101);
```

(The 'linspace' function returns a linearly spaced vector such that, for the above example, 101 time points are generated with even spacing between 0 and 50.)

Additionally, we specify the initial condition, i.e., counts for each model species at the start of the simulation or solution. The SSIT default initial condition of '[0]' would only work for a single-species model, and even then may or may not be appropriate depending on the model. For our 4-state STL1 model (Box 1), each of the five species are given an initial count:

```
STL1_4state.initialCondition = [1;0;0;0;0];
```

The SSIT allows for the initial condition to be probabilistic by specifying the 'initialCondition' property as a set of possible states and the 'initialProbs' property as a vector of the corresponding probabilities. For FSP implementations, the initial condition can also be set as the steady state probability distribution assuming rate constants that are fixed at a user-specified time.

2.2.1 The Chemical Master Equation (CME)

Discrete state stochastic processes describe a system that evolves through time with some degree of randomness, as its transitions between countable states proceed in time under the governance of probabilistic rules [69, 70]. These processes are characterized by a sequence of random variables, typically indexed by integers, where each variable represents the state of the system at a particular time [69]. Discrete state stochastic processes are common models for systems whose states, or counts, are updated according to probabilistic rules at distinct fixed or random time intervals [7, 8, 69].

The Chemical Master Equation (CME), also known as the Forward Kolmogorov equation, is a fundamental approach to describe the evolution of these discrete state stochastic processes by computing the probabilities of each distinct system state over time [5]. In this representation, states are defined as the discrete set of all possible configurations that the system can take over time. For example, a subset of the possible states for the Bursting Gene Model from Fig. 3 is shown in Fig. 4. In that example, each state is defined by three numbers corresponding to the amounts of 'G_{OFF}', 'G_{OFF}', and 'mRNA' molecules. The transitions between states are determined by stoichiometric updates. As there is only one gene that is being turned off ('G_{OFF}') or on ('G_{ON}') in our model, there is a natural bound on the first two species of the model so that the first two numbers in each state may only be '0' or '1'. However, the third number that provides a value for the number of mRNA in a cell is unbounded in the positive direction.

CME solutions for reaction systems have been used to understand the dynamics of a wide variety of systems with stochastic natures, particularly useful within confined environments such as cells and nanopores [71, 72]. However, as noted for the 'mRNA' copy number in the Bursting Gene model, the system often has a theoretically infinite number of possible states and thus requires a decision as to how the system states may be bounded into a manageable subset without incurring unreasonable approximation errors. Regardless, there are multiple differing tactics of investigating the properties and

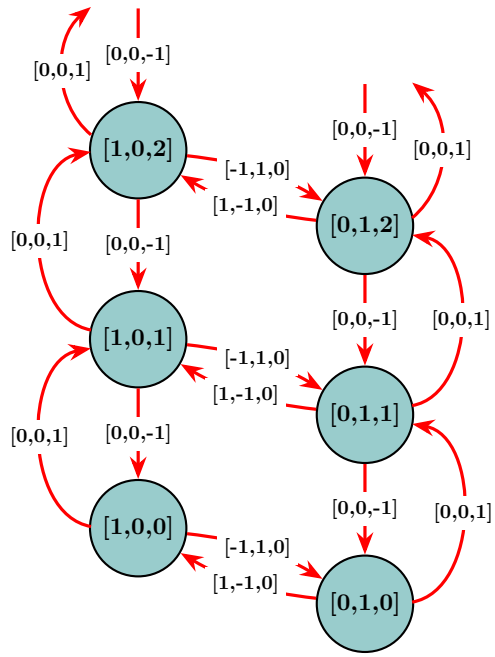

Fig 4. The Bursting Gene Model from Fig. 3 represented as a discrete state Markov model. The model has three species: a gene in its inactive form ("G_{OFF}"), the activated gene ("G_{ON}"), and mRNA. For this simple model, the gene only switches states between "G_{OFF}" and ("G_{ON}"), but there can, in theory, be any positive number of mRNA molecules.

responses of reaction networks governed by the CME, from simulation to approximation.

Master equations characterize the changing probabilities of these states dynamic networks through time and are useful to (a) probe the probability of a combination of states at any given time and (b) infer rates at which the system transitions between states [73, 74]. In the Chemical Master Equations (CME) [6, 61], or forward Kolmogorov equations [75, 76], these transition rates represent the rates of reactions or interactions between molecular species - e.g., rates at which a gene is expressed or an mRNA codon translated - and the states represent the count of each type of molecule at a particular time in the stochastic evolution of the system [5, 8]. CMEs capture the intrinsic heterogeneity, noise and variability found in cellular processes such as transcription and translation and are uniquely disposed to identify the emergences of sparse sub-populations as they are highly stochastic in nature [77, 78].

The CME is composed of a set of linear ordinary differential equations where the probability mass $p(\mathbf{x}_i, t)$ is described through time t for all states of the system $\mathbf{x}_i$ [79]. A reaction $\mu \in \{1, ..., R\}$ occurs at time t probabilistically according to the reaction propensities, or rates, determined by function $w_\mu(t, \mathbf{x}, \boldsymbol{\theta})$, with model parameters $\boldsymbol{\theta} = [\theta_1, \theta_2, ..., \theta_n] \in \mathbb{R}_{\geq 0}$. A reaction in the CME is treated as a Markovian jump (illustrated in Fig. 4 for a 3-species model) that moves the system between the two different states. The states $\mathbf{x} \in \mathbb{X}$ in the CME represent the counts of each molecule (species) type, so for a single-species model $\mathbf{x} \in \mathbb{X}$ is represented as a scalar, where $\mathbb{X} \subseteq \mathbb{N}_{\geq 0}$ and for multi-species models with d species, $\mathbf{x}$ is a d-element vector and $\mathbb{X} \subseteq \mathbb{N}^d_{\geq 0}$. The new state at time $t + \epsilon$, to which the system moves when a reaction occurs, is updated according to

$$\mathbf{x}(t + \epsilon) = \mathbf{x}(t) + \boldsymbol{\nu}_\mu, \tag{2}$$

where $\boldsymbol{\nu}_\mu$ is the stoichiometric vector coinciding with reaction $\mu \in [0, 1, \ldots, M]$. [29] The probability mass $p(\mathbf{x}_i, t)$ is thus updated for each interval of time such that:

$$\frac{d}{dt}p(\mathbf{x}_i, t) = \underbrace{-\sum_{\mu=1}^{M} p(\mathbf{x}_i, t) w_\mu(\mathbf{x}_i, t)}_{\text{Transitions } \textit{from} \text{ state } \mathbf{x}_i} + \underbrace{\sum_{\mu=1}^{M} p(\mathbf{x}_i - \boldsymbol{\nu}_\mu, t) w_\mu(\mathbf{x}_i - \boldsymbol{\nu}_\mu, t)}_{\text{Transitions } \textit{to} \text{ state } \mathbf{x}_i}. \tag{3}$$

The probability mass vector for the whole state space of the CME can be written as

$$\mathbf{p} \equiv [p(\mathbf{x}_1), p(\mathbf{x}_2), ...]^{\mathsf{T}} \tag{4}$$

[79], and an *infinitesimal generator matrix* $\mathbf{A}$ can be constructed to capture the propensity sums, where $A_{i,j}$ denotes the (i, j)th scalar entry of $\mathbf{A}$ such that

$$A_{ij} = \left\{ \begin{array}{cl} -\sum_{\mu=1}^{M} w_\mu(\mathbf{x}_j) & \text{for } i = j, \\ w_\mu(\mathbf{x}_j) & \text{for } (i, j) \text{ such that } \mathbf{x}_i = \mathbf{x}_j + \boldsymbol{\nu}_\mu, \\ 0 & \text{otherwise} \end{array} \right\}. \tag{5}$$

Therefore, the CME of Equation 3 can be reformulated more concisely as

$$\frac{d}{dt}\mathbf{p}(t) = \mathbf{A}\mathbf{p}(t). \tag{6}$$

The CME is a useful method for probing stochastic discrete state time evolutions. However, solving a full CME analytically is only possible for simple, often uninteresting, systems, and the combinatorial explosion of possible states in more complex, realistic models renders exact solutions infeasible.

2.2.2 Ordinary Differential Equations (ODEs)

Ordinary differential equations (ODEs) may be used for species whose populations are large enough that the changes in their sizes may be approximated by averaging over time; the sums of their changes from their initial states are sufficient due to a lack of stochasticity [80–82].

The Chemical Master Equation (CME) solution can be approximated using first-order ODEs to estimate the expected changes to all species in the state vector, i.e., $\mathbb{E}\{\mathbf{x}\}$ with respect to time such that:

$$\frac{d\mathbb{E}\{\mathbf{x}\}}{dt} = \mathbf{S}\mathbb{E}\{w(\mathbf{x})\} \approxeq \mathbf{S}w(\mathbb{E}\{\mathbf{x}\}), \tag{7}$$

where $\mathbf{S}$ represents the stoichiometry matrix and $w(\mathbf{x}) = [w_1(\mathbf{x}, t), \ldots, w_N(\mathbf{x}, t)]^{\mathrm{T}}$ is a vector containing propensities for the potential reactions of the system [81, 82]. This approximation is exact in the case where $w(\mathbf{x})$ is constant or linear in $\mathbf{x}$, such that $\mathbb{E}\{w(\mathbf{x})\} = w(\mathbb{E}\{\mathbf{x}\})$. The matrix-vector product $\mathbf{S}w$ sums over all changes to $\mathbf{x}$ that may occur according to all possible reactions (i.e., summing over each reaction's propensity and stoichiometric update) with respect to t. The Supporting Information shows the ODEs in component form for the simple Bursting Gene model (Fig. 3) and 4-state STL1 model (Box 1).

An example for solving the 4-state STL1 ODEs and plotting their solutions is provided in Box 10 and in more detail in `example_2_SolveSSITModels_ODE.m`, which also contains examples for the simple Bursting Gene model (Fig. 3) and a simplified STL1 model for comparison.

Note: An ODE solver is used to solve the system of ODEs. MATLAB provides a suite of precise, fast ODE solvers for both stiff and non-stiff solutions, e.g., ode23s and ode45 [83], which are readily employable in the SSIT by user specification. When propensity functions are defined as continuous and differentiable functions, the SSIT symbolically computes the Jacobians for use in ODE solvers; this allow for significant computational savings for numerically stiff systems.

Example Box 10: Deterministic Solutions (ODEs)

We demonstrate how to solve ODEs for the 4-state STL1 model (see Box 1) in the SSIT below:

```
% Set solution scheme to ODEs:
STL1_4state.solutionScheme = 'ODE';

% Solve ODEs:
STL1_4state.Solutions = STL1_4state.solve;

% Plot ODE solutions for mRNA:
STL1_4state.plotODE(STL1_4state.species(5), STL1_4state.tSpan, {'linewidth',4},...
    TitleFontSize=26, Title='4-state STL1 (mRNA)', AxisLabelSize=20,...
    TickLabelSize=20, LegendFontSize=20, LegendLocation='east',...
    Colors=[0.23,0.67,0.20], XLabel='Time', YLabel='Molecule Count')

% Plot ODE solutions for the four gene states:
STL1_4state.plotODE(STL1_4state.species(1:4), STL1_4state.tSpan, {'linewidth',4},...
    TitleFontSize=26, Title='4-state STL1 (gene states)', AxisLabelSize=20,...
    TickLabelSize=20, LegendFontSize=20, LegendLocation='east',...
    XLabel='Time', YLabel='Molecule Count')
```

The ODE solutions for the five species in the 4-state STL1 model are shown below. Note: We plot the full time interval $t \in [0,50]$ and acknowledge that the dynamics for $t \leq 1$ are dominated by a stiff fast mode that quickly decays, after which the slower gene-state dynamics of interest become clearly visible.

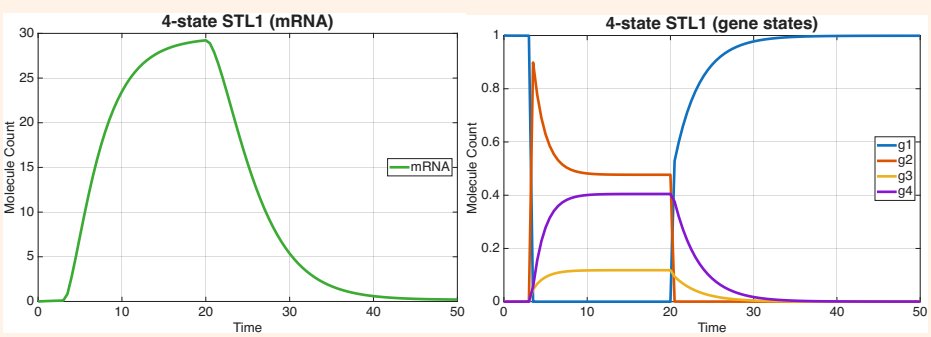

Figure B10. ODE solution plots for the 4-state STL1 model from Box 1 showing the average number of mRNA (left) and the fraction of cells in the $\mathbf{g}_1 - \mathbf{g}_4$ states (right).

2.2.3 Moment Closure

Moment-based approaches are efficient ways approximate the time evolution of the system using low-order statistics by deriving exact or approximately closed moment solutions from the Chemical Master Equation (CME), assuming functional dependence between all cumulants of the probability distributions (the mean, variance, and so forth). For linear propensities, the moment closure is exact, and for non-linear reaction

networks, the closure bases the approximation on a proposed distribution or generating function (e.g., Gaussian, log-normal, cumulant, or entropy-based), procuring solvable ODE systems [84]. The closure thereby posits a parametric form or constraint in place of the unknown higher moments. However, closures rely on assumptions [85] that may be inappropriate in some cases and lead to inaccurate predictions, e.g., when Gaussian assumptions are applied to distributions that are skewed, multimodal, or heavy-tailed [26, 27].

The SSIT offers the ability to compute simple closure approximations. For example, if the joint state distribution can reasonably be assumed as multivariate normal, then the CME might be approximated using Gaussian closures by calculating all third and higher-order moments using the means, covariances, and the assumption of a multivariate normal distribution. An example for computing moment solutions from our 4-state STL1 model (Box 1) is provided in the Supporting Information and `example_18_Moments.m`.

2.2.4 Stochastic Simulation Algorithm (SSA)

The Stochastic Simulation Algorithm (SSA) [6, 7] is one of the most common methods for exploring the dynamics of reaction networks, computing relevant statistics, and validating mechanistic models. The SSA simulates exact sample paths through reaction networks, probabilistically selecting reactions to occur in a given time interval based on propensities and applying stoichiometric updates to the system states. The SSA is simple to implement and, unlike methods which use ODEs to average over population changes with respect to time, the SSA provides Monte-Carlo estimates and can yield important insights regarding low-probability events and heterogeneous behavior. Unfortunately, to resolve these same desirable results, Monte-Carlo sampling can require high numbers of trajectories, making the SSA approach computationally impractical for some systems.

While efficient, continuous, and deterministic solutions for computing system states as a function of time may be inappropriate when the system is built on small, discrete populations whose population sizes change dynamically through time and can thus exhibit highly variable behavior - as is the case at the level of gene expression and cell signaling, for examples, where the numbers of species in the model may be as low as tens of mRNA or proteins [1, 3, 4]. In such cases, the stochastic fluctuations of the model species populations may become significant and unpredictable by deterministic, averaging models [77, 82].

A collection of many SSA trajectories (details about generating SSA trajectories are provided in the Supporting Information) may be computed by simulating the procedure with newly generated random numbers for each new trajectory, and subsequent statistical analyses may be performed on these trajectories, including computing the likelihood of the data.

The SSA is precise in that it tracks every individual reaction over time and updates the resultant states accordingly [86], but as such it does not scale well in efficiency when required to track large numbers of reactions. Time-leaping methods such as the τ-leaping method [86, 87] approximate the solution by incrementing a specified interval of time and grouping reaction events within the time intervals. Naturally, these methods are more efficient but risk missing key reactions and result in a substantial loss of accuracy for systems containing low species copy numbers or propensities that lead to rapid, large changes of species counts.

The SSIT solver for the SSA supports parallelization and GPU use on capable devices. The SSIT also makes use of a modified version of the Extrande [88] method to handle time-varying reaction rates in the system propensity functions. Box 11 and

`example_3_SolveSSITModels_SSA.m` illustrate how to use the SSIT to simulate SSA trajectories.

Example Box 11: Generate Stochastic Trajectories with the SSA

Here, we demonstrate how to use SSA in the SSIT for the 4-state STL1 model (Box 1). Plots of the SSA results are shown in Fig B11.

```
% Set solution scheme to SSA:
STL1_4state.solutionScheme = 'SSA';

% Set the number of simulations performed per experiment:
STL1_4state.ssaOptions.nSimsPerExpt=100;

% Use a negative initial time to allow model to equilibrate to an initial
% steady state prior to starting the subsequent simulation (burn-in):
STL1_4state.tSpan = [-100,STL1_4state.tSpan];

% Set the initial time:
STL1_4state.initialTime = STL1_4state.tSpan(1);

% Run iterations in parallel with multiple cores, or execute serially:
STL1_4state.ssaOptions.useParallel = true;

% Run SSA:
STL1_4state.Solutions = STL1_4state.solve;
```

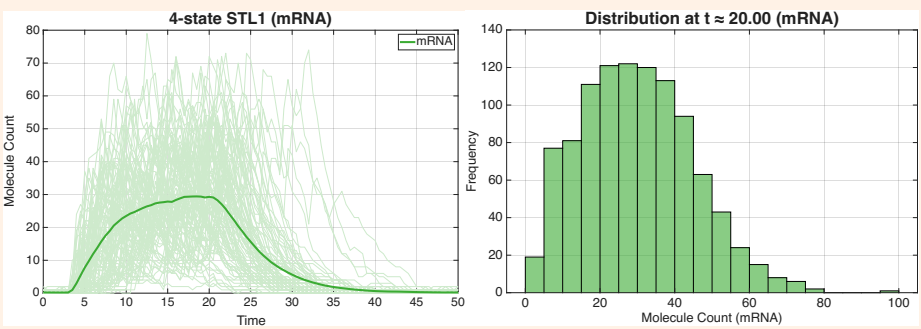

Figure B11. SSA means and trajectories for the 5 species of the 4-state STL1 model from Box 1 (left) and means only, zoomed in on the 4 gene states (right).

2.2.5 Finite State Projection (FSP)

The Finite State Projection (FSP) algorithm [13] yields an exact analytical solution in the case of finite distinct population vectors and closely approximates a reduced, finite space of an otherwise possibly infinite set of states while simultaneously calculating the error of its approximation. The FSP is precise and fast under the SSIT implementation in MATLAB, and thus has proven highly valuable in the solving of CMEs for single-cell data [15, 25, 27, 68, 79].

The FSP algorithm divides the total states of the CME into two categories: a set of states determined to be most realistically (most probable to be) visited over the time interval of interest, for which the FSP algorithm provides solutions, and one or more absorbing, sink states that represents error (see the state labeled "$\sigma_{\to\infty}(t)$)" in Fig. 2). Probability flows between the states of interest and into the error sinks, from which it does not return. The FSP then solves a subset of linear ordinary differential equations that provide the solutions to the evolution of the probability density vectors relevant to the FSP states, a subset of the whole CME. In doing so, FSP provides efficient results

and a lower bound to the exact solution [13, 58].

Formally, FSP solutions are computed by first defining all enumerations over all possible states reachable by the system as: $\{\mathbf{x}_1, \mathbf{x}_2, \ldots\} \subseteq \mathbb{X}$. Next, a finite set of indices $J = \{j_1, j_2, ..., j_N\}$ are specified and used to determine a finite set of states $\mathbb{X}_J = \{\mathbf{x}_{j_1}, \ldots, \mathbf{x}_{j_N}\} \subseteq \mathbb{X}$. Naturally, the complement set $\mathbb{X}_{J'}$ contains all of the states that were *not* included in $\mathbb{X}_J$. The full CME can then be rearranged to write as:

$$\frac{d}{dt}\begin{bmatrix} P_J(t) \\ P_{J'}(t) \end{bmatrix} = \begin{bmatrix} A_{JJ} & A_{JJ'} \\ A_{J'J} & A_{J'J'} \end{bmatrix} \begin{bmatrix} P_J(t) \\ P_{J'}(t) \end{bmatrix}. \tag{8}$$

The most defining feature of the FSP approach is that the second set of states is replaced by one or more absorbing sinks that add to form $z(t) = \left|P_J - P_J^{\text{FSP}}\right|_1 + |P'_J|_1$, into which the probabilities of system states flow but are no longer individually tracked and therefore are lost from the finite state set $\mathbb{X}_J$ [79]. The total probability mass that ends up in the subset of states $z(t)$ is accounted for as the total error - loss of probability mass as a function of time from the finite state truncation $\mathbb{X}_J$ of the CME - and this formalization gives rise to the finite dimensional master equation:

$$\frac{d}{dt}\begin{bmatrix} P_J^{\text{FSP}}(t) \\ z(t) \end{bmatrix} = \begin{bmatrix} A_{JJ} & 0 \\ -1^T A_{JJ} & 0 \end{bmatrix} \begin{bmatrix} P_J^{\text{FSP}}(t) \\ z(t) \end{bmatrix}. \tag{9}$$

The solutions of the FSP algorithm yield several notable provisions, the first of which has been mentioned but is worth reiterating, the key of which are listed below:

(a) The FSP provides a strict lower bound on the exact CME solution ($P_J^{\text{FSP}} \leq P_J$) for every possible state of the process;

(b) The FSP provides the exact total absolute error of the finite approximation of the full CME, $\sum_i (P_i(t) - P_i^{\text{FSP}}(t)) = z(t)$;

(c) The error resulting from CME truncation, $z(t)$, decreases monotonically with each additional state that is added to the set $\mathbb{X}_J$;

(d) The exact cumulative probability distribution for the time to escape any finite set $\mathbb{X}_J$ is given by $z(t)$.

Characteristics (a)-(c) of the FSP approximation listed above enable the setting of a maximum tolerable CME error as a monotonically increasing function, $\varepsilon(t) > 0$, where $\varepsilon(t_1) \leq \varepsilon(t_2)$ for every $t_2 > t_1$. The FSP algorithm is then implemented so that it proceeds through the following steps [13, 15, 68, 79]:

1. An initial state space $\mathbb{X}_J$ is set, and the initial distribution along this space is extracted from $P(0)$;
2. The probabilities $P(t)$ and the FSP approximation error $z(t)$ are integrated forward in time using Eq. 9. If $z(t)$ exceeds the maximum error threshold function $\epsilon(t)$ at any time, the integration is paused, and the size of the truncated CME state space, $\mathbb{X}_J$, is expanded by adding states from $z(t)$ - thus shrinking $z(t)$ and lowering the total error - before continuing the integration;
3. The process is repeated until the final time, t_N, at which point the approximation guarantees that the FSP solution is below the error threshold at all times.

In practice, the SSIT uses multiple absorbing sinks [15], where each sink corresponds to the set of all states that violate a defined polynomial inequality:

$$J_{\text{sink-k}} \equiv \{i\} \text{ such that } f_k(\mathbf{x_k}) \geq b_k,$$

the FSP region is defined as

$$J \equiv \{i\} \text{ such that } f_k(\mathbf{x_k}) < b_k \text{ for all } k \in \{1, 2, \ldots, K\},$$

and the FSP approximation becomes:

$$\frac{d}{dt}\begin{bmatrix} P_J^{\text{FSP}}(t) \\ z_1(t) \\ z_2(t) \\ \vdots \\ z_K(t) \end{bmatrix} = \begin{bmatrix} A_{JJ} & 0 & 0 & 0 & 0 \\ 1^T A_{J_1 J} & 0 & 0 & 0 & 0 \\ 1^T A_{J_2 J} & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1^T A_{J_K J} & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_J^{\text{FSP}}(t) \\ z_1(t) \\ z_2(t) \\ \vdots \\ z_K(t) \end{bmatrix}, \tag{10}$$

By keeping track of how much probability mass exits through each sink at each time step, the SSIT can automatically adjust the polynomial bounds b_k to recover that probability mass in that and subsequent time steps.

Box 12 and `example_2_SolveSSITModels_FSP.m` show examples for computing FSP solutions in the SSIT.

Example Box 12: Directly Solve the CME with FSP

Below is an example for computing FSP solutions in the SSIT, using our 4-state STL1 model (Box 1), with results shown in Fig. B12.

```
% Ensure the solution scheme is set to FSP (default):
STL1_4state.solutionScheme = 'FSP';

% Set FSP 1-norm error tolerance:
STL1_4state.fspOptions.fspTol = 1e-4;

% Guess initial bounds on FSP StateSpace:
STL1_4state.fspOptions.bounds = [1,1,1,1,200];

% Approximate the steady state for the initial distribution:
STL1_4state.fspOptions.initApproxSS = true;
STL1_4state.tSpan = [0:5:60];
STL1_4state.initialTime = 0;

% Compile and store the given reaction propensities:
STL1_4state = STL1_4state.formPropensitiesGeneral('STL1_4state');

% Solve Model:
[~,~,STL1_4state] = STL1_4state.solve;
```

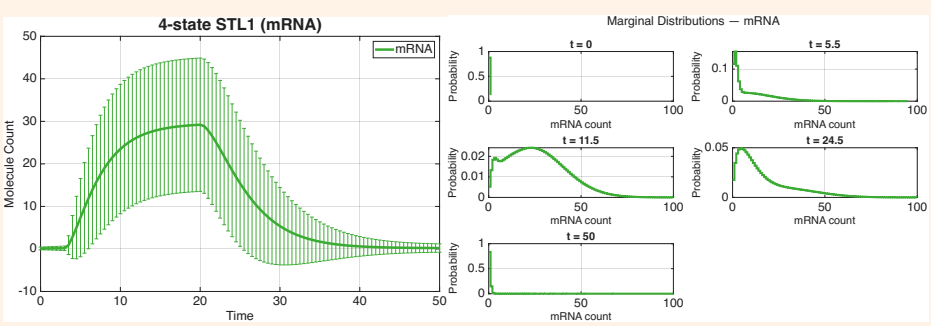

Figure B12. FSP means and standard deviations (top) and marginal distributions at select time points (bottom) for the 4-state STL1 model from Box 1.

2.2.6 Escape (First Passage) and Waiting Time Calculations

In many applications, we are interested not only in the instantaneous probability distribution on states, but also in the *escape* behavior of the process [89]. For a continuous-time Markov chain $X(t)$ with state space S and a subset $J \subset S$, the (random) escape time from J is defined as

$$\tau_J = \inf\{t > 0 \mid X(t) \notin J\}. \tag{11}$$

The corresponding cumulative escape probability (or escape-time distribution function) is

$$F_J(t) \;=\; p(\tau_J \leq t), \tag{12}$$

which quantifies the fraction of trajectories that have left the subset J by time t.

In the FSP formulation, the CME is approximated by a finite subset ($\mathbb{X}_J \subset \mathbb{X}$) and all states in the complement ($\mathbb{X}_{J'}$) are replaced by an absorbing sink variable $z(t)$, as in Eq. 9. $P_J^{\mathrm{FSP}}(t)$ is the probability vector over the truncated state set $\mathbb{X}_J$, A_{JJ} is the restriction of the generator to $\mathbb{X}_J$, and $\mathbf{1}$ is the vector of ones of appropriate length. By construction,

$$z(t) \;=\; 1 - \|P_J^{\mathrm{FSP}}(t)\|_1 \tag{13}$$

is exactly the cumulative probability of having left $\mathbb{X}_J$ by time t, and therefore

$$z(t) \;=\; F_J(t) \tag{14}$$

for the full CME.

To separate the desired escape behavior from numerical truncation error, it is useful to further decompose the state space into three disjoint and exhaustive subsets,

$$S = \mathbb{X}_J \;\cup\; \mathbb{X}_{J_e} \;\cup\; \mathbb{X}_{(J \cup J_e)'},$$

where X_J is the set of non-absorbing states of the FSP projection, $X_{J_{\mathrm{esc}}}$ is the collection of all states that satisfy the intended escape criteria, and $X_{J'} \equiv X_{(J \cup J_{\mathrm{esc}})'}$ contains all remaining states that will be combined into an auxiliary absorbing "error sink". Introducing two variables for the escape and error sinks, we can write the modified FSP as:

$$\frac{d}{dt}\begin{bmatrix} P_J^{\mathrm{FSP}}(t) \\ z_{\mathrm{esc}}(t) \\ z(t) \end{bmatrix} = \begin{bmatrix} A_{JJ} & 0 & 0 \\ 1^T A_{J_{\mathrm{esc}}J} & 0 & 0 \\ 1^T A_{J'J} & 0 & 0 \end{bmatrix} \begin{bmatrix} P_J^{\mathrm{FSP}}(t) \\ z_{\mathrm{esc}}(t) \\ z(t) \end{bmatrix}, \tag{15}$$

where A_{JJ} contains information about all reactions that are contained within X_J, $A_{J_{\mathrm{esc}}J}$ contains the reactions that transition $X_J \to X_{J_{\mathrm{esc}}}$, and $A_{J'J}$ contains the reactions that transition $X_J \to X_{J'}$. With this reformulation, z_{esc} is now a strict *lower bound* estimate on the cumulative escape probability, P_J^{FSP} is a strict lower-bound FSP estimate for the probability mass that has not yet escaped (i.e., P_J), and $z(t)$ is the total error of the FSP approximation:

$$z(t) = \left|P_J - P_J^{\mathrm{FSP}}\right|_1 + |P_{J'}|_1 + |P_{\mathrm{esc}} - z_{\mathrm{esc}}|. \tag{16}$$

Classical quantities such as the mean escape time from an initial state i to an absorbing target set J_1 can be recovered from the escape-time distribution via

$$\mathbb{E}[\tau_{i \to J_1}] = \int_0^\infty \left(1 - F_{J_1}(t)\right) dt. \tag{17}$$

Box 13 and `example_5_SolveSSITModels_EscapeTimes.m` show examples for computing escape time solutions.

Example Box 13: Computing First Passage Time Probabilities with the FSP

Below, we provide an example for computing escape time solutions that solves for the time until the 'mRNA' (from our 4-state STL1 model, Box 1) count reaches 100. It is worth noting that although the 4-state STL1 model is non-ergodic/non-stationary in the time scale we consider, the calculation of escape time distributions from within transient responses to Hog1 input is both still useful to consider and appropriate, as they are defined in terms of when trajectories leave a chosen set of states.
The cumulative distribution function and probability distribution functions are shown in Fig. 13.

```
% Set the times at which distributions will be computed:
STL1_4state.tSpan = linspace(0,100,200);

% Solve for time for mRNA to reach 100:
STL1_4state.fspOptions.escapeSinks.f = {'mRNA'};
STL1_4state.fspOptions.escapeSinks.b = 100;
[~,~,STL1_4state] = STL1_4state.solve;
```

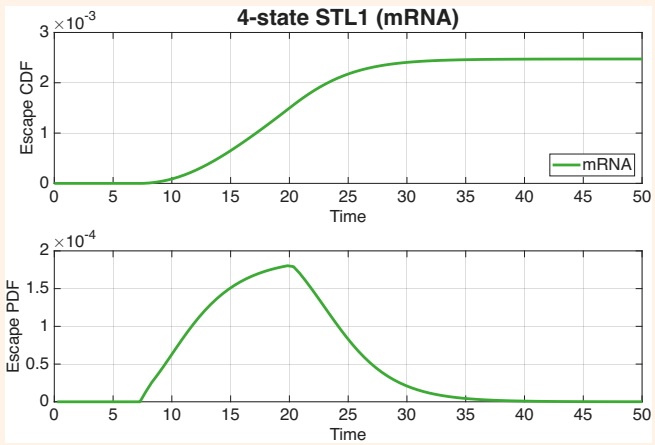

Figure B13. Escape CDF and PDF for time to $mRNA = 100$ for the 4-state STL1 model.

2.2.7 Sensitivity Analysis

Parameter sensitivity analysis provides quantified insights on how the model parameters affect outputs, and consequently, which parameters have greater effects than others, the extent of parameter identifiability, and which experimental conditions may yield the most information about underlying mechanisms [90, 91]. As there are different methods for computing derivatives, there are different approaches to computing the sensitivities of a model output with respect to its parameters. The SSIT focuses on the sensitivity of the CME solutions to parameter changes:

$$s_k(t;\theta) = \frac{\partial \mathbf{p}(t;\boldsymbol{\theta})}{\partial \theta_k},$$

where $\mathbf{p}(t;\boldsymbol{\theta})$ denotes the probability vector (or other model output) at time t. To estimate this quantity, the SSIT provides two approaches: finite difference, and forward integration as follows:

Finite differences. A basic approach is to approximate the sensitivity $s_i(t;\theta)$ using small perturbations h along the unit basis vector e_i:

$$\mathbf{s}_i(t;\boldsymbol{\theta}) \approx \frac{\mathbf{p}(t;\boldsymbol{\theta} + h/2\, e_i) - \mathbf{p}(t;\boldsymbol{\theta} - h/2\, e_i)}{h}. \tag{18}$$

For stochastic reaction networks, variance-reduced couplings (e.g., common random numbers, pathwise/likelihood-ratio hybrids) are often used when $\mathbf{P}$ is estimated by simulation; for FSP, (18) is computed by re-solving on the same truncated state space.

Forward (FSP) sensitivities. The forward integration approach uses a rearangement of the time derivative of the sensitivity function:

$$\frac{d}{dt}\mathbf{s}_k(t) = \frac{d}{dt}\frac{\partial \mathbf{p}}{\partial \theta_k} = \frac{\partial}{\partial \theta_k}\frac{d\mathbf{p}}{dt} = \frac{\partial}{\partial \theta_k}(\mathbf{A}\mathbf{p}) \tag{19}$$

$$= \mathbf{A}\frac{\partial \mathbf{p}}{\partial \theta_k} + \frac{\partial \mathbf{A}}{\partial \theta_k}\mathbf{p} = \mathbf{A}\mathbf{s}_k + \frac{\partial \mathbf{A}}{\partial \theta_k}\mathbf{p}, \tag{20}$$

which can be solved simultaneously with $\mathbf{p}(t)$ and using the initial condition $\mathbf{s}_k(t_0) = \partial_{\theta_k}\mathbf{p}(t_0;\boldsymbol{\theta})$.

FSP escape/absorbing states obey an analogous sensitivity balance [13]. Computation time for the forward sensitivity and the finite difference approach scales linearly with the number of parameters [92].

These sensitivities provide gradients of any distribution-based objective: log-likelihoods for maximum likelihood estimation (Section 2.3.3) and Bayesian inference (Section 2.3.4), distance metrics, or Fisher Information Matrix (Section 2.2.8) entries.

Box 14 and `example_6_SensitivityAnalysis.m` show how to compute sensitivity solutions in the SSIT.

Example Box 14: Performing FSP-Based Sensitivity Analysis

Below, we show an example usage of the SSIT's sensitivity solver for our 4-state STL1 model from Box 1 and show solution plots for parameter sensitivities by mRNA count at $t = 25$ in Fig. B14.

```
% Set solution scheme to FSP sensitivity:
STL1_4state.solutionScheme = 'fspSens';

% Solve the sensitivity problem:
[~,~,STL1_4state] = STL1_4state.solve;
```

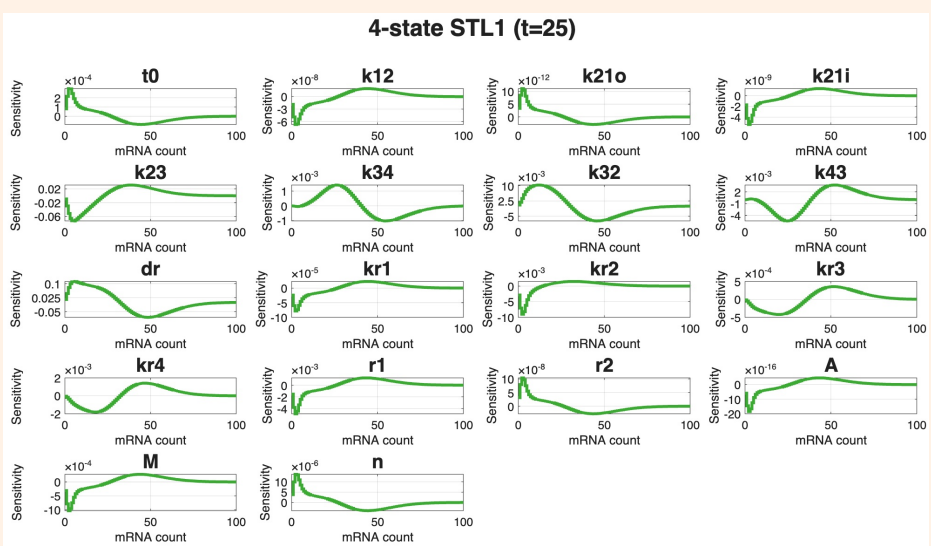

Figure B14. Sensitivities at $t = 25$ of each 4-state STL1 model parameter by mRNA count from the 0.2 nM dataset [24].

2.2.8 Fisher Information Matrix (FIM)

The Fisher Information Matrix (FIM) is a statistical tool that quantifies how much information an observable random variable contains about unknown model parameters $\boldsymbol{\theta}$ [93]. In other words, the FIM characterizes how tightly an experiment can, in principle, constrain $\boldsymbol{\theta}$ by measuring the local curvature (precision) of the likelihood around a parameter value [27]. When a maximum likelihood estimate (MLE) $\hat{\boldsymbol{\theta}}$ has been computed from data, the FIM determines the asymptotic covariance of $\hat{\boldsymbol{\theta}}$. In a Bayesian setting, under standard regularity conditions and a sufficiently smooth/weakly informative prior, the posterior distribution becomes approximately Gaussian for large samples with covariance governed by the FIM [94]. Thus, the FIM is a natural tool for assessing parameter identifiability, for diagnosing stiffness in parameter directions, and for making decisions about sequential experiments [58].

The utility of an FSP-based FIM approach for single-cell experiments was demonstrated by Fox *et al.* [27]. They used the FSP algorithm to compute the FIM by solving the Chemical Master Equation (CME) for constitutive gene expression, including in cases where expression was strongly non-Gaussian. Unlike previous approaches that approximate population distributions as Gaussian [95, 96], which may be suitable only for large-population limits, the Fox *et al.* FSP-FIM framework works directly with the full discrete distribution and therefore captures the inherent stochasticity of small-copy-number systems.

Intuitively, the tighter a (locally) Gaussian distribution for $\hat{\boldsymbol{\theta}}$, the more information the data provide about the underlying model parameters $\boldsymbol{\theta}$. For sufficiently large sample size N_c, the covariance $\Sigma_{\hat{\boldsymbol{\theta}}}$ of an unbiased MLE $\hat{\boldsymbol{\theta}}$ satisfies

$$\Sigma_{\hat{\boldsymbol{\theta}}} \;\approx\; I(\boldsymbol{\theta}^*)^{-1}, \tag{21}$$

where $I(\boldsymbol{\theta}^*)$ is the FIM evaluated at the true parameter $\boldsymbol{\theta}^*$. The matrix $I(\boldsymbol{\theta}^*)^{-1}$ sets the Cramér–Rao lower bound (CRB) on $\Sigma_{\hat{\boldsymbol{\theta}}}$ [97]; moreover,

$$\sqrt{N_c}\,\big(\hat{\boldsymbol{\theta}} - \boldsymbol{\theta}^*\big) \xrightarrow{d} \mathcal{N}\big(0,\, I(\boldsymbol{\theta}^*)^{-1}\big), \tag{22}$$

as $N_c \to \infty$ [27]. Diagonalizing the FIM as $I(\boldsymbol{\theta}^*) = Q\Lambda Q^\top$, the eigenvectors (columns of Q) define principal directions in parameter space, and the eigenvalues in Λ determine the scale of uncertainty along those directions: large eigenvalues correspond to well-constrained combinations of parameters, while small eigenvalues indicate poorly constrained directions.

To connect with the likelihood, define the *score* (or *informant*)

$$u(\boldsymbol{\theta_k}; \mathcal{D}) \;=\; \frac{\partial}{\partial \theta_k}\, \log \mathcal{L}(\boldsymbol{\theta}; \mathcal{D}), \tag{23}$$

where $\mathcal{L}(\boldsymbol{\theta}; \mathcal{D})$ is the likelihood for data $\mathcal{D}$, (see Section 2.3.2 for likelihood calculation). The FIM is then the covariance (equivalently, the second central moment) of the score:

$$I_{jk}(\boldsymbol{\theta}) \;=\; \mathrm{Var}\big(s(\boldsymbol{\theta}; \mathcal{D})\big) \;=\; \mathbb{E}_{\mathcal{D}}\big[\, u(\boldsymbol{\theta}; \mathcal{D})\, u(\boldsymbol{\theta}; \mathcal{D})^\top \,\big|\, \boldsymbol{\theta}\big]\,, \tag{24}$$

which, iterating over parameter combinations (i.e., in component form), gives

$$I_{jk}(\boldsymbol{\theta}) \;=\; \mathbb{E}_{\mathcal{D}}\left[\frac{\partial \log L(\boldsymbol{\theta}; \mathcal{D})}{\partial \theta_j}\, \frac{\partial \log L(\boldsymbol{\theta}; \mathcal{D})}{\partial \theta_k}\right]. \tag{25}$$

Under standard regularity assumptions, this also equals the negative expected Hessian of the log-likelihood.

For the FSP-based CME model (Section 2.2.5), we consider independent single-cell measurements drawn from the probability distribution $p(x, t_m; \boldsymbol{\theta})$ at one or more

observation times t_m. For a single measurement of a cell state X with probability mass function $p(x;\boldsymbol{\theta})$, the contribution to the score is

$$\frac{\partial}{\partial \theta_i} \log p(X;\boldsymbol{\theta}) \;=\; \frac{1}{p(X;\boldsymbol{\theta})} \frac{\partial p(X;\boldsymbol{\theta})}{\partial \theta_i}. \tag{26}$$

Using the sensitivities defined in Section 2.2.7, $s_i(x,t_m) = \partial_{\theta_i} p(x,t_m;\boldsymbol{\theta})$, the FIM for N_c independent cells observed at a fixed time t_m can be written as

$$I_{ij}(\boldsymbol{\theta}) \;=\; N_c \sum_{x \in \mathcal{X}_{\mathrm{FSP}}} \frac{1}{p(x,t_m;\boldsymbol{\theta})}\, s_i(x,t_m;\boldsymbol{\theta})\, s_j(x,t_m;\boldsymbol{\theta}). \tag{27}$$

When multiple observation times and experimental conditions are used, their contributions are additive: one sums expressions of the form (27) over all times and conditions, with appropriate cell counts for each. Because (27) is derived directly from the FSP probabilities and their sensitivities, it inherits the FSP's ability to represent non-Gaussian, multimodal, or highly skewed distributions, making it well-suited for optimal design and identifiability analysis in single-cell experiments [98].

Box 15 and `example_7_FIM.m` give examples for computing the FIM.

Example Box 15: Computing the Fisher Information Matrix

Below is an example for computing the FIM sub matrix for the free parameters in our 4-state STL1 model (Box 1), as the five Hog1 signal parameters ('r1', 'r2', 'A', 'M', and 'n') are experimentally known and can thus be fixed. In Fig. 15, we show heat maps, per-θ information, and eigenvalue spectra for both the full FIM and free FIM results.

```
% Compute the FIM sub matrix for free parameters:
STL1_4state_fimResults = STL1_4state_FIM.computeFIM([],'log',[],freePars);

% Generate a count of measured cells:
cellCounts = 1000*ones(size(STL1_4state_FIM.tSpan));

% Evaluate the provided experiment design (in "cellCounts")
% and produce an array of FIMs (one for each parameter set):
[STL1_4state_fimTotal,STL1_4state_mleCovEstimate,STL1_4state_fimMetrics] = ...
    STL1_4state_FIM.evaluateExperiment(STL1_4state_fimResults,cellCounts)
```

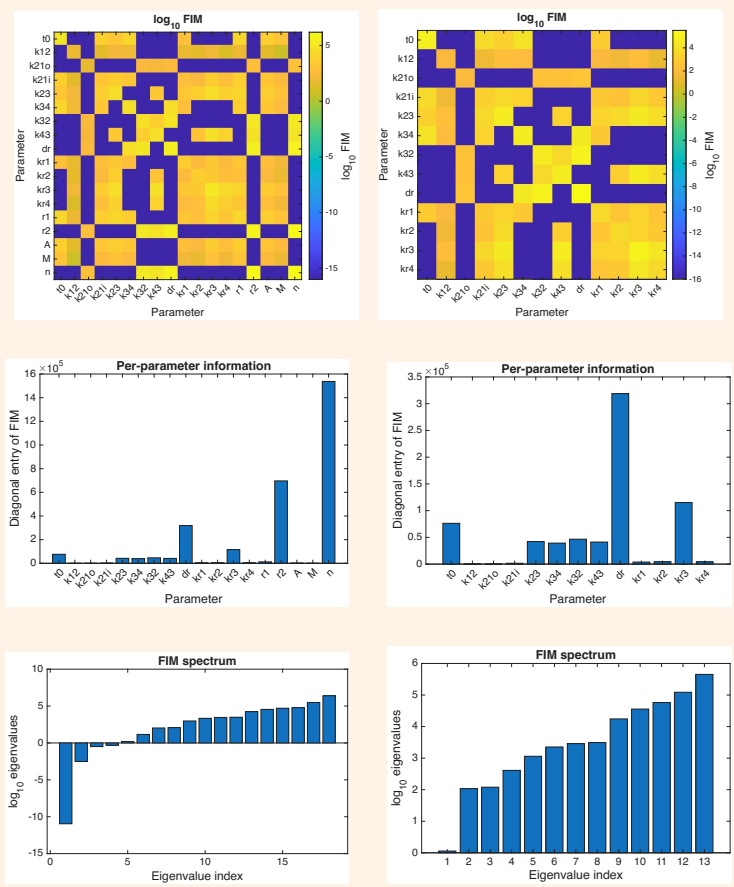

Figure B15. (Top) θ-heat maps; (middle) per-θ information; and (bottom) FIM spectra for (left) full FIM results and (right) results using only free parameters.

2.2.9 FIM Optimality Criteria and Experiment Design

The Fisher Information Matrix $I(\boldsymbol{\theta};\xi)$ for a given experimental design ξ (e.g., choice of input signals, measurement times, doses, or imaging settings) quantifies how much

information that design can provide about the model parameters $\boldsymbol{\theta}$. [27, 95, 99] In practice, we use the FIM to compare candidate designs and to select experiments that are expected to yield the most informative data, either with respect to all parameters or to a subset of parameters of particular interest [98–100].

Several scalar *optimality criteria* compress the FIM into a single design score, each emphasizing different aspects of parameter uncertainty [101, 102]. The SSIT offers:

- **E-optimality** (default) - maximizes the smallest eigenvalue of $I(\boldsymbol{\theta};\xi)$, ensuring that no direction in parameter space is too poorly estimated (which can be advantageous or disadvantageous depending on the presence of weakly identifiable parameters).
- **D-optimality** - maximizes the expected determinant, $\det I(\boldsymbol{\theta};\xi)$ - equivalently, minimizes $\det I(\boldsymbol{\theta};\xi)^{-1}$, the volume of the confidence ellipsoid in parameter space and thus reduces overall uncertainty in all linear combinations of parameters.
- **$\mathbf{D_s}$-optimality** - maximizes the determinant of a sub-matrix of the FIM to reduce uncertainty for a particular subset of parameters. This may be especially useful for hierarchical models; the SSIT provides $\mathrm{D_s}$-style criteria for subsets of or grouped parameters.
- **Trace-based optimality** - maximizes the trace of the FIM, $\mathrm{tr}(I(\boldsymbol{\theta};\xi))$. Trace criteria is similar to **A-optimality**, which minimizes $\mathrm{tr}\big(I(\boldsymbol{\theta};\xi)^{-1}\big)$, the average variance across parameters. A-optimality is often viewed as a more conservative variant of D-optimality because it tends to avoid extreme experimental conditions - e.g., very large doses or impractical imaging regimes.

Other FIM-based criteria target different goals. **G-optimality** minimizes the worst-case prediction variance over a specified design or prediction space, prioritizing uniform predictive accuracy rather than parameter precision alone. **T-optimality** and related model-discrimination criteria focus on maximizing differences in model predictions, thereby favoring designs that most clearly distinguish between competing mechanistic hypotheses [99, 103].

Within the FSP-FIM framework [27], each of these criteria can be evaluated by combining the distribution-level sensitivities $\mathbf{S}_i(\mathbf{x}, t; \boldsymbol{\theta})$ with the experiment-specific cell counts and observation times, and then optimizing over ξ to guide sequential experiment design [98].

In Box 16, we optimize the sampling of 13k cells across 13 time points, from 0 to 60 minutes, using five different criteria (the full example code is provided in `example_7b_FIM_ExperimentDesign.m`). It is clearly important to understand and choose the best optimality criteria for one's specific research questions.

Example Box 16: Design Optimal Experiments Using the FIM

Here is an example for computing the optimal number of cells using the FIM.

```
% Compute FIM results:
fimResults = STL1_4state_design.computeFIM([],'log',[]);

% Get the number of cells from loaded experimental data using 'nCells':
cellCounts_data = STL1_4state_design.dataSet.nCells * ...
                  ones(size(STL1_4state_design.tSpan));

%% Experiment Design
% Find the FIM-based designs for a total cells

% Compute the optimal number of cells from the FIM results using different
% design criteria:  'Trace' maximizes the trace of the FIM; 'DetCovariance'
% minimizes the expected determinant of MLE covariance; 'Smallest Eigenvalue'
% maximizes the smallest e.val of the FIM; and 'TR[$<i_1>,<i_2>$,...]'
% maximizes the determinant of the FIM for the specified indices.  The latter
% is shown for different parameter combinations, where 'Tr[9:13]' are the
% mRNA-specific parameters 'dr' and 'kr1','kr2','kr3', and 'kr4' (degradation
% and transcription reactions).  All other parameters are assumed to be known
% and fixed.

nCol = sum(cellCounts);
nTotal = nCol(1);
nCellsOpt_detCov = ...
    STL1_4state_design.optimizeCellCounts(fimResults,nTotal,'DetCovariance');
nCellsOpt_trace = STL1_4state_design.optimizeCellCounts(fimResults,nTotal,'Trace');
nCellsOpt_tr = STL1_4state_design.optimizeCellCounts(fimResults,nTotal,'tr[1:8]');
nCellsOpt_trR = STL1_4state_design.optimizeCellCounts(fimResults,nTotal,'tr[9:13]');
nCellsOpt_trI = STL1_4state_design.optimizeCellCounts(fimResults,nTotal,'tr[14:18]');
```

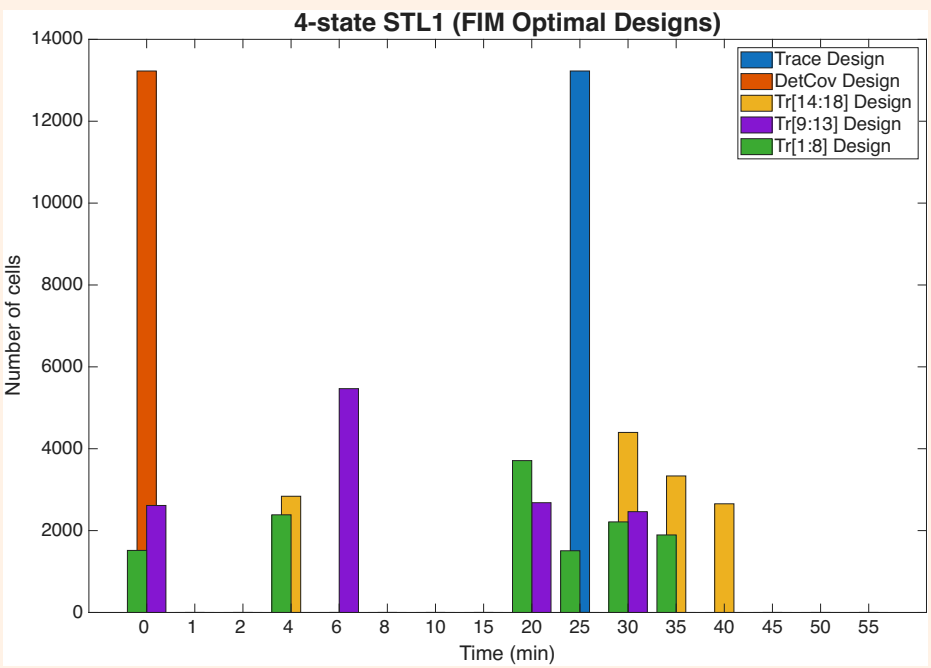

Figure B16: The optimal number of cells per time point based on different FIM criteria: 'Trace' is trace-based optimality and maximizes the trace of the FIM; 'DetCovariance' is D-optimality and minimizes the expected determinant of MLE covariance; 'Smallest Eigenvalue' is E-optimality and maximizes the smallest e.val of the FIM; 'tr[$< i_1 >, < i_2 >$,...]' is Ds-optimality and maximizes the determinant of the FIM for the specified indices. The latter is shown for different parameter combinations, where 'Tr[9:13]' are the mRNA-specific parameters 'dr' and 'kr1','kr2','kr3', and 'kr4' (degradation and transcription reactions).

2.3 The SSIT provides integrated capabilities to load, process, and fit experimental data

In this section, we discuss and demonstrate how to load experimental or saved simulated data, associate it with a model in the SSIT, and fit the model parameters to the data using experimental STL1 data [24] and our 4-state STL1 model (Box 1).

2.3.1 Data Loading and Handling

The SSIT reads count data, or other numeric data that may be interpreted as count data by the SSIT, such as total or average fluorescence intensities measures whose resulting distorted probability distributions can be transformed by the use of the SSIT's *probabilistic distortion operators* (PDOs).

File types

Data files that can be read as a table or a cell by MATLAB's 'readtable' and 'iscell' functions can be read by the SSIT, including: text files (.txt and .csv) and Microsoft Excel files (.xls),

Loading

The 'loadData' function is used to load data into the SSIT. The SSIT has built-in filtering capabilities, called simply by specifying which strings should be searched for and which values should be used to filter.

Single-Cell Experiments and Imaging

Although the SSIT is general and can be used for the modeling of any discrete stochastic system, for the sake of simplicity, we focus on single-cell experiments to illustrate the usage and underlying methodology of the SSIT and its associated packages. This choice is motivated by the rapidly growing body of quantitative experiments that capture the *in situ* heterogeneity and intricacies of molecular dynamics at the most fundamental level of biology [79, 104, 105].

The Chemical Master Equation has been successfully applied to study numerous single-cell processes, e.g., in the elucidation of transcriptional and translational bursting and the contribution of each reaction's effects on RNA or protein levels within a cell population [24], to infer the kinetic parameters driving gene regulatory networks from time-series data [68], and to describe phenotypic variability between single cells [106].

A particularly powerful approach to capturing intracellular process data, and therefore the dynamic processes occurring within cells, is to use fluorescence microscopy [107, 108]. Single-molecule fluorescent imaging involves the attachment of probes [109–111] to individual molecules and enables their tracking and localization within individual cells. Single-molecule fluorescence in-situ hybridization (smFISH) is a technique for measuring gene expression and localizing RNA *in situ* by attaching fluorescent complementary DNA oligos to individual mRNA molecules [109, 112]. The cells are then imaged under an epi-fluorescence microscope, where each DNA probe on its individual mRNA is shown as a single bright spot. Each spot can then be counted, and the total number of spots is treated as a direct measurement of the number of mRNA molecules that were transcribed in the cell [113]. However, counting individual spots can be tricky due to the 3D nature of cells, not solely to determine localization but also to avoid miscounts due to overlapping spots.

Single-cell experiments have transformed biological research by enabling the characterization of cellular heterogeneity at unprecedented resolution. Unlike traditional bulk assays that report only population-averaged measurements, single-cell technologies quantify molecular features such as gene expression, protein abundance, and epigenetic

modifications in individual cells. Advances in single-cell RNA sequencing (scRNA-seq) now make it possible to capture the transcriptomic landscape by profiling thousands to millions of cells in a single experiment, uncovering rare cell populations, delineating developmental trajectories, and revealing cell-specific responses to perturbations [114–116]. These approaches have also expanded beyond transcriptomics to include chromatin accessibility [117], DNA methylation [118], and protein levels [119], providing a more integrated view of cellular state and function.

However, single-cell data pose unique computational and statistical challenges, including technical noise, dropout events, and high dimensionality. Addressing these issues requires specialized modeling approaches, such as probabilistic frameworks, dimensionality reduction methods, and tools for data integration and imputation. The SSIT is applicable to many of these settings: even continuous measurements can be discretized, and the associated measurement error can be explicitly modeled and quantified using probabilistic distortion operators (PDOs) and related error-handling functions [29], (see Section 2.4.3).

In this paper, we use measurements of nascent STL1 mRNA expressed as part of the mitogen-activated protein kinase stress response in yeast cells [24, 26] and from scRNA-seq data composed of 151 genes in breast cancer cells responding to a treatment of dexamethasone [120] in demonstratation of how to create, solve, and fit models; perform statistical inference; account for extrinsic noise; and design a best next experiment in the SSIT. Box 17 and `example_8_LoadingandFittingData_DataLoading.m` illustrate how to load and filter data in one command.

Example Box 17: Load Data and Associate with Model

Below is the SSIT loadData usage to load STL1 mRNA count data from the osmotic shock experiments in Neuert *et al.* [24] and associate the data with the 4-state STL1 model (Box 1). The first input is the location and name of the data file, in this case a .csv file. The second input links the name(s) of the model species with the appropriate name(s) of the data file columns; in this case, our data consists of counts for the model species 'mRNA' which are provided in the column 'RNA_STL1_total_TS3Full'. The third input is optional and may be used for filtering the data by conditions specified within the data file; in this case, only the data from the second replica of 0.2M NaCl concentration experiments are applied to the SSIT model named 'STL1_4state_data'.

```
% Note: Ensure the search path is correct on the local machine:
STL1_4state = STL1_4state.loadData('data/filtered_data_2M_NaCl_Step.csv',...
                                   {'mRNA','RNA_STL1_total_TS3Full'},...
                                   {'Replica',1;'Condition','0.2M_NaCl_Step'});
```

2.3.2 Likelihood Calculation

Once data has been loaded into SSIT and associated with a model, the data $\mathcal{D}$ can be used to confine the model parameters $\boldsymbol{\theta}$ to values that most likely represent the underlying biological process which generated the data. Often this involves the calculation of a likelihood using an appropriate likelihood function $\mathcal{L}(\mathcal{D}|\boldsymbol{\theta})$, which provides a quantification of the probability of observing the data under particular values of model parameters [121]. In other words, the likelihood function $\mathcal{L}(\mathcal{D}|\boldsymbol{\theta})$ yields insight pertaining to the degree of support shown by the data D as a function of various combinations of parameter values $\boldsymbol{\theta}$. The exact specification of the likelihood function depends on the type of data being studied, and the subsequent use of the likelihood once

computed is dependent on the complexity (dimensionality, information content, etc.) of the data, the available computation power, and the researcher's statisical philosophy.

Here, we provide an example of an appropriate likelihood function that would fit the type of data for which SSIT has been constructed. First, we describe observed molecular counts as $(d_{t_1}, d_{t_2}, \ldots, d_{t_n}) = \mathcal{D}$ taken from single cell experimental data for a specific cell measured at time points $t = [t_1, t_2, \ldots, t_n]$. In our 'Model' example, these data are the numbers of mRNA from a particular cell at time points t. The likelihood function is:

$$\mathcal{L}(\mathcal{D}|\boldsymbol{\theta}) = \Pr(d_{t_1}, d_{t_2}, \ldots, d_{t_n} \,|\, \boldsymbol{\theta}). \tag{28}$$

Note that in this formulation the data $\mathcal{D}$ is fixed, and the likelihood is dependent only on the model parameters, $\boldsymbol{\theta}$.

The likelihood function in Eq. 28 can be estimated directly from the system's state probabilities, i.e., when (a) an exact Chemical Master Equation (CME) solution can be computed, or (b) the CME can be approximated, such as with Finite State Projection (FSP) as discussed in Section 2.2. [13, 15, 27]

For example, having obtained the probabilities $p_x(t) = \Pr(d_{t_1}, d_{t_2}, \ldots, d_{t_n} \,|\, \boldsymbol{\theta})$ of CME states, then $\Pr(d_{t_1}|\boldsymbol{\theta})$ can be computed as the probability distribution at the first time point t_1. Then probabilities at subsequent time points can be computed by continuing the solution forward in time and conditioning upon the previous solution. The second term is therefore $\Pr(d_{t_2}|d_{t_1}, \boldsymbol{\theta})$ and the nth term is $\Pr(d_{t_n}|d_{t_1}, d_{t_2}, \ldots, d_{t_{n-1}}, \boldsymbol{\theta})$, with the full iteration for calculating Eq. 28:

$$\mathcal{L}(\mathcal{D}|\boldsymbol{\theta}) = \Pr(d_{t_1}|\boldsymbol{\theta}) \cdot \Pr(d_{t_2}|d_{t_1}, \boldsymbol{\theta}) \cdots \Pr(d_{t_n}|d_{t_1}, d_{t_2}, \ldots, d_{t_{n-1}}, \boldsymbol{\theta}). \tag{29}$$

Generally, the procedure for computing the likelihoods of time-series data as in Eq. 29 is called filtering, and includes the Kalman filter [122] and HMM filter, the algorithm outlined above. This particular forward-time conditional description for calculating the likelihood based on state spaces is also known as the forward algorithm for state-space models [123–125].

Following Eq. 28 and 29 will yield the likelihood for data observed in a single cell, which does not provide much information about the biological system in question, especially when the system is based on highly stochastic processes and thus cells can exhibit highly variable dynamics. Fortunately, if we can appropriately assume in our current model that the processes taking place within each cell are independent to those of the other cells from which we are taking snapshot data, such as may be the case for smFISH and sequencing data, then it is straightforward to use Eq. 28/29 to compute the likelihood based on molecular count data taken from multiple independent cells at the same time. The independence between cells is a key feature because this means we can simply multiply the likelihoods for each molecular count at each time point from each single cell. Across N_c independent single-cell measurements, the likelihood can be reduced simply to:

$$\mathcal{L}(\mathcal{D}|\boldsymbol{\theta}) = \Pr(d_{t_1}|\boldsymbol{\theta}) \cdot \Pr(d_{t_2}|\boldsymbol{\theta}) \cdots \Pr(d_{t_n}|\boldsymbol{\theta}) = \prod_{c=1}^{N_c} \Pr(d_c|\boldsymbol{\theta}), \tag{30}$$

with d_c referring to the cth cell, a formulation that has been widely used to fit stochastic gene-expression models to single-cell snapshot data [24, 25, 68]. For mathematical simplicity, we often use the log-likelihood form,

$$\log\mathcal{L}(\mathcal{D}|\boldsymbol{\theta}) = \sum_{i=1}^{N_c} \log \Pr(d_c|\boldsymbol{\theta}). \tag{31}$$

The likelihood function $\mathcal{L}(\mathcal{D}|\boldsymbol{\theta})$ is computed based on a particular set of model parameters $\boldsymbol{\theta}$, the values of which must be iteratively sampled if not known *a priori* - which is always the case if the data is not simulated with known parameter values - and the likelihood recomputed for each sample of $\boldsymbol{\theta}$. It might be obvious, then, that one method of determining the model parameter values that best fit the data is to find the set $\boldsymbol{\theta}$ for which the likelihood of the data is the highest. In other words, the set of parameter values $\boldsymbol{\theta}$ that maximizes the likelihood of the data, $\mathcal{L}(\mathcal{D}|\boldsymbol{\theta})$.

2.3.3 Maximum Likelihood Estimation

A maximum likelihood estimate (MLE) is the maximum value of the likelihood function, $\mathcal{L}(\mathcal{D}|\boldsymbol{\theta})$ [121]. An MLE is meant to reveal the set of parameter values $\boldsymbol{\theta}$ that make the observed data $\mathcal{D}$ most probable and is often represented with argmax [126], an argument that returns the maximum value of the function:

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \left(\log \mathcal{L}(\mathcal{D}|\boldsymbol{\theta}) \right). \tag{32}$$

More details about log-likelihood maximization in the Supporting Information.

Box 18 and `example_9_LoadingandFittingData_MLE.m` demonstrate the use of maximum likelihood estimation to fit model parameters in the SSIT.

Example Box 18: Fit Model Parameters (MLE)

An example of fitting model parameters to loaded data using maximum likelihood estimation (MLE) is shown below. The results are shown in Fig. B18.

```
% Set fitOptions, with the maximum allowable number of iterations to fit:
fitOptions = optimset('Display','iter','MaxIter',2000);

% Specify which parameters to fit (all but Hog1 input parameters, which are fixed):
fitPars = 13;   STL1_4state.fittingOptions.modelVarsToFit = 1:fitPars;

% Store parameters for fitting:
STL1_4state_pars = cell2mat(STL1_4state.parameters(1:fitpars,2));

% Search to Find the MLE:
[STL1_4state_pars,STL1_4state_likelihood] = ...
    STL1_4state.maximizeLikelihood(STL1_4state_pars,fitOptions);

% Update parameters:
for i=1:length(STL1_4state_pars)
    STL1_4state.parameters{i,2} = STL1_4state_pars(i);
end
```

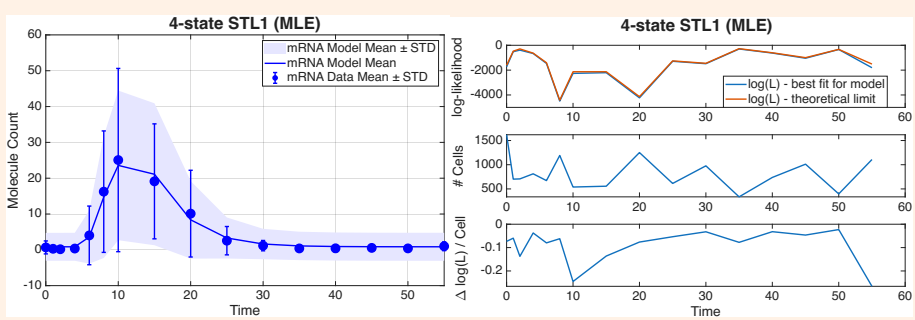

Figure B18: Trajectories of means and standard deviations for STL1 data [24] and the 4-state STL1 model fit to the data by MLE.

The higher the dimensionality of the model, the more parameter values there are to be estimated such that the likelihood function reaches its global maximum. The more parameters that must be estimated, the richer the data must be in information. When the systems being examined are stochastic in nature, there will inevitably be increased noise inherent in our data - in addition to any extraneous noise we add to it by our experimental, data collecting, and data processing choices - leading to *parameter uncertainty*. In other words, the parameter set yielded by $\hat{\boldsymbol{\theta}}$ often does not actually represent the "true" parameters of the model or underlying the biological processes. We should therefore take precautions to account for uncertainty in our parameter estimates. We can use techniques such as Metropolis-Hastings to account for parameter uncertainty [127–129].

2.3.4 Bayesian Inference

By its nature Bayesian inference provides measures of uncertainty surrounding parameter estimates and in the case of discrete variables (in our case, model parameters $\boldsymbol{\theta}$) will yield probabilities over every hypothesis for how the data was generated, when computationally feasible. For continuous variables ($\boldsymbol{\theta}$), Bayesian inference yields posterior probabilities over the sampled space taken from a *posterior probability distribution*, or *posterior* [130, 131].

Bayesian inference is a method that takes a prior belief or knowledge $\Pr(\boldsymbol{\theta})$ about the set of model parameters $\boldsymbol{\theta}$ *a priori* to observing the data from our most recent experiment or simulations and updates that state of knowledge about $\boldsymbol{\theta}$ based on the likelihood $\mathcal{L}(\mathcal{D}|\boldsymbol{\theta})$ of observing the current data $\mathcal{D}$ given a set of $\boldsymbol{\theta}$ values by computing the conditional probability $\Pr(\boldsymbol{\theta}|\mathcal{D})$ *a posteriori* to observing some data, $\mathcal{D}$, and normalizing by the probability of observing the data, $\Pr(\mathcal{D})$, i.e.,

$$\Pr(\boldsymbol{\theta}|\mathcal{D}) = \frac{\mathcal{L}(\mathcal{D}|\boldsymbol{\theta}) \cdot \Pr(\boldsymbol{\theta})}{\Pr(\mathcal{D})}, \tag{33}$$

which is commonly referred to as Bayes' theorem [132, 133].

A common method employed to sample the posterior probability distribution is Markov Chain Monte Carlo (MCMC), particularly the Metropolis-Hastings (MH) algorithm [127–129]. We elaborate on Bayesian inference, MCMC, and MH in the Supporting Information. As per Eq. 21, the inverse of the FIM, $I(\theta)^{-1}$, can be used to determine the covariance of the multivariate normal proposal distribution for the MHA, which may improve the sampling efficiency. Example code for using $I(\theta)^{-1}$ as a proposal distribution for Metropolis-Hastings is also provided in the Supporting Information.

Boxes 19-20 and `example_10_LoadingandFittingData_MH.m` show an example of how to sample parameters using Metropolis-Hastings to produce the improved model fits and what to expect from the results.

Example Box 19: Fit Model Parameters (Metropolis-Hastings): Code

```
% Specify how many model parameters will be fit:
fitpars = 13;   STL1_4state_MH.fittingOptions.modelVarsToFit = 1:fitpars;

%% Specify Prior as log-normal distribution with wide uncertainty
% Prior log-mean:
mu_log10 = [0.5,2,5,3.5,-0.4,1,0.2,0.4,-0.5,-1.3,-0.1,2,0.5];

% Prior log-standard deviation:
sig_log10 = 2*ones(1,fitpars);

% Prior:
STL1_4state_MH.fittingOptions.logPrior = ...
    @(x)-sum((log10(x)-mu_log10).^2./(2*sig_log10.^2));

% Create first parameter guess:
STL1_4state_MH_pars = [STL1_4state_MH.parameters{:,2}];

% Run Metropolis-Hastings (can be adjusted for better sampling
- generally aim for acceptance ratio around 0.3-0.4):
proposalWidthScale = 0.01;
MHOptions.proposalDistribution  = @(x)x+proposalWidthScale*randn(size(x));

% Set MH runtime options (number of samples, burnin, thinning, etc.):
MHOptions.numberOfSamples = 2000; MHOptions.burnin = 500; MHOptions.thin = 2;

% Run Metropolis-Hastings:
[STL1_4state_MH_pars,~,STL1_4state_MH_MHResults] = ...
    STL1_4state_MH.maximizeLikelihood([], MHOptions, 'MetropolisHastings');

% Store MH parameters in model:
STL1_4state_MH.parameters([1:fitpars],2) =  num2cell(STL1_4state_MH_pars);
```

Example Box 20: Fit Model Parameters (Metropolis-Hastings)

To sample parameter uncertainty and improve the fit of our 4-state STL1 model parameters (Box 1) to STL1 data [24], we iterate between computing maximum likelihood estimates and sampling by Metropolis-Hastings rejection scheme. Example code is provided in the following example block, and results are shown below in Fig. B20.

```
% Plot results:
STL1_4state_MH.plotMHResults(STL1_4state_MH_MHResults);
STL1_4state_MH.plotFits([], "all", [], {'linewidth',2},...
     Title='4-state STL1', YLabel='Molecule Count',...
     LegendLocation='northeast', LegendFontSize=18, ProbXLim = [0 80],...
     TimePoints=[0 8 10 15 30 55], TitleFontSize=24, AxisLabelSize=20);
```

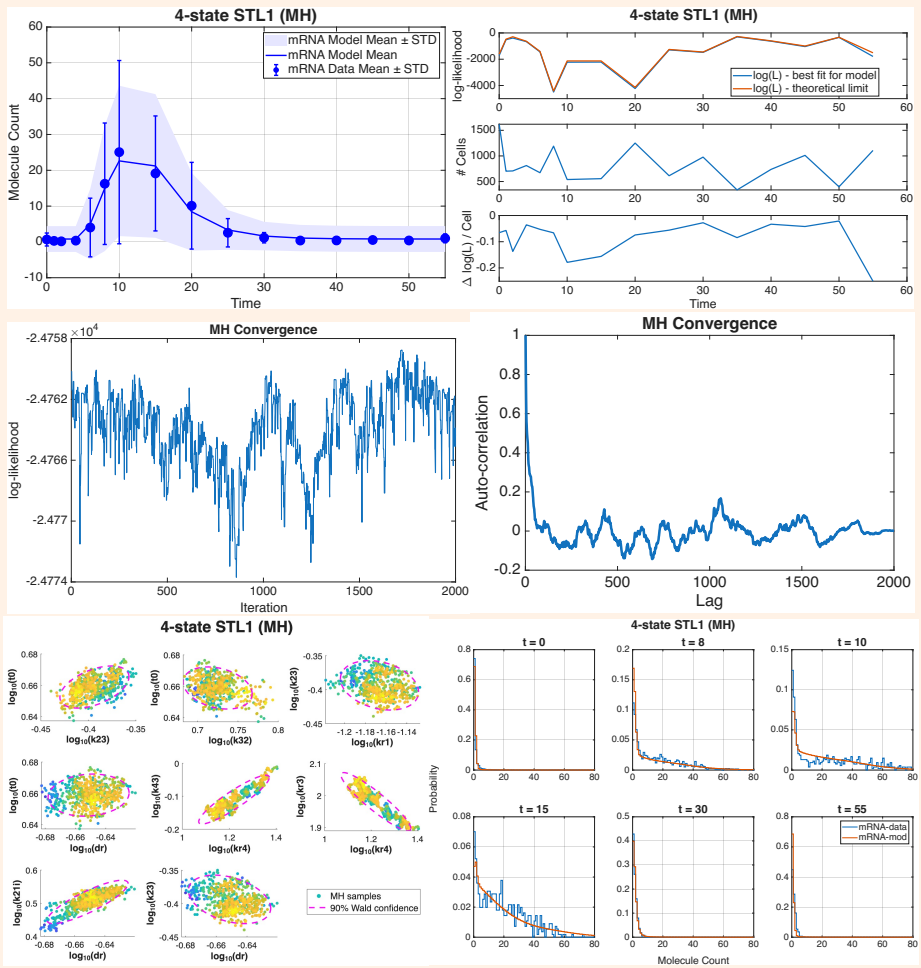

Figure B20: Trajectories of means and standard deviations for STL1 data; log-likelihoods ($\log(\mathcal{L})$) for best fit for model vs. theoretical limit; MH convergence by $\log(\mathcal{L})$ and autolag; a subset of sampled model parameters; and probability distributions at select time points.

2.3.5 Approximate Bayesian Computation

The SSIT also offers Approximate Bayesian Computation (ABC) for likelihood-free inference for cases in which a tractable likelihood is either unobtainable or infeasible to compute [62, 134]. Details about ABC implementation in the SSIT are provided in the Supporting Information.

2.3.6 Cross-Validation

The SSIT provides tools for cross-validation, where the model can be fit to different combinations of time points or replicas to determine how much parameters change and how well the model can predict some data when trained on other data. Additional details are included in the Supporting Information.

2.4 The SSIT provides advanced tools to create and analyze more complex combinations of models and datasets

In addition to sophisticated and time-varying propensity functions, the SSIT handles the following complexities: solving reduced models for increased computational efficiency (Section 2.4.1), including hybridized ODE-FSP solutions (Section 2.4.2); handling of probability distributions that have been distorted by data produced by particular choices in experimental design, measurement, and data processing (Section 2.4.3); and fitting multiple models and datasets (Section 2.4.4).

For Sections 2.4.3 and 2.4.4, we use an additional example dataset to showcase complex modeling in the SSIT, as these data benefit even more from probabilistic distortion handling (2.4.3) and multiple model-fitting (2.4.4) that assumes some shared parameters than the Hog1 dataset. This additional example is centered around single-cell RNA-seq (scRNA-seq) measurements representing transcriptional responses of different genes across five distinct time points following dexamethasone treatment in T47D A1-2 breast cancer cells [120]. Additional detail is provided in the Supporting Information).

2.4.1 Additional Model Reduction Strategies

Model reduction techniques simplify complex mathematical models while preserving essential dynamical behavior. These strategies are particularly valuable in systems biology, control theory, chemical kinetics, and engineering, where full-scale models may be computationally expensive or analytically intractable due to high dimensionality or nonlinearities [135, 136]. The goal is to create a reduced model that is easier to simulate, analyze, and interpret, without sacrificing predictive accuracy in relevant contexts.

The FSP approximation provides an exact model reduction method to reduce the (potentially infinite) set of states of the Chemical Master Equation to a finite set while satisfying a strict error tolerance [13, 27, 137]. However, even after the FSP reduction many problems remain computationally burdensome or intractable. Fortunately, subsequent approximate reductions are also possible through application of additional projection and lifting operations. For most of these, one proposes a lower dimensional vector $\mathbf{q}$ such that $\mathbf{p} \approx \mathbf{\Phi q}$. Substituting this approximation into 3, allows for one to formulate a reduced set of equations:

$$\mathbf{\Phi} \frac{d\mathbf{q}}{dt} = \mathbf{A\Phi q},$$

which yields

$$\frac{d\mathbf{q}}{dt} = \mathbf{\Phi}^{\dagger}\mathbf{A\Phi q}, \tag{34}$$

where $\mathbf{\Phi}^\dagger$ is the pseudo inverse of $\mathbf{\Phi}$. This reduced differential equation can then be with the projected initial condition $\mathbf{q}(0) \equiv \mathbf{\Phi}^\dagger \mathbf{p}(0)$ and solved using standard ODE integrators. One of the most powerful of these reduction strategies uses Krylov projections, where the projection operator is chosen as orthonormal basis spanning the space $[\mathbf{p}, \mathbf{Ap}, \mathbf{A^p}, \ldots, \mathbf{A}^m\mathbf{p}]$, which can be found efficiently using a modified implementation of Sidje's Expokit [60]. For time invariant problems, the SSIT automatically uses Expokit both for the solution of the original FSP (Eq. 9) and for the solution of reduced linear ODEs (Eq. 34).

In addition to Expokit, the SSIT provide three categories of methods to define $\mathbf{\Phi}$, each using a different model reduction strategy: (1) time-scale separations that average over fast state dynamics, (2) local approximations that assume smooth probabilities functions nearby regions of state-space, and (3) data-driven learning of distribution shapes based on previous solutions.

Time-Scale Separation Methods: The "Quasi-Steady State Assumption" (QSSA) approach assumes that one or more species are "fast" compared to the others. These species are then assumed to equilibrate to their steady state probability distributions (conditioned on the slow species) allowing for the calculation of average reaction rates for the remaining slower species [14]. The SSIT model reduction scheme "Eigen Decomposition" (ED) employs a sparse eigenvalue decomposition to find the N largest (i.e., least negative) eigenvalues of the infinitesimal generator matrix $\mathbf{A}$ and then projects the master equation onto this lower dimensional space.

State Space Smoothing Methods: The "Linear State Lumping" (LNSL) and "Logarithmic State Lumping" (LGSL) model reduction strategies form coarse rectangular grids with linearly or logarithmically distributed points (within the current FSP bounds) and then combines these multiple state variables into fewer "lumped" states [66]. All states within each lumped region are assumed to have equal probabilities. The "Lin Lump QSSA" (LNQSSA) and "Log Lump QSSA" (LGQSSA) methods combine this lumping strategy with the time scale separation assumption to treat the states within each at quasi-steady equilibrium and then averages to find the rates of transitions out of the lumped states.

Data-Driven Learning Methods: The "Proper Orthogonal Decomposition" (POD) approach uses one or more previous full solutions of the CME at discrete time points (e.g., at similar values during a parameter search) and applies singular value decomposition (SVD) to identify a lower-dimensional subspace that best represents the dominant probability mass structures of those solutions in the least-squares sense. In other words $\mathbf{\Phi}_{\text{PDO}}$ is an orthonormal matrix that spans the snapshot set $[\mathbf{p}(t_1), \mathbf{p}(t_2), ...]$. The "Dynamic Mode Decomposition" (DMD) approach also uses a set of previous solutions, but instead of identifying modes that maximize variance in the snapshot distributions, it constructs a best-fit linear operator that maps one time snapshot to the next. Specifically, it seeks to determine an operator $\mathbf{Q} \approx \exp(\mathbf{A}\Delta t)$ such that $\mathbf{p}(t_{k+1}) \approx \mathbf{Q}\,\mathbf{p}(t_k)$, and computes its eigen-decomposition. The resulting matrix $\mathbf{\Phi}_{\text{DMD}}$ has columns that approximate eigenvectors of $\mathbf{Q}$, so that $\mathbf{Q}\mathbf{\Phi}_{\text{DMD}} \approx \mathbf{\Phi}_{\text{DMD}}\mathbf{\Lambda}$. Thus, span$\{\mathbf{\Phi}_{\text{DMD}}\}$ approximates an invariant subspace of the evolution operator, rather than merely the span of the snapshots themselves. The method "POD2" combines both of these approaches and defines $\mathbf{\Phi}$ to span the combined set $[\mathbf{p}(t_1), ..., \mathbf{p}(t_N), \mathbf{Ap}(t_1), ..., \mathbf{Ap}(t_N)]$.

We note that for the current implementations of the SSIT, the QSSA, ED, LNQSSA, LGQSSA, DMD, POD2 methods assume that the infinitesimal generator matrix $\mathbf{A}$ is constant, and are only appropriate for time invariant problems. Because the Hog model example is time varying, the Supporting Information provides examples of the remaining three methods: POD, LNSL, and LGSL.

2.4.2 Hybrid ODE-FSP Solutions

The SSIT can assign upstream species to deterministic ODE solvers for efficiency, while maintaining full stochastic solutions for species of interest whose copy numbers are influenced by noise, in turn influencing the dynamics of the system (e.g., species with low copy numbers or switch-like species).

2.4.3 Data Distortion Handling

Single-cell experiments are highly versatile and modifiable, broad in scope with nearly limitless possible designs but also varying greatly in cost with respect to both time and expense [58] and, accordingly, the degree to which they may yield information about the underlying models [27] or introduce errors [29, 120].

Most of the time, the results of single-cell experiments are assumed to have negligible measurement and data processing errors and model development tends to assume ideal data as well [58]. However, while stochastic models are adopted with the idea of handling intrinsic noise, extrinsic noise can also have significant effects on inferred molecule counts and probability distributions [29]. Below are examples of the types measurement errors which may contribute to data noise and impact resulting analyses, model inference, and subsequent experimental design.

- Microscope resolution
- Photobleaching
- Optical filters
- Laser intensity
- Camera exposure time
- Imaging modality (z-stacking, max projection, etc.)
- Image processing (spot counting, intensity averaging, etc.)
- Delays due to drag/inducer diffusion or nuclear import

Capture inefficiencies such as allelic and transcript dropout in genotyping, single-cell DNA assays, and single-cell RNA sequencing are other examples of data distortions, yielding false zero or low counts for the dropout gene in the affected cell(s).

Vo *et al.* [29] laid the groundwork for quantifying and accounting for the effects of measurement and image processing errors on the accuracy and precision of model inference from single-cell fluorescence microscopy experimental data using conditional probability-based operators called *probabilistic distortion operators (PDOs)*. A PDO is simply a Markov kernel that transforms a distorted probability distribution, $\mathbf{p}_Y(t)$, computed using methods such as FSP (2.2.5) from data into which errors have been introduced by certain experiment design, measurement, or data processing choices, into a distribution more closely resembling the true distribution, $\mathbf{p}_X(t)$, or, more realistically, the distribution which would have been inferred from data containing fewer errors due to different design and/or processing choices.

If we considered a measurement distorted by noise taken at time t to be $y(t)$ realized from the vector $Y(t)$ which results from random distortions of the true process $X(t)$ [29], then the relationship between $\mathbf{p}_Y(t)$ and $\mathbf{p}_X(t)$ can be described by the linear transformation

$$\mathbf{p}_Y(t,\theta) = \Pr(Y(t) = y | X(t) = x, \theta)\mathbf{p}_X(t,\theta), \tag{35}$$

for each distorted measurement y and "true" measurement x, where Pr is either a probability mass or probability density depending on whether Y is discrete or continuous. The matrix $\mathbf{C}_{X\to Y}$ containing all possible observations y indexing its rows is $\mathbf{C}_{X\to Y}(t;\theta) := \Pr(Y(t) = y | X(t) = x, \theta)$, the PDO.

The steps of the data distortion handling (i.e., probability distribution transformation) process are as follows:

1. Collect molecular imaging data from chosen experiment.

2. Fit models to the data using the Finite State Projection (FSP) algorithm to efficiently solve Chemical Master Equations (CME) and yield the likelihood of the data and exact error.
3. Calibrate models to robustly handle measurement errors by computing and applying a distortion kernel $[C_{ij}]$.
4. Use Fisher information matrix (FIM) to determine how much information is obtainable from distorted experiments and approximate the sensitivity of the CME to changes in parameters.
5. Apply Bayesian experiment design methods to determine the most cost- and time-efficient experiments that can achieve sufficient information about model parameters and states of the single-cell systems under investigation.

In Box 21 and `example_Seq1_BatchFitManyGenes.m` we provide example code for fitting a Binomial PDO to scRNA-seq data [120]. We also include `example_14_ComplexModels_PDO.m` for an example on how to calibrate PDO parameters to data.

To illustrate the application of a *probabilistic distortion operator* (PDO) on a real dataset, we treat the single-cell RNA-seq measurements from Hoffman *et al.* [120] as distorted observations of the underlying, "true" transcript counts. In particular, we interpret the observed unique molecular identifier (UMI) counts as the result of imperfect capture and detection of true mRNA molecules and model this measurement process using the "missing spot" Binomial PDO [29].

From Eq. 35, the observation model can be written as

$$\mathbf{p}y(t,\theta) = \mathbf{C}_{\mathbf{X}\to\mathbf{Y}} \cdot \mathbf{p}_x(t,\theta), \tag{36}$$

where $\mathbf{C}_{\mathbf{X}\to\mathbf{Y}}$ is the PDO encoding the conditional probabilities of observing y UMIs given a true count x. Under the Binomial detection model, each column of $\mathbf{C}_{\mathbf{X}\to\mathbf{Y}}$ is given by

$$\mathbf{C}_{yx}(\theta) \;=\; P(y \mid x,\theta) \;=\; \binom{x}{y} p^{\,y}\,(1-p)^{\,x-y}, \qquad 0 \le y \le x, \tag{37}$$

with p representing the effective capture probability (and $1-p$ the corresponding missing-molecule rate). As before, a single row of the PDO may be denoted by $\mathbf{c}y = \mathbf{C}yx(\theta)$. The probability of observing a specific UMI count y_i at time t is then obtained by

$$P(y_i \mid t,\theta) = \mathbf{c}_{y_i} \cdot \mathbf{p}_x(t,\theta). \tag{38}$$

Droplet-based UMI scRNA-seq provides a sparse and incomplete sampling of cellular mRNA content; many transcripts present in the cell are not captured, reverse-transcribed, or sequenced, and low-copy transcripts are therefore frequently recorded as zeros. Hoffman *et al.* [120] reported $29,448$ UMIs per cell in their scRNA-seq measurements. A "typical" estimate of mRNAs/cell is 2×10^5, which would be $p \approx 29,448/200,000$. For our purposes, we adopt a conservative value of $p = 0.05$ (i.e., a 95% "dropout" or missing-molecule rate).

For the scRNA-seq data, we fix p, generate the PDO, and apply it to the model template prior to loading and fitting any data. As a case where one might one to calibrate a PDO - i.e., the hyperparameters that tune the conditional probability distribution - we also demonstrate the application of a Binomial PDO to the STL1 data [24] previously discussed. In the latter case, we use the nuclear mRNA count as the incomplete data with "missing" counts and the full-cell data as the "true" counts.

Example Box 21: Use Probabilistic Distortion Operators

An example of generating the Binomial PDO with an experimentally obtained failure probability 1 - p for the scRNA-seq [120] data is shown below (and in Fig. 21 *left*):

```
%% Set PDO to Binomial for RNA:
Model_Template.pdoOptions.type = 'Binomial';
Model_Template.pdoOptions.unobservedSpecies = 'onGene';
Model_Template.pdoOptions.props.CaptureProbabilityS1 = 0;    % Gene State
                                                             % not measured
Model_Template.pdoOptions.props.CaptureProbabilityS2 = 0.05; % 95% drop out
[~,Model_Template] = Model_Template.generatePDO();
```

And example of calibrating a Binomial PDO to data, in this case yeast STL1 [24] data, is shown below (Fig. 21 *right*):

```
STL1_4state_PDO_nuc = ...
 STL1_4state_PDO_nuc.calibratePDO('data/filtered_data_2M_NaCl_Step.csv',...
    {'mRNA'}, {'RNA_STL1_total_TS3Full'}, {'RNA_STL1_nuc_TS3Full'},...
    'Binomial', true, parGuess, {'Replica',1}, FontSize=24,...
    Title="4-state STL1 (PDO: Nuclear mRNA)", LegendLocation="northwest",...
    XLabel="Total mRNA counts", YLabel="Nuclear mRNA counts");
```

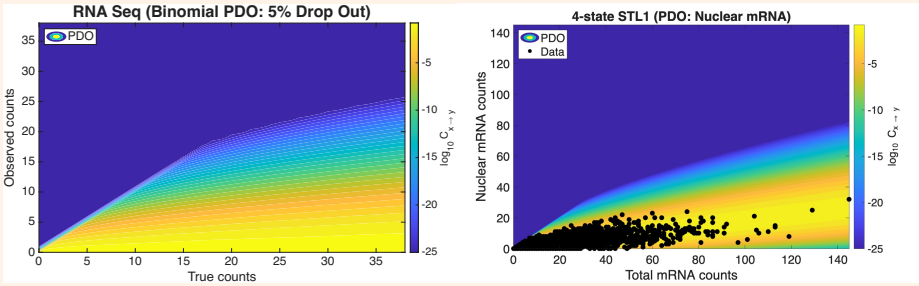

Figure B21: Conditional probability mass function contour maps of $\log_{10} C_{x \to y}$ for (left) the Binomial PDO with a conservative assumption of 95% dropout for scRNA-seq, and (right) a Binomial PDO calibrated to yeast STL1 data.

2.4.4 Fitting Multiple Models and datasets with Shared Parameters

SSITMultiModel is a class included within the SSIT package that allows for combined parameter fitting across multiple datasets or models. Each dataset is associated with its own instance of SSIT model but assumed to share a common subset of parameters. Examples that might benefit from SSITMultiModel include different experimental replicas or single-cell RNA seq (scRNA-seq) data from different genes.

In Box 22 and `example_scRNAseq_1_BatchFitManyGenes.m`, we show an example demonstrating the use of SSITMultiModel to fit shared upstream input signal parameters but distinct bursting gene parameters for four different genes from scRNA-seq data [120]. In the Supporting Information and `example_15_ComplexModels_MultiModel.m`, we include examples using SSITMultiModel to fit different subsets of parameters of the 4-state STL1 model (Box 1) to two different experimental replicas [24].

Example Box 22: Multi Model

```
% Select which models to include in multimodel:
Models = {Model_DUSP1,Model_RUNX1,Model_BIRC3,Model_TSC22D3};

% Define how parameters are assigned to sub-models:
ParInds = {[1,2,3:9],[1,2,7*1+(3:9)],[1,2,7*2+(3:9)],[1,2,7*3+(3:9)]};

% Set a constraint on model parameters:
Constraint = @(x) -var(log10([x(3:9);x(7*1+(3:9));x(7*2+(3:9));x(7*3+(3:9))]));

% Create and initialize multimodel:
combinedModel = SSITMultiModel(Models, ParInds, Constraint);
combinedModel = combinedModel.initializeStateSpaces();

% Fit the multimodel:
fitOptions = optimset('Display','iter','MaxIter',1000);
for i = 1:10
    [~,~,~,combinedModel] = combinedModel.maximizeLikelihood([],fitOptions);
    save('seqModels/CombinedModel4Genes','combinedModel');
end
```

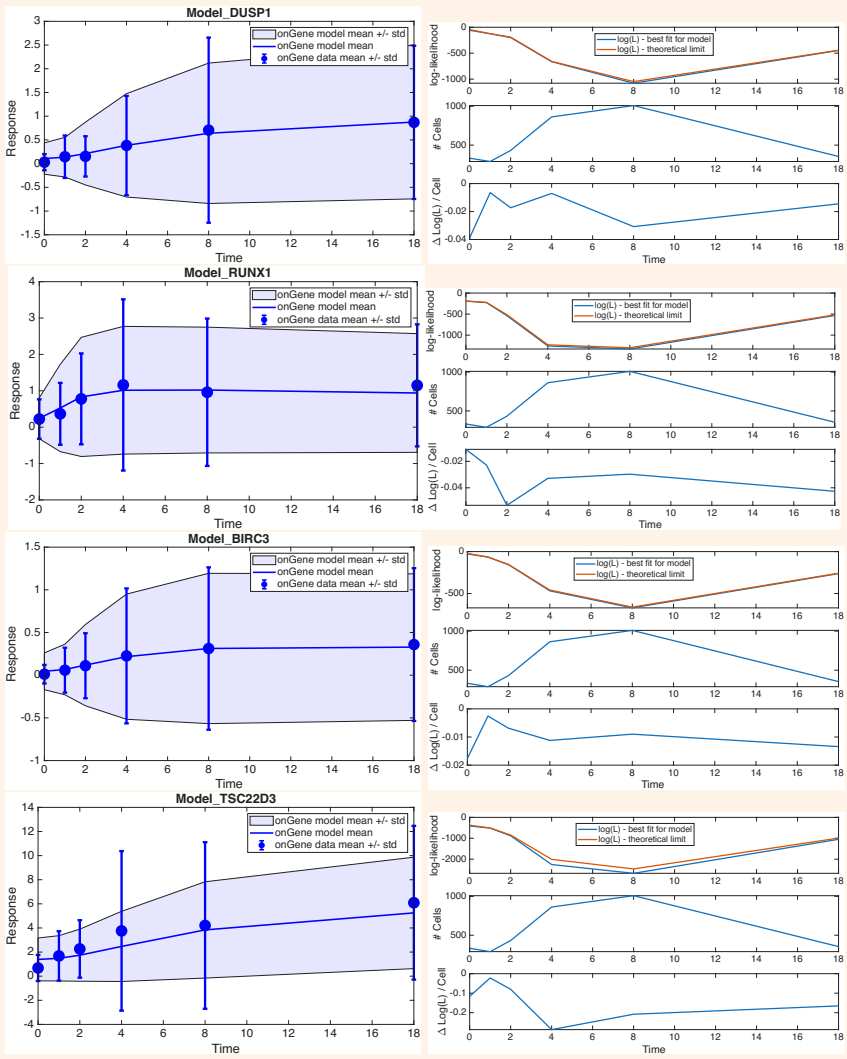

Figure B22: SSITMultiModel results from four different models, each representing a different gene (DUSP1, RUNX1, BIRC3, TSC22D3), fit to single-cell RNA sequencing data: (left) trajectories of means and standard deviations for each model and its associated data; (right) log-likelihoods, number of cells, and log-likelihoods/cell through time.

2.5 The SSIT can be extended to generate complex workflows using pipelines and cluster computing options.

The SSIT can generate condensed pipelines that make it easy to launch long or repeated analyses in the background or on a high-performance computing (HPC) cluster. More computationally intensive processes that are handy to compute in the form of an SSIT pipeline include sampling using Metropolis-Hastings (2.3.4, Supporting Information), cross-validation (2.3.6), and parameter sweeps; these can be exported to a stand-alone command that runs non-interactively under the operating system's job scheduler.

The SSIT provides a utility function `generateCommandLinePipeline(saveFileIn, modelName, dummy, Pipeline, pipelineArgs, saveFileOut, logFile, runNow, runOnCluster, clusterPrefix)`. This function serializes the model and pipeline configuration into a single MATLAB command-line call of the form:

```
matlab -nodisplay -nosplash -nodesktop -r "addpath(...);
SSIT(saveFileIn,modelName,[],Pipeline,pipelineArgs,saveFileOut);
exit;"matlab -nodisplay -nosplash -nodesktop -r "addpath(...);
SSIT(saveFileIn, modelName, [], Pipeline, pipelineArgs, saveFileOut);
exit;"
```

and returns it as a shell command string. The `pipelineArgs` input is converted into a MATLAB `struct` and passed directly to the named pipeline function, ensuring that the same configuration can be used interactively and in batch mode. Before execution, the command automatically adds the SSIT root directory and the temporary propensity-function directory (`tmpPropensityFunctions`) to the MATLAB path, so the job is self-contained and does not rely on a user's interactive MATLAB session.

When `runOnCluster = false`, `generateCommandLinePipeline` builds a command that calls the local MATLAB executable (determined from `matlabroot`), redirects standard output and errors to the specified log file, and runs the job in the background. This is useful for launching long-running analyses from a workstation or login node without blocking the MATLAB session. When `runOnCluster = true`, the function instead wraps the MATLAB call in a user-specified cluster prefix string `clusterPrefix`. By default, SSIT is configured for Slurm-based clusters with a prefix

```
sbatch --cpus-per-task=1 --output=#LOG --error=#ERR --wrap="module load
matlab-2022b; srun matlab -nodisplay -nosplash -nodesktop -r sbatch
--cpus-per-task=1 --output=#LOG --error=#ERR --wrap="module load
matlab-2022b; srun matlab -nodisplay -nosplash -nodesktop -r
```

where the placeholders `#LOG` and `#ERR` are automatically replaced by the chosen log file and a corresponding error log (prefixed with `err_`). Users can customize `clusterPrefix` to match their local scheduler (e.g., adding memory or time limits, changing the MATLAB module name, or swapping `sbatch`/`srun` for other job-launch commands). The resulting string is a complete `sbatch` command that can either be copied and pasted into a terminal or executed directly from MATLAB by setting `runNow = true`, in which case SSIT calls `system(cmd)`.

Box 23 and `example_scRNAseq_2_BatchFitManyGenes.m` provide example code to generate pipelines for fitting models to data. These pipelines can then be run from within the MATLAB environment, from a local command line, or on a high-performance cluster. The Supporting Information and `example_scRNAseq_3_BatchFitManyGenes.m` show code for generating a figure (Fig. 23) to compare the fitted parameters across scRNA-seq genes.

Example Box 23: Generate and Run Pipelines (Local and HPC)

Below is example code for generating a pipeline to fit all 151 genes from the 500 most highly expressed genes selected from the scRNA-seq dataset provided by Hoffman *et al.* [120]. Fig. 23 shows the mean fold change for comparison of the key parameters K_{ON} and K_{OFF} between all genes, each of which was fit as its own model for bursting RNA transcription. (The code for generating Fig. 23 is shown in the Supporting Information.)

```
%% Call Pipeline to Fit Model
% Specify pipeline to apply to model and arguments
% ("../src/examplePipelines/fittingPipelineExample.m")
Pipeline = 'fittingPipelineExample';
pipelineArgs.maxIter = 1000;
pipelineArgs.display = 'iter';
pipelineArgs.makePlot = false;
pipelineArgs.nRounds = 1;

%% Launch cluster jobs for all genes.
DataFileName = 'data/Raw_DEX_UpRegulatedGenes_ForSSIT.csv';
TAB = readtable(DataFileName);
geneNames = fields(TAB);

for iRound = 1:5
    for iGene = 2:length(geneNames)-4
        modelName = ['Model_',geneNames{iGene}];
        saveName = ['seqModels/',modelName];
        logfile = ['logFiles/log',modelName];
        if iGene==2
            load(saveName)
            eval([modelName,'.formPropensitiesGeneral;']);
        end
        cmd = SSIT.generateCommandLinePipeline(saveName,modelName,...
              Pipeline, pipelineArgs=pipelineArgs,saveFileOut=saveName,...
              logFile=logfile,runNow=true,runOnCluster=true);
        pause(60); % Pause to allow for matlab licenses to reset.
    end
end
```

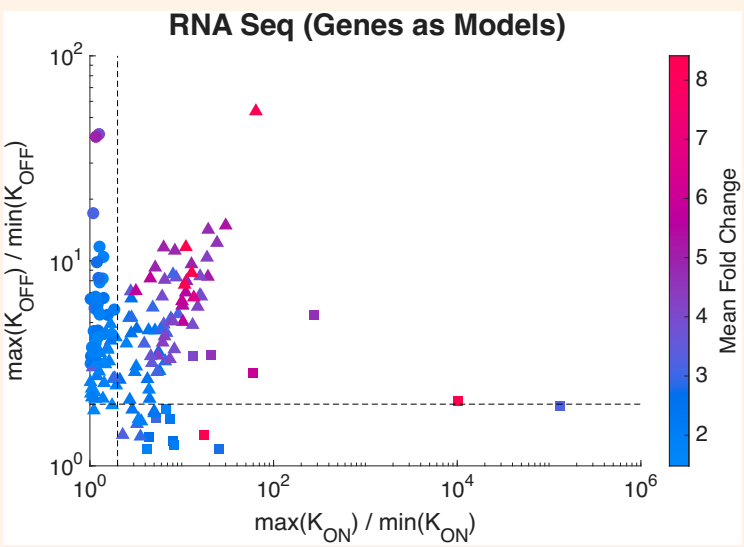

Figure B22: SSITMultiModel results from four different models, each representing a different gene (DUSP1, RUNX1, BIRC3, TSC22D3), fit to single-cell RNA sequencing data: (left) trajectories of means and standard deviations for each model and its associated data; (right) log-likelihoods, number of cells, and log-likelihoods/cell through time.

2.6 The SSIT provides a graphical user interface (GUI) for novice users.

In addition to its application programming interface (API), the SSIT also provides a graphical user interface (GUI) built in MATLAB for interactive utilization of the main modeling and inference workflows. The SSIT GUI offers an intuitive and easy-to-use button- and tab-based environment for users rather than requiring the writing and working with scripts that can be extensive in some cases. Instead, users can load preset models or define or define their own species, parameters, propensities, stoichiometry, input expressions, and initial conditions (2.1); load, filter, and associate experimental or simulated data (2.3.1); perform simulations and solve probability distributions by ODEs (2.2.2), SSA (2.2.4), FSP (2.2.5), and hybrid methods (2.4.2); set likelihood (2.3.2) or loss functions (2.3.5); perform parameter inference through maximum likelihood estimation (2.3.3), Bayesian inference (2.3.4), or Approximate Bayesian Computation (2.3.5); compute sensitivities (2.2.7) and FIM (2.2.8); and visualize results, including simulated trajectories and FSP distributions, comparisons with empirical data, model fits and predictions, uncertainty measurements, and summary diagnostics and statistics. Models and settings specified in the GUI can be saved and used for command-line execution or batch scripts.

3 Discussion

The Stochastic System Identification Toolkit (SSIT) offers a platform for fast, accurate, and versatile mechanistic modeling solutions, parameter inference, information and uncertainty quantification, distortion handling, and quantitative experimental design. This tight integration between forward solvers, inference algorithms, and assessments of both model and experimental data is a central practical advantage of the SSIT.

3.1 Stochastic modeling and inference

Stochastic modeling is crucial for systems that experience significant fluctuations from mean behavior through time, thus affecting the inference of the underlying model and rate parameters which drive the system. Such systems include those with low copy numbers and high intrinsic noise. In the SSIT, simple, commonly used mechanistic models such as birth-death and toggle switch are already specified and readily called. Models can also be built from scratch by defining species, stoichiometry, and propensity functions, and the SSIT allows for highly complex functions (including logical statements, etc.) and time-varying input signals (2.3.1).

The Chemical Master Equation (CME) is a fundamental approach for describing the time evolution of probability distributions over discrete system states (2.2.1). Unfortunately, most interesting systems involve an infinite state space or a finite state space that is too large to produce computationally tractable numerical solutions. The Finite State Projection (FSP) approximation (2.2.5) truncates the full CME according to an modifiable error tolerance and computes not only the rigorously bounded subspace but also keep track of the total absolute error resulting from the loss of probability to sink states. Other important properties of the FSP lead to exact calculation of escape times and likelihood bounding for model fitting and parameter inference. FSP is therefore a powerful tool for modeling stochastic systems, but its computational efficiency and scalability depend on the specific implementation of its state space truncation routine and evolution through time.

The SSIT specializes in producing efficient solutions of FSP truncations to the CME and integrating them into broader workflows for parameter inference, information and

uncertainty quantification, and experiment design. The SSIT has also been carefully designed to efficiently simulate trajectories using parallel and GPU implementations of the Stochastic Simulation Algorithm (SSA) for exact solutions (2.2.4) and optionally averages system dynamics over time by deterministic ordinary differential equations (ODEs) - supposing large, well-mixed populations with negligible fluctuation or randomness (2.2.2) - hybridized deterministic-stochastic solutions (2.4.2), or moment closure approximations (2.2.3).

The SSIT provides full statistical inference tools for fitting mechanistic models to experimental data and estimating parameters, including maximum likelihood estimation (MLE) (2.3.3) for point estimation, Bayesian posterior sampling (2.3.4), and likelihood-free Approximate Bayesian Computation (2.3.5).

3.2 Sensitivity analysis, Fisher Information, and experiment design

The SSIT pairs probability distributions over system states with parameter sensitivity calculations (2.2.7) to compute Fisher Information (2.2.8), which quantifies the amount of information accessible about the values of the model parameters and can thus be used for informative experiment design (2.2.9). By computing Fisher Information matrices (FIM) over multiple realistic experimental and measurement constraints and using one of various scalar optimality criteria (2.2.9), the SSIT enables the tailoring of designs to achieve specific goals such as: minimizing overall parameter uncertainty, avoiding poorly identified directions, maximizing predictive accuracy over design space, or discriminating between competing mechanistic models/hypotheses. In practice, users can specify an experimental design variable (e.g., drug dosage, measurement times, or number of cells), and the SSIT evaluates FIM-based design scores by propagating FSP solutions and their sensitivities. The design scores can then be used in the SSIT within search or optimization routines to propose more informative experiments, or within sequential experiment design loops where the design is updated after each round of data collection and inference.

The pipeline from data and model to experiment design is fully transparent in the SSIT, and unlike in other popular methods [138], this transparency includes uncertainty measurements (2.3.4-2.3.6), model assessment and comparison, tunable distortion handling (2.4.3), and the option to automatically generate both local and cluster computing pipeline scripts (2.5).

3.3 Model assessment

The SSIT includes tools to critically assess model fit and to quantify parameter uncertainty. Likelihoods (2.3.2), Bayes factors, and Akaike and Bayesian Informaton Criteria are all comparable using varying mechanistic models and/or independent experimental measurements across different time points and experimental conditions. Building on this, the SSIT's cross-validation module (2.3.6) provides a systematic way to assess predictive performance: data are partitioned into training and test subsets, MLEs (2.3.3) are obtained on each training partition, and their predictive log-likelihoods are evaluated on held-out data. The resulting cross-validation error summarizes how well the inferred model generalizes to unseen data and can reveal overfitting or model misspecification.

Additionally, the SSITMultiModel class (2.4.4) and cross-validation multi-model creator enable fitting the same mechanistic model to multiple replicas, doses, genetic variants, or other criteria while allowing selected parameters to vary between groups. The log-normal penalty formulation on replica-to-replica parameter deviations provides a principled way to encode prior expectations about batch variation or biological

heterogeneity. This framework allows users to quantify which parameters are stable across conditions and which are sensitive to experimental context, and to test hypotheses about shared vs. condition-specific mechanisms.

3.4 Model reduction

For many realistic systems, the CME or FSP state space can be extremely large and an infeasible number of SSA trajectories are required for statistical significance. The SSIT therefore provides several model reduction strategies (2.4.1) that can be applied to accelerate both simulation and inference while preserving key dynamic behavior of the system.

The FSP itself acts as a controlled truncation of the CME, limiting the state space while bounding the truncation error. In addition, hybrid deterministic–stochastic methods (2.4.2) allow users to treat specified species (e.g., highly abundant species or those that undergo relatively fast reactions) with upstream reactions deterministically via ODEs, while computing fully stochastic realisations for others. This division can dramatically reduce computational cost without sacrificing fidelity in the most informative parts of the system.

Other model reduction (2.4.1) options include quasi-steady-state approximations (QSSA) for fast intermediates, lumping strategies (e.g., Log Lump QSSA) to aggregate states or compartments, and projection-based methods such as eigen-decomposition, proper orthogonal decomposition (POD), or dynamic mode decomposition (DMD) to construct low-dimensional bases from FSP solutions.

3.5 Distortion handling

The SSIT also contains options for handling extrinsic noise due to cell-to-cell variability or to measurement inaccuracies. Analysis of upstream cellular variability is achieved by sampling parameters over arbitrary distributions, whereas downstream measurement noise is analyzed using probabilistic distortion operators (PDOs, 2.4.3). In particular, the latter type of extrinsic noise - introduced by experimental protocols, measurement instrumentation or procedures, and downstream data-processing choices - distorts the observed data, subsequently transforming the corresponding probability distributions describing changes to the system states through time, and thereby interfering with the accuracy and precision of parameter inference, mechanistic insights, and predictions. Fortunately, in many cases, particular distortions introduced by cheaper, faster designs are systematic and therefore learnable and revertible to less distorted distributions by way of conditional probability. In the SSIT, such conditional distributions can be used as operators (i.e., PDOs) that map latent state distributions to observed data distributions through calibration from reference data. Users can choose from built-in parametric forms (e.g., affine-Poisson) or define custom distortion models.

PDOs can be incorporated directly into the SSIT modeling pipeline (2.5) to improve parameter estimates (2.3.2-,2.3.5), uncertainty quantification (2.3.4-2.3.6), and FSP-FIM-based experiment design (2.2.9). The SSIT thus jointly accounts for intrinsic stochasticity as well as additional extrinsic distortion when fitting models, making predictions, and designing subsequent experiments.

3.6 Pipelines, cluster computing, and the SSIT GUI

The SSIT generates simple, customizable pipelines (2.5) for execution in the background on local machines with optional parallelization or wrapped inside a schedular command (e.g., `sbatch` on `slurm`) to be run on cluster nodes. These scripts are particularly useful for analyses that involve larger systems with numerous parameters to fit, many forward

simulations, long sampling chains to convergence, and multiple experiment design considerations.

The SSIT graphical user interface (GUI) (2.6) allows users to load preset models or define their own (2.1); load experimental or simulated data (2.3.1); simulate and solve models using ODEs (2.2.2), SSA (2.2.4), FSP (2.2.5), and hybrid methods (2.4.2); define likelihood (2.3.2); obtain point estimates in parameter space through maximum likelihood estimation (2.3.3) or infer posterior distributions by Bayesian inference (2.3.4); perform sensitivity analysis (2.2.7); compute FIMs (2.2.8); and optimize experiments (2.2.9); and visualize results, including simulated trajectories and FSP distributions, comparisons with empirical data, model fits and predictions, uncertainty measurements, and summary diagnostics and statistics. Models and settings specified in the GUI can be saved and used for command-line execution or batch scripts.

3.7 Data analyses

In this paper, we demonstrated features of the SSIT on two experimental datasets consisting of RNA count data from: (a) *Saccharomyces cerevisiae* yeast cells, focusing on the STL1 gene, in response to NaCl [24]; and (b) 151 genes from breast cancer cells in response, measured by single-cell RNA sequencing (scRNA-seq), to dexamethasone [120]. The STL1 data was modeled and fit using a complex input expression (Box 4) and four distinct gene states, each with their own transcription rate. The scRNA-seq data analysis made use of the SSIT's complex modeling capabilities, including the incorporation of a binomial probabilistic distortion operator to handle dropout and other causes of a failure to detect counts as well as SSITMultiModel, which fit multiple genes with two shared parameters and seven distinct parameters per gene.

3.8 Comparison to other software

Tables 1 and 2 summarize a representative selection of software tools for simulating and analyzing reaction networks. Table 1 emphasizes the diversity of approaches, with many focusing primarily on deterministic ordinary differential equation (ODE) models, Gillespie's Stochastic Simulation Algorithm (SSA) for sampling trajectories, or moment-based approximations. Comparatively few packages offer direct numerical solutions of the Chemical Master Equation (CME) and fewer still offer efficient implementations of the Finite State Projection [13] method for approximating the CME with rigorous error bounds, let alone fully developed pipelines from modeling and data processing to full statistical inference, distortion handling, and experiment design. Table 2 provides a more detailed comparison between the software in Table 1 that support FSP-based CME solutions, highlighting these additional tools offered by the SSIT.

Packages like Catalyst [37] (with FiniteStateProjection.jl) and CERENA [38] provide strong CME/ODE/moment capabilities, while CmePy [139] focuses narrowly on FSP and StochSS [52]/URDME [54] target SSA and diffusive FSP for spatial RDMEs. The SSIT complements these tools by combining (a) direct CME solutions by efficient FSP computations, iteratively expanding and contracting the truncation bounds while recomputing the resulting error, with (b) full statistical inference using likelihoods or posterior probability sampling, (c) experiment design using FSP-FIM and Bayesian utilities, (d) model-reduction & acceleration tools (hybrid ODE–FSP, POD, sparse grids, quasi-steady-state, etc.), and (e) explicit handling of extrinsic noise to allow for cost effective, efficient experimentation. Unlike model-construction libraries (e.g., BioCRNpyler [140], PySB [141]) or constraint-based frameworks (e.g., COBRA [142]), the SSIT is purpose-built for stochastic, mechanistic, time-evolving analyses and the data-model alignment issues that arise in modern experiments.

3.9 Reproducibility and interoperability

The SSIT reads models in the XML format of SBML [57, 143] and can interoperate with SimBiology [50]. The SSIT can be run from command line, through its own Graphical User Interface, or interactively within the MATLAB environment. The SSIT supports reproducible analyses with explicit random-seed control, versioned example scripts, and automated testing. To further ensure reproducibility, the SSIT contains a set of automated tests that check SSIT command line results against known analytical results, and these that are executed every time a push is made to SSIT on GitHub, and test results are visible to the public at: `https://github.com/MunskyGroup/SSIT/actions`. The graphical functions of the SSIT GUI can be tested by running the function `testGui.m` in the `tests` folder.

3.10 Limitations and future directions

At present, the SSIT targets well-mixed systems. The current version does not yet include spatial Reaction-Diffusion Master Equation models or DFSP, although diffusive modeling in the SSIT is currently under development. Second, while hybridization, POD/sparse-grid surrogates, and N-step/SSA-guided expansion extend FSP to moderately large systems, very high-dimensional networks can still be challenging.

Finally, the SSIT's core implementation in MATLAB eases integration with SimBiology/SBML pipelines and allows for the use of built-in toolkits to expand the SSIT's capabilities in the future, but MATLAB may not match the raw performance of low-level Julia/C++ stacks under some regimes. Wrapping core solvers in Julia/C++ could broaden performance and providing portable Python-based workflows may allow for broader adoption.

4 Conclusion

The Stochastic System Identification Toolkit (SSIT) is an open-source, versatile, actively developed software package based in MATLAB that provides an integrated pipeline from (i) mechanistic model building and simulation; to (ii) data importation, filtering by condition, and connection to model species; to (iii) model fitting/parameter inference based on likelihood, Bayesian, and simulation/loss function; to (iv) data distortion handling conditional probability operators [29]; to (v) parameter sensitivity analysis, calculation of Fisher Information [27], and experiment design using FSP-FIM [27]. Additionally, the SSIT includes several model reduction techniques and options for automated parallelization within the MATLAB environment, on command line, or via high-performance cluster computing to increase speed.

The SSIT makes use of built-in MATLAB packages while remaining fully open-source and readily accessible, with full documentation and detailed examples. The source code is freely available with README, download and installation instructions, and example scripts for every tool discussed in this paper at: `http://github.com/MunskyGroup/SSIT/`.

5 Acknowledgments

This research was supported by the NIH National Institute for General Medical Sciences (Award R35GM124747) and the National Science Foundation (Award MCB-1941870). The authors would like to thank Rachel Keating, Torin Moore, Stuart McKnight, Ethan

Green, Eric Ron, and Joshua Cook for feedback and assistance with testing throughout the development of the the SSIT.

References

1. Elowitz MB, Levine AJ, Siggia ED, Swain PS. Stochastic gene expression in a single cell. Science. 2002;297(5584):1183–1186. doi:10.1126/science.1070919.

2. Swain PS, Elowitz MB, Siggia ED. Intrinsic and extrinsic contributions to stochasticity in gene expression. Proceedings of the National Academy of Sciences of the United States of America. 2002;99(20):12795–12800. doi:10.1073/PNAS.162041399.

3. Paulsson J. Models of stochastic gene expression. Physics of Life Reviews. 2005;2(2):157–175. doi:10.1016/j.plrev.2005.03.003.

4. McAdams HH, Arkin A. Stochastic mechanisms in gene expression. Proceedings of the National Academy of Sciences of the United States of America. 1997;94(3):814–819. doi:10.1073/pnas.94.3.814.

5. Mcquarrie DA, Gillespie DT. Stochastic theory and simulations of chemical kinetics. J Appl Probab. 1967;478(413-478):1140.

6. Gillespie DT. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. Journal of Computational Physics. 1976;22(4):403–434. doi:https://doi.org/10.1016/0021-9991(76)90041-3.

7. Gillespie DT. Exact stochastic simulation of coupled chemical reactions. The Journal of Physical Chemistry. 1977;81(25):2340–2361. doi:10.1021/j100540a008.

8. Wilkinson DJ. Stochastic modelling for quantitative description of heterogeneous biological system. Nature Reviews Genetics. 2009;10(2):122–133.

9. Gupta A, Mikelson J, Khammash M. A finite state projection algorithm for the stationary solution of the chemical master equation. The Journal of Chemical Physics. 2017;147(15):154101. doi:10.1063/1.5006484.

10. Cao Y, Terebus A, Liang J. State space truncation with quantified errors for accurate solutions to discrete Chemical Master Equation. Bulletin of Mathematical Biology. 2016;78(4):617–661. doi:10.1007/s11538-016-0149-1.

11. Cao Y, Terebus A, Liang J. Accurate Chemical Master Equation Solution Using Multi-Finite Buffers. Multiscale Modeling & Simulation. 2016;14(2):923–963. doi:10.1137/15M1034180.

12. Kim J, Dark J, Enciso G, Sindi S. Slack reactants: A state-space truncation framework to estimate quantitative behavior of the chemical master equation. The Journal of Chemical Physics. 2020;153(5):054117. doi:10.1063/5.0014736.

13. Munsky B, Khammash M. The finite state projection algorithm for the solution of the chemical master equation. Journal of Chemical Physics. 2006;124(4):44104. doi:10.1063/1.2145882/561868.

14. Peleš S, Munsky B, Khammash M. Reduction and Solution of the Chemical Master Equation Using Time Scale Separation and Finite State Projection. The Journal of Chemical Physics. 2006;125(20):204104. doi:10.1063/1.2397685.

15. Fox Z, Neuert G, Munsky B. Finite state projection based bounds to compare chemical master equation models using single-cell data. The Journal of chemical physics. 2016;145(7):074101.

16. Munsky B, Khammash M. A multiple time interval finite state projection algorithm for the solution to the chemical master equation. Journal of Computational Physics. 2007;226(1):818–835.

17. Wolf V, Goel R, Mateescu M, Henziner TA. Solving the chemical master equation using sliding windows. BMC Systems Biology. 2010;4(42). doi:10.1186/1752-0509-4-42.

18. Burrage K, Hegland M, Macnamara S, Sidje R, et al. A Krylov-based finite state projection algorithm for solving the chemical master equation arising in the discrete modelling of biological systems. In: Proc. of The AA Markov 150th Anniversary Meeting. 21-37; 2006.

19. Munsky B, Khammash M. The Finite State Projection Approach for the Analysis of Stochastic Noise in Gene Networks. IEEE Transactions on Automatic Control. 2008;53:201–214. doi:10.1109/TAC.2007.911361.

20. Vo HD, Fox ZR, Baetica AA, Munsky BE. Bayesian Estimation for Stochastic Gene Expression Using Multifidelity Models. The Journal of Physical Chemistry B. 2019;123(10):2217–2234. doi:10.1021/acs.jpcb.8b10946.

21. Kazeev V, Khammash M, Nip M, Schwab C. Direct Solution of the Chemical Master Equation Using Quantized Tensor Trains. PLOS Computional Biology. 2014;doi:10.1371/journal.pcbi.1003359.

22. Dinh T, Sidje RB. An adaptive solution to the chemical master equation using quantized tensor trains with sliding windows. Physical Biology. 2020;17(6):065014. doi:10.1088/1478-3975/aba1d2.

23. Lou C, Stanton B, Chen YJ, Munsky B, Voigt CA. Ribozyme-based insulator parts buffer synthetic circuits from genetic context. Nature Biotechnology. 2012;30(11):1137–1142. doi:10.1038/nbt.2401.

24. Neuert G, Munsky B, Tan RZ, Teytelman L, Khammash M, Van Oudenaarden A. Systematic identification of signal-activated stochastic gene regulation. Science. 2013;339(6119):584–587.

25. Munsky B, Fox Z, Neuert G. Integrating single-molecule experiments and discrete stochastic models to understand heterogeneous gene transcription dynamics. Methods. 2015;85:12–21.

26. Munsky B, Li G, Fox ZR, Shepherd DP, Neuert G. Distribution shapes govern the discovery of predictive models for gene regulation. Proceedings of the National Academy of Sciences. 2018;115(29):7533–7538. doi:10.1073/pnas.1804060115.

27. Fox ZR, Munsky B. The finite state projection based Fisher information matrix approach to estimate information and optimize single-cell experiments. PLOS Computational Biology. 2019;15(1):e1006365. doi:10.1371/JOURNAL.PCBI.1006365.

28. Kalb D, Vo HD, Adikari S, Hong-Geller E, Munsky B, Werner J. Visualization and modeling of inhibition of IL-1beta and TNF-alpha mRNA transcription at the single-cell level. Scientific Reports. 2021;11(1):13692. doi:10.1038/s41598-021-92846-0.

29. Vo H, Forero-Quintero LS, Aguilera LU, Munksy B. Analysis and design of single-cell experiments to harvest fluctuation information while rejecting measurement noise. Frontiers in Cell and Developmental Biology. 2023;11. doi:10.3389/fcell.2023.1133994.

30. Warne DJ, Baker RE, Simpson MJ. Simulation and Inference Algorithms for Stochastic Biochemical Reaction Networks: From Basic Concepts to State-of-the-Art. Journal of the Royal Society Interface. 2019;16(151):20180943. doi:10.1098/rsif.2018.0943.

31. Karr JR, Malik-Sheriff RS, Osborne J, Gonzalez-Parra G, Forgoston E, Bowness R, et al. Model Integration in Computational Biology: The Role of Reproducibility, Credibility and Utility. Frontiers in Systems Biology. 2022;2:822606. doi:10.3389/fsysb.2022.822606.

32. Puniya BL, Verma M, Damiani C, Bakr S, Dräger A. Perspectives on Computational Modeling of Biological Systems and the Significance of the SysMod Community. Bioinformatics Advances. 2024;4(1):vbae090. doi:10.1093/bioadv/vbae090.

33. Sanft KR, Wu S, Roh M, Fu J, Lim RK, Petzold LR. StochKit2: software for discrete stochastic simulation of biochemical systems with events. Bioinformatics. 2011;27(17):2457–2458. doi:10.1093/bioinformatics/btr401.

34. Fröhlich F, Weindl D, Schälte Y, Pathirana D, Paszkowski Ł, Lines GT, et al. AMICI: High-Performance Sensitivity Analysis for Large Ordinary Differential Equation Models. Bioinformatics. 2021;doi:10.1093/bioinformatics/btab227.

35. Balsa-Canto E, Henriques D, Gábor A, Banga JR. AMIGO2, a toolbox for dynamic modeling, optimization and control in systems biology. Bioinformatics. 2016;32(21):3357–3359. doi:10.1093/bioinformatics/btw411.

36. Harris LA, Hogg JS, Tapia JJ, Sekar JAP, Gupta S, Korsunsky I, et al. BioNetGen 2.2: advances in rule-based modeling. Bioinformatics. 2016;doi:10.1093/bioinformatics/btw469.

37. Loman TE, Ma Y, Ilin V, Gowda S, Korsbo N, Yewale N, et al. Catalyst: Fast and flexible modeling of reaction networks. PLOS Computational Biology. 2023;19(10):1–19. doi:10.1371/journal.pcbi.1011530.

38. Kazeroonian A, Fröhlich F, Raue A, Theis FJ, Hasenauer J. CERENA: ChEmical REaction Network Analyzer—A Toolbox for the Simulation and Analysis of Stochastic Chemical Kinetics. PLoS One. 2016;doi:10.1371/journal.pone.0146732.

39. Hoops S, Sahle S, Gauges R, Lee C, Pahle J, Simus N, et al. COPASI—a COmplex PAthway SImulator. Bioinformatics. 2006;22. doi:10.1093/bioinformatics/btl485.

40. Raue A, Steiert B, Schelker M, Kreutz C, Maiwald T, Hass H, et al. Data2Dynamics: a modeling environment tailored to parameter estimation in dynamical systems. Bioinformatics. 2015;31(21):3558–3560. doi:10.1093/bioinformatics/btv405.

41. Boersch-Supan PH, Ryan SJ, Johnson LR, Fitzjohn R. deBInfer: Bayesian inference for dynamical models of biological systems in R. Methods in Ecology and Evolution. 2017;8:511–518. doi:10.1111/2041-210X.12679.

42. Ramsey S, Orrell D, Bolouri H. Dizzy: Stochastic Simulation of Large-Scale Genetic Regulatory Networks. Journal of Bioinformatics and Computational Biology. 2005;3(2):415–36. doi:10.1142/s0219720005001132.

43. Kaschek D, Mader W, Fehling-Kaschek M, Rosenblatt M, Timmer J. Dynamical Modeling, Parameter Estimation, and Uncertainty Analysis in R. Journal of Statistical Software. 2019;88. doi:10.18637/jss.v088.i10.

44. Matthew S, Carter F, Cooper J, Dippel M, Green E, Hodges S, et al. GillesPy2: A Biochemical Modeling Framework for Simulation Driven Biological Discovery. Letters in Biomathematics. 2023;10:87–103.

45. Watanbe L, Nguyen T, Zhang M, Zundel Z, Zhang Z, Madsen C, et al. iBioSim 3: A Tool for Model-Based Genetic Circuit Design. ACS Synthetic Biology. 2018;8. doi:10.1021/acssynbio.8b00078.

46. Sequeiros C, Pájaro M, Vázquez C, Banga JR, Otero-Muras I. IDESS: a toolbox for identification and automated design of stochastic gene circuits. Bioinformatics. 2023;39(11):btad682. doi:10.1093/bioinformatics/btad682.

47. Thomas P, Matuscheck H, Grima R. Intrinsic Noise Analyzer: A Software Package for the Exploration of Stochastic Biochemical Kinetics Using the System Size Expansion. PLOS Computational Biology. 2012;doi:10.1371/journal.pone.0038518.

48. Cravo F, Prakash G, Függer M, Nowak T. MobsPy: A programming language for biochemical reaction networks. PLOS Computational Biology. 2025;21(5):1–33. doi:10.1371/journal.pcbi.1013024.

49. van Aalst M, Ebenhöh, Matuszyńska A. Constructing and analysing dynamic models with modelbase v1.2.3: a software update. BMC Bioinformatics. 2021;22. doi:10.1186/s12859-021-04122-7.

50. MathWorks. SimBiology Documentation; 2025. `https://www.mathworks.com/products/simbiology.html`.

51. Maarleveld TR, Olivier BG, Bruggeman FJ. StochPy: A Comprehensive, User-Friendly Tool for Simulating Stochastic Biological Processes. PLOS ONE. 2013;8(11):1–10. doi:10.1371/journal.pone.0079345.

52. Drawert B, Hellander A, Bales B, Banerjee D, Bellesia G, Daigle Jr BJ, et al. Stochastic Simulation Service: Bridging the Gap between the Computational Expert and the Biologist. PLoS Computational Biology. 2016;doi:10.1371/journal.pcbi.1005220.

53. Choi K, Medley JK, Kö"nig M, Stocking K, Smith L, Gu S, et al. Tellurium: An Extensible Python-based Modeling Environment for Systems and Synthetic Biology. Biosystems. 2019;171:74–79. doi:10.1016/j.biosystems.2018.07.006.

54. Drawert B, Engblom S, Hellander A. URDME: a modular framework for stochastic simulation of reaction-transport processes in complex geometries. BMC Systems Biology. 2012;6. doi:10.1186/1752-0509-6-76.

55. Voigt CA. Synthetic biology 2020–2030: six commercially-available products that are changing our world. Nature Communications. 2020;11(1):6379. doi:10.1038/s41467-020-20122-2.

56. Hucka M, Finney A, Sauro HM, Bolouri H, Doyle JC, Kitano H, et al. The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. Bioinformatics. 2003;19(4):524–531. doi:10.1093/bioinformatics/btg015.

57. SBML Team. The Systems Biology Markup Language (SBML); 2024. `https://sbml.org`.

58. Cook J, Ron E, Svetlov D, Aguilera LU, Munsky B. Sequential design of single-cell experiments to identify discrete stochastic models for gene expression. In: 2024 IEEE 63rd Conference on Decision and Control (CDC). Milan, Italy: IEEE; 2024. p. 7778–7785.

59. Bader BW, Kolda TG. Efficient MATLAB Computations with Sparse and Factored Tensors. SIAM Journal on Scientific Computing. 2007;30(1):205–231. doi:10.1137/060676489.

60. Sidje RB. EXPOKIT: A Software Package for Computing Matrix Exponentials. ACM Transactions on Mathematical Software. 1998;24(1):130–156. doi:10.1145/285861.285868.

61. Gillespie DT. A rigorous derivation of the chemical master equation. Physica A: Statistical Mechanics and its Applications. 1992;188(1-3):404–425. doi:10.1016/0378-4371(92)90283-V.

62. Toni T, Welch D, Strelkowa N, Ipsen A, Stumpf MPH. Approximate Bayesian computation scheme for parameter inference and model selection in dynamical systems. Journal of the Royal Society Interface. 2009;6(31):187–202. doi:10.1098/rsif.2008.0172.

63. Liepe J, Kirk P, Filippi S, Toni T, Barnes CP, Stumpf MPH. A framework for parameter estimation and model selection from experimental data in systems biology using approximate Bayesian computation. Nature Protocols. 2014;9(2):439–456. doi:10.1038/nprot.2014.025.

64. Kass RE, Raftery AE. Bayes Factors. Journal of the American Statistical Association. 1995;90(430):773–795. doi:10.1080/01621459.1995.10476572.

65. Vehtari A, Gelman A, Gabry J. Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. Statistics and Computing. 2017;27(5):1413–1432. doi:10.1007/s11222-016-9696-4.

66. Tapia J, Faeder JR, Munsky B. Adaptive Coarse-Graining for Transient and Quasi-Equilibrium Analyses of Stochastic Gene Regulation. In: Proceedings of the 51st IEEE Conference on Decision and Control (CDC). Maui, HI, USA; 2012. p. 5361–5366.

67. Peccoud J, Ycart B. Markovian Modeling of Gene-Product Synthesis. Theoretical Population Biology. 1995;48(2):222–234.

68. Munsky B, Neuert G, Van Oudenaarden A. Using gene expression noise to understand gene regulation. Science. 2012;336(6078):183–187.

69. Ross SM. Stochastic Processes. 2nd ed. John Wiley & Sons; 1995.

70. Norris JR. Markov Chains. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge: Cambridge University Press; 1997.

71. Ge H, Qian H. In: Dubitzky W, Wolkenhauer O, Cho KH, Yokota H, editors. Chemical Master Equation. New York, NY: Springer New York; 2013. p. 396–399. Available from: https://doi.org/10.1007/978-1-4419-9863-7_278.

72. Craig JM, Laszlo AH, Nova IC, Gundlach JH. Modelling single-molecule kinetics of helicase translocation using high-resolution nanopore tweezers (SPRNT). Essays in Biochemistry. 2022;65(1):109–127. doi:10.1042/EBC20200027.

73. Nordsieck A, Lamb WE, Uhlenbeck GE. On the theory of cosmic-ray showers I the furry model and the fluctuation problem. Physica. 1940;7(4):344–360. doi:https://doi.org/10.1016/S0031-8914(40)90102-1.

74. Nordsieck A, Lamb WE, Uhlenbeck GE. On the theory of cosmic-ray showers I the furry model and the fluctuation problem. Physica. 1940;7(4):344–360. doi:https://doi.org/10.1016/S0031-8914(40)90102-1.

75. Kolmogoroff A. Über die analytischen Methoden in der Wahrscheinlichkeitsrechnung. Mathematische Annalen. 1931;104:415–458. doi:https://doi.org/10.1007/BF01457949.

76. Feller W. On Boundaries and Lateral Conditions for the Kolmogorov Differential Equations. Annals of Mathematics. 1957;65(3):527–570. doi:https://doi.org/10.2307/1970064.

77. Raj A, van Oudenaarden A. Nature, nurture, or chance: stochastic gene expression and its consequences. Cell. 2008;135(2):216–226. doi:10.1016/j.cell.2008.09.050.

78. Eldar A, Elowitz MB. Functional roles for noise in genetic circuits. Nature. 2010;467(7312):167–173.

79. Aguilera LU, Weber LM, Ron E, King CR, Öcal K, Popinga A, et al. Methods in Quantitative Biology - from Analysis of Single-Cell Microscopy Images to Inference of Predictive Models for Stochastic Gene Expression. 2025;Submitted.

80. Kurtz TG. The Relationship between Stochastic and Deterministic Models for Chemical Reactions. The Journal of Chemical Physics. 1972;57(7):2976–2978. doi:10.1063/1.1678692.

81. Anderson DF, Kurtz TG. Stochastic Analysis of Biochemical Systems. Mathematical Biosciences Institute Lecture Series. Springer; 2015.

82. Wilkinson DJ. Stochastic Modelling for Systems Biology. 3rd ed. Chapman & Hall/CRC Mathematical and Computational Biology. Boca Raton, FL: CRC Press; 2019.

83. Shampine LF, Reichelt MW. The MATLAB ODE Suite. SIAM Journal on Scientific Computing. 1997;18(1):1–22. doi:10.1137/S1064827594276424.

84. Engblom S. Computing the moments of high dimensional solutions of the master equation. Applied Mathematics and Computation. 2006;180(2):498–515. doi:10.1016/j.amc.2005.12.032.

85. Gillespie CS. Moment-closure approximations for mass-action models. IET Systems Biology. 2009;3(1):52–58. doi:10.1049/iet-syb:20070060.

86. Gillespie DJ. Approximate accelerated stochastic simulation of chemically reacting systems. The Journal of Chemical Physics. 2001;115(4):1716–1733. doi:10.1063/1.1378322.

87. Rathinam M, Petzold LR, Cao Y, Gillespie DT. Stiffness in stochastic chemically reacting systems: The implicit tau-leaping method. The Journal of Chemical Physics. 2003;119(24):12784–12794. doi:/10.1063/1.1627296.

88. Voliotis M, Thomas P, Grima R, Bowsher CG. Stochastic Simulation of Biomolecular Networks in Dynamic Environments. PLOS Computational Biology. 2016;12(6):1–18. doi:10.1371/journal.pcbi.1004923.

89. Munsky B, Khammash M. Transient analysis of stochastic switches and trajectories with applications to gene regulatory networks. IET Systems Biology. 2008;2(5):323–333. doi:10.1049/iet-syb:20070082.

90. Gunawan R, Cao Y, Petzold L, Doyle FJ. Sensitivity analysis of discrete stochastic systems. Biophysical journal. 2005;88(4):2530–2540.

91. Degasperi A, Gilmore S. Sensitivity analysis of stochastic models of bistable biochemical reactions. In: Formal Methods for Computational Systems Biology: 8th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2008 Bertinoro, Italy, June 2-7, 2008 Advanced Lectures 8. Springer; 2008. p. 1–20.

92. Stapor P, Fröhlich F, Hasenauer J. Optimization and profile calculation of ODE models using second order adjoint sensitivity analysis. Bioinformatics. 2018;34(13):i151–i159. doi:10.1093/bioinformatics/bty230.

93. Fisher RA. Theory of Statistical Estimation. Mathematical Proceedings of the Cambridge Philosophical Society. 1925;22(5):700–725. doi:10.1017/S0305004100009580.

94. Le Cam L. In: Asymptotic Methods in Statistical Decision Theory. New York: Springer; 1986. p. 618–621.

95. Komorowski M, Costa MJ, Rand DA, Stumpf MP. Sensitivity, robustness, and identifiability in stochastic chemical kinetics models. Proceedings of the National Academy of Sciences. 2011;108(21):8645–8650.

96. Ruess J, Milias-Argeitis A, Lygeros J. Designing experiments to understand the variability in biochemical reaction networks. Journal of The Royal Society Interface. 2013;10(88):20130588.

97. Casella G, Berger RL. Statistical Inference. Pacific Grove, CA: Brooks/Cole; 1990.

98. Fox ZR, Neuert G, Munsky B. Optimal Design of Single-Cell Experiments within Temporally Fluctuating Environments. Complexity. 2020;2020:1–15. doi:10.1155/2020/8536365.

99. Kreutz C, Timmer J. Systems Biology: Experimental Design. FEBS Journal. 2009;276(4):923–942. doi:10.1111/j.1742-4658.2008.06843.x.

100. Banga JR, Balsa-Canto E. Parameter Estimation and Optimal Experimental Design. Essays in Biochemistry. 2008;45:195–209. doi:10.1042/bse0450195.

101. Atkinson AC, Donev AN, Tobias RD. Optimum Experimental Designs, with SAS. Oxford Statistical Science Series. Oxford: Oxford University Press; 2007.

102. Pukelsheim F. Optimal Design of Experiments. New York: John Wiley & Sons; 1993.

103. Mélykúti B, August E, Papachristodoulou A, El-Samad M. Discriminating Between Rival Biochemical Network Models: Three Approaches to Optimal Experiment Design. BMC Systems Biology. 2010;4:38. doi:10.1186/1752-0509-4-38.

104. Huang S. Non-genetic heterogeneity of cells in development: more than just noise. Development. 2009;136(23):3853–3862. doi:/10.1242/dev.035139.

105. Trapnell C, Cacchiarelli D, Grimsby J, Pokharel P, Li S, Morse M, et al. The dynamics and regulators of cell fate decisions are revealed by pseudotemporal ordering of single cells. Nature Biotechnology. 2014;32:381–386. doi:/10.1038/nbt.2859.

106. Ruess J, Parise F, Milias-Argeitis A, Khammash M, Lygeros J. Iterative experiment design guides the characterization of a light-inducible gene expression circuit. Proceedings of the National Academy of Sciences. 2015;112(26):8148–8153.

107. Pitchiaya S, Heinicke LA, Custer TC, Walter NG. Single molecule fluorescence approaches shed light on intracellular RNAs. Chemical Reviews. 2014;114(6):3224–3265. doi:10.1021/cr400496q.

108. Lakadamyali M, Cosma MP. Visualizing the genome in high resolution challenges our textbook understanding. Nature Methods. 2020;586(7829):372–382. doi:/10.1038/s41592-020-0758-3.

109. Raj A, Van Den Bogaard P, Rifkin SA, Van Oudenaarden A, Tyagi S. Imaging individual mRNA molecules using multiple singly labeled probes. Nature methods. 2008;5(10):877–879.

110. Tsanov N, Samacoits A, Chouaib R, Traboulsi AM, Gostan T, Weber C, et al. smiFISH and FISH-quant–a flexible single RNA detection approach with super-resolution capability. Nucleic acids research. 2016;44(22):e165–e165.

111. Gómez-Schiavon M, Chen LF, West AE, Buchler NE. BayFish: Bayesian inference of transcription dynamics from population snapshots of single-molecule RNA FISH in single cells. Genome biology. 2017;18:1–12.

112. Femino AM, Fay FS, Fogarty K, Singer RH. Visualization of single RNA transcripts in situ. Science. 1998;280(5363):585–590.

113. Shepherd DP, Li N, Micheva-Viteva SN, Munsky B, Hong-Geller E, Werner JH. Counting small RNA in pathogenic bacteria. Analytical chemistry. 2013;85(10):4938–4943.

114. Tang F, Barbacioru C, Wang Y, Nordman E, Lee C, Xu N, et al. mRNA-Seq whole-transcriptome analysis of a single cell. Nature Methods. 2009;6(5):377–382. doi:10.1038/nmeth.1315.

115. Macosko EZ, Basu A, Satija R, Nemesh J, Shekhar K, Goldman M, et al. Highly parallel genome-wide expression profiling of individual cells using nanoliter droplets. Cell. 2015;161(5):1202–1214. doi:10.1016/j.cell.2015.05.002.

116. Cao J, Packer JS, Ramani V, Cusanovich DA, Huynh C, Daza R, et al. Comprehensive single-cell transcriptional profiling of a multicellular organism. Science. 2017;357(6352):661–667. doi:10.1126/science.aam8940.

117. Buenrostro JD, Wu B, Litzenburger UM, Ruff D, Gonzales ML, Snyder MP, et al. Single-cell chromatin accessibility reveals principles of regulatory variation. Nature. 2015;523(7561):486–490. doi:10.1038/nature14590.

118. Smallwood SA, Lee HJ, Angermueller C, Krueger F, Saadeh H, Peat J, et al. Single-cell genome-wide bisulfite sequencing for assessing epigenetic heterogeneity. Nature Methods. 2014;11(8):817–820. doi:10.1038/nmeth.3035.

119. Giesen C, Wang HAO, Schapiro D, Zivanovic N, Jacobs A, Hattendorf B, et al. Highly multiplexed imaging of tumor tissues with subcellular resolution by mass cytometry. Nature Methods. 2014;11(4):417–422. doi:10.1038/nmeth.2869.

120. Hoffman JA, Papas BN, Trotter KW, Archer TK. Single-cell RNA sequencing reveals a heterogeneous response to Glucocorticoids in breast cancer cells. Communications Biology. 2020;3(1):126. doi:10.1038/s42003-020-0837-0.

121. Fisher RA. On the mathematical foundations of theoretical statistics. Philosophical Transactions of the Royal Society of London Series A, Containing Papers of a Mathematical or Physical Character. 1922;222:309–368. doi:10.1098/rsta.1922.0009.

122. Kalman RE. A New Approach to Linear Filtering and Prediction Problems. Transactions of the ASME–Journal of Basic Engineering. 1960;82(Series D):35–45.

123. Rabiner LR. A tutorial on hidden Markov models and selected applications in speech recognition. Proceedings of the IEEE. 1989;77(2):257–286. doi:10.1109/5.18626.

124. Cappé O, Moulines E, Rydén T. Inference in hidden markov models. In: Proceedings of EUSFLAT Conference; 2009. p. 14–16.

125. Sarkka S, Solin A, Hartikainen J. Spatiotemporal learning via infinite-dimensional Bayesian filtering and smoothing: A look at Gaussian process regression through Kalman filtering. IEEE Signal Processing Magazine. 2013;30(4):51–61.

126. Lehmann EL, Casella G. Theory of Point Estimation. 2nd ed. Springer Texts in Statistics. New York: Springer; 1998.

127. Metropolis N, Rosenbluth AW, Rosenbluth MN, Teller AH, Teller E. Equation of state calculations by fast computing machines. The Journal of Chemical Physics. 1953;21(6):1087–1092. doi:10.1063/1.1699114.

128. Hastings WK. Monte Carlo sampling methods using Markov chains and their applications. Biometrika. 1970;57(1):97–109. doi:10.1093/biomet/57.1.97.

129. Robert CP, Casella G. Monte Carlo Statistical Methods. 2nd ed. Springer Texts in Statistics. New York: Springer; 2004.

130. Gelman A, Carlin JB, Stern HS, Dunson DB, Vehtari A, Rubin DB. Bayesian Data Analysis. 3rd ed. Boca Raton, FL: CRC Press; 2013.

131. Bernardo JM, Smith AFM. Bayesian Theory. Chichester: John Wiley & Sons; 1994.

132. Bayes T. An Essay Towards Solving a Problem in the Doctrine of Chances. Philosophical Transactions of the Royal Society of London. 1763;53:370–418. doi:10.1098/rstl.1763.0053.

133. Lee PM. Bayesian Statistics: An Introduction. 4th ed. Wiley; 2012.

134. Sunnåker M, Busetto AG, Numminen E, Corander J, Foll M, Dessimoz C. Approximate Bayesian Computation. PLoS Computational Biology. 2013;9(1):e1002803. doi:10.1371/journal.pcbi.1002803.

135. Antoulas AC. Approximation of Large-Scale Dynamical Systems. Advances in Design and Control. Philadelphia: SIAM; 2005.

136. Quarteroni A, Manzoni A, Negri F. Reduced Basis Methods for Partial Differential Equations. La Matematica per il 3+2. Cham: Springer; 2015.

137. Munsky B, Khammash M, Petzold LR. Quantitative Biology: Theory, Computational Methods, and Models. Cambridge, MA: MIT Press; 2018.

138. Banks AM, Whitfield CJ, Brown SR, Fulton DA, Goodchild SA, Grant C, et al. Key reaction components affect the kinetics and performance robustness of cell-free protein synthesis reactions. Computational and Structural Biotechnology Journal. 2022;20:218–229. doi:10.1016/j.csbj.2021.12.013.

139. Hegland M, Fletcher-Costin R. Cmepy v0.3 documentation;. Available from: `https://fcostin.github.io/cmepy/`.

140. Poole W, Pandey A, Shur A, Tuza ZA, Murray RM. BioCRNpyler: Compiling chemical reaction networks from biomolecular parts in diverse contexts. PLOS Computational Biology. 2022;18(4):1–19. doi:10.1371/journal.pcbi.1009987.

141. Lopez CF, Muhlich JL, Bachman JA, Sorger PK. Programming biological models in Python using PySB. Molecular Systems Biology. 2013;9(1):646. doi:https://doi.org/10.1038/msb.2013.1.

142. Heirendt L, Arreckx S, Pfau T, Mendoza SN, Richelle A, Heinken A, et al. Creation and analysis of biochemical constraint-based models using the COBRA Toolbox v.3.0. Nature Protocols. 2019;14(3). doi:10.1038/s41596-018-0098-2.

143. Bergmann FT, Adams R, Moodie S, Cooper J, Glont M, Golebiewski M, et al. SBML Level 3 Package: Layout, Version 1 Core. Journal of Integrative Bioinformatics. 2014;11(2):246. doi:10.2390/biecoll-jib-2014-246.

144. Poole W, Pandey A, Shur A, Tuza ZA, Murray RM. BioCRNpyler: Compiling chemical reaction networks from biomolecular parts in diverse contexts. PLoS Computational Biology. 2022;doi:10.1371/journal.pcbi.1009987.

145. Gupta A, Schwab C, Khammash M. DeepCME: A deep learning framework for computing solution statistics of the chemical master equation. PLOS Computational Biology. 2021;doi:10.1371/journal.pcbi.1009623.

146. Schnoerr D, Sanguinetti G, Grima R. Comparison of different moment-closure approximations for stochastic chemical kinetics. The Journal of Chemical Physics. 2015;143. doi:10.1063/1.4934990.

147. Fröhlich F, Weindl D, Schälte Y, Pathirana D, Paszkowski Ł, Lines GT, et al. Sukys, Augustinas and Öcal, Kaan and Grima, Ramon. iScience. 2022;25.

148. Pájaro M, Otero-Muras I, Vázquez C, Alonso AA. SELANSI: a toolbox for simulation of stochastic gene regulatory networks. Bioinformatics. 2017;34(5):893–895. doi:10.1093/bioinformatics/btx645.

149. Lapin M, Mikeev L, Wolf V. SHAVE: Stochastic Hybrid Analysis of Markov Population Models. In: Proceedings of the 14th International Conference on Hybrid Systems: Computation and Control (HSCC '11). New York, NY, USA: ACM; 2011. p. 311–312. Available from: `https://dl.acm.org/doi/10.1145/1967701.1967746`.

150. Hespanha JP. `StochDynTools` — a MATLAB toolbox to compute moment dynamics for stochastic networks of bio-chemical reactions; 2007. Available at `http://www.ece.ucsb.edu/~hespanha`.

151. Sukys A, Grima R. MomentClosure.jl: automated moment closure approximations in Julia. Bioinformatics. 2021;38(1):289–290. doi:10.1093/bioinformatics/btab469.

152. Ocal K, Sukys A. FiniteStateProjection.jl; 2025. Available from: `https://github.com/SciML/FiniteStateProjection.jl`.

Approach:		**ODEs**	**Moments**	**SSA**	**CME (FSP)**	**CME (other)**	**Hybrid**
AMICI	[34]	✓					
AMIGO2	[35]	✓					
BioCRNpyler*	[144]						
BioNetGen†	[36]	✓		✓			
Catalyst‡	[37]	✓	✓	✓	✓		✓
CERENA	[38]	✓	✓	✓	✓		✓
CmePy§	[139]				✓		
COPASI	[39]	✓		✓			✓
Data2Dynamics	[40]	✓					
deBInfer	[41]	✓					
DeepCME	[145]					✓	
Dizzy	[42]	✓	∼	✓			
dMod	[43]	✓					
GillesPy2	[44]	✓		✓			✓
iBioSim 3	[45]	✓		✓			
IDESS	[46]			✓		✓	
iNA	[47]	✓	✓	✓			
MobsPy	[48]	✓		✓			
modelbase	[49]	✓					
MOCA	[146]		✓				
Nessie	[147]					✓	
SELANSI	[148]					✓	
SHAVE	[149]		✓			✓	✓
SimBiology	[50]	✓		✓			
StochDynTools	[150]		✓				
StochKit2	[33]			✓			
StochPy	[51]			✓			
StochSS	[52]	✓		✓	✓¶		
Tellurium	[53]	✓		✓			
URDME	[54]			✓	✓¶	✓¶	
The SSIT		✓		✓	✓		✓

Table 1. *Model-builder only; † "hybrid" in this table refers to the determination of some subpopulations as deterministic and some as stochastic, whereas BioNetGen uses "hybrid" to refer to particle/population simulation; ‡with the MomentClosure.jl [151] and FiniteStateProjection.jl [152] add-ons; §currently inaccessible; ∼Linear Noise Approximation only; ¶Note: the StochSS method for using FSP is to use Diffusive Finite State Projection for spatial Reaction-Diffusion Master Equations, not the well-mixed CME.

	Catalyst [37]	CERENA [38]	CmePy [139]	StochSS [52]	URDME [54]	**The SSIT**
CME solutions						
Escape/waiting times						✓
State space adjust*						
N-step expansion [13]			✓			✓
SSA						✓
Polynomial bounds						✓
Estimation of θ						
ABC				✓		✓
Distance metric (RMSE)						
Likelihood						✓
Monte Carlo method(s)						✓
Cross validation						✓
Sensitivities/FIM						
Finite difference						✓
Forward sensitivity		✓				✓
Fisher Information						✓
Measurement noise						
PDO [29]						✓
Model Reduction						
Hybrid ODE-CME						✓
POD						✓
Sparse grid						✓
Quasi-steady state						✓
Eigen value decomp.						✓
Experiment Design						
FSP-FIM						✓
Bayesian						✓
Propensities						
Time-varying	✓	✓	✓	(✓)	✓	✓
Logicals						✓
Non-linear	✓	✓	✓	✓	✓	✓
Compatibility						
SBML	✓	✓		✓	✓	✓
SimBiology						✓
Source Code/API						
Julia	✓					
Python			✓	(✓)		
MATLAB		✓			✓	✓
User Interface						
GUI				✓		✓
Text/XML	✓	✓		✓	✓	✓
IDE/Commandline	✓	✓	✓		✓	✓
Accessibility						
GitHub	✓	✓		✓	✓	✓
Standalone website	✓	✓		✓	✓	✓

Table 2