IOP Publishing Phys. Biol. 13 (2016) 035003 doi:10.1088/1478-3975/13/3/035003

Physical Biology

PAPER

Understanding the finite state projection and related methods for solving the chemical master equation

RECEIVED
16 November 2015

REVISED
25 March 2016

ACCEPTED FOR PUBLICATION
13 April 2016

PUBLISHED
13 May 2016

Khanh N Dinh and Roger B Sidje[1]

Department of Mathematics University of Alabama Tuscaloosa, Alabama 35487, USA
[1] Author to whom any correspondence should be addressed.

E-mail: kdinh@crimson.ua.edu **and** roger.b.sidje@ua.edu

Keywords: ODE solver, chemical master equation, gene regulation, finite state projection

Abstract

The finite state projection (FSP) method has enabled us to solve the chemical master equation of some biological models that were considered out of reach not long ago. Since the original FSP method, much effort has gone into transforming it into an adaptive time-stepping algorithm as well as studying its accuracy. Some of the improvements include the multiple time interval FSP, the sliding windows, and most notably the Krylov-FSP approach. Our goal in this tutorial is to give the reader an overview of the current methods that build on the FSP.

1. Introduction

A familiar approach to modeling a complex reaction network is to find the master equation that describes the joint probability density function of the population of the reactants over time. In their extensive review [1], Goutsias and Jenkinson listed many fields of science in which such an equation is of considerable importance, such as ecological networks, pharmacokinetic networks and social networks.

In systems biology, the equation is called the chemical master equation (CME) [2], and its solution is the probability distribution of finding the system in all possible states (i.e., all possible integer-valued populations of the reacting chemical species). It is easy to see why solving the CME is a formidable task: if the copy numbers of species in the system are not bounded, then there are potentially infinitely many states that the system can occupy. Even if we apply a bound on the species numbers, the size of the CME increases exponentially with the numbers of species and therefore solving for the probability distribution of all of them is numerically very expensive.

Despite this so-called 'curse of dimensionality', solving the CME has been of great interest to the systems biology community, because unlike deterministic models, the CME captures the randomness of the biological processes. It has been shown in different biological contexts that single molecular events may significantly impact the process, which underlines the importance of stochasticity in these models.

The main approach to solving the CME has often been using Monte Carlo algorithms, of which Gillespie's stochastic stimulation algorithm (SSA) [3] has been extensively employed. However, the SSA can be very slow, considering the numerous runs needed to average the probability distribution. Several modifications to improve its efficiency include the tau-leaping method [4, 5] or slow-scale SSA [6].

The finite state projection (FSP) method [7] is a different approach to provide an approximation to the actual probability distribution at the end time as well as the transient probabilities with a guaranteed accuracy.

Soon after the FSP, an adaptive time-stepping version was introduced [8, 9] and followed by other variants. These algorithms have proved to give accurate probability distributions, and their speed has been helpful in gene regulation problems where a set of parameters have to be found, which can only be achieved by computing the distributions for many parameter sets and choosing the distribution closest to experimental data [10]. The FSP approach is also valid for cases where reaction rates vary over time. However, the solutions need to be found by applying ODE techniques, and certain FSP improvements may no longer be applicable. We therefore only consider the cases where reaction rates are constant in this paper.

Despite the huge numerical improvements of the FSP and its variants, to the authors' knowledge, there has been no survey of the different approaches. This paper is meant to fulfill that purpose.

The paper is organized as follows: section 2 describes the notation used throughout the paper, formalizes the CME, and comments on popular Monte Carlo methods. Section 3 outlines the original FSP method. Section 4 describes a time-stepping framework, where each step consists of three stages, and we show how all variants of the FSP aimed at improving one or more of these stages. From there, the variants are presented in sections 5–7, following how they fit in the time-stepping framework. Note that each method is accompanied by a simplified pseudocode for ease of readability. Another approach for solving the CME via tensor decomposition is discussed in section 8. An illustrative example is given in section 9 to demonstrate how to generate the CME matrix for a specific biological case. Section 10 lists some available software for solving the CME by using the FSP. The strengths and weaknesses of solving the CME are discussed in section 11. We finish with some concluding remarks in section 12.

2. Chemical master equation

Consider a chemical reaction system consisting of N molecular species $S_1, \ldots, S_N$ that interact through M reactions of the form

$$R_k: \; a_{1k}S_1 + \ldots + a_{Nk}S_N \xrightarrow{c_k} b_{1k}S_1 + \ldots + b_{Nk}S_N$$

for $k = 1, \ldots, M$. The c_k are *reaction rate constants*, which are scale factors for how likely such a collision of the reactants results in a reaction. At any time, the system can be described as the numbers of copies of each species. We then can define the *state vector* of the system as the vector of these numbers:

$$\boldsymbol{x} = (x_1, \ldots, x_N)^T,$$

where x_l is a count for species S_l. We denote $\boldsymbol{x}(t)$ as the state of the system at time t.

The *propensity function* $\alpha_k(\boldsymbol{x}(t))$ of reaction R_k at the current state $\boldsymbol{x}(t)$ is defined so that the probability of such a reaction occurring during the infinitesimal time interval $[t, t + dt)$ is equal to $\alpha_k(\boldsymbol{x}(t))dt$.

When reaction R_k happens, the state vector is updated as

$$\boldsymbol{x}(t + dt) \leftarrow \boldsymbol{x}(t) + \boldsymbol{\nu}_k, \tag{1}$$

where the *stoichiometric vector* $\boldsymbol{\nu}_k$, representing the change in species numbers, is defined as

$$\boldsymbol{\nu}_k = (b_{1k} - a_{1k}, \ldots, b_{Nk} - a_{Nk})^T.$$

We are now interested in the probability that the system is at state $\boldsymbol{x}$ at time t, which we denote $P(\boldsymbol{x}, t) = \text{Prob}\{\boldsymbol{x}(t) = \boldsymbol{x}\}$. Assuming that we know the number of each species at $t = 0$ (from which we can deduce $P(\boldsymbol{x}, 0)$), the CME [2] dictates that

$$\frac{dP(\boldsymbol{x}, t)}{dt} = \sum_{k=1}^{M} \alpha_k(\boldsymbol{x} - \boldsymbol{\nu}_k) P(\boldsymbol{x} - \boldsymbol{\nu}_k, t) - \sum_{k=1}^{M} \alpha_k(\boldsymbol{x}) P(\boldsymbol{x}, t). \tag{2}$$

The expression is clear if we note that $\alpha_k(\boldsymbol{x} - \boldsymbol{\nu}_k)dt$ is the probability for state $\boldsymbol{x} - \boldsymbol{\nu}_k$ to transition to state $\boldsymbol{x}$ through reaction R_k during $[t, t + dt)$, and $(\sum_{k=1}^{M} \alpha_k(\boldsymbol{x}))dt$ is the probability for the system to escape from state $\boldsymbol{x}$ through any reaction during that same time period.

Let $\boldsymbol{X}$ be the set of all possible states, if we order these states as $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$, where $\boldsymbol{x}_i = (x_{1i}, \ldots, x_{Ni})^T$ and n is the total number of states, then (2) defines a set of ODEs governing the change in $\boldsymbol{p}(t) = (P(\boldsymbol{x}_1, t), \ldots, P(\boldsymbol{x}_n, t))^T$:

$$\dot{\boldsymbol{p}}(t) = \boldsymbol{A} \cdot \boldsymbol{p}(t), \tag{3}$$

where here we set $\boldsymbol{p}(0) = (1, 0, \ldots, 0)^T$ by assuming that the system is at state $\boldsymbol{x}_1$ at $t = 0$. The transition rate matrix $\boldsymbol{A} = [a_{ij}] \in \mathbb{R}^{n \times n}$ is defined as

$$a_{ij} = \begin{cases} -\sum_{k=1}^{M} \alpha_k(\boldsymbol{x}_j), & \text{if } i = j \\ \alpha_k(\boldsymbol{x}_j), & \text{if } \boldsymbol{x}_i = \boldsymbol{x}_j + \boldsymbol{\nu}_k \\ 0, & \text{otherwise} \end{cases}.$$

From (3) we can find the probability vector at the end point t_f:

$$\boldsymbol{p}(t_f) = \exp(t_f \boldsymbol{A})\boldsymbol{p}(0),$$

where the exponential matrix is defined as

$$\exp(t_f \boldsymbol{A}) = \sum_{m=0}^{\infty} \frac{(t_f \boldsymbol{A})^m}{m!}.$$

An explicit formula for $\boldsymbol{p}(t)$ can be given only in extremely simple cases, such as monomolecular reaction systems [11], therefore it is necessary to numerically solve at a prescribed end point t_f.

The enormous size of the CME usually makes it too challenging to solve directly. The stochastic simulation algorithm (SSA) takes a piecemeal approach by computing single realizations of the state vector rather than an entire probability distribution. For each realization, the algorithm updates the state as in (1), by randomly choosing the time between events, dt, and the next reaction index, k. The τ-leap variant that seeks to improve the efficiency of the SSA consists in allowing a larger time between events, $\tau \leftarrow dt$, so that more that one reaction can be accumulated in the state update.

As noted before, the cost of these methods is compounded by the multiple runs that have to be done to average the results in a Monte Carlo manner. More details about these algorithms can be found in [3–5], and further improvements in [6, 12, 13].

3. The original finite state projection

Unlike SSA, the FSP method seeks to directly approximate the probability density function that is the

solution of (3). We define $\boldsymbol{X}_J$ to be a finite subset of states in $\boldsymbol{X}$, where J is the index set of those states. Consider all the other states not in $\boldsymbol{X}$ as only one state, which we call the *sink state* G. Let $\boldsymbol{A}_J$ be a submatrix of $\boldsymbol{A}$ containing only the elements on the rows and columns indexed by J, and $\boldsymbol{p}(\boldsymbol{X}_J, t)$ contains only the probabilities of states indexed in J at time t. The FSP method approximates $\boldsymbol{p}(\boldsymbol{X}, t)$ by $\boldsymbol{p}_J^{\mathrm{FSP}}(t) = \boldsymbol{p}^{\mathrm{FSP}}(\boldsymbol{X}_J, t)$, which follows the master equation

$$\begin{pmatrix} \dot{\boldsymbol{p}}_J^{\mathrm{FSP}}(t) \\ \dot{g}(t) \end{pmatrix} = \begin{pmatrix} \boldsymbol{A}_J & \boldsymbol{0} \\ -\mathbf{1}^T \boldsymbol{A}_J & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{p}_J^{\mathrm{FSP}}(t) \\ g(t) \end{pmatrix}, \tag{4}$$

with initial distribution

$$\begin{pmatrix} \boldsymbol{p}_J^{\mathrm{FSP}}(0) \\ g(0) \end{pmatrix} = \begin{pmatrix} \boldsymbol{p}(\boldsymbol{X}_J, 0) \\ 1 - \sum \boldsymbol{p}(\boldsymbol{X}_J, 0) \end{pmatrix}.$$

The theoretical solution to this set of ODEs can be proven to be

$$\begin{cases} \boldsymbol{p}_J^{\mathrm{FSP}}(t) = \exp(t\boldsymbol{A}_J)\boldsymbol{p}_J^{\mathrm{FSP}}(0) \\ g(t) = \mathbf{1}^T(\boldsymbol{I} - \exp(t\boldsymbol{A}_J))\boldsymbol{P}_J^{\mathrm{FSP}}(0) + g(0) \end{cases}. \tag{5}$$

Note that since $\boldsymbol{A}_J$ is extracted from $\boldsymbol{A}$, $\boldsymbol{p}_J^{\mathrm{FSP}}(t)$ has a different statistical meaning from $\boldsymbol{p}(\boldsymbol{X}_J, t)$: the elements in $\boldsymbol{p}_J^{\mathrm{FSP}}(t)$ are the probabilities that the system is at the corresponding states at t and has never left $\boldsymbol{X}_J$ during $[0, t)$. On the other hand, $g(t)$ is the probability that the system visited the sink state G at least once during $[0, t)$ (also note that if $\boldsymbol{X}_J$ contains the initial state then $g(0) = 0$). In [7] the authors showed that these facts imply

$$\boldsymbol{p}(\boldsymbol{X}_J, t) \geqslant \boldsymbol{p}_{J_2}^{\mathrm{FSP}}(t) \geqslant \boldsymbol{p}_{J_1}^{FSP}(t) \geqslant \boldsymbol{0} \tag{6}$$

if $J_1 \subset J_2$, and

$$\left\| \begin{pmatrix} \boldsymbol{p}(\boldsymbol{X}_J, t) \\ \boldsymbol{p}(\boldsymbol{X}_{J'}, t) \end{pmatrix} - \begin{pmatrix} \boldsymbol{p}_J^{\mathrm{FSP}}(t) \\ \boldsymbol{0} \end{pmatrix} \right\|_1 = g(t) \tag{7}$$

where J' is the index set of the states not in J. The significance of these two properties cannot be overstated: (6) guarantees that the FSP approximation only improves monotonically element-wise if we keep expanding the state space, and (7) gives us the exact evaluation of the 1-norm error.

The original FSP algorithm follows directly from the solution (5) and these two observations.

Original FSP algorithm. (B Munsky, M Khammash, [7], 2006)

Find the FSP approximation with 1-norm error less than ϵ at t_f:

1: Initialize $i = 0$, J_0 and $\boldsymbol{A}_{J_0}$.
2: Approximate $\exp(t_f \boldsymbol{A}_{J_i})\boldsymbol{p}(\boldsymbol{X}_{J_i}, 0)$.
If $\mathbf{1}^T \exp(t_f \boldsymbol{A}_{J_i})\boldsymbol{p}(\boldsymbol{X}_{J_i}, 0) \geqslant 1 - \epsilon$, then stop.
We have the approximation
$\boldsymbol{p}(\boldsymbol{X}_{J_i}, t_f) \approx \exp(t_f \boldsymbol{A}_{J_i})\boldsymbol{p}(\boldsymbol{X}_{J_i}, 0)$,
with
$\|\boldsymbol{p}(\boldsymbol{X}_{J_i}, t_f) - \exp(t_f \boldsymbol{A}_{J_i})\boldsymbol{p}(\boldsymbol{X}_{J_i}, 0)\|_1 \leqslant \epsilon$
3: Increment i and add more states into J_i, then return to step 2.

4. Time-stepping FSP variations

There has been great interest in modifying the FSP algorithm into a time-stepping scheme. Many approaches that have been developed for this purpose follow the same framework demonstrated in the following variable time-stepping FSP algorithm.

The algorithm divides $[0, t_f]$ into small intervals

$$0 = t_0 < t_1 < \ldots < t_{K+1} = t_f.$$

At every interval $[t_k, t_{k+1}]$, the quest is then to find the index set J_{k+1} of the most likely states. The probability vector of $\boldsymbol{X}_{J_{k+1}}$ at t_{k+1} is then calculated by

$$\boldsymbol{p}_{J_{k+1}}^{\mathrm{FSP}}(t_{k+1}) = \exp(\tau_k \boldsymbol{A}_{J_{k+1}})\boldsymbol{p}_{J_k}^{\mathrm{FSP}}(t_k) \tag{8}$$

and the algorithm moves on to the next time interval.

Though not always the case, the problem of approximating (8) can be even further reduced before solving, especially when special properties of the reactions or the species are realized. In these cases, the reduced system will be preconditioned in step 2 of the algorithm. Apart from that, steps 1 and 3 are taken from the original FSP algorithm. Note that our notation for the FSP approximation is $\boldsymbol{p}_{J_k}(t_k) = \boldsymbol{p}_{J_k}^{\mathrm{FSP}}(t_k) = \boldsymbol{p}^{\mathrm{FSP}}(\boldsymbol{X}_{J_k}, t_k)$ and not to be confused with the true solution $\boldsymbol{p}(\boldsymbol{X}_{J_k}, t_k)$.

Variable time-stepping FSP algorithm. (K Burrage, M Hegland, S MacNamara, R Sidje, [8], 2006. B Munsky, M Khammash, [9], 2007.)

0: Start from $k = 0$, $t_k = t_0$.
1: Find the time step τ_k, and the state space $\boldsymbol{X}_{J_{k+1}}$ containing states most likely over $[t_k, t_{k+1}]$ where $t_{k+1} = t_k + \tau_k$.
2: Precondition the reduced system before solving.
3: Approximate
$\boldsymbol{p}_{J_{k+1}}(t_{k+1}) \approx \exp(\tau_k \boldsymbol{A}_{J_{k+1}})\boldsymbol{p}_{J_k}(t_k)$
4: If $t_{k+1} < t_f$, set $k = k + 1$ and go to step 1.

Note that the variable time-stepping FSP algorithm serves more as a framework, because each step from 1 to 3 can be modified for the specific biological problem. The next three sections will drill further into these steps: section 5 considers different strategies to update the state space, section 6 discusses some methods to precondition the FSP, and section 7 introduces some techniques to compute the action of matrix exponential on a vector while taking advantage of the fact that the entire matrix exponential is not needed.

5. Update the state space

It is clear that a crucial part of the variable time-stepping FSP algorithm is to find a strategy for updating the state space: it needs to contain enough states that contribute the most to the probability mass,

but a too large state space results in a bigger $\boldsymbol{A}_J$ and a more time-consuming evaluation of its matrix exponential.

5.1. Update by r-step reachability

The original paper by Munsky and Khammash [7] has a suggestion for how to expand the state space through the concept of reachability. They made an observation that, the system ends up in a state at t_{k+1} by jumping from a state in t_k through only finitely many states in $\boldsymbol{X}$. Therefore, much of the probability mass at t_{k+1} will be contained in the set of the states either in $\boldsymbol{X}_{J_k}$ or can be reached from a state in $\boldsymbol{X}_{J_k}$ within one reaction, we call the index set of these states $\mathcal{R}_1(J_k)$:

$$\boldsymbol{X}_{\mathcal{R}_1(J_k)} = \boldsymbol{X}_{J_k} \bigcup_{i=1}^{M} \{\boldsymbol{x} + \boldsymbol{\nu}_i \in \boldsymbol{X}; \boldsymbol{x} \in \boldsymbol{X}_{J_k}\}.$$

Inductively, the index set of states either in $\boldsymbol{X}_{J_k}$ or within r reactions from $\boldsymbol{X}_{J_k}$ can be defined as $\mathcal{R}_r(J_k) = \mathcal{R}_1(\mathcal{R}_{r-1}(J_k))$.

Update by 5.1. r-step reachability (B Munsky, M Khammash, [7], 2006)

0: $J_{k+1} = J_k$
1: Repeat r times:
$J_{k+1} = \mathcal{R}_1(J_{k+1})$.

5.2. Update by multiple absorbing states

One disadvantage of the r-step reachability algorithm is that of the states that can be reached from the current state space, the probabilities of many are so low that including them in the state space only results in a big matrix without a noticeable improvement in error. The multiple absorbing states method, discussed in [9], attempts to solve the problem. Instead of grouping all the states not in $\boldsymbol{X}_{J_k}$ into only one sink state, the method divides them into L sink states $G_1, \ldots, G_L$ such that G_i contains all states that escape $\boldsymbol{X}_{J_k}$ through reaction R_i. Letting $g_i(t)$ be the probability that the system escapes to G_i from $\boldsymbol{X}_{J_k}$, and $\boldsymbol{g}(t) = (g_1(t), \ldots, g_L(t))^T$, the system then follows the ODEs

$$\begin{pmatrix} \dot{\boldsymbol{p}}_{J_k}(t) \\ \dot{\boldsymbol{g}}(t) \end{pmatrix} = \begin{pmatrix} \boldsymbol{A}_{J_k} & \boldsymbol{0} \\ \boldsymbol{Q} & \boldsymbol{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{p}_{J_k}(t) \\ \boldsymbol{g}(t) \end{pmatrix} \tag{9}$$

instead of (4), where

$$\boldsymbol{Q}(i,j) = \begin{cases} \alpha_i(\boldsymbol{x}_j), & \text{if } (\boldsymbol{x}_j + \boldsymbol{\nu}_i) \notin \boldsymbol{X}_{J_k} \\ 0, & \text{otherwise} \end{cases},$$

The solution of (9) at t_{k+1} is

$$\begin{cases} \boldsymbol{p}_{J_k}(t_{k+1}) = \exp(\tau_k \boldsymbol{A}_{J_k}) \boldsymbol{p}_{J_k}(t_k) \\ \boldsymbol{g}(t_{k+1}) = \left(\int_0^{\tau} \boldsymbol{Q} \exp(\xi \boldsymbol{A}_{J_k}) d\xi \right) \boldsymbol{p}_{J_k}(t_k) + \boldsymbol{g}(t_k), \end{cases} \tag{10}$$

As we can see, the set of ODEs (9) does not give us a better approximation for $\boldsymbol{p}_{J_k}(t)$, but it gives us information about what reactions leak the most probability mass from $\boldsymbol{X}_{J_k}$. To expand the state space we then only consider the directions of the reactions contributing much to the probability mass drop. In [14], the absorbing states are defined by an arbitrary set of nonlinear inequalities, which can avoid the stiffness of the matrix $\boldsymbol{A}_{J_k}$ and therefore can be much more efficient.

Update by 5.2. r-step reachability and multiple absorbing states (B Munsky, M Khammash, [9], 2007, [14], 2011)

0: Start from $k = 0$, time-step τ_k,
tolerance on error ϵ_k,
1: Define a set of functions $\{f_1, \ldots, f_L\}$ and a set of bounds $\{b_1, \ldots, b_L\}$.
2: $\boldsymbol{X}_{J_{k+1}}$ is defined as all states $\boldsymbol{x}$ so that
$f_i(\boldsymbol{x}) \leqslant b_i, \quad i = 1, \ldots, L,$
and G_i contains the all states exiting $\boldsymbol{X}_{J_{k+1}}$ through inequality f_i.
Note: The next two stages in the Variable time-stepping FSP algorithm solve for $\boldsymbol{p}_{J_k}(t_{k+1})$ and $\boldsymbol{g}(t_{k+1})$ by applying (10). In case $\mathbf{1}^T \boldsymbol{g}(t_{k+1}) > \epsilon_k$, we can redo the process and choose a larger b_i when $g_i(t_{k+1})$ is big.

5.3. Update by sliding windows

The sliding windows algorithm updates the state space from $\boldsymbol{X}_{J_k}$ at t_k to $\boldsymbol{X}_{J_{k+1}}$ at t_{k+1} by constructing a window $\boldsymbol{W}_{k+1} \subset \boldsymbol{X}$ that aims to contain the most probability mass during $[t_k, t_{k+1}]$. We then can solve for the probability distribution of $\boldsymbol{W}_{k+1}$ at t_{k+1}, but the whole vector is not stored. Only the states in $\boldsymbol{W}_{k+1}$ having a considerable probability at t_{k+1} are kept in $\boldsymbol{X}_{J_{k+1}}$. Therefore the algorithm can avoid having a big state space, and as a result, computing the probability vector for the next time step can be more efficient.

$\boldsymbol{W}_{k+1}$ is constructed by realizing which states are frequently visited by the system during $[t_k, t_{k+1}]$. The task is done by applying a stochastic approach to estimate the largest and smallest populations attained by each species over $[t_k, t_{k+1}]$, and then use these extremes to form the boundaries of the window $\boldsymbol{W}_{k+1}$.

Since the exact transient dynamics during $[t_k, t_{k+1}]$ is not needed, the extremes are found by a crude random approximation during this time period, instead of using the SSA which could be slow. The time interval $[t_k, t_{k+1}]$ is divided into small equal intervals

$$[t_k, t_k + \Delta, t_k + 2\Delta, \ldots, t_{k+1}].$$

During each interval $[t^{(l)}, t^{(l)} + \Delta]$, the propensity of each reaction R_i is assumed to remain constant to $\alpha_i(\boldsymbol{x}^{(l)})$ where $\boldsymbol{x}^{(l)}$ is the state at $t^{(l)}$. This means that the number of reactions R_i to occur during $[t^{(l)}, t^{(l)} + \Delta]$ is Poisson distributed, with parameter $\alpha_i(\boldsymbol{x}^{(l)})\Delta$. Taking into account that the standard

deviation of the Poisson distribution is $\sqrt{\alpha_i(\boldsymbol{x}^{(l)})\Delta}$, statistically it can be assumed that the actual number of reactions R_i happening during $[t^{(l)},\ t^{(l)}+\Delta]$ is at most

$$\kappa_i^+(\boldsymbol{x}^{(l)},\ \Delta) = \alpha_i(\boldsymbol{x}^{(l)})\Delta + \sqrt{\alpha_i(\boldsymbol{x}^{(l)})\Delta}$$

and at least

$$\kappa_i^-(\boldsymbol{x}^{(l)},\ \Delta) = \max(0,\ \alpha_i(\boldsymbol{x}^{(l)})\Delta - \sqrt{\alpha_i(\boldsymbol{x}^{(l)})\Delta}).$$

Because the interest is in building a window containing considerable probability mass, either of these two extremes is assumed to have happened, for each reaction of type R_i. Continuing until reaching t_{k+1}, we have one trajectory where the worst case scenario happens at each step. The maximum and minimum of x_d, the number of the dth species of state $\boldsymbol{x}$, along all trajectories are then the boundaries for the window $\boldsymbol{W}_{k+1}$.

Update by 5.3. Sliding windows (V Wolf, R Goel, *et al* [15], 2010)

0: Start with $\boldsymbol{X}_{J_k}$ at t_k, time-step τ_k, parameter δ.
1: Find the extremes of each dimension d for the window, b_d^+ and b_d^-. These are estimates of the largest and smallest populations attained by the dth species over $[t_k,\ t_{k+1}]$.
2: The state window will be $\boldsymbol{W}_{k+1} = \boldsymbol{X}_{J_k} \cup \{\boldsymbol{x} \in \boldsymbol{X}\colon b_d^- \leqslant x_d \leqslant b_d^+\}$.
3: Solve for $\boldsymbol{p}(\boldsymbol{W}_{k+1},\ t_{k+1}) = \exp(\tau_k \boldsymbol{A}_{\boldsymbol{W}_{k+1}})\boldsymbol{p}(\boldsymbol{W}_{k+1},\ t_k)$.
4: $\boldsymbol{X}_{J_{k+1}} = \{\boldsymbol{x} \in \boldsymbol{W}_{k+1} : P(\boldsymbol{x},\ t_{k+1}) > \delta\}$, and $\boldsymbol{p}(\boldsymbol{X}_{J_{k+1}},\ t_{k+1})$ is truncated accordingly from $\boldsymbol{p}(\boldsymbol{W}_{k+1},\ t_{k+1})$.

5.4. Update by the optimal state space

The 'optimal' FSP method [16] underlines the problem that methods such as r-step reachability has and that the sliding windows tries to solve: expanding the state space without removing any or most improbable states can result in an unnecessarily big problem to solve. The proposal of the Optimal FSP method is intuitive: after expanding the state space from $\boldsymbol{X}_{J_k}$ at time t_k to $\boldsymbol{X}_{J_{k+1}}$ at time t_{k+1} using conventional methods, we solve for $\boldsymbol{p}_{k+1}(t_{k+1})$ and then remove the states in $\boldsymbol{X}_{J_{k+1}}$ whose probabilities at t_{k+1} are too small.

The question that arises is then how many states do we remove at each step. The algorithm proposes the following approach, consisting of two steps:

- Find the state space $J_{k+1}^{\epsilon/2}$ by any FSP method (the paper applied r-step reachability) so that the 1-norm error at t_{k+1} is less than a prescribed $\frac{\epsilon}{2}$.
- Find and delete the states in $J_{k+1}^{\epsilon/2}$ with the smallest probabilities at t_{k+1} that add up to $\frac{\epsilon}{2}$, resulting in the state space J_{k+1}.

J_{k+1} is then guaranteed to have the 1-norm error at t_{k+1} of at most ϵ.

We need to point out that the method is only 'optimal' in the sense that the resulting state space has the fewest elements while still ensuring that the the 1-norm error is less than a prescribed ϵ.

However, the fact that the first step in the method applies r-step reachability with error $\epsilon/2$ instead of ϵ as in the original r-step reachability ensures that finding the state space will be much slower than other methods, although approximating $\boldsymbol{p}_{k+1}(t_{k+1})$ is faster.

Update by 5.4. Optimal FSP method (V Sunkara, M Hegland, [16], 2012)

0: Start from $\boldsymbol{X}_{J_k}$ at t_k, time-step τ_k and tolerence ϵ_k.
1: Apply r-step reachability to find J_{k+1} so that $1 - \mathbf{1}^T\boldsymbol{p}_{k+1}(t_{k+1}) < \frac{\epsilon_k}{2}$.
2: Sort $\boldsymbol{p}_{k+1}(t_{k+1})$ in descending order, then remove the states with the smallest probabilities at t_{k+1} into J' so that $\mathbf{1}^T\boldsymbol{p}(\boldsymbol{X}_{J'},\ t_{k+1})$ is as close to $\frac{\epsilon_k}{2}$ as possible.
3: Compress the state space at t_{k+1}: $J_{k+1} \leftarrow J_{k+1} - J'$. J_{k+1} is the smallest state index set so that $1 - \mathbf{1}^T\boldsymbol{p}_{k+1}(t_{k+1}) < \epsilon_k$.

5.5. Update by GORDE

The Gated One Reaction Domain Expansion (GORDE) is another method targeted at making r-step reachability more efficient. However, instead of optimizing the state space after solving for the probability vector at t_{k+1} like the Optimal FSP method, GORDE estimates the probabilities of the likely reachable states using a *gating function*.

The main disadvantage of r-step reachability is that it does not evaluate the likelihood of the new states when expanding the state space. The result is that the algorithm expands in the directions of the unlikely states as much as in the directions of the more probable ones. GORDE seeks to solve this by assigning every state $\boldsymbol{x}$ that is m steps from $\boldsymbol{X}_{J_k}$ with the *gating value* $u_m(\boldsymbol{x})$, the probability that the system travels from $\boldsymbol{X}_{J_k}$ to $\boldsymbol{x}$ in exactly m reactions during $[t_k,\ t_{k+1}]$ and has stayed there since. $u_m(\boldsymbol{x})$ is therefore an upper bound of and a crude approximation for $P(\boldsymbol{x},\ t_{k+1})$. It acts like a weight function integrated in r-step reachability: the algorithm only expands in the directions of states $\boldsymbol{x}$ with larger $u_m(\boldsymbol{x})$.

The most practical fact about the gating function is that it can be calculated inductively in m:

$$u_m(\boldsymbol{x}) = \sum_{i=1}^{M} \frac{\alpha_i(\boldsymbol{x}-\boldsymbol{\nu}_i)}{\alpha_{\text{sum}}(\boldsymbol{x}-\boldsymbol{\nu}_i)} \times (1 - e^{-\alpha_{\text{sum}}(\boldsymbol{x}-\boldsymbol{\nu}_i)\tau_k}) u_{m-1}(\boldsymbol{x}-\boldsymbol{\nu}_i). \quad (11)$$

The algorithm for GORDE is then as follows: the gating values for states in $\boldsymbol{X}_{J_k}$ are initialized as $u_0(\boldsymbol{X}_{J_k}) = \boldsymbol{p}_k(t_k)$. $\boldsymbol{X}_{J_k}$ is then expanded in 1-step into $\widetilde{\boldsymbol{\nabla}}_1$, and the gating values for $\widetilde{\boldsymbol{\nabla}}_1$ are then evaluated by

(11). If the gating values for the entire $\widetilde{\nabla}_1$ sum up to be less than a prescribed tolerence ϵ_k, then the algorithm stops and $\boldsymbol{X}_{J_{k+1}} = \boldsymbol{X}_{J_k} \cup \widetilde{\nabla}_1$. If not, choose the smallest set $\nabla_1 \subset \widetilde{\nabla}_1$ containing the highest gating values and expand only from that set in 1-step into $\widetilde{\nabla}_2$, with the new tolerence equals ϵ_k substracted by the gating values of states in $\widetilde{\nabla}_1 - \nabla_1$ and redo the whole process. In the end, if the gating values of all states in $\widetilde{\nabla}_m$ add up to less than the tolerence, then the state space at t_{k+1} is

$$\boldsymbol{X}_{J_{k+1}} = \nabla_0 \bigcup \ldots \bigcup \nabla_{m-1} \bigcup \widetilde{\nabla}_m.$$

Update by 5.5. GORDE (V Sunkara, [17], 2013)

- Start from $\boldsymbol{X}_{J_k}$ at t_k, time-step τ_k and tolerence ϵ_k.
- Initialize $\nabla_0 = \boldsymbol{X}_{J_k}$, $u_0(\boldsymbol{x}) = P(\boldsymbol{x}, t_k)$ and $\tau_0 = \epsilon_k$.
- For $m = 1, 2, \ldots$
  1. Expand ∇_{m-1} by 1-step reachability: $\widetilde{\nabla}_m \leftarrow \mathcal{R}_1(\nabla_{m-1})$.
  2. Compute the gating function for $\boldsymbol{x} \in \widetilde{\nabla}_m$:
     $u_m(\boldsymbol{x}) = \sum_{i=1}^{M} \frac{\alpha_i(\boldsymbol{x} - \boldsymbol{\nu}_i)}{\alpha_{\text{sum}}(\boldsymbol{x} - \boldsymbol{\nu}_i)} \times (1 - e^{-\alpha_{\text{sum}}(\boldsymbol{x}-\boldsymbol{\nu}_i)\tau_k}) u_{m-1}(\boldsymbol{x} - \boldsymbol{\nu}_i)$
     with $\alpha_{\text{sum}} = \sum_{i=1}^{M} \alpha_i$.
  3. Compress $\widetilde{\nabla}_m$ into the smallest ∇_m so that $\sum_{\boldsymbol{x} \in \widetilde{\nabla}_m - \nabla_m} u_m(\boldsymbol{x}) < \tau_{m-1}$.
  4. Stop the loop if $|\nabla_m| = 0$. Otherwise update $\tau_m = \tau_{m-1} - \sum_{\boldsymbol{x} \in \widetilde{\nabla}_m - \nabla_m} u_m(\boldsymbol{x})$.
- The next state space at t_{k+1} is $\boldsymbol{X}_{J_{k+1}} = \nabla_0 \cup \ldots \cup \nabla_{m-1} \cup \widetilde{\nabla}_m$.

5.6. Update by SSA

The SSA-driven method [18] is a combination of the r-step reachability and an approach similar to sliding windows. However, instead of forming the boundaries of a hyper-rectangle as in the sliding windows method, it uses SSA trajectories to build a collection of sets that can possibly be disjoint.

At every step, the method eliminates the states in $\boldsymbol{X}_{J_k}$ that have become improbable. It does so by applying the condition

$$J_{k+\frac{1}{3}} = \{i \in J_k : \mu(\boldsymbol{x}_i) \geqslant \epsilon\},$$

where $\mu(\boldsymbol{x}_i)$ is a dropping criterion, e.g., $\mu(\boldsymbol{x}_i) = \max_{j=k-\ell,\ldots,k} p_i(t_j)$ calculates the highest probability $\boldsymbol{x}_i$ has in the last ℓ steps.

The method then runs the SSA from the states in $J_{k+\frac{1}{3}}$ and saves all the states visited along the trajectories as $J_{k+\frac{2}{3}}$. To smooth these random trajectories the method further applies r-step reachability on $J_{k+\frac{2}{3}}$. The result of this will be the state space J_{k+1} at t_{k+1}.

A feature central to the efficiency of the method is its 'lazy evaluation'. The SSA runs are only performed when deemed necessary by an error control mechanism, and even then, the SSA trajectories are only extended as far as needed for the suitability of J_{k+1}.

Update by 5.6. SSA-driven method (R Sidje, H Vo, [18], 2015)

0: Start from $\boldsymbol{X}_{J_k}$ at t_k, time-step τ_k, parameters ℓ and r, and the tolerence ϵ.
1: Eliminate the states in $\boldsymbol{X}_{J_k}$ with low probabilities:
$J_{k+\frac{1}{3}} = \{i \in J_k : \mu(\boldsymbol{x}_i) \geqslant \epsilon\}$, where
$\mu(\boldsymbol{x}_i) = \max_{j=k-\ell,\ldots,k} p_i(t_j)$.
2: $J_{k+\frac{2}{3}} = \bigcup_{i \in J_{k+\frac{1}{3}}} \text{SSA}(\boldsymbol{x}_i, t_k, \tau_k)$.
3: J_{k+1} is the r-step expansion of $J_{k+\frac{2}{3}}$.

6. Precondition the FSP

The original FSP algorithm proceeds with the matrix exponential right after finding the state space. This can sometimes be slow, either because there is a difference in the magnitudes of the reaction rate constants, which causes the matrix to be stiff, or the state space is simply too large. Time scale separation [19] improves the numerical performance in the former case, by applying the perturbation theory, and aggregation [20] solves the latter case by simply grouping states together.

6.1. Precondition by time scale separation

Time scale separation is based on the authors' observation in [19] that in some biological cases, some reactions can have higher propensities and therefore happen more frequently than other reactions. The result is that there are clusters of states, the states within the same cluster can transition regularly to each other, and transitions between clusters are rare, which implies that the generator $\boldsymbol{A}$ can be divided into

$$\boldsymbol{A} = \boldsymbol{H} + \epsilon \boldsymbol{V},$$

where $\boldsymbol{H}$ is a block diagonal matrix, and $\epsilon \ll 1$. Each block of $\boldsymbol{H}$ represents a proper master equation for the states in one cluster, and $\epsilon \boldsymbol{V}$ contains the transition rates between these clusters.

Since the blocks of $\boldsymbol{H}$ represent ODEs for a proper master equation, and their dimensions are much less than the dimension of $\boldsymbol{A}$, it is computationally cheap to compute their eigensystems, which is equivalent to having the eigensystem of $\boldsymbol{H}$:

$$\boldsymbol{S}: \quad \boldsymbol{S}^{-1}\boldsymbol{H}\boldsymbol{S} = \tilde{\boldsymbol{\Lambda}}$$
$\boldsymbol{S}$ has same block diagonal structure as $\boldsymbol{H}$
$\tilde{\boldsymbol{\Lambda}} = diag(\tilde{\lambda}_1, \ldots, \tilde{\lambda}_n)$

We rearrange $\tilde{\lambda}_1, \ldots, \tilde{\lambda}_n$ into a decreasing sequence

$$Re(\lambda_1) \geqslant \ldots \geqslant Re(\lambda_n).$$

Notice that if $\boldsymbol{H}$ has m blocks, then $\lambda_1 = \ldots = \lambda_m = 0$, Let $\boldsymbol{S}^R \in \mathbb{R}^{N \times m}$ and $\boldsymbol{S}^L \in \mathbb{R}^{m \times N}$ contain the right and left eigenvectors of $\boldsymbol{H}$ for these zero eigenvalues (which construct the right and left null-spaces of $\boldsymbol{H}$), respectively. Note that the left eigenvectors are all $\mathbf{1}$.

Using perturbation theory, the authors observed that the projection

$$\widetilde{V} = S^L V S^R$$
$$\mathcal{V}(t) = S^R \exp(\epsilon \widetilde{V} t) S^L \quad (12)$$

gives the asymptotic approximation to the problem:

$$t > T(\epsilon): \quad \|p(t) - \mathcal{V}(t)p(0)\| = \mathcal{O}(\epsilon) \quad (13)$$

The time it takes for the approximation to be applicable is estimated to be

$$T(\epsilon) \sim ln(\epsilon / Re(\lambda_{m+1})).$$

If the distinction between the clusters of states is clear, then it is guaranteed that $Re(\lambda_{i \geqslant m}) \ll -\epsilon$, and $T(\epsilon)$ is small. However, there can be cases where the λ_i's decrease only mildly at first, and $T(\epsilon)$ can be larger than the end point t_f. When such a problem is encountered, we can simply add in the right and left eigenvector of H corresponding to λ_{m+1} in S^L and S^R (the left eigenvector no longer being **1**). The projection (12) then satisfies (13) with

$$T(\epsilon) \sim ln(\epsilon / Re(\lambda_{m+2})).$$

If the new time restriction is still too big, we can keep adding in eigenvectors of H until the condition $T(\epsilon) < \tau$ is satisfied.

Precondition by 6.1. Time scale separation (S Peles, B Munsky, M Khammash, [19], 2006)

0: Start with $A_{J_{k+1}}$ and time-step τ_k. Separate $A_{J_{k+1}}$ into H and ϵV, and m is the number of blocks in H.
1: Find S^R, S^L corresponding to $\lambda_1, \ldots, \lambda_m$.
2: $T(\epsilon) = ln(\epsilon / Re(\lambda_{m+1}))$. Continue if $T(\epsilon) < \tau_k$, otherwise increase m and return to step 1.
3: Compute $\widetilde{V} = S^L V S^R$.
4: Approximate $\exp(\epsilon \tau_k \widetilde{V})$.
5: Compute $\mathcal{V}(\tau_k) = S^R \exp(\epsilon \tau_k \widetilde{V}) S^L$, then we have $p_{k+1}(t_{k+1}) \approx \mathcal{V}(\tau_k) p_k(t_k)$.

6.2. Precondition by aggregation

There are many cases where the FSP matrix $A_{J_{k+1}}$, already reduced from A, is still too large to store or compute. One method to even further reduce the size of the matrix is aggregation [20].

The method partitions J_{k+1} into a small number of disjoint subsets

$$J_{k+1} = \bigcup_{\ell} J_{k+1,\ell}.$$

Let y_ℓ be one state representing the whole set $X_{J_{k+1,\ell}}$, and its probability equals the probability mass of $X_{J_{k+1,\ell}}$:

$$P(y_\ell, t) = \sum_{x \in X_{J_{k+1,\ell}}} P(x, t).$$

Let the *aggregation operator* E define this conversion from the probability vector of $X_{J_{k+1,\ell}}$ into the probability distribution of $Y = \{y_\ell\}$:

$$p(Y, t) = E \cdot p(X_{J_{k+1}}, t),$$

and the *disaggregation operator* F, which approximates $p(X_{J_{k+1,\ell}}, t)$ from $p(Y, t)$, is the right inverse of E:

$$E \cdot F = I_Y.$$

The Markov generator for Y can be reduced from $A_{J_{k+1}}$:

$$B = E \cdot A_{J_{k+1}} \cdot F.$$

It can be shown that B represents a proper master equation: its off-diagonal elements are nonnegative and each column sums up to 0. The ODEs for the distribution of Y are then

$$\begin{cases} \dot{p}(Y, t) = B \cdot p(Y, t), & t > t_k \\ p(Y, t_k) = E \cdot p(X_{J_{k+1}}, t_k) \end{cases}$$

which we can solve using techniques in section 7, then $p(X_{J_{k+1,\ell}}, t_{k+1})$ can be deduced by

$$p(X_{J_{k+1}}, t_{k+1}) = F \cdot p(Y, t_{k+1}), \quad (14)$$

which means that the states in each $X_{J_{k+1,\ell}}$ will have the same probability. In practice, if the shape of the probability distribution is known, then different strategies to disaggregate from $p(Y, t)$ to $p(X_{J_{k+1}}, t)$ can be employed instead of (14), most notably interpolation. We refer to [21] for different methods toward this end.

Precondition by 6.2. Aggregation (M Hegland, C Burden, L Santoso, S MacNamara, H Booth, [20], 2005)

0: Start with $X_{J_{k+1}}$ and $A_{J_{k+1}}$ and τ_k.
1: Divide $X_{J_{k+1}}$ into $X_{J_{k+1,\ell}}$'s.
2: Find the operators E and F based on the aggregation.
3: Compute
$B = E \cdot A_{J_{k+1}} \cdot F$,
and
$p(Y, t_k) = E \cdot p_{k+1}(t_k)$.
4: Solve for
$p(Y, t_{k+1}) = \exp(\tau_k B) p(Y, t_k)$.
5: We have
$p_{k+1}(t_{k+1}) \approx F \cdot p(Y, t_{k+1})$,
or by other methods, as reported in [21].

7. Approximate the solution

The last stage in each step of the adaptive time-stepping FSP algorithm is to approximate $p_{k+1}(t_{k+1}) = \exp(\tau_k A_{J_{k+1}}) \cdot p_k(t_k)$. Obviously, p_{k+1} is the solution of

$$\dot{p}(t) = A_{J_{k+1}} \cdot p(t) \quad (15)$$

at t_{k+1}, where $p(t_k) = p_k(t_k)$. Therefore, standard ODE solution techniques can be applied, for instance Runge–Kutta methods.

Here we will address some techniques to solve (15) which take advantage of the fact that $A_{J_{k+1}}$ is constant over time. Krylov subspace techniques have proved very efficient for solving this kind of problem for large

sparse $\boldsymbol{A}_{J_{k+1}}$, by projecting it down to a small dense matrix, the exponential of which can then be approximated by Padé or Chebyshev polynomials.

A different approach to solving this problem is the uniformization method, which truncates the Taylor series expansion of the matrix.

7.1. Approximate by uniformization

Even if $\boldsymbol{A}_{J_{k+1}}$ is preconditioned or not, each step in the variable time-stepping FSP algorithm ends with approximating the probability vector at time t_{k+1}, in the form of a matrix exponential multiplied by a vector:

$$\boldsymbol{p}_{J_{k+1}}(t_{k+1}) \approx \exp(\tau_k \boldsymbol{A}_{J_{k+1}})\boldsymbol{p}_{J_k}(t_k).$$

We will simplify the notation so that the problem is approximating

$$\exp(\tau \boldsymbol{A})\boldsymbol{v}, \tag{16}$$

where the matrix exponential is defined by

$$\exp(\tau \boldsymbol{A}) = \sum_{k=0}^{\infty} \frac{(\tau \boldsymbol{A})^k}{k!}. \tag{17}$$

The idea behind uniformization is to use a truncation of the series that avoids roundoff errors. To do so, the method applies the transformation

$$\boldsymbol{P} = \boldsymbol{I} + \frac{1}{\alpha}\boldsymbol{A}$$
$$\alpha = \max_i |\boldsymbol{A}_{ii}|$$

where $\boldsymbol{P}$ now has nonnegative entries, and then approximates (16) using

$$\begin{aligned}\exp(\tau \boldsymbol{A})\boldsymbol{v} &= \exp(\alpha\tau(\boldsymbol{P} - \boldsymbol{I}))\boldsymbol{v} \\ &= e^{-\alpha\tau}\exp(\alpha\tau \boldsymbol{P})\boldsymbol{v} \\ &\approx \sum_{k=0}^{\ell} e^{-\alpha\tau}\frac{(\alpha\tau)^k}{k!}\boldsymbol{P}^k\boldsymbol{v}.\end{aligned}$$

The last aspect of the uniformization method to consider is choosing the number of iterations ℓ. Noting that the way $\boldsymbol{P}$ is defined implies that $0 \leqslant \boldsymbol{P} \leqslant 1$ component-wise, and $\|\boldsymbol{P}\|_1 = 1$, from which it can be shown that

$$\left\| \exp(\tau \boldsymbol{A})\boldsymbol{v} - \sum_{k=0}^{\ell} e^{-\alpha\tau}\frac{(\alpha\tau)^k}{k!}\boldsymbol{P}^k\boldsymbol{v} \right\|_1 \leqslant \epsilon$$

if

$$1 - \sum_{k=0}^{\ell} e^{-\alpha\tau}\frac{(\alpha\tau)^k}{k!} \leqslant \epsilon.$$

In practice, the integration interval $[t_k,\ t_{k+1}]$ is usually subdivided to avoid overflow issues. This is often done by choosing a parameter θ ([22] suggests $\theta = 100$) and writing the solution as

$$m = \left\lceil \frac{\alpha\tau}{\theta} \right\rceil$$
$$\bar{\tau} = \frac{\tau}{m}$$
$$\exp(\tau \boldsymbol{A})\boldsymbol{v} = (e^{-\alpha\bar{\tau}}\exp(\alpha\bar{\tau}\boldsymbol{P}))^m\boldsymbol{v}$$

then evaluating the last equation by starting from $\boldsymbol{\omega} = \boldsymbol{v}$ and iterating

$$\boldsymbol{\omega} \leftarrow e^{-\alpha\bar{\tau}}\exp(\alpha\bar{\tau}\boldsymbol{P})\boldsymbol{\omega}$$

m times.

Approximate by 7.1. Uniformization (W Grassmann, [23], 1977 D Gross, D Miller, [24], 1984)

0: Start with $\boldsymbol{p}_k(t_k)$, $\boldsymbol{A}_{J_{k+1}}$, τ_k and parameter θ
1: Initialize $\alpha = \max|\mathrm{diag}(\boldsymbol{A}_{J_{k+1}})|$ and
$\boldsymbol{P} = \boldsymbol{I} + \frac{\boldsymbol{A}_{J_{k+1}}}{\alpha}$.
2: Find parameters $m = \left\lceil \frac{\alpha\tau}{\theta} \right\rceil$ and $\bar{\tau} = \frac{\tau_k}{m}$.
3: Choose ℓ so that
$1 - \sum_{k=0}^{\ell} e^{-\alpha\bar{\tau}}\frac{(\alpha\bar{\tau})^k}{k!} \leqslant \epsilon$.
4: Start with $\boldsymbol{\omega} = \boldsymbol{P}_k(t_k)$.
5: Iterate
$\boldsymbol{\omega} \leftarrow \sum_{l=0}^{\ell} e^{-\alpha\tau}\frac{(\alpha\bar{\tau})^l}{l!}\boldsymbol{P}^k\boldsymbol{\omega}$
m times.
6: Return $\boldsymbol{p}_{k+1}(t_{k+1}) = \boldsymbol{\omega}$.

7.2. Approximate by Krylov-based techniques

One of the most effective methods to approximate the solution in the form of (16) is by using Krylov-based techniques. Given a vector $\boldsymbol{v}$ and matrix $\boldsymbol{A}$, we define the Krylov subspace of order m to be

$$\mathcal{K}_m(\boldsymbol{A},\ \boldsymbol{v}) = span\{\boldsymbol{v},\ \boldsymbol{A}\boldsymbol{v}, \ldots, \boldsymbol{A}^{m-1}\boldsymbol{v}\}.$$

The well-known Arnoldi process produces an orthonormal basis $\boldsymbol{V}_m$ of $\mathcal{K}_m(\boldsymbol{A},\ \boldsymbol{v})$, and an upper Hessenberg matrix $\boldsymbol{H}_m$. The Krylov approximation is then

$$\exp(\tau \boldsymbol{A})\boldsymbol{v} \approx \beta \boldsymbol{V}_m \exp(\tau \boldsymbol{H}_m)\boldsymbol{e}_1 \tag{18}$$

where $\beta = \|\boldsymbol{v}\|_2$ and $\boldsymbol{e}_1 = (1,\ 0, \ldots, 0)^T$.

We still need to approximate $\exp(\tau \boldsymbol{H}_m)$ in this equation. This can be done using the Padé approximation, together with scaling and squaring. This is a much cheaper problem than approximating $\exp(\tau \boldsymbol{A})$, since it has been shown in experiments that (18) yields a good approximation even when $m = 40$ or less.

The authors in [8] proposed integrating the Krylov method in the time-stepping FSP algorithm more than just to approximate $\boldsymbol{P}_{k+1}(t_{k+1})$. They did so by examining the local error

$$\Gamma = \boldsymbol{1}^T \boldsymbol{p}_{k+1}(t_{k+1})$$

and decreasing the step-size by half and reapplying Krylov method if the condition

$$\Gamma > 1 - \epsilon\frac{t_{k\ +\ 1}}{t_f} \tag{19}$$

fails to be satisfied. The algorithm keeps halving the step-size until we have (19). Only then, the algorithm moves on to the new time point.

At the new time point, the algorithm only expands the state space if the time-step was reduced in the last time interval. This guarantees that the algorithm does not waste time finding and operating with a larger matrix unless it has to.

Approximate by 7.2. Krylov subspace

0: Start with the Krylov order m, $\boldsymbol{p}_k(t_k)$, $\boldsymbol{A}_{J_{k+1}}$, and an initial choice for τ_k.
1: Apply the Arnoldi process to find $\boldsymbol{V}_m$ and $\boldsymbol{H}_m$ for $\mathcal{K}_m(\boldsymbol{A}_{J_{k+1}}, \boldsymbol{p}_k(t_k))$.
2: Approximate the solution at $t_{k+1} = t_k + \tau_k$:
$\boldsymbol{p}_{k+1}(t_{k+1}) \approx \beta \boldsymbol{V}_m \exp(\tau_k \boldsymbol{H}_m)\boldsymbol{e}_1$,
where $\beta = \|\boldsymbol{p}_k(t_k)\|$ $\exp(\tau \boldsymbol{H}_m)$ is approximated by the Padé approximation, with scaling and squaring.
3: $\Gamma = \mathbf{1}^T \boldsymbol{p}_{k+1}(t_{k+1})$.
4: Stop if $\Gamma > 1 - \epsilon \frac{t_{k+1}}{t_f}$ (see note), otherwise reduce τ_k by half and return to Step 2.
Note: For the next step in the time-stepping FSP algorithm, only expand the state space if halving happened in this step.

8. Tensor decomposition alternative

As will be shown in the example in section 9, the biggest disadvantage of solving the CME is the 'curse of dimensionality': the size of matrix $\boldsymbol{A}$ is great even when there are only a few species at play. In such cases, even if $\boldsymbol{A}$ is sparse, operating with it can be costly. In recent years, a lot of work has been done in decomposing the CME matrix using tensors, which promises a reduction in storage space.

In this section, we assume the FSP to take the form of a hyper-rectangle

$$\boldsymbol{X}_{J_{k+1}} = \{0, 1, \ldots, n_1\} \times \ldots \times \{0, 1, \ldots, n_N\}. \quad (20)$$

As a slight abuse of notation, we will use $\boldsymbol{A}$ instead of $\boldsymbol{A}_J$ to refer to the submatrix corresponding to the hyper-rectangle.

As a motivation, we note that under mild conditions the infinitesimal generator $\boldsymbol{A}$ can be decomposed into a sum of tensor products [25–27]

$$\boldsymbol{A} = \sum_{k=1}^{M} \otimes_{i=1}^{N} \boldsymbol{S}_k^i \boldsymbol{M}_k^i - \otimes_{i=1}^{N} \boldsymbol{M}_k^i, \quad (21)$$

where each term in the sum corresponds to a chemical reaction, the matrix $\boldsymbol{S}_k^i \in \mathbb{R}^{n_i \times n_i}$ is the 'shifted-diagonal' matrix corresponding to the change in species i when reaction k happens, and the diagonal matrix $\boldsymbol{M}_k^i \in \mathbb{R}^{n_i \times n_i}$ that stores the values of i-factor in the propensity function α_k. This allows the matrix $\boldsymbol{A}$ to be stored in $O(MN \max(n_i))$ terms instead of $O(Mn_1 n_2 \ldots n_N)$ terms of a straightforward sparse storage. Moreover, each constituent matrix can be subject to further compression techniques that improve further the memory management. We now turn to some tensor-based techniques that seek to compress the long probability vector $\boldsymbol{p}$.

The fact that $\boldsymbol{p}$ is indexed by the multi-dimensional states allows it to be reshaped into an $n_1 \times \ldots \times n_N$-dimensional array (or tensor), making the CME a natural target for tensor decompositions. The earliest attempt to apply tensor decomposition in the CME context that we know of is by Hegland and Garcke in 2011 [25], who sought approximations of the form

$$\boldsymbol{p} \approx \sum_{j=1}^{r} \otimes_{i=1}^{N} \boldsymbol{p}^i, \quad (22)$$

where the number of terms r is called the *rank* of the tensor decomposition and is made as small as possible for a prescribed tolerance. The approximation can be stored in $O(rN \max(n_i))$ and this avoids the curse of dimensionality if r is small. About one year later, Dolgov and Khoromskij proposed a different approach that used the Tensor Train (TT) format [28]. The TT decomposition breaks $\boldsymbol{p}$ into the 3-dimensional boxes $\boldsymbol{G}_0, \boldsymbol{G}_1, \ldots, \boldsymbol{G}_N$ where $\boldsymbol{G}_i \in \mathbb{R}^{n_i \times r_i \times n_{i+1}}$ (with $n_0 = n_{N+1} = 1$). The numbers r_i are called the TT-ranks of $\boldsymbol{p}$ and each $\boldsymbol{G}_i$ a TT-core. The probability at any state $(x_1, \ldots, x_N)$ is recovered by

$$\boldsymbol{p}(x_1, \ldots, x_N) = \sum_{i_0=1}^{r_0} \cdots \sum_{i_{N-1}=1}^{r_{N-1}} \times \boldsymbol{G}_0(i_0, x_1)\boldsymbol{G}_1(x_1, i_1, x_2) \cdots \boldsymbol{G}_N(x_N, i_N),$$

which is an instance of *tensor contraction*. The TT approach reduces the $O(n_1 \cdots n_N)$ storage of the full tensor $\boldsymbol{p}$ into $O(Nn^2 r_{tt})$ where $n = \max\{n_i\}$ and $r_{tt} = \max\{r_i\}$. If r_{tt} is small the compression rate is tremendous. The quantized tensor train (QTT) format used in Kazeev *et al* [27] takes the compression of the TT approach further by reshaping the already high-dimensional probability tensor $\boldsymbol{p}$ into an even higher-dimensional tensor with many virtual dimensions (as opposed to the physical dimensions represented by the chemical species). The TT decomposition is then applied on top of this reshaped tensor to achieve a higher compression rate. This is perhaps one of the most fascinating features of the tensor approaches that can potentially turn the curse of dimensionality into a blessing (to paraphrase [29]). Finally, we note that the techniques described in this paragraph can be applied equally to compress the matrix $\boldsymbol{A}$ itself to potentially overcome the memory explosion issue in solving the CME.

So far we have only mentioned the compression strategies for the large matrix and vector in the CME using tensor decompositions. The challenge is to design numerical schemes that maintain the benefits brought by these techniques. Unfortunately, classical matrix methods do not lend themselves easily to the new formats. The work of Dolgov in adapting the

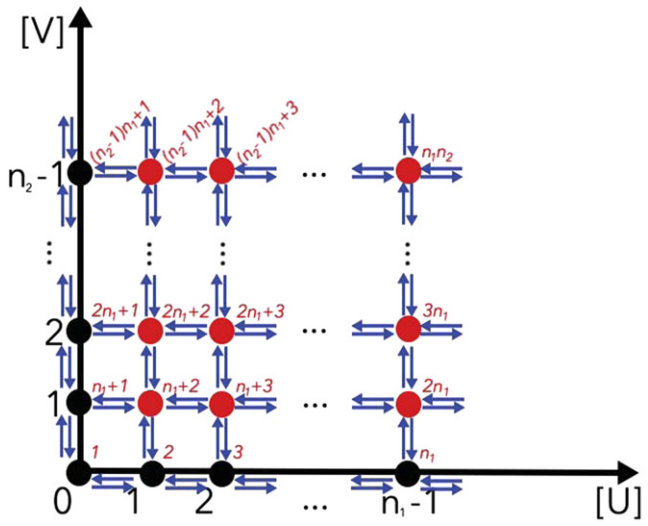

Figure 1. A lattice describing all possible states for the stochastic gene toggle model, in example 1. The limits in the FSP for gene U is n_1, and gene V is n_2. The red numbers are the state indices.

GMRES method to TT format reveals incompatibility between Krylov subspace methods and TT decomposition: the TT-ranks of the Krylov vectors given by the Arnoldi iteration increase even though both $\boldsymbol{A}$ and $\boldsymbol{p}$ have low TT-ranks. There are, however, promising tools being developed and analyzed for the tensor format such as the Density matrix renormalization group (DMRG) solver for linear systems in tensor format. Based on this, implicit time-stepping schemes can be employed to integrate the CME. This is essentially the approach of Kazeev *et al* [27], where the *hp-discontinuous Galerkin* scheme is applied successfully in many non-trivial CME problems in quantized TT format. Alternatively, classical–time-stepping schemes like implicit Euler can be used to form a global linear system in tensor format to solve once and for all for the snapshots of the time-dependent probability distribution as done by [30]. Such scheme would have been costly in traditional matrix-vector format, but becomes much more feasible in tensor format due to its strong compression ability. We refer to the numerical results in the cited paper that show the prospects of this new approach.

The tensor decomposition approaches to the CME are just a wave in the growing currents of tensor techniques with wide applications in different fields of science. We refer to the reviews by [29–30].

9. Illustrative example

We will consider the stochastic gene toggle model, which is a simple model usually employed for its interesting properties. This example is meant to showcase how the matrix used in the CME is generated. There are two species in the model, U and V, and the production of each has a negative feedback on the production of the other species. The four reactions that U and V participate in and their propensities are: [18, 32]

$$\varnothing \xrightarrow{k_1} \mathrm{U}: \quad \alpha_1 = k_1$$
$$\mathrm{U} \xrightarrow{k_2} \varnothing: \quad \alpha_2 = k_2[\mathrm{U}]$$
$$\varnothing \xrightarrow{k_3} \mathrm{V}: \quad \alpha_3 = k_3$$
$$\mathrm{V} \xrightarrow{k_4} \varnothing: \quad \alpha_4 = k_4[\mathrm{V}]$$

The parameters, taken from [33], showcase the feedback between the two species: $k_1 = 0.2 + \frac{4}{1+[\mathrm{V}]^3}$, $k_2 = 1.09$, $k_3 = 0.2 + \frac{4}{1+[\mathrm{U}]^3}$, $k_4 = 1$. k_1 and k_3 can be thought of as functions of [V] and [U], respectively. We set the initial state as $\boldsymbol{x}_0 = ([\mathrm{U}], [\mathrm{V}]) = (85, 5)$.

Since the numbers of U and V can be any non-negative numbers, we need a bound so that the FSP matrix is finite dimensional. Let $n_1 - 1$ and $n_2 - 1$ be the upper bounds on [U] and [V], so there are a total of $n_1 n_2$ states that we keep track of (because [U] and [V] can be 0).

Each state of the system consists of the numbers of genes U and V. To identify the state with a single number, we need the following indexing formula

$$i([\mathrm{U}], [\mathrm{V}]) = [\mathrm{U}] + 1 + n_1[\mathrm{V}].$$

The states and reactions are shown in figure 1. The state index $i([\mathrm{U}], [\mathrm{V}])$ is shown in red.

We will now formulate the ODE describing the evolution of the probabilities over time. The vector form of the ODE is

$$\dot{\boldsymbol{p}}(t) = \boldsymbol{A} \cdot \boldsymbol{p}(t),$$

where matrix $\boldsymbol{A}$ in $\mathbb{R}^{n_1 n_2 \times n_1 n_2}$, has the block form of

$$\begin{bmatrix} \boldsymbol{D}_0 & \boldsymbol{C}_1 & \boldsymbol{0} & \dots & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{B} & \boldsymbol{D}_1 & \boldsymbol{C}_2 & \dots & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{B} & \boldsymbol{D}_2 & \dots & \boldsymbol{0} & \boldsymbol{0} \\ \vdots & \vdots & \vdots & \ddots & & \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & & \boldsymbol{D}_{n_2-2} & \boldsymbol{C}_{n_2-1} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & & \boldsymbol{B} & \boldsymbol{D}_{n_2-1} \end{bmatrix},$$

where

$$\boldsymbol{B} = \mathrm{diag}(k_3(0), \dots, k_3(n_1 - 1)) \in \mathbb{R}^{n_1 \times n_1}$$

accounts for the production of V, which can be thought of as an upward transition in figure 1. On the other hand,

$$\boldsymbol{C}_m = \mathrm{diag}(m \cdot k_4, \dots, m \cdot k_4) \in \mathbb{R}^{n_1 \times n_1}$$

describes the downward transition of states in row m of figure 1, through the death of one gene V.

Finally, the diagonal blocks are computed as

$$\begin{aligned} \boldsymbol{D}_0 &= \boldsymbol{T}_0 - \boldsymbol{B} \\ \boldsymbol{D}_m &= \boldsymbol{T}_m - \boldsymbol{B} - \boldsymbol{C}_m, \quad m \geqslant 1 \end{aligned}$$

where $\boldsymbol{T}_m$ depicts the left and right transitions within row m of figure 1, through the birth or death of one gene U:

$$\boldsymbol{T}_m(i, j) = \begin{cases} -k_1(m) & i = j = 1 \\ -k_1(m) - (j-1) \cdot k_2 & i = j > 1 \\ (j-1) \cdot k_2 & j = i + 1 \\ k_1(m) & j = i - 1 \end{cases}$$

The gene toggle model is well-known for its bistability: there are two different stable steady modes that the system can converge to, which can be seen easily using the SSA-driven FSP method [18], shown in the last row of figure 2.

The first row in figure 2 shows the distributions from applying the SSA with 10^5 trajectories. All of the trajectories converge to only one of the two steady states. The reason is transparent: the result from the SSA-driven FSP method informs us that the probabilities of the states in the second mode are between 10^{-8} and 10^{-13}. Therefore many more trajectories would be needed for the SSA to reach bimodality. This is shown more clearly when 10^8 trajectories are simulated by the SSA, as shown in the second row of figure 2.

Therefore this example shows that in some cases, solving the CME by using the FSP can be both more accurate and much faster than applying the SSA or other stochastic methods.

10. Softwares

Here we discuss some softwares and packages that illustrate the methodology of the FSP and serve as an efficient means to solve the CME.

10.1. FSP Toolkit

FSP Toolkit is a MATLAB software for solving the CME for two species using the FSP, which can be found at

http://cnls.lanl.gov/~munsky/Software.html

Reference [14] describes the numerical method in detail. The initial state space is defined by a set of nonlinear inequalities, and it is expanded by applying r-step reachability and multiple absorbing states. A number of stochastic phenomena involving two species in biological systems are illustrated in the software, including activation through linear regulation, activation with a convex or concave function, and toggle switch. The toolkit is very well explained and therefore recommended as a valuable resource for people new to the FSP approach.

10.2. CMEPy

CMEPy is a Python package for solving the CME, which can be downloaded and installed at

http://fcostin.github.io/cmepy/index.html

The program expands the state space by r-step reachability. It can also solve for the case where the propensities are time-dependent, but only when a separation of variables can be applied:

$$\alpha_k(\boldsymbol{x}, t) = \phi_k(\boldsymbol{x})\theta(t).$$

10.3. Expokit

As mentioned in section 7, Expokit is one of the most efficient software to calculate the matrix exponential of either small dense or very large sparse matrices. The package is written in Fortran and MATLAB [34], and can be found at

http://www.maths.uq.edu.au/expokit/

It is the basis of ongoing solution techniques of the FSP.

11. Discussion

The FSP is an especially effective method in a number of gene expression regulation problems for several reasons. First of all, there are few species involved and upper limits on these species numbers are usually given either from experimental data or theoretical biology, implying that the size of CME may occasionally be manageable. Secondly, the goal in these problems is usually to find the model that explains the experimental data, and to find the model parameter (i.e. a vector of reaction rates) that results in the probability distributions that best fit the data at different times. Since thousands or even millions of different model parameters have to be computed and compared, the probability distributions have to be solved efficiently and fast, in which case the FSP has an advantage over kinetic Monte Carlo simulations, which require large numbers of simulations.

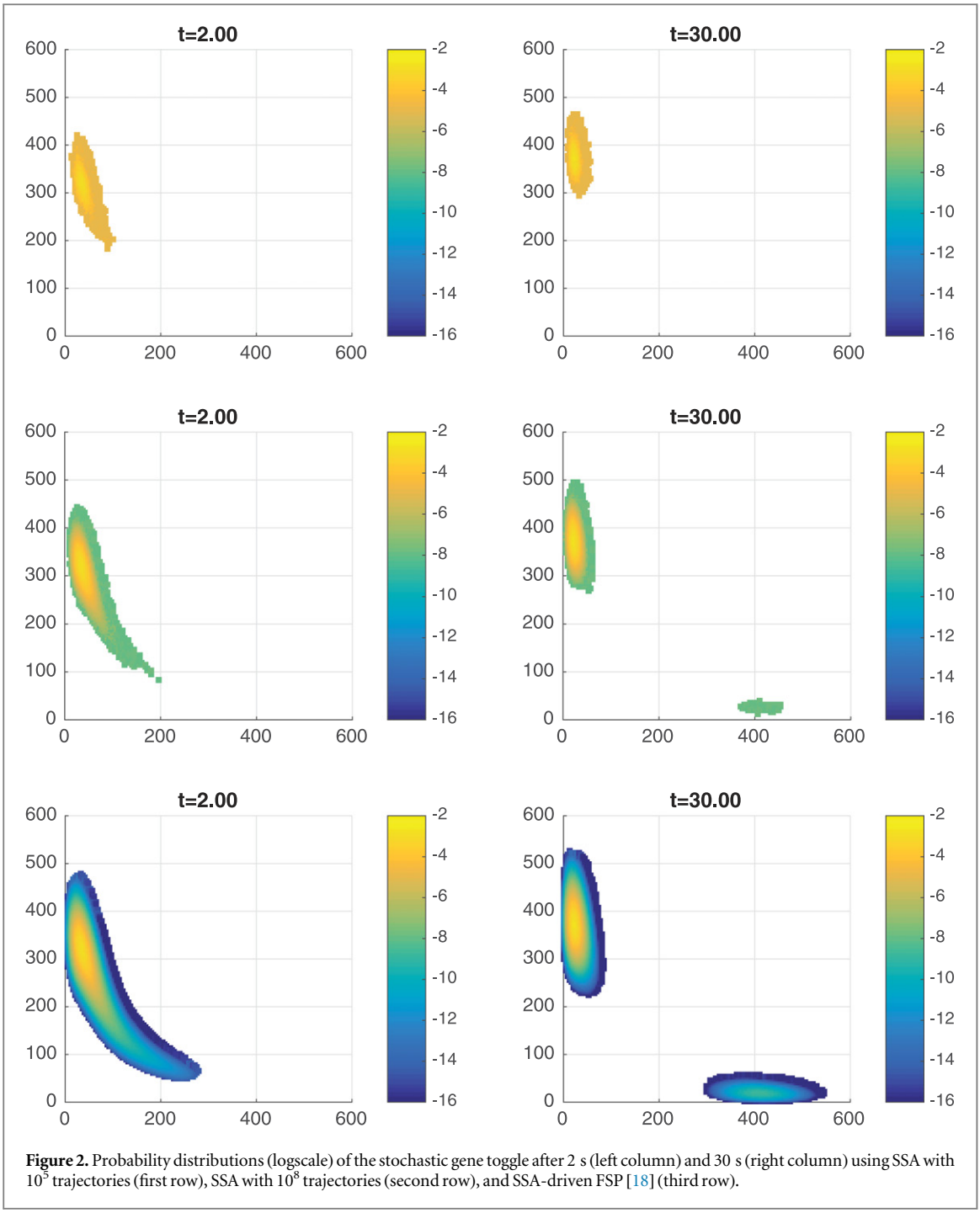

Figure 2. Probability distributions (logscale) of the stochastic gene toggle after 2 s (left column) and 30 s (right column) using SSA with 10^5 trajectories (first row), SSA with 10^8 trajectories (second row), and SSA-driven FSP [18] (third row).

We refer to [35] which contains a number of good examples where the FSP is applied in real-life gene expression regulation problems.

However, the 'curse of dimensionality' makes solving the CME numerically difficult if not impossible in the case where there are many species. An example for such a case is the regulation of protein p53 [36], where there are six species of interest, interacting with each other through 11 reactions. A bound B is set to be the maximum number of molecules of each species that the cell can contain. It is then obvious that the number of states that the system can be in is roughly B^6, where B can be thousands. We then end up with a very large, although sparse, matrix $\boldsymbol{A}$ for the CME. In practice, when we solve the CME using real parameters as reported in [36], the probability mass requires a projection of over 4 million states and at each time point, expanding the state space to the next time step would explode the projection up to over 15 million states.

When such a huge system is encountered, the FSP method fails and the best numerical methods to employ are the SSA and other Monte Carlo methods.

12. Conclusion

The amount of research efforts in the last few years that built on the finite state projection method is the most

convincing evidence of the importance of the method in solving the chemical master equation. Variants of the original algorithm have led to tremendous improvements. This tutorial offered a review of these methods in a systematic fashion. We outlined the core ideas behind the variants, and highlighted similarities and differences between them. We note that in addition to biological applications, the FSP is found useful in other areas as well [37].

Acknowledgments

Work supported by NSF grant DMS-1320849. The authors thank the referees for their helpful comments.

References

[1] Goutsias J and Jenkinson G 2013 Markovian dynamics on complex reaction networks *Phys. Rep.* **529** 199–264
[2] Gillespie D 1992 A rigorous derivation of the chemical master equation *Physica* A **188** 404–25
[3] Gillespie D 1976 A general method for numerically simulating the stochastic time evolution of coupled chemical reactions *J. Comput. Phys.* **22** 403–34
[4] Gillespie D 2001 Approximate accelerated stochastic simulation of chemically reacting systems *J. Chem. Phys.* **115** 1716–33
[5] Cao Y, Gillespie D and Petzold L 2006 Efficient step size selection for the tau-leaping simulation method *J. Chem. Phys.* **124** 044109
[6] Cao Y, Gillespie D and Petzold L 2005 The slow-scale stochastic simulation algorithm *J. Chem. Phys.* **122** 14116
[7] Munsky B and Khammash M 2006 The finite state projection algorithm for the solution of the chemical master equation *J. Chem. Phys.* **124** 044104
[8] Burrage K, Hegland M, Macnamara S and Sidje R 2006 A Krylov-based finite state projection algorithm for solving the chemical master equation arising in the discrete modelling of biological systems *Markov Anniversary Meeting: an international conference to celebrate the 150th anniversary of the birth of A.A. Markov* pp 21–38
[9] Munsky B and Khammash M 2007 A multiple time interval finite state projection algorithm for the solution to the chemical master equation *J. Comput. Phys.* **226** 818–35
[10] Neuert G, Munsky B, Tan R Z, Teytelman L, Khammash M, Oudenaarden A V and van Oudenaarden A 2013 Systematic identification of signal-activated stochastic gene regulation *Science* **339** 584–7
[11] Jahnke T and Huisinga W 2007 Solving the chemical master equation for monomolecular reaction systems analytically *J. Math. Biol.* **54** 1–26
[12] Cao Y, Gillespie D and Petzold L 2005 Avoiding negative populations in explicit Poisson tau-leaping *J. Chem. Phys.* **123** 054104
[13] Chatterjee A, Vlachos D and Katsoulakis M 2005 Binomial distribution based tau-leap accelerated stochastic simulation *J. Chem. Phys.* **122** 24112
[14] Munsky B 2011 *Modeling cellular variability* (Oxford: Taylor and Francis) pp 233–66
[15] Wolf V, Goel R, Mateescu M and Henzinger T 2010 Solving the chemical master equation using sliding windows *BMC Syst. Biol.* **4** 42
[16] Sunkara V and Hegland M 2010 An optimal finite state projection method *Procedia Comput. Sci.* **1** 1579–86
[17] Sunkara V 2013 Analysis and numerics of the Chemical Master equation *PhD thesis* Australian National University
[18] Sidje R B and Vo H D 2015 Solving the chemical master equation by a fast adaptive finite state projection based on the stochastic simulation algorithm *Math. Biosci.* **269** 10–6
[19] Peles S, Munsky B and Khammash M 2006 Reduction and solution of the chemical master equation using time scale separation and finite state projection *J. Chem. Phys.* **125** 204104
[20] Hegland M, Burden C, Santoso L, MacNamara S and Booth H 2007 A solver for the stochastic master equation applied to gene regulatory networks *J. Comput. Appl. Math.* **205** 708–24
[21] Tapia J J, Faeder J R and Munsky B 2012 Adaptive coarse-graining for transient and quasi-equilibrium analyses of stochastic gene regulation *In Decision and Control (CDC) 2012 IEEE 51st Annual Conference on* pp 5361–6
[22] Sidje R B, Burrage K and MacNamara S 2007 Inexact uniformization method for computing transient distributions of Markov chains *SIAM J. Sci. Comput.* **29** 2562–80
[23] Grassmann W K 1977 Transient solutions in markovian queueing systems *Comput. Oper. Res.* **4** 47–53
[24] Gross D and Miller D R 1982 The randomization technique as a modeling tool and solution procedure for transient Markov processes *Oper. Res.* **32** 343
[25] Hegland M and Garcke J 2011 On the numerical solution of the chemical master equation with sums of rank one tensors *In ANZIAM J.* **52** C628–43
[26] Wolf V 2007 Modelling of Biochemical Reactions by Stochastic Automata Networks, Electron *Notes Theor. Comput. Sci.* **171** 197–208
[27] Kazeev V, Khammash M, Nip M and Schwab C 2014 Direct solution of the Chemical Master equation using quantized tensor trains *PLoS Comput. Biol.* **10** e1003359
[28] Dolgov S and Khoromskij B 2015 Simultaneous state-time approximation of the chemical master equation using tensor product formats *Numer. Linear Algebra Appl.* **22** 197–219
[29] Cichocki A 2014 Era of big data processing: a new approach via tensor networks and tensor decompositions arXiv:1403.2048
[30] Dolgov S and Khoromskij B 2013 Simultaneous state-time approximation of the chemical master equation using tensor product formats arXiv:1311.3143
[31] Kolda T G and Bader B W 2008 Tensor Decompositions and Applications *SIAM Rev.* **51** 455–500
[32] MacNamara S, Burrage K and Sidje R B 2008 Multiscale modeling of chemical kinetics via the Master equation *Multiscale Model. Simul.* **6** 1146–68
[33] Tian T and Burrage K 2006 Stochastic models for regulatory networks of the genetic toggle switch *Proc. Natl Acad. Sci. USA* **103** 8372–7
[34] Sidje R B 1998 Expokit: A software package for computing matrix exponentials *ACM Trans. Math. Softw.* **24** 130–56
[35] Munsky B, Fox Z and Neuert G 2015 Integrating single-molecule experiments and discrete stochastic models to understand heterogeneous gene transcription dynamics *Methods* **85** 12–21
[36] Leenders G B and Tuszynski J A 2013 Stochastic and deterministic models of cellular p53 regulation *Front. Oncol.* **3** 64
[37] Ramaswamy S, Lakerveld R, Barton P I and Stephanopoulos G 2015 Controlled formation of nanostructures with desired geometries: 3. dynamic modeling and simulation of directed self-assembly of nanoparticles through adaptive finite state projection *Ind. Eng. Chem. Res.* **54** 4371–84