Received: 5 June 2017 | Revised: 3 January 2018 | Accepted: 5 January 2018

DOI: 10.1002/nla.2158

RESEARCH ARTICLE

WILEY

On compact vector formats in the solution of the chemical master equation with backward differentiation

Tuğrul Dayar[1] | M. Can Orhan[2]

[1]Department of Computer Engineering, Bilkent University, Ankara, Turkey

[2]Kanava Technologies, Ankara, Turkey

Correspondence
Tuğrul Dayar, Department of Computer Engineering, Bilkent University, TR–06800 Bilkent, Ankara, Turkey.
Email: tugrul@cs.bilkent.edu.tr

Funding information
Alexander von Humboldt Foundation; Scientific and Technological Research Council of Turkey, Grant/Award Number: 2211-A

Summary

A stochastic chemical system with multiple types of molecules interacting through reaction channels can be modeled as a continuous-time Markov chain with a countably infinite multidimensional state space. Starting from an initial probability distribution, the time evolution of the probability distribution associated with this continuous-time Markov chain is described by a system of ordinary differential equations, known as the chemical master equation (CME). This paper shows how one can solve the CME using backward differentiation. In doing this, a novel approach to truncate the state space at each time step using a prediction vector is proposed. The infinitesimal generator matrix associated with the truncated state space is represented compactly, and exactly, using a sum of Kronecker products of matrices associated with molecules. This exact representation is already compact and does not require a low-rank approximation in the hierarchical Tucker decomposition (HTD) format. During transient analysis, compact solution vectors in HTD format are employed with the exact, compact, and truncated generated matrices in Kronecker form, and the linear systems are solved with the Jacobi method using fixed or adaptive rank control strategies on the compact vectors. Results of simulation on benchmark models are compared with those of the proposed solver and another version, which works with compact vectors and highly accurate low-rank approximations of the truncated generator matrices in quantized tensor train format and solves the linear systems with the density matrix renormalization group method. Results indicate that there is a reason to solve the CME numerically, and adaptive rank control strategies on compact vectors in HTD format improve time and memory requirements significantly.

KEYWORDS

backward differentiation, chemical master equation, compact vector, continuous-time Markov chain, Kronecker product

1 | INTRODUCTION

A stochastic chemical system with multiple types of molecules interacting through reaction channels can be modeled as a continuous-time Markov chain (CTMC) with a countably infinite multidimensional state space. In this context, a state is a row vector with elements representing the number of molecules of each type present in the system. A state

transition corresponds to a reaction channel that is expressed by a state change vector and a state-dependent transition rate function. Starting from an initial probability distribution, time evolution of the probability distribution associated with the multidimensional CTMC is described as a system of linear first-order ordinary differential equations (ODEs), known as the chemical master equation (CME; or forward Kolmogorov equation). The goal is to compute the transient probability distribution of the stochastic chemical system at a particular time by solving the CME whose coefficient matrix is the infinitesimal generator of its underlying CTMC. A detailed overview on transient analysis of stochastic chemical systems may be obtained, for instance, from the extensive review by Goutsias et al.[1]

Analytical solution of the CME is possible only in a few simple cases[2–5]; therefore, other approaches are considered. A popular approach is to use the stochastic simulation algorithm (SSA)[6,7] in which sample trajectories are generated by simulating reaction events. SSA is said to be exact in the sense that every reaction event is simulated. Approximation techniques that avoid simulating every reaction event such as tau-leaping methods[8,9] and system-partitioning methods[10–12] are suggested to reduce computational complexity by trading off accuracy. A drawback of simulation techniques is that they require a large number of trajectories to obtain sufficiently accurate estimates and they are inefficient especially for estimating probability distributions.

Another approach is to solve the CME numerically. This is a formidable task because the size of the CME increases exponentially with the number of dimensions even when an upper bound is applied to the number of molecules in the system. Munsky et al.[13] proposed the finite state projection (FSP) algorithm to approximate the actual probability distribution. The FSP algorithm gradually expands the truncated state space by adding states that are reachable within a certain number of transitions and solves the truncated CME until the accuracy of the approximate solution meets a prespecified tolerance. The algorithm uses a number of molecule types, that is, the number of dimensions, in limiting reachability during the truncated state space expansion process. Unfortunately, this strategy yields a truncated state space including many states with negligible probability. Furthermore, the matrix scaling and powering method (see pp. 416–417 in the work of Stewart[14]) employed to solve the truncated CME becomes inefficient as the size of the system increases. Hence, several variants of the FSP algorithm such as multiple time interval FSP,[15] Krylov-FSP,[16,17] optimal FSP,[18] and sliding window[19] methods are proposed to solve the CME more efficiently. These methods differ from each other by the choice of the truncated state space and the technique employed to solve the truncated CME. We refer to the survey by Dinh et al.[20] for an overview of the FSP algorithm and its variants.

Accurate approximation of the probability distribution requires the truncated state space to be large in many cases. Some numerical methods take a hybrid approach and combine the CME solver with other techniques to improve execution time and memory requirement by trading off accuracy.[21,22] Other numerical methods reduce the degrees of freedom, that is, the number of unknowns, in the CME by sparse grids,[23,24] discrete orthogonal polynomials,[25,26] and sparse wavelets.[27,28] In essence, these methods represent the probability distribution as a linear combination of basis functions. Although the degrees of freedom become smaller, these methods still suffer from the curse of dimensionality, that is, memory requirements and computational cost scaling exponentially with the number of dimensions.

In order to overcome the curse of dimensionality, low-rank approximation techniques that represent multidimensional arrays (or tensors) compactly are employed in many application areas (see, for instance, the work of Grasedyck et al.[29]). Canonical polyadic (CP) decomposition[30,31] is a compact representation of a tensor, where the multidimensional array is given as a sum of Kronecker products of smaller vectors or rank-one tensors. Jahnke et al.[32] projected the dynamics of the system onto a manifold of all tensors in CP decomposition format with a predetermined maximal tensor rank. The rank bound is chosen based on the initial distribution and kept constant. However, they advise an adaptive rank control strategy because it is not possible to estimate the rank of a vector in CP decomposition format that approximates the solution of the CME with a desired accuracy. Hegland et al.[33] employed the implicit Euler method to solve the CME and store probability vectors in CP decomposition format. Therein, a bound on the ranks is not chosen, and they suggested to reduce ranks after algebraic tensor operations using various procedures.[34] However, this approach appears to pose problems in practice because there is no robust and efficient algorithm to approximate the CP decomposition with low ranks using any desirable accuracy for tensors having more than two dimensions.

The Tucker decomposition[35] represents a d-dimensional tensor as the multiplication of a d-dimensional core tensor and a basis matrix along each dimension. The higher-order singular value decomposition (HOSVD)[36] provides a simple and nearly optimal solution to approximate a tensor in the Tucker decomposition format with a user-controlled error. However, the memory requirement of the core tensor scales exponentially with the number of dimensions. The tensor train (TT) decomposition[37] is a low-rank approximation technique that avoids this exponential dependence on the number of dimensions, d. The TT decomposition is extended to the quantized TT (QTT) decomposition,[38] which is shown to uncover similarity patterns in data up to the finest resolution level. Assuming that there are m states to represent each

molecule, a quantization size of 2 is used, and m is a power of 2; QTT provides the way for the best possible compression rate for data within the chosen and rank-structured dimensional splitting model in the auxiliary dimension $D = d\log_2 m$. We refer to the paper by Khoromskij[39] for an overview of related tensor numerical methods. A Kronecker-based solver for a multiparticle probabilistic model in the TT and QTT formats has first been applied to the Fokker–Planck equation,[40] which is closely related to the CME.

The truncated generator matrix associated with the CME has been shown to have low TT and QTT ranks,[41,42] and it is possible to represent the solution of the truncated CME compactly in QTT format and reduce memory requirements of solution vectors.[42,43] It seems that a compact vector in QTT format can be multiplied by a matrix only in the same format, and the linear system at each step can be solved using an iterative method such as the density matrix renormalization group (DMRG) method.[44] DMRG is an optimization-based method that is also used in solving linear systems and has been used before to solve the CME with the QTT format.[43] At each DMRG iteration (or sweep), core matrices corresponding to consecutive virtual dimensions are aggregated; each aggregated system is solved directly or using an iterative method such as generalized minimal residual, and the result of the aggregated system is disaggregated. The peak efficiency of the QTT approach is achieved if quantization in the time variable is also implemented,[42] the so-called time–space approximation scheme.

Another low-rank approximation technique that admits user-controlled error is the hierarchical Tucker decomposition (HTD).[45–48] HTD may be considered as a generalization of the TT decomposition in which basis and core matrices are related through a tree structure with logarithmic depth in the number of dimensions that reduces the complexity of the Tucker decomposition by hierarchically splitting the core tensor into core matrices. An advantage of the HTD representation for our purposes is that the multiplication of a compact solution vector in HTD format with a sum of Kronecker products is a well-defined operation[49] and has recently been used in the steady-state analysis of Kronecker-based CTMCs.[50] In the CME setting, such a Kronecker-based Markovian analysis with vectors in HTD format is possible under the separability assumption of transition rates.[7,25,32,43,51,52] To the best of our knowledge, the HTD format has not been employed in the transient analysis of the CME so far.

Backward differentiation formulae (BDF; see pp. 446–448 in the work of Stewart[14]) are a class of implicit multistep methods to solve ODEs. The k-step BDF method, denoted BDFk, keeps the approximations of solutions at k previous time steps and computes the solution at the current time step by solving a linear system. BDFk methods have local truncation error proportional to the kth power of the step size and, therefore, are said to be of order k. They have two major advantages in solving the CME. First, BDFk provides an efficient solver for stiff systems in which the solution contains rapidly decaying transient terms. Stiffness generally manifests itself when reaction rates occur at different time scales, and this is the case for many realistic systems. Second, a prediction of the solution can be given as a linear combination of previous solutions (see p. 150 in the work of Ascher et al.[53]), and local truncation error can be obtained from the difference between prediction and solution at that step.[54] The prediction vector provides a good estimate of the solution, and contrary to the FSP algorithm, it is possible to choose a truncated state space that includes most of the probability mass without having to compute the solution of the truncated CME at that step.

In this paper, we propose a BDFk solver for the CME, which uses a prediction vector to truncate the countably infinite state space at each time step. Then, the truncated generator matrix associated with the CME at each time step is represented compactly, and exactly, using a sum of Kronecker products of matrices associated with molecules.[55] This exact representation, which has been used in the Kronecker-based modeling and analysis of finite multidimensional CTMCs[56] for many years, is already compact and does not require a low-rank approximation for compactness in the HTD format. Following recent results from the steady-state analyses of Kronecker-based CTMCs with compact vectors,[50] however, solution vectors in the HTD format are employed in the proposed BDFk solver with the exact, compact, and truncated generator matrix in Kronecker form during transient analysis. The linear system at each time step of BDFk is solved iteratively by the Jacobi method, which splits the coefficient matrix into diagonal and negated off-diagonal matrices at the outset so that the iteration vector is multiplied by the negated off-diagonal matrix and inverse of the diagonal matrix at each iteration. Computation of the inverse of the diagonal matrix at the outset is convenient. To that end, the diagonal of the coefficient matrix is obtained in HTD format, and the elementwise reciprocal of this compact vector is computed iteratively using the Newton–Schulz method.[47] If an iteration vector is not close to the solution, an accurate computation of the vector should not be necessary. Furthermore, the imposed accuracy may yield additional time and memory requirements. Therefore, we propose adaptive rank control strategies in Newton–Schulz and Jacobi iterations that allow relatively inaccurate computation of iteration vectors.

In summary, at each time step of the proposed BDFk solver, truncation takes place on the state space due to its countable infiniteness, in the solver due its use of a finite number of backward differences, and in the low-rank approximation of solution vectors to control the error in the approximation. The proposed approach that uses compact vectors in HTD

format does not need to compute a low-rank approximation to the truncated generator matrix at each time step of the transient solver. This is expected to yield memory and time savings. Furthermore, the Jacobi iterative method used in solving the linear system requires only a small number of supplementary vectors compared with more sophisticated iterative solvers, and this is expected to result in further savings, especially when only a few iterations of the method end up being executed at each time step.

Although we follow in our research the basic algorithms from the `htucker`[47,48] and `tt-toolbox`[57] Matlab toolboxes, the code is implemented in C[58] within the NSolve package of the Abstract Petri Net Notation toolbox,[59,60] which is used in the analysis of Kronecker-based CTMCs, and calls some LAPACK functions.[61] Results of simulation[62] on benchmark models are compared with those of the proposed solver and another version, which works with compact vectors and highly accurate low-rank approximations of the truncated generator in QTT format and solves the linear system at each time step with the DMRG method. Results indicate that there is a reason to solve the CME numerically, and adaptive rank control strategies on compact solution vectors in HTD format improve time and memory requirements significantly when fixed rank control strategies lead to inefficient ODE solvers.

Throughout the paper, calligraphic uppercase letters are used for sets. Vectors and matrices are represented respectively with boldface lowercase and boldface uppercase letters. All vectors are row vectors except $\mathbf{e}$ and $\mathbf{e}_h$, which represent a column vector of ones and the hth column of the identity matrix, respectively. $\mathbf{1}$ denotes a row vector of ones, and $\mathbf{I}$ denotes the identity matrix. Lengths of vectors and sizes of identity matrices are determined from the contexts in which they are used. $\mathbf{A}(\mathcal{X}, \mathcal{Y})$ is the submatrix of $\mathbf{A}$ where rows and columns are selected respectively from the sets $\mathcal{X}$ and $\mathcal{Y}$, and $\mathbf{A}(x, y)$ is the value of matrix $\mathbf{A}$ corresponding to element (x, y). Consistently, $\mathbf{a}(\mathbf{x})$ is the value of vector $\mathbf{a}$ at state $\mathbf{x}$, and $\mathbf{a}(\mathcal{X})$ is the subvector of $\mathbf{a}$ including the values of $\mathbf{a}(\mathbf{x})$ for $\mathbf{x} \in \mathcal{X}$. diag($\mathbf{a}$) denotes a matrix with the entries of vector $\mathbf{a}$ along its diagonal and 0 elsewhere. The sets of integers, positive integers, and nonnegative integers are denoted by $\mathbb{Z}$, $\mathbb{Z}_{>0}$, $\mathbb{Z}_{\geq 0}$, and the sets of reals, positive reals, and nonnegative reals are denoted by $\mathbb{R}$, $\mathbb{R}_{>0}$, $\mathbb{R}_{\geq 0}$, respectively. Finally, ^{T}, $\times$, $\otimes$, and $\star$ denote respectively the transposition, Cartesian product, Kronecker product, and elementwise multiplication operators.

The next section provides background on CME, BDF, and HTD. The third section describes implementation details of the proposed CME solver. The fourth section discusses results of numerical experiments, and the fifth section concludes the paper.

2 | BACKGROUND

2.1 | Chemical master equation

We consider a CTMC describing the dynamics of a stochastic chemical system with d types of molecules interacting through J reaction channels. Let $\{S(t) = (S_1(t), \ldots, S_d(t)) \in \mathcal{S},\ t \geq 0\}$ be the stochastic process associated with the d-dimensional CTMC, where $\mathcal{S}_h \subseteq \mathbb{Z}_{\geq 0}$ is the set of states that molecules of type h can be in, and $\mathcal{S} = \times_{h=1}^{d} \mathcal{S}_h$ is the product state space of the system, which is potentially infinite. The underlying CTMC can be described by J transition classes each corresponding to a reaction channel as in the next definition.[55]

Definition 1. A pair $(\alpha^{(j)}, \nu^{(j)})$ is said to be transition class $j \in \{1, \ldots, J\}$, where $\alpha^{(j)} : \mathcal{S} \to \mathbb{R}_{>0}$ and $\nu^{(j)} \in \mathbb{Z}^{1\times d}$ are respectively the transition rate function and the state change vector.

We assume the separability of transition rate function α. That is, there exist $\gamma^{(j)} \in \mathbb{R}_{>0}$ and $\alpha_h^{(j)} : \mathcal{S}_h \to \mathbb{R}_{\geq 0}$ such that

$$\alpha^{(j)}(\mathbf{s}) = \gamma^{(j)} \prod_{h=1}^{d} \alpha_h^{(j)}(s_h)$$

for $\mathbf{s} = (s_1, \ldots, s_d) \in \mathcal{S}$ and $j = 1, \ldots, J$. This separability assumption holds for elementary reactions that are also building blocks in more complicated reaction kinetics.[7,25,32,43,51,52]

It is possible to represent the infinitesimal generator matrix underlying the multidimensional CTMC as a sum of Kronecker products of matrices associated with molecules. The following relates molecules and transition classes by defining transition matrices.[55]

Definition 2. The transition rate matrix of molecule $h \in \{1, \ldots, d\}$ for transition class $j \in \{1, \ldots, J\}$, denoted by $\mathbf{S}_h^{(j)} \in \mathbb{R}_{\geq 0}^{|\mathcal{S}_h|\times|\mathcal{S}_h|}$, is given elementwise as

$$\mathbf{S}_h^{(j)}(s_h, s_h') = \begin{cases} \alpha_h^{(j)}(s_h) & \text{if } s_h' = s_h + \nu_h^{(j)} \\ 0 & \text{otherwise} \end{cases} \quad \text{for } s_h, s_h' \in \mathcal{S}_h.$$

The infinitesimal generator matrix includes a diagonal corrector that ensures that row sums are zero; therefore, it is convenient to let $\mathbf{D}_h^{(j)} = \text{diag}(\mathbf{S}_h^{(j)}\mathbf{e}) \in \mathbb{R}_{\geq 0}^{|S_h|\times|S_h|}$ for $h = 1, \dots, d$ and $j = 1, \dots, J$. Then, the infinitesimal generator matrix can be written as

$$\mathbf{Q} = \sum_{j=1}^{J}\mathbf{S}^{(j)} - \sum_{j=1}^{J}\mathbf{D}^{(j)} \quad \text{with} \quad \mathbf{S}^{(j)} = \gamma^{(j)}\bigotimes_{h=1}^{d}\mathbf{S}_h^{(j)} \quad \text{and} \quad \mathbf{D}^{(j)} = \gamma^{(j)}\bigotimes_{h=1}^{d}\mathbf{D}_h^{(j)} \quad \text{for } j = 1, \dots, J.$$

Finally, the CME that describes the time evolution of the stochastic chemical system is given by

$$\frac{\partial \mathbf{p}_t}{\partial t} = \mathbf{p}_t\mathbf{Q} \quad \text{with initial condition} \quad \mathbf{p}_t\big|_{t=0} = \mathbf{p}_0,$$

where $\mathbf{p}_t \in \mathbb{R}_{\geq 0}^{1\times|S|}$ is the probability distribution (row) vector of the system at time $t \geq 0$, satisfying $\mathbf{p}_t\mathbf{e} = 1$.

Having defined the initial value problem to be solved, let us now turn to its solution through BDF.

2.2 | Backward differentiation formulae

BDF are a class of implicit multistep methods that are widely used to cope with stiffness in the solution of ODEs. In this class of solvers, time is discretized and solutions at intermediate time steps are approximated using backward differences (see p. 125 in the work of Ascher et al.[53]). It is not possible to use a BDF solver when the ODE system has a countably infinite number of equations and unknowns; however, the ODE system may be truncated by assuming that the probability of the system being at a state outside the finite truncated state space is zero.

Let us denote by $\mathcal{R}_n$ the truncated state space at time step n such that $\mathcal{R}_n \subset \mathcal{S}$ and $\mathcal{R}_n$ is finite. In order to avoid using two levels of subscripts with $\mathbf{p}_t$ when describing the BDF method, we omit t and the step size, Δt, in the notation, and let the vector $\mathbf{p}_{n-i} \in \mathbb{R}_{\geq 0}^{1\times|\mathcal{R}_{n-i}|}$ hold the probabilities of the system being at states in $\mathcal{R}_{n-i}$ at time $(t_n - i\Delta t)$ for $i = 0, \dots, k$. Then, for a constant step size $\Delta t > 0$, the BDF method of order k, denoted BDFk, at time step n is given by

$$\sum_{i=1}^{k}\frac{1}{i}\nabla^i\mathbf{p}_n = \Delta t\mathbf{p}_n\mathbf{Q}_n \quad \text{with} \quad \mathbf{Q}_n = \mathbf{Q}(\mathcal{R}_n, \mathcal{R}_n),$$

where the backward difference vectors $\nabla^i\mathbf{p}_n \in \mathbb{R}^{1\times|\mathcal{R}_n|}$ are recursively defined as

$$\nabla^0\mathbf{p}_n = \mathbf{p}_n \quad \text{for } n \in \mathbb{Z}_{\geq 0} \quad \text{and} \quad \nabla^i\mathbf{p}_n = \nabla^{i-1}\mathbf{p}_n - \nabla^{i-1}\mathbf{p}_{n-1} \quad \text{for} \quad i = 1, \dots, k \text{ and } n \in \mathbb{Z}_{>0}.$$

Here, $\mathbf{p}_n \in \mathbb{R}_{\geq 0}^{1\times|\mathcal{R}_n|}$ is the solution to be obtained at time step n.

Note that BDFk may be rewritten as a linear system in the form

$$\mathbf{p}_n(\mathbf{I} - \theta_k\Delta t\mathbf{Q}_n) = \sum_{i=0}^{k-1}\beta_{k,i}\nabla^i\mathbf{p}_{n-1}, \tag{1}$$

where θ_k and $\beta_{k,0}, \dots, \beta_{k,k-1}$ are real numbers defined by BDF order k. For instance, $\theta_5 = 60/137$, $\beta_{5,0} = 1$, $\beta_{5,1} = 77/137$, $\beta_{5,2} = 47/137$, $\beta_{5,3} = 27/137$, and $\beta_{5,4} = 12/137$ when $k = 5$.

The leading term of the BDFk local truncation error can be approximated as

$$\mathbf{r}_n \approx \frac{1}{k+1}\left(\mathbf{p}_n - \mathbf{p}_n^{(0)}\right), \tag{2}$$

where

$$\mathbf{p}_n^{(0)} = \sum_{i=0}^{k}\nabla^i\mathbf{p}_{n-1}$$

is the prediction vector.[54]

The probability of the system being in a state changes over time. Hence, at each time step, the truncated state space $\mathcal{R}_n$ needs to be chosen such that the probability of the system being outside $\mathcal{R}_n$ is small. One possible approach is solving (1) and expanding the truncated state space until it includes most of the probability mass as in the FSP algorithm.[13] However, it is possible to take a different and more efficient approach. The prediction vector $\mathbf{p}_n^{(0)} \in \mathbb{R}_{\geq 0}^{1\times|\mathcal{R}_n|}$ is a good estimate of the solution at time $t_n = t_{n-1} + \Delta t$. Hence, the truncated state space $\mathcal{R}_n$ will capture most of the probability mass at time t_n if $\sum_{\mathbf{s}\notin\mathcal{R}_n}\mathbf{p}_n^{(0)}(\mathbf{s})$ is small. This approach allows us to choose the truncated state space before solving (1) because the backward difference vectors $\nabla^0\mathbf{p}_{n-1}, \dots, \nabla^k\mathbf{p}_{n-1}$ are obtained at the previous time step, $n-1$.

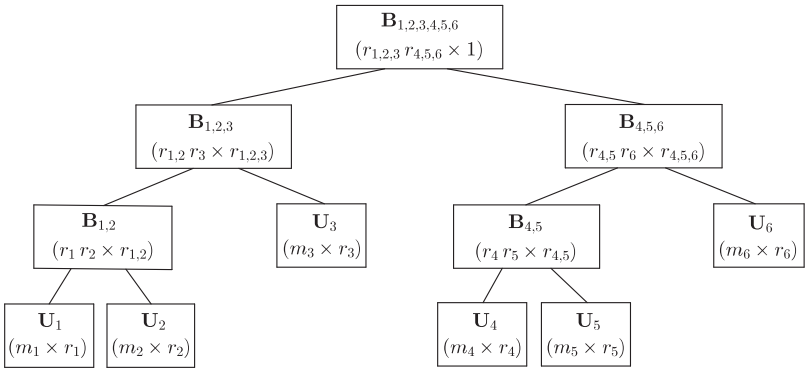

FIGURE 1 Matrices forming $\mathbf{x}$ in hierarchical Tucker decomposition format for $d = 6$

Backward differences are also very suitable for adjusting the step size.[54] Letting $\mathbf{P}$ and $\mathbf{P}^*$ denote respectively the matrices with backward differences for old step size Δt and new step size $\Delta t^* \neq \Delta t$ as in

$$\mathbf{P} = \left[\left(\nabla^1 \mathbf{p}_{n-1}\right)^T, \ldots, \left(\nabla^k \mathbf{p}_{n-1}\right)^T\right] \quad \text{and} \quad \mathbf{P}^* = \left[\left(\nabla^{*1} \mathbf{p}_{n-1}\right)^T, \ldots, \left(\nabla^{*k} \mathbf{p}_{n-1}\right)^T\right],$$

the new backward difference vectors may be obtained from $\mathbf{P}^* = \mathbf{P}\bar{\mathbf{R}}^*\bar{\mathbf{U}}^*$, where $\bar{\mathbf{R}}^*$ and $\bar{\mathbf{U}}^*$ are $(k \times k)$ matrices given elementwise as

$$\bar{\mathbf{R}}^*(i,i') = \frac{1}{i!}\prod_{l=0}^{i-1}\left(l - i'(\Delta t^*/\Delta t)\right) \quad \text{and} \quad \bar{\mathbf{U}}^*(i,i') = \frac{1}{i!}\prod_{l=0}^{i-1}(l - i') \quad \text{for } i, i' = 1, \ldots, k.$$

The next section provides a short description of HTD and operations that are used in the implementation of BDFk.

2.3 | Hierarchical Tucker decomposition

A column vector $\mathbf{x}$ of length $\prod_{h=1}^{d} m_h$ in Tucker decomposition format[35] can be expressed as

$$\mathbf{x} = (\mathbf{U}_1 \otimes \cdots \otimes \mathbf{U}_d)\mathbf{c},$$

where $\mathbf{U}_h$ is an $(m_h \times r_h)$ orthogonal basis matrix for dimension h with $h = 1, \ldots, d$, and $\mathbf{c}$ is a vector of length $\prod_{h=1}^{d} r_h$. Memory requirement for this representation scales exponentially with d.

HTD[45–48] employs a hierarchical splitting of dimensions $1, \ldots, d$ to represent vector $\mathbf{c}$ using smaller matrices.

Definition 3. Let $\mathcal{T}$ be a binary tree with each node represented by a subset of $\{1, \ldots, d\}$. Then, $\mathcal{T}$ is said to be a dimension tree if the root node is $\{1, \ldots, d\}$, each parent node is a disjoint union of its children, and there are d leaf nodes with each leaf node represented by a different singleton from $\{1, \ldots, d\}$.

Given orthogonal matrices $\mathbf{U}_{\bar{t}}$, $\mathbf{U}_{\bar{t}_l}$, and $\mathbf{U}_{\bar{t}_r}$, there exists a $(r_{\bar{t}_l} r_{\bar{t}_r} \times r_{\bar{t}})$ transfer matrix $\mathbf{B}_{\bar{t}}$ such that

$$\mathbf{U}_{\bar{t}} = (\mathbf{U}_{\bar{t}_l} \otimes \mathbf{U}_{\bar{t}_r})\mathbf{B}_{\bar{t}}, \tag{3}$$

where $\bar{t}_l$ and $\bar{t}_r$ are the left and right children of nonleaf node $\bar{t}$ (see lemma 2.1 in the work of Kressner et al.[48]), respectively. Then, by recursively substituting in (3), vector $\mathbf{x} = \mathbf{U}_{1,\ldots,d}$ can be represented by d orthogonal basis matrices $\mathbf{U}_1, \ldots, \mathbf{U}_d$ and $(d-1)$ transfer matrices $\mathbf{B}_{\bar{t}}$ for $\bar{t} \in \mathcal{N}(\mathcal{T})$, where $\mathcal{N}(\mathcal{T})$ is the set of nonleaf nodes of binary tree $\mathcal{T}$.

A possible decomposition for vector $\mathbf{x}$ with $d = 6$ can be written as

$$\mathbf{x} = (\mathbf{U}_1 \otimes \mathbf{U}_2 \otimes \mathbf{U}_3 \otimes \mathbf{U}_4 \otimes \mathbf{U}_5 \otimes \mathbf{U}_6)\mathbf{c} \quad \text{with} \quad \mathbf{c} = (\mathbf{B}_{1,2} \otimes \mathbf{I}_{r_3} \otimes \mathbf{B}_{4,5} \otimes \mathbf{I}_{r_6})(\mathbf{B}_{1,2,3} \otimes \mathbf{B}_{4,5,6})\mathbf{B}_{1,2,3,4,5,6}$$

for the corresponding dimension tree in Figure 1.

Memory requirement for matrices in the dimension tree associated with the HTD format is bounded by the following[48]:

$$(dm_{\max}r_{\max} + (d-2)r_{\max}^3 + r_{\max}^2) \quad \text{with} \quad r_{\max} = \max_{\bar{t}}(r_{\bar{t}}) \quad \text{and} \quad m_{\max} = \max(m_1, \ldots, m_d).$$

A matricization of dimensions corresponding to $\bar{t}$ may be obtained by taking $\bar{t}$ as the row index and $\bar{s} = \{1, \ldots, d\}\backslash\{\bar{t}\}$ as the column index of the d-dimensional vector $\mathbf{x}$. The orthogonal matrix $\mathbf{U}_{\bar{t}}$ has in its columns the singular vectors associated with the largest $r_{\bar{t}}$ singular values (see pp. 76–79 in the work of Golub et al.[63]) of the matricization of dimensions

corresponding to $\bar{t}$. Hence, we have the concepts of "hierarchy of matricizations" and "HOSVD", and $r_{\bar{t}}$ is the rank of the truncated HOSVD.[36,47,64]

Now, we provide short explanations for operations that are used in the implementation of BDFk in HTD format.

2.3.1 | Representing a Kronecker product of vectors in HTD format

Let $\mathbf{x} = \otimes_{h=1}^{d} \mathbf{x}_h$ be a vector, where $\mathbf{x}_h$ is a vector of length m_h. Then, $\mathbf{x}$ may be represented in HTD format with basis matrices $\mathbf{U}_h = \mathbf{x}_h$ for $h = 1, \dots, d$ and transfer matrices $\mathbf{B}_{\bar{t}} = 1$.

2.3.2 | Aggregation of dimensions of a vector in HTD format

Let $\mathbf{x}$ be a vector in HTD format with basis matrices $\mathbf{U}_{h'}$ for $h' = 1, \dots, d$ and transfer matrices $\mathbf{B}_{\bar{t}}$ forming $\mathbf{c}$. Then, the vector obtained by aggregating all dimensions except dimension h of vector $\mathbf{x}$ can be written as

$$\mathbf{g}_h^{(\mathbf{x})} = \left(\bigotimes_{h'=1}^{h-1} \mathbf{1}_{m_{h'}} \otimes \mathbf{I}_{m_h} \otimes \bigotimes_{h'=h+1}^{d} \mathbf{1}_{m_{h'}} \right) \mathbf{x} = \left(\bigotimes_{h'=1}^{h-1} \mathbf{1}_{m_{h'}} \mathbf{U}_{h'} \otimes \mathbf{U}_h \otimes \bigotimes_{h'=h+1}^{d} \mathbf{1}_{m_{h'}} \mathbf{U}_{h'} \right) \mathbf{c}.$$

The vector $\mathbf{g}_h^{(\mathbf{x})} \in \mathbb{R}^{|S_h| \times 1}$ is in HTD format with basis matrices

$$\tilde{\mathbf{U}}_{h'} = \begin{cases} \mathbf{U}_h & \text{if } h' = h \\ \mathbf{1}_{m_{h'}} \mathbf{U}_{h'} & \text{otherwise} \end{cases}$$

and transfer matrices $\mathbf{B}_{\bar{t}}$ forming $\mathbf{c}$. The flat representation of $\mathbf{g}_h^{(\mathbf{x})}$ may be obtained by computing the basis matrices at the leaves and by moving toward the root in $O(r_{\max}(dm_{\max} + r_{\max}^2 + m_{\max} r_{\max}))$ floating-point operations (flops).

2.3.3 | Multiplication of a vector in HTD format with a sum of Kronecker products

Let $\mathbf{A} = \sum_{j=1}^{J} \otimes_{h=1}^{d} \mathbf{A}_h^{(j)}$ and $\mathbf{x}$ be a vector in HTD format with orthogonal basis matrices $\mathbf{U}_h$ and transfer matrices $\mathbf{B}_{\bar{t}}$ forming $\mathbf{c}$. Then, the vector $\mathbf{y} = \mathbf{A}\mathbf{x}$ may be written as

$$\mathbf{y} = \sum_{j=1}^{J} \mathbf{y}^{(j)} \quad \text{with} \quad \mathbf{y}^{(j)} = \left(\bigotimes_{h=1}^{d} \mathbf{A}_h^{(j)} \mathbf{U}_h \right) \mathbf{c}.$$

Hence, the vector $\mathbf{y}^{(j)}$ is in HTD format with basis matrices $\mathbf{A}_h^{(j)} \mathbf{U}_h$ and transfer matrices $\mathbf{B}_{\bar{t}}$ forming $\mathbf{c}$.

2.3.4 | Addition of vectors in HTD format

The addition of vectors in HTD format is studied before.[47] Therein, three algorithms are presented, and the best approach seems to be adding and truncating each vector one by one (see algorithm 7 on p. 23 in the work of Kressner et al.[47]). The significance of algorithm 7 is that it is possible to impose an accuracy of tol on the truncated vector by choosing the rank $r_{\bar{t}}$ in node $\bar{t}$ based on dropping the smallest singular values whose squared sum is less than or equal to $tol^2/(2d-3)$ (see pp. 18–19 in the work of Kressner et al.[47]) when adding two vectors. The number of flops for adding J vectors with this algorithm becomes $O(dJ^2 r_{\max}^2 (m_{\max} + r_{\max}^2 + J r_{\max}))$.

2.3.5 | Computing the 2-norm of a vector in HTD format

The computation of the maximum value in magnitude of the elements of a vector in HTD format seems to be costly. Therefore, instead of the maximum (i.e., infinity) norm of vector $\mathbf{x}$ in HTD format, we consider the computation of the 2-norm of vector $\mathbf{x}$ given by $\|\mathbf{x}\|_2 = \sqrt{\mathbf{x}^T \mathbf{x}}$. The scalar $\|\mathbf{x}\|_2$ can be obtained by computing the inner products of two compact vectors in HTD format (see algorithm 2 on p. 14 in the work of Kressner et al.[47]). The number of flops for this algorithm is $O(r_{\max}^2 (d r_{\max}^2 + m_{\max}))$.

2.3.6 | Computing the elementwise multiplication of two vectors in HTD format

The elementwise multiplication of two matrices $\mathbf{X}$ and $\mathbf{Y}$ with given SVDs (see pp. 76–79 in the work of Golub et al.[63])

$$\mathbf{X} = \mathbf{U}_\mathbf{X} \Sigma_\mathbf{X} \mathbf{V}_\mathbf{X}^T \quad \text{and} \quad \mathbf{Y} = \mathbf{U}_\mathbf{Y} \Sigma_\mathbf{Y} \mathbf{V}_\mathbf{Y}^T \quad \text{results in} \quad \mathbf{X} \star \mathbf{Y} := \left(\mathbf{U}_\mathbf{X} \odot^T \mathbf{U}_\mathbf{Y} \right) (\Sigma_\mathbf{X} \otimes \Sigma_\mathbf{Y}) \left(\mathbf{V}_\mathbf{X} \odot^T \mathbf{V}_\mathbf{Y} \right)^T,$$

where $\odot^T$ denotes a transposed variant of the Khatri–Rao product.[34] More specifically, the ith row of the $(m \times r_{\mathbf{A}} r_{\mathbf{B}})$ matrix $\mathbf{A} \odot^T \mathbf{B}$ is the Kronecker product of the the ith rows of the $(m \times r_{\mathbf{A}})$ matrix $\mathbf{A}$ and the $(m \times r_{\mathbf{B}})$ matrix $\mathbf{B}$. This implies that the elementwise multiplication $\mathbf{X} \star \mathbf{Y}$ has rank equal to the multiplication of the ranks of the matrices $\mathbf{X}$ and $\mathbf{Y}$.

The situation for elementwise multiplication of two vectors $\mathbf{x}$ and $\mathbf{y}$ in HTD format is no different if SVD is replaced with HOSVD (see pp. 24–25 in the work of Kressner et al.[47]). As for the multiplication of two vectors in HTD format, the vector $\mathbf{x} \star \mathbf{y}$ may be truncated directly while computing the basis and transfer matrices. During truncation, it is possible to impose an accuracy of tol on the truncated HOSVD by choosing the rank $r_{\bar{t}}$ in node $\bar{t}$ based on dropping the smallest singular values whose squared sum is less than or equal to $tol^2/(2d-3)$, as it is done in the addition of two vectors.

2.3.7 | Computing the elementwise reciprocal of a vector in HTD format

In the Jacobi method, the iteration vector needs to be multiplied by the reciprocal of diagonal elements of the coefficient matrix. Because it is costly to generate each element of a vector in HTD format, an iterative method such as Newton–Schulz method (see p. 28 in the work of Kressner et al.[47]) can be employed to compute the elementwise reciprocal of a vector in HTD format as in Algorithm 1. In this method, the elementwise reciprocal vector $\mathbf{xinv}$ is initialized to a vector with each element having the reciprocal of the 2-norm of the input vector $\mathbf{x}$. The algorithm stops if the relative 2-norm of $(\mathbf{e} - \mathbf{x} \star \mathbf{xinv}^{(it-1)})$ is smaller than the tolerance rcp_tol as in `elem_reciprocal.m` of `htucker_toolbox`[47] or if the change in the norm is larger than the tolerance rcp_chg_tol and rcp_max_it iterations has already been performed.

Now, we can move to implementation issues regarding HTD-based BDFk transient solvers for CTMCs with infinite multidimensional state spaces.

Algorithm 1 Elementwise reciprocal computation of vector $\mathbf{x}$ in HTD format

Input: Input vector in HTD format: $\mathbf{x}$
Output: Elementwise reciprocal of input vector in HTD format: $\mathbf{xinv}$

function ELEMENTWISERECIPROCAL($\mathbf{x}$)
  Set each element of $\mathbf{xinv}$ to $1/\|\mathbf{x}\|_2$;
  $it \leftarrow 0$; $nrm\_chg \leftarrow 0$; $rcp\_stop \leftarrow FALSE$;
  do
    $it \leftarrow it + 1$; $\mathbf{xinv}^{(it-1)} \leftarrow \mathbf{xinv}$;
    $\mathbf{d} \leftarrow \mathbf{e} - \mathbf{x} \star \mathbf{xinv}^{(it-1)}$; $nrm^{(it)} \leftarrow \|\mathbf{d}\|_2/\|\mathbf{e}\|_2$;
    $\mathbf{xinv} \leftarrow \mathbf{xinv}^{(it-1)} + \mathbf{xinv}^{(it-1)} \star \mathbf{d}$;
    if $it > 1$ **then**
      $nrm\_chg \leftarrow nrm^{(it)}/nrm^{(it-1)}$;
    end if
    if $nrm\_chg > rcp\_chg\_tol$ AND $it \geq rcp\_max\_it$ **then**
      $rcp\_stop \leftarrow TRUE$;
    end if
  while $nrm^{(it)} \geq rcp\_tol$ AND $rcp\_stop = FALSE$
  return $\mathbf{xinv}$;
end function

3 | IMPLEMENTATION ISSUES

The BDFk method is employed to compute the probability distribution at time t_f starting from initial state $\mathbf{s}_0 \in S$ at time 0. The time interval $[0, t_f]$ is discretized, and at each time step, a truncated state space is chosen, the linear system in (1) is solved, and the BDFk local truncation error in (2) is computed. If this error is not within the prespecified error tolerances of err_tol_1 and err_tol_2, the above sequence of operations is repeated for a new step size. Otherwise, the backward difference vectors for the next step are computed and the step ends. The solutions at the first k time steps 1 through k to start BDFk at time step $k+1$ are obtained using the embedded Runge–Kutta method due to Fehlberg, written RKF$k-1(k)$, which is an order k method without the error estimate.[14,53] The following subsections elaborate on the details of Algorithm 2.

3.1 | Choosing the truncated state space

Storing the infinitesimal generator matrix in Kronecker form requires the truncated state space to be represented as a union of Cartesian products of subsets of subsystem state spaces.[49,56] Two algorithms for partitioning an arbitrary

multidimensional state space into Cartesian products have been proposed.[65] However, these two algorithms are relatively time consuming and do not seem to be suitable for our purposes in this context. Efficiently partitioning an arbitrary truncated state space at each time step requires further research and is out of the scope of this paper. Here, we take a naïve approach and let the truncated state space at time step n be a Cartesian product of consecutive integers as in sliding windows,[19] that is, we let

$$\mathcal{R}_n = \times_{h=1}^{d} \mathcal{R}_{n,h},$$

where $\mathcal{R}_{n,h}$ is a set of consecutive integers representing the set of states along dimension h at time step n such that $\mathcal{R}_{n,h} \subset \mathcal{S}_h$ and $\mathcal{R}_{n,h}$ is finite.

Algorithm 2 BDFk method at step n

Input: Truncated state space: $\mathcal{R}_{n-1}$
Backward difference vectors: $\nabla^0\mathbf{p}_{n-1}, \dots, \nabla^k\mathbf{p}_{n-1}$
Output: Truncated state space: $\mathcal{R}_n$
Backward difference vectors: $\nabla^0\mathbf{p}_n, \dots, \nabla^k\mathbf{p}_n$

function BDF_step ($\mathcal{R}_{n-1}, \nabla^0\mathbf{p}_{n-1}, \dots, \nabla^k\mathbf{p}_{n-1}$)
  accept_sol ← *FALSE*
  while *accept_sol* = *FALSE* **do**
    $\mathbf{p}_n^{(0)} \leftarrow \sum_{i=0}^{k} \nabla^i\mathbf{p}_{n-1}$; Compute truncated state space $\mathcal{R}_n$ from $\mathbf{p}_n^{(0)}$;
    Obtain $\mathbf{p}_n^{(0)}, \nabla^0\mathbf{p}_{n-1}, \dots, \nabla^k\mathbf{p}_{n-1}$ for the new truncated state space $\mathcal{R}_n$;
    $\mathbf{minv}_n \leftarrow$ ELEMENTWISERECIPROCAL$(\mathbf{e} + \theta_k\Delta t \sum_{j=1}^{J} \gamma^{(j)} \bigotimes_{h=1}^{d} \mathbf{D}_h^{(j)}(\mathcal{R}_{n,h}, \mathcal{R}_{n,h})\mathbf{e})$;
    $\mathbf{N}_n \leftarrow \theta_k\Delta t \sum_{j=1}^{J} \gamma^{(j)} \bigotimes_{h=1}^{d} \mathbf{S}_h^{(j)}(\mathcal{R}_{n,h}, \mathcal{R}_{n,h})$; $\mathbf{p}_n^{(rhs)} \leftarrow \sum_{i=0}^{k-1} \beta_{k,i}\nabla^i\mathbf{p}_{n-1}$;
    $it \leftarrow 0$; $nrm\_chg \leftarrow 0$; *jac_stop* ← *FALSE*;
    while *jac_stop* = *FALSE* **do**
      $it \leftarrow it + 1$; $\mathbf{p}_n^{(it)} \leftarrow (\mathbf{p}_n^{(rhs)} + \mathbf{p}_n^{(it-1)}\mathbf{N}_n) \star \mathbf{minv}_n^T$;
      $nrm^{(it)} \leftarrow \|\mathbf{p}_n^{(rhs)} - \mathbf{p}_n^{(it)}(\mathbf{I} - \theta_k\Delta t\mathbf{Q}_n)\|_2$;
      if $it > 1$ **then**
        $nrm\_chg \leftarrow nrm^{(it)}/nrm^{(it-1)}$;
      end if
      if $it \geq jac\_max\_it$ OR $nrm^{(it)} < jac\_tol$ OR $nrm\_chg > jac\_chg\_tol$ **then**
        jac_stop ← *TRUE*;
      end if
    end while
    $\mathbf{p}_n \leftarrow \mathbf{p}_n^{(it)}$; $lte_n \leftarrow \frac{\|\mathbf{p}_n - \mathbf{p}_n^{(0)}\|_2}{k+1}$; $\rho \leftarrow \left(\frac{err\_tol_1 + err\_tol_2}{2\ lte_n}\right)^{1/(k+1)}$;
    if $err\_tol_1 < lte_n$ AND $lte_n < err\_tol_2$ **then**
      accept_sol ← *TRUE*;
    else
      $\Delta t^* \leftarrow$Max$(\rho\ \Delta t, \Delta t_{min})$;
      if $\Delta t \neq \Delta t^*$ **then**
        $\Delta t \leftarrow \Delta t^*$; $\mathcal{R}_{n-1} \leftarrow \mathcal{R}_n$;
        Recompute $\nabla^0\mathbf{p}_{n-1}, \dots, \nabla^k\mathbf{p}_{n-1}$ for the new step size Δt;
      end if
    end if
  end while
  Compute $\nabla^0\mathbf{p}_n, \dots, \nabla^k\mathbf{p}_n$ from $\nabla^0\mathbf{p}_{n-1}, \dots, \nabla^k\mathbf{p}_{n-1}$ and $\mathbf{p}_n$;
  return $(\mathcal{R}_n, \nabla^0\mathbf{p}_n, \dots, \nabla^k\mathbf{p}_n)$;
end function

The truncated state space $\mathcal{R}_n$ at time step n needs to be chosen such that it includes most of the probability mass. Here, we use the prediction vector to estimate the probability distribution without solving the linear system in BDFk and obtaining the probability distribution as discussed in Section 2.2. To that end, we first compute the flat vectors $\mathbf{g}_1^{((\mathbf{p}_n^{(0)})^T)}, \dots, \mathbf{g}_d^{((\mathbf{p}_n^{(0)})^T)}$ that are defined in Section 2.3.2. Assuming that multidimensional states are mapped to nonnegative integers lexicographically, we choose $\mathcal{R}_{n,h}$ such that

$$\mathcal{R}_{n,h} = \left\{\min(\tilde{\mathcal{R}}_{n,h}) - \omega|\tilde{\mathcal{R}}_{n,h}|,\ \dots, \max(\tilde{\mathcal{R}}_{n,h}) + \omega|\tilde{\mathcal{R}}_{n,h}|\right\} \cap \mathcal{S}_h,$$

where *state_trunc_tol* is a prespecified state truncation tolerance, $\tilde{\mathcal{R}}_{n,h}$ is a minimal set that satisfies

$$\sum_{s_h \notin \tilde{\mathcal{R}}_{n,h}} \mathbf{g}_h^{\left(\left(\mathbf{p}_n^{(0)}\right)^T\right)}(s_h) < \frac{state\_trunc\_tol}{d},$$

and $\omega \in (0,1)$ is a safety factor. The probability mass that remains outside $\mathcal{R}_n$ due to truncation along each dimension is bounded by *state_trunc_tol*/d. Hence, the probability of the system being outside $\mathcal{R}_n$ is smaller than *state_trunc_tol*, that is, $\mathcal{R}_n$ satisfies

$$\sum_{\mathbf{s}\notin\mathcal{R}_n} \mathbf{p}_n^{(0)}(\mathbf{s}) < state\_trunc\_tol.$$

We remark that in this way, it is possible to express $\mathbf{Q}_n$ in Kronecker form using the smaller molecule matrices $\mathbf{S}_h^{(j)}(\mathcal{R}_{n,h},\mathcal{R}_{n,h}) \in \mathbb{R}_{\geq 0}^{|\mathcal{R}_{n,h}|\times|\mathcal{R}_{n,h}|}$ and $\mathbf{D}_h^{(j)}(\mathcal{R}_{n,h},\mathcal{R}_{n,h}) \in \mathbb{R}_{\geq 0}^{|\mathcal{R}_{n,h}|\times|\mathcal{R}_{n,h}|}$ as in

$$\mathbf{Q}_n = \sum_{j=1}^{J} \gamma^{(j)} \bigotimes_{h=1}^{d} \mathbf{S}_h^{(j)}(\mathcal{R}_{n,h},\mathcal{R}_{n,h}) - \sum_{j=1}^{J} \gamma^{(j)} \bigotimes_{h=1}^{d} \mathbf{D}_h^{(j)}(\mathcal{R}_{n,h},\mathcal{R}_{n,h}).$$

Now, observe that the matrices $\mathbf{S}_h^{(j)}(\mathcal{R}_{n,h},\mathcal{R}_{n,h})$ and $\mathbf{D}_h^{(j)}(\mathcal{R}_{n,h},\mathcal{R}_{n,h})$ that contribute to $\mathbf{Q}_n$ are not only finite but also sparse, and those associated with molecules that do not change their states in transition class j are identity.[55,56]

The elements of the vectors obtained at time step $n-1$ (i.e., backward difference vectors $\nabla^0\mathbf{p}_{n-1}, \ldots, \nabla^k\mathbf{p}_{n-1}$ and prediction vector $\mathbf{p}_n^{(0)}$) correspond to states in the truncated state space $\mathcal{R}_{n-1}$. These vectors need to be updated so that they become incident to $\mathcal{R}_n$ (see Line 5 of Algorithm 2). The value of an element in the updated vectors will be zero if it corresponds to a state in $\mathcal{R}_n \backslash \mathcal{R}_{n-1}$. The other elements of the updated vectors have the same values as those of the corresponding elements of the respective vectors obtained at time step $n-1$. The rows of new basis and transfer matrices are either zero or can be obtained by copying the corresponding rows from the old basis and transfer matrices. In conclusion, the updating of vectors requires no flops.

3.2 | Solving the linear system in BDF*k*

We employ the classical point iterative Jacobi method for solving the linear system in BDF*k* after the truncated state space $\mathcal{R}_n$ is chosen and backward difference vectors and the prediction vector are updated. The Jacobi method can be written as

$$\mathbf{p}_n^{(it)} := \left(\mathbf{p}_n^{(rhs)} + \mathbf{p}_n^{(it-1)}\mathbf{N}_n\right) \star \mathbf{minv}_n^T \quad \text{for } it = 1, 2, \ldots,$$

starting with the prediction vector $\mathbf{p}_n^{(0)}$, where

$$\mathbf{p}_n^{(rhs)} = \sum_{i=0}^{k-1} \beta_{k,i} \nabla^i \mathbf{p}_{n-1}, \quad \mathbf{N}_n = \theta_k \Delta t \sum_{j=1}^{J} \gamma^{(j)} \bigotimes_{h=1}^{d} \mathbf{S}_h^{(j)}(\mathcal{R}_{n,h},\mathcal{R}_{n,h}),$$

and $\mathbf{minv}_n \in \mathbb{R}_{>0}^{|\mathcal{R}_n|\times 1}$ is the elementwise reciprocal of

$$\mathbf{m}_n = \mathbf{e} + \theta_k \Delta t \sum_{j=1}^{J} \gamma^{(j)} \bigotimes_{h=1}^{d} \mathbf{D}_h^{(j)}(\mathcal{R}_{n,h},\mathcal{R}_{n,h})\mathbf{e}.$$

At the outset, $\mathbf{minv}_n$ is computed by the Newton–Schulz method (see Line 6 of Algorithm 2). Then, the iteration vector $\mathbf{p}_n^{(it)}$ and the 2-norm of the residual, $nrm^{(it)}$, are computed at each Jacobi iteration, it. Jacobi stops if the number of iterations becomes large, the iteration vector converges, or stagnation is detected (see Line 16 of Algorithm 2).

3.3 | Adjusting the step size

The local truncation error of BDF*k* at each step can be controlled by adjusting the step size. To that end, from (2), we compute the local truncation error in 2-norm using

$$lte_n = \frac{\left\|\mathbf{p}_n - \mathbf{p}_n^{(0)}\right\|_2}{k+1}$$

after the Jacobi method provides the solution $\mathbf{p}_n$ at time step n. If lte_n turns out to be between the error tolerances err_tol_1 and err_tol_2, then the solution is accepted. Otherwise, a candidate step size is chosen as $\Delta t^* = \max(\rho\ \Delta t, \Delta t_{\min})$, where $\Delta t_{\min}$ is the minimum step size allowed, and

$$\rho = \left(\frac{err\_tol_1 + err\_tol_2}{2 \quad lte_n}\right)^{1/(k+1)}$$

is the optimal step size coefficient for the average of err_tol_1 and err_tol_2. If the candidate step size is different than the current step size, then the step size will be updated and backward difference vectors for the new step size will be computed as discussed in Section 2.2 (see Line 26 of Algorithm 2). We remark that the step size gets adjusted to compute the solution at time t_f when the last probability distribution vector is obtained for time t_n that is sufficiently close to the final time t_f.

3.4 | Rank control strategies

An accuracy of *tol* can be imposed on the truncation of a vector in HTD format by choosing the truncation error tolerance as $tol/\sqrt{2d-3}$.[47,48] In the original methods of `htucker`, a fixed rank control strategy is considered. Therein, the ranks are determined by a truncation error tolerance and a fixed rank bound. The fixed rank bound is used to avoid very large ranks that cause the methods to take an unacceptably long time. The bound needs to be chosen relatively large, for example, 1,000 in `htucker`, so that the ranks are determined by the truncation error tolerance in most cases. In the fixed rank control strategy, iteration vectors are computed accurately even if they are not close to the solution during early iteration steps. However, the ranks of iteration vectors can become unnecessarily large in some cases. In order to avoid this situation, we propose to use adaptive rank control strategies for the Newton–Schulz and Jacobi methods by introducing adaptive rank bounds.

Adaptive rank control strategies need to be devised in such a way that the rank bound is increased only when it is necessary. Based on the observation that the iterative method does not converge or that it converges relatively slowly in the case of a small rank bound, we increase the rank bound if the ratio of two consecutive residual norms, that is, *nrm_chg* in Algorithms 1 and 2, is larger than a prespecified tolerance. In the Newton–Schulz method, the rank bound is initialized to 1 and increased if $nrm\_chg > rcp\_chg\_tol$. In the Jacobi method, the rank bound is initialized to the maximum rank of the prediction vector $\mathbf{p}_n^{(0)}$ and increased if $nrm\_chg > jac\_chg\_tol$. In the latter case, the stopping criterion is modified so that Jacobi iterations assume stagnation if $nrm\_chg > jac\_chg\_tol$ at two consecutive iterations.

The truncation of a vector may be inaccurate if the rank is determined by a bound instead of a truncation error tolerance. Hence, errors may be underestimated if the backward difference vectors, the prediction vector, the BDF*k* local truncation error, the right-hand side vector of BDF*k*, and the residual vectors are not computed accurately. Therefore, we truncate these vectors only using truncation error tolerances. More precisely, fixed and adaptive rank control strategies are only considered while computing the elementwise multiplication $\mathbf{x} \star \mathbf{xinv}^{(it-1)}$ (see Line 6 of Algorithm 1), the vector **xinv** (see Line 7 of Algorithm 1), and the iteration vector $\mathbf{p}_n^{(it)}$ (see Line 10 of Algorithm 2).

Regarding error tolerances, the same error tolerance is used while truncating a vector in both the fixed and the adaptive rank control strategies. We have chosen $rcp\_tol/\sqrt{2d-3}$ and $jac\_tol/\sqrt{2d-3}$ as truncation error tolerances in the Newton–Schulz and Jacobi methods, respectively. Other vectors are truncated with a truncation error tolerance of $tol/\sqrt{2d-3}$, which imposes an accuracy of *tol* on the HTD format.

4 | NUMERICAL RESULTS

Although we follow in our research the basic algorithms from the `htucker`[47,48] and `tt-toolbox`[57] Matlab toolboxes, the code is implemented in C[58] within the NSolve package of the Abstract Petri Net Notation toolbox[59,60] and calls some LAPACK functions.[61] In this study, we consider a BDF*k* solver of order 5, that is, $k = 5$, which is also the default order choice in MATLAB's `ode15s`.[54] The first five solutions for BDF5 are obtained by using the RKF4(5) method of order 5 as discussed at the beginning of Section 3.

4.1 | Experimental framework

The numerical experiments are performed on an Intel Core i7 2.6 GHz processor with 16 GB of main memory under Linux; however, only a single processor is used. A time limit of 1,000 s is imposed, and execution will stop at the end of a BDF5 step if execution time exceeds this limit.

Parameters are chosen either as constants or as functions of the accuracy tolerance, *tol*. Because backward difference vectors may include an error of *tol*, we allow for the same error in BDF5 and state space truncation by letting

$$err\_tol_1 = tol/5, \quad err\_tol_2 = tol, \quad \text{and} \quad state\_trunc\_tol = tol$$

(see the subsections on adjusting the step size and choosing the truncated state space in Section 3). The minimum step size, $\Delta t_{\min}$, is set to `1e-6` based on the assumption that it is sufficiently small. The initial step size is set to `1e-5`, which happens to be the value of the step size used in the first five time steps with RKF4(5). The safety factor ω associated with choosing the truncated state space is set to $1/20$ based on the observation that it is sufficiently small and still yields accurate results. The parameters in the Newton–Schulz and Jacobi methods are chosen as

$$rcp\_tol = jac\_tol = tol/2 \quad \text{and} \quad rcp\_chg\_tol = jac\_chg\_tol = 1/2,$$

so that these methods are efficient yet still relatively accurate (see the subsection on computing the elementwise reciprocal of a vector in HTD format in Section 2 and the subsection on rank control strategies in Section 3).

Numerical experiments are devised to investigate the effects of using the proposed adaptive rank control strategies. We consider three different HTD-based solvers. HTD_F implements fixed rank control strategies in Newton–Schulz and Jacobi methods. HTD_M implements a mixed policy in which adaptive and fixed rank control strategies are used in Newton–Schulz and Jacobi methods, respectively. HTD_A implements adaptive rank control strategies in both iterative methods. In fixed rank control strategies, the fixed rank bound is set to 1,000 as in `htucker`.

Besides the HTD format, we have also implemented the BDFk solver[58] with vectors in QTT and transposed QTT (QT3) formats[43] with a quantization size of 2 for comparison purposes. Note that the QTT format quantizes each dimension h into l_h subdimensions (or levels) for $h = 1, \dots, d$. Now, let a state in the quantized representation be denoted by $s_{\text{dimension,subdimension}}$. Then, the quantized state vector for QTT is given by $(s_{1,1}, \dots, s_{1,l_1}, s_{2,1}, \dots, s_{2,l_2}, \dots, s_{d,1}, \dots, s_{d,l_d})$. As can be seen, QTT uses the natural ordering of (dimension, subdimension) associated with quantized state indices lexicographically. The QT3 format is a version of QTT that uses the transposed ordering (subdimension, dimension) lexicographically in ordering quantized states in the state vector. Assuming that there are m states to represent each molecule, a quantization size of 2 is used, and m is a power of 2, implying $l_h = \log_2 m$ for $h = 1, \dots, d$; the quantized state representation for the QT3 format is given by $(s_{1,1}, \dots, s_{d,1}, s_{1,2}, \dots, s_{d,2}, \dots, s_{1,l_1}, \dots, s_{d,l_d})$.

The QTT- and QT3-based BDF5 solvers both use the DMRG method to solve the linear system at each time step. Our implementation of DMRG follows the `dmrg_solve3.m` function of the `tt-toolbox`.[57] We use the same parameters as the default parameters in `dmrg_solve3.m` except for two of them. Therein, the default aggregation order changes in consecutive iterations, and the aggregated systems are solved directly if the order of the system is smaller than a threshold of 2,500. In our experiments, the former choice leads to slower convergence and the latter choice results in solving relatively large systems directly. Hence, we use the same aggregation order in all iterations and solve the aggregated systems directly if the order of the system is smaller than 50.[66] Furthermore, in order not to put the QTT- and QT3-based BDF5 solvers at a disadvantage, we use a truncation tolerance of `1e-14` in computing the low-rank approximation to $\mathbf{Q}_n$ at time step n,[43] although we have also experimented with a truncation tolerance of `1e-16`. In both cases, the ranks associated with the approximation of $\mathbf{Q}_n$ have turned out to be modest. In our experiments, the QTT-based solver has outperformed the QT3-based solver in terms of time, memory, and accuracy. Therefore, we do not report results with the latter.

4.2 | Assessment of results

We compare the mean number of molecules obtained by the proposed BDF5 solvers and SSA in order to assess the accuracy of computed solutions. For each model, we have generated 10^8 trajectories using the benchmark implementation StochKit.[62] StochKit chooses the optimal SSA method for the given model and reports the mean and the variance of the number of molecules of each type at time t_f. Letting $\mu_h^{(\text{SSA})}$ and $\sigma_h^{(\text{SSA})}$ denote the mean and the standard deviation of the number of molecules of type h obtained by the simulation, respectively, the confidence interval (see pp. 68–71 in the work of Asmussen et al. [67]) corresponding to a confidence probability of 95% is given by

$$\left(\mu_h^{(\text{SSA})} - \tau_h^{(\text{SSA})}, \mu_h^{(\text{SSA})} + \tau_h^{(\text{SSA})}\right), \quad \text{where} \quad \tau_h^{(\text{SSA})} = 1.96\frac{\sigma_h^{(\text{SSA})}}{10^4}.$$

Then, it is said that simulation has achieved an accuracy of $\max_h(\tau_h^{(\text{SSA})}/\mu_h^{(\text{SSA})})$. Letting μ_h denote the mean number of type h molecules obtained by a BDF5 solver at time t_f,

$$re_\mu = \max_h \left(\frac{\left|\mu_h - \mu_h^{(\text{SSA})}\right|}{\mu_h^{(\text{SSA})}} \right)$$

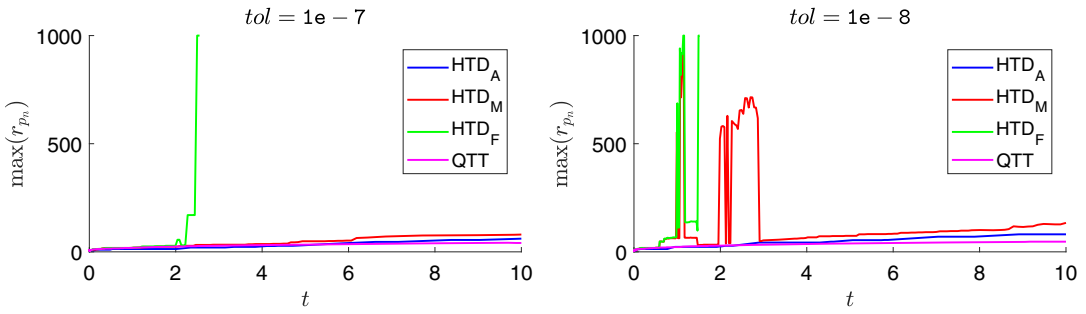

FIGURE 2 Maximum ranks of solution vectors for the metabolite synthesis model with $d = 4$. HTD = hierarchical Tucker decomposition; QTT = quantized TT

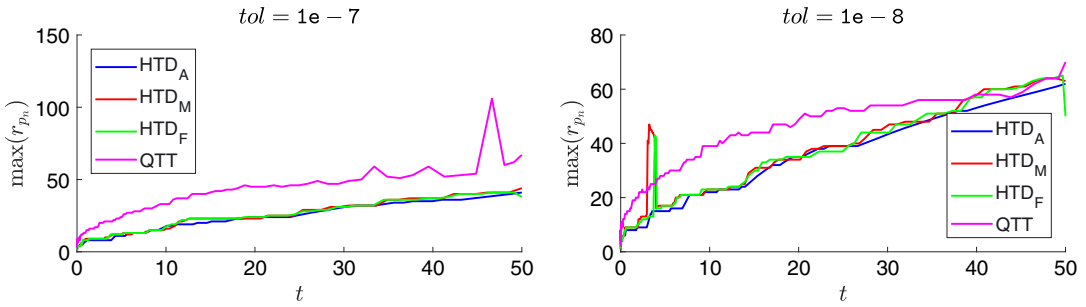

FIGURE 3 Maximum ranks of solution vectors for the extended toggle switch model with $d = 4$. HTD = hierarchical Tucker decomposition; QTT = quantized TT

provides the maximum relative error in the computation of the mean, assuming that simulation is at least as accurate as BDF5.

We present the numerical results in Tables 2, 4, 6, and 8 and the maximum ranks of solution vectors in Figures 2–5. The first four columns of the tables give the solver name, the accuracy tolerance *tol*, the time t_e for which the last solution vector is computed by 1,000 s, and the number of time steps N_e taken by the computation. Column $\max_n(|\mathcal{R}_n|)$ lists the maximum truncated state space size used through the computation, column $\max_n(r_{\mathbf{p}_n})$ lists the maximum of the ranks of solution vectors, and column $\max_n(r_{\mathbf{Q}_n})$ lists the maximum rank associated with the QTT approximation of the truncated generated matrix. We remark that $\max_n(|\mathcal{R}_n|)$ corresponds to the maximum core tensor dimension associated with $\mathbf{Q}_n$ raised to the power of 2 for the QTT-based solver. Column "It" reports the average number of Jacobi iterations per time step for HTD-based solvers and the average number of DMRG iterations per time step for the QTT-based solver. Columns *nnz* and "Time" give the number of floating-point array elements allocated and the execution time in seconds, respectively. The next two columns report BDF5 and state space truncation errors, where

$$tte = \sum_{n=1}^{N_e} lte_n \quad \text{and} \quad \Delta\mathbf{p} = \sum_{n=1}^{N_e} |\mathbf{p}_n\mathbf{e} - \mathbf{p}_{n-1}\mathbf{e}|.$$

Finally, the last column reports the relative error, re_μ, in the computation with respect to the simulation if the solution at time t_f can be computed by 1,000 s. The values under columns $\max_n(|\mathcal{R}_n|)$, *nnz*, *tte*, $\Delta\mathbf{p}$, and re_μ are rounded to one decimal digit of accuracy and reported using scientific notation.

The maximum truncated state space size, $\max_n(|\mathcal{R}_n|)$, is an important measure for the HTD format to see the advantage of the compact representation over the flat representation. BDF5 with the Jacobi method needs nine vectors as long as the truncated state space size (six backward difference vectors and three auxiliary vectors to store the solution, prediction, and diagonal vectors). Hence, the number of nonzeros allocated in BDF5 with flat vectors would be roughly nine times that of $\max_n(|\mathcal{R}_n|)$. Note that $\max_n(|\mathcal{R}_n|)$ for the QTT-based solver can be misleading in this respect. Because $|\mathcal{R}_{n,h}|$ does not need to be a power of 2 for $n = 1, \ldots, N_e$ and $h = 1, \ldots, d$, the values of $\max_n(|\mathcal{R}_n|)$ for the QTT format end up being much larger than their counterparts for the HTD format.

We remark due to an earlier result that the state space truncation error at each step will be bounded by the total probability mass outside $\mathcal{R}_n$ if the truncated system is solved exactly (see theorem 2 in the work of Munsky et al.[13]). However,

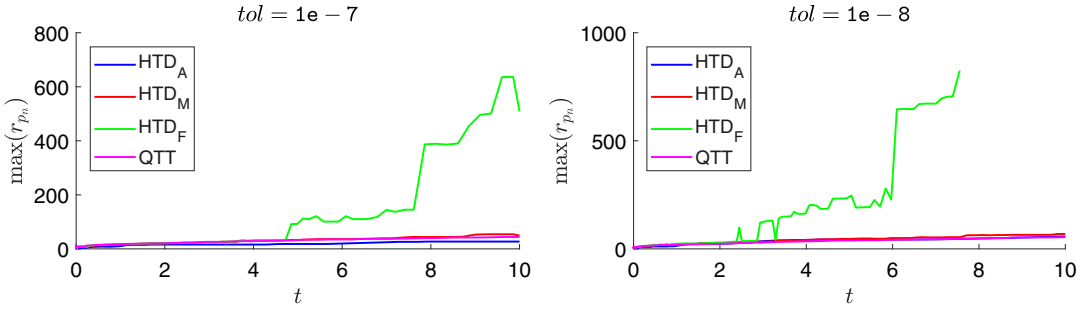

FIGURE 4 Maximum ranks of solution vectors for the lambda phage model with $d = 5$. HTD = hierarchical Tucker decomposition; QTT = quantized TT

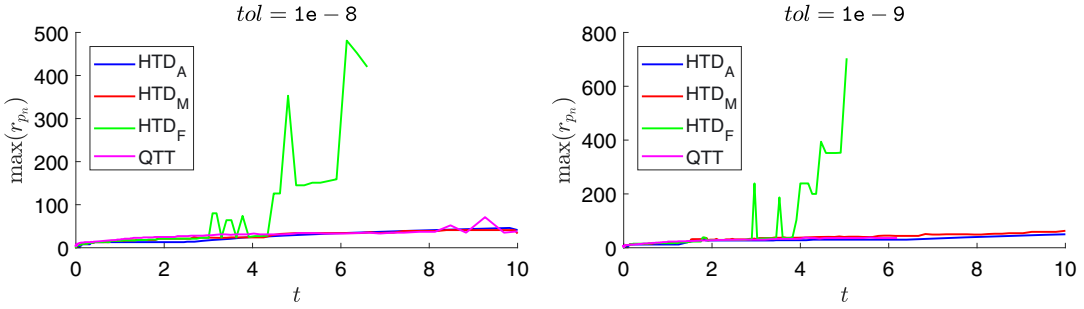

FIGURE 5 Maximum ranks of solution vectors for the cascade model with $d = 6$. HTD = hierarchical Tucker decomposition; QTT = quantized TT

this value appears to be negative in some models due to numerical errors (see, for instance, the work of Kazeev et al.[43]). Therefore, instead of computing the probability mass outside $\mathcal{R}_n$ at the end of execution, we report $\Delta\mathbf{p}$ as the state space truncation error. Note that the local state space truncation error at time step n, that is, $|\mathbf{p}_n\mathbf{e} - \mathbf{p}_{n-1}\mathbf{e}|$, is dominated by other numerical errors such as the BDFk truncation error if the probability mass outside $\mathcal{R}_{n-1}$ and $\mathcal{R}_n$ is sufficiently large. Hence, $\Delta\mathbf{p}$ is in fact an upper bound on the numerical error in the computed result in 1-norm upon the stopping of the execution of the solver. We therefore refer to $\Delta\mathbf{p}$ throughout the discussion as the accuracy of the CME solver. Numerical results also validate this interpretation because $\Delta\mathbf{p}$ and re_μ are close to each other when simulation is at least as accurate as the CME solver.

We remark that the QTT-based solver achieves a higher accuracy in terms of $\Delta\mathbf{p}$ than HTD-based solvers for the same value of *tol*, and the ratios of *tte* and $\Delta\mathbf{p}$ are different for HTD- and QTT-based solvers. This phenomenon seems to be due to the structure of QTT and the method chosen to solve the linear system in BDFk, and it may be the subject of future research. Now, we discuss the models used in the numerical experiments and their particular results.

4.3 | Results of models

Model 1. (Metabolite synthesis)
A biological process associated with metabolite synthesis involving two metabolites and two enzymes is considered.[55,68] In this model, molecules interact through the nine transition classes given in Table 1. Metabolites S_1 and S_2 are synthesized respectively by enzymes S_3 and S_4 through the first two transition classes. The third transition class corresponds to the consumption of metabolites, and the metabolites are degraded through the fourth and fifth transition classes. The last four transition classes correspond to the production and destruction of enzymes S_3 and S_4. Here, $d = 4$, $\mathbf{s} = (s_1, s_2, s_3, s_4)$, $J = 9$, and $k_A, k_B, K_I, k_2, \mu, K_R, k_{E_A}, k_{E_B} \in \mathbb{R}_{>0}$.

In the numerical experiments, we let $(k_A, k_B, K_I, k_2, \mu, K_R, k_{E_A}, k_{E_B}) = (0.3, 0.3, 60, 0.001, 0.002, 30, 0.02, 0.02)$[68] for initial state $\mathbf{s}_0 = (10, 10, 10, 10)$ and final time $t_f = 10$. Simulation yields mean values with an accuracy of `3e−5` in 898 s. Table 2 presents the numerical results for the metabolite synthesis model.

Among the HTD-based solvers, HTD_F fails to compute the solution at time t_f because fixed rank control in the Newton–Schulz method results in solution vectors with large ranks. Time and memory requirements decrease significantly when adaptive rank control is used in Newton–Schulz method instead of fixed rank control. When adaptive

TABLE 1 Transition classes of the metabolite synthesis model with $d = 4$

j	$\gamma^{(j)}$	$\alpha_1^{(j)}(s_1)$	$\alpha_2^{(j)}(s_2)$	$\alpha_3^{(j)}(s_3)$	$\alpha_4^{(j)}(s_4)$	$\nu^{(j)}$
1	$k_A K_I$	$\frac{1}{s_1+K_I}$	1	s_3	1	$\mathbf{e}_1^T$
2	$k_B K_I$	1	$\frac{1}{s_2+K_I}$	1	s_4	$\mathbf{e}_2^T$
3	k_2	s_1	s_2	1	1	$(-\mathbf{e}_1 - \mathbf{e}_2)^T$
4	μ	s_1	1	1	1	$-\mathbf{e}_1^T$
5	μ	1	s_2	1	1	$-\mathbf{e}_2^T$
6	$k_{E_A} K_R$	$\frac{1}{s_1+K_R}$	1	1	1	$\mathbf{e}_3^T$
7	$k_{E_B} K_R$	1	$\frac{1}{s_2+K_R}$	1	1	$\mathbf{e}_4^T$
8	μ	1	1	s_3	1	$-\mathbf{e}_3^T$
9	μ	1	1	1	s_4	$-\mathbf{e}_4^T$

TABLE 2 Numerical results for the metabolite synthesis model with $d = 4$

Solver	*tol*	t_e	N_e	$\max_n(\lvert\mathcal{R}_n\rvert)$	$\max_n(r_{\mathbf{p}_n})$	$\max_n(r_{\mathbf{Q}_n})$	It	*nnz*	Time	*tte*	$\Delta\mathbf{p}$	re_μ
HTD_A	1e−6	10.00	205	3e5	21	N/A	1.5	7e4	7	1e−4	2e−3	2e−3
HTD_M		10.00	221	3e5	30	N/A	1.3	8e4	8	1e−4	3e−3	3e−3
HTD_F		5.84	167	2e5	656	N/A	1.4	1e8	1,491	8e−5	1e−3	–
QTT		10.00	151	1e8	33	13	2.3	2e6	69	5e−5	3e−5	2e−5
HTD_A	1e−7	10.00	223	5e5	60	N/A	1.9	2e5	33	9e−6	1e−4	1e−4
HTD_M		10.00	227	5e5	79	N/A	1.5	4e5	41	1e−5	1e−4	1e−4
HTD_F		2.55	161	1e5	1,000	N/A	1.3	8e7	1,075	6e−6	4e−5	–
QTT		10.00	215	1e8	42	13	2.3	3e6	200	7e−6	5e−6	1e−5
HTD_A	1e−8	10.00	314	7e5	81	N/A	1.9	6e5	144	1e−6	2e−5	2e−5
HTD_M		10.00	306	7e5	960	N/A	1.5	8e6	581	1e−6	2e−5	2e−5
HTD_F		1.52	178	1e5	1,000	N/A	1.5	6e7	1,240	8e−7	2e−6	–
QTT		10.00	303	1e8	47	14	2.2	6e6	469	1e−6	7e−7	1e−5

Note. HTD = hierarchical Tucker decomposition; QTT = quantized TT.

rank control is also used in the Jacobi method, improvements in time and memory become as high as 75% and 92% for *tol* = 1e−8, respectively.

Figure 2 shows that the maximum rank of solution vectors changes smoothly over time when HTD_A is used. Regarding the QTT format, the maximum rank associated with the approximation of $\mathbf{Q}_n$ is modest, and the maximum rank associated with solution vectors turns out to be the best among all solvers for decreasing values of *tol*.

Memory requirements of the QTT-based solver are at least an order of magnitude higher than those of solvers using HTD_A and HTD_M. Note that the number of nonzeros allocated for flat vectors would roughly be 22≈(9×5e5/23e55) and 10≈(9×7e5/6e5) times that of compact vectors allocated in HTD_A for accuracy tolerances of 1e−7 and 1e−8, respectively. Furthermore, the QTT-based solver is clearly slower than the HTD_A-based solver, although the discrepancy becomes relatively smaller as *tol* decreases. On the other hand, the accuracy of the QTT-based solver is higher than those of HTD-based solvers for all values of *tol*. We remark that when *tol* =1e−8, the accuracy of simulation is reached by HTD_A and HTD_M. The QTT-based solver reaches the same accuracy early on, when *tol* =1e−6.

Model 2. (Extended toggle switch)

A system of stochastic chemical kinetics modeling the biological process associated with a toggle switch is considered.[28] In this model, S_1 and S_2 are two mutually repressing gene products and express the proteins S_3 and S_4, respectively. These four molecules interact through the eight transition classes given in Table 3. The first four transition classes correspond respectively to the production of S_1, the production of S_2, the destruction of S_1, and the destruction of S_2. The fifth and sixth transition classes correspond to the expression of gene products, and the last two transition classes correspond to the destruction of gene products. Here, $d = 4$, $\mathbf{s} = (s_1, s_2, s_3, s_4)$, $J = 8$, and a_{11}, a_{12}, a_{21}, a_{22}, a_3, a_4, a_5, a_6, a_7, $a_8 \in \mathbb{R}_{>0}$.

TABLE 3 Transition classes of the extended toggle switch model with $d = 4$

j	$\gamma^{(j)}$	$\alpha_1^{(j)}(s_1)$	$\alpha_2^{(j)}(s_2)$	$\alpha_3^{(j)}(s_3)$	$\alpha_4^{(j)}(s_4)$	$\nu^{(j)}$
1	a_{11}	1	$\frac{1}{s_2^2+a_{12}}$	1	1	$\mathbf{e}_1^T$
2	a_{21}	$\frac{1}{s_1^2+a_{22}}$	1	1	1	$\mathbf{e}_2^T$
3	a_3	s_1	1	1	1	$-\mathbf{e}_1^T$
4	a_4	1	s_2	1	1	$-\mathbf{e}_2^T$
5	a_5	s_1	1	1	1	$\mathbf{e}_3^T$
6	a_6	1	s_2	1	1	$\mathbf{e}_4^T$
7	a_7	1	1	s_3	1	$-\mathbf{e}_3^T$
8	a_8	1	1	1	s_4	$-\mathbf{e}_4^T$

TABLE 4 Numerical results for the extended toggle switch model with $d = 4$

Solver	*tol*	t_e	N_e	$\max_n(\lvert\mathcal{R}_n\rvert)$	$\max_n(r_{\mathbf{p}_n})$	$\max_n(r_{\mathbf{Q}_n})$	It	*nnz*	Time	*tte*	Δp	re_μ
HTD_A	1e−6	50.00	173	7e5	28	N/A	2.1	1e5	6	7e−5	1e−3	2e−3
HTD_M		50.00	170	7e5	28	N/A	1.7	1e5	5	7e−5	1e−3	2e−3
HTD_F		50.00	170	7e5	28	N/A	1.7	1e5	5	7e−5	1e−3	2e−3
QTT		50.00	134	1e8	80	11	2.8	6e6	257	5e−5	8e−5	7e−5
HTD_A	1e−7	50.00	165	9e5	40	N/A	2.1	2e5	21	6e−6	2e−4	3e−4
HTD_M		50.00	161	9e5	41	N/A	1.7	2e5	15	6e−6	2e−4	3e−4
HTD_F		50.00	162	9e5	41	N/A	1.7	2e5	16	6e−6	2e−4	3e−4
QTT		50.00	163	3e8	106	12	2.5	2e7	977	6e−6	1e−5	6e−5
HTD_A	1e−8	50.00	220	1e6	61	N/A	2.2	6e5	75	9e−7	2e−5	8e−5
HTD_M		50.00	218	1e6	64	N/A	1.7	6e5	53	9e−7	2e−5	8e−5
HTD_F		50.00	223	1e6	65	N/A	1.7	6e5	58	9e−7	2e−5	8e−5
QTT		50.00	220	1e8	64	12	2.3	7e6	863	8e−7	8e−7	5e−5

Note. HTD = hierarchical Tucker decomposition; QTT = quantized TT.

In the numerical experiments, we let

$$(a_{11}, a_{12}, a_{21}, a_{22}, a_3, a_4, a_5, a_6, a_7, a_8) = (10, 30, 10, 30, 0.017, 0.017, 0.01, 0.01, 0.01, 0.01)^{28}$$

for initial state $\mathbf{s}_0 = (10, 10, 10, 10)$ and final time $t_f = 50$. Simulation yields mean values with an accuracy of 7e−5 in 726 s. Table 4 presents the numerical results for the extended toggle switch model.

In this model, rank control strategies for HTD-based solvers do not influence the maximum ranks of solution vectors much (see Figure 3). Adaptive rank control in the Jacobi method increases the average number of iterations it performs due to additional iterations that get executed after rank bounds are increased. However, this increase is not offset by a sizable (if any) decrease in the ranks of solution vectors. Consequently, HTD_A ends up being slower than the other HTD-based solvers. Regarding the QTT format, ranks associated with the approximation of $\mathbf{Q}_n$ are again modest; however, the maximum rank associated with solution vectors turns out to be the largest among all solvers except for $tol = 1e-8$.

The QTT-based solver is more accurate than HTD-based solvers for the same value of *tol* in this model as well. The accuracy of simulation is reached by HTD-based solvers when $tol = 1e-8$ and by the QTT-based solver for all values of *tol* used. However, compared with HTD-based solvers, the QTT-based solver requires about an order of magnitude more memory and significantly more time to compute the solution as accurately as simulation computes mean values. The number of nonzeros allocated for flat vectors would roughly be 40 and 15 times that of compact vectors allocated in HTD format for accuracy tolerances of 1e−7 and 1e−8, respectively.

Model 3. (Lambda phage)

The biological process associated with the life cycle of bacteriophage-λ is a naturally occurring toggle switch that consists of two phases.[23,32,69] Transition from one phase to the other is controlled by competing proteins S_1 and S_2. In this model, molecules interact through 10 transition classes given in Table 5. Proteins S_1 and S_2 are generated through the first and the third transition classes, respectively. Other proteins are generated through the fifth, seventh, and ninth

TABLE 5 Transition classes of the lambda phage model with $d = 5$

j	$\gamma^{(j)}$	$\alpha_1^{(j)}(s_1)$	$\alpha_2^{(j)}(s_2)$	$\alpha_3^{(j)}(s_3)$	$\alpha_4^{(j)}(s_4)$	$\alpha_5^{(j)}(s_5)$	$\nu^{(j)}$
1	$a_1 b_1$	1	$\frac{1}{s_2+b_1}$	1	1	1	$\mathbf{e}_1^T$
2	δ_1	s_1	1	1	1	1	$-\mathbf{e}_1^T$
3	b_2	$\frac{1}{s_1+b_2}$	1	1	1	$a_2 s_5 + 1$	$\mathbf{e}_2^T$
4	δ_2	1	s_2	1	1	1	$-\mathbf{e}_2^T$
5	a_3	1	$\frac{s_2}{s_2+1}$	1	1	1	$\mathbf{e}_3^T$
6	δ_3	1	1	s_3	1	1	$-\mathbf{e}_3^T$
7	a_4	1	1	$\frac{s_3}{s_3+1}$	1	1	$\mathbf{e}_4^T$
8	δ_4	1	1	1	s_4	1	$-\mathbf{e}_4^T$
9	a_5	1	1	$\frac{s_3}{s_3+1}$	1	1	$\mathbf{e}_5^T$
10	δ_5	1	1	1	$\frac{1}{s_4+1}$	s_5	$-\mathbf{e}_5^T$

TABLE 6 Numerical results for the lambda phage model with $d = 5$

Solver	*tol*	t_e	N_e	$max_n(\lvert\mathcal{R}_n\rvert)$	$max_n(r_{\mathbf{p}_n})$	$max_n(r_{\mathbf{Q}_n})$	It	*nnz*	Time	*tte*	$\Delta\mathbf{p}$	re_μ
HTD_A	1e−6	10.00	138	4e6	11	N/A	1.5	3e4	3	5e−5	8e−4	8e−4
HTD_M		10.00	128	4e6	32	N/A	1.4	8e4	4	5e−5	6e−4	8e−4
HTD_F		10.00	139	4e6	56	N/A	1.4	2e5	5	5e−5	7e−4	7e−4
QTT		10.00	109	4e9	41	13	2.5	3e6	128	4e−5	4e−5	3e−5
HTD_A	1e−7	10.00	146	7e6	27	N/A	1.6	1e5	9	6e−6	1e−4	2e−4
HTD_M		10.00	152	7e6	54	N/A	1.4	2e5	12	5e−6	1e−4	1e−4
HTD_F		10.00	144	6e6	637	N/A	1.3	1e7	356	5e−6	1e−4	1e−4
QTT		10.00	137	9e9	44	14	2.4	5e6	300	5e−6	4e−6	2e−5
HTD_A	1e−8	10.00	201	9e6	57	N/A	1.9	4e5	53	8e−7	2e−5	3e−5
HTD_M		10.00	192	9e6	69	N/A	1.5	4e5	42	8e−7	2e−5	3e−5
HTD_F		7.55	183	6e6	705	N/A	1.4	2e7	1,127	7e−7	1e−5	–
QTT		10.00	181	9e9	54	15	2.2	8e6	683	6e−7	6e−7	2e−5

Note. HTD = hierarchical Tucker decomposition; QTT = quantized TT.

transition classes. Other transition classes correspond to degradation of proteins. Here, $d = 5$, $\mathbf{s} = (s_1, s_2, s_3, s_4, s_5)$, $J = 10$, and $a_1, a_2, a_3, a_4, a_5, \delta_1, \delta_2, \delta_3, \delta_4, \delta_5, b_1, b_2 \in \mathbb{R}_{>0}$.

In the numerical experiments, we let

$$(a_1, a_2, a_3, a_4, a_5, \delta_1, \delta_2, \delta_3, \delta_4, \delta_5, b_1, b_2) = (0.5, 1.0, 0.15, 0.3, 0.3, 0.0025, 0.0007, 0.0231, 0.01, 0.01, 0.12, 0.6)^{23}$$

for initial state $\mathbf{s}_0 = (10, 10, 10, 10, 10)$ and final time $t_f = 10$. Simulation yields mean values with an accuracy of 6e−5 in 324 s. Table 6 presents the numerical results for the lambda phage model.

HTD_F fails to compute the solution at time t_f for $tol = 1e-8$. Time and memory requirements of HTD-based solvers become relatively smaller when adaptive rank control is used in the Newton–Schulz method instead of fixed rank control as *tol* decreases. It appears that the increase in the average number of iterations of Jacobi for smaller values of *tol* is more or less offset by a relative decrease in the maximum ranks of solution vectors for HTD_A and HTD_M, suggesting execution times that are in the same order. Figure 4 shows that the maximum rank of solution vectors changes smoothly over time with the HTD_A, HTD_M, and QTT formats.

The QTT-based solver requires more time and memory than the HTD-based solvers with adaptive rank control in Newton–Schulz method to compute the solution as accurately as simulation computes mean values. QTT is outperformed by HTD_A and HTD_M, which require at least an order of smaller number of nonzeros; however, the discrepancy in time becomes relatively smaller as *tol* decreases. In this model, the number of nonzeros allocated for flat vectors would roughly be 630 and 202 times that of compact vectors allocated in HTD_A for accuracy tolerances of $1e-7$ and $1e-8$, respectively.

Model 4. (Cascade)

Consider a system of stochastic chemical kinetics modeling the biological process in which adjacent genes produce

TABLE 7 Transition classes of the cascade model with $d = 6$

j	$\gamma^{(j)}$	$\alpha_1^{(j)}(s_1)$	$\alpha_2^{(j)}(s_2)$	$\alpha_3^{(j)}(s_3)$	$\alpha_4^{(j)}(s_4)$	$\alpha_5^{(j)}(s_5)$	$\alpha_6^{(j)}(s_6)$	$\nu^{(j)}$
1	a	1	1	1	1	1	1	$\mathbf{e}_1^T$
2	μ	s_1	1	1	1	1	1	$-\mathbf{e}_1^T$
3	b	$\frac{s_1}{bs_1+c}$	1	1	1	1	1	$\mathbf{e}_2^T$
4	μ	1	s_2	1	1	1	1	$-\mathbf{e}_2^T$
5	b	1	$\frac{s_2}{bs_2+c}$	1	1	1	1	$\mathbf{e}_3^T$
6	μ	1	1	s_3	1	1	1	$-\mathbf{e}_3^T$
7	b	1	1	$\frac{s_3}{bs_3+c}$	1	1	1	$\mathbf{e}_4^T$
8	μ	1	1	1	s_4	1	1	$-\mathbf{e}_4^T$
9	b	1	1	1	$\frac{s_4}{bs_4+c}$	1	1	$\mathbf{e}_5^T$
10	μ	1	1	1	1	s_5	1	$-\mathbf{e}_5^T$
11	b	1	1	1	1	$\frac{s_5}{bs_5+c}$	1	$\mathbf{e}_6^T$
12	μ	1	1	1	1	1	s_6	$-\mathbf{e}_6^T$

TABLE 8 Numerical results for the cascade model with $d = 6$

Solver	*tol*	t_e	N_e	max($\lvert\mathcal{R}_n\rvert$)	max($r_{\mathbf{p}_n}$)	max($r_{\mathbf{Q}_n}$)	It	*nnz*	Time	*tte*	Δp	re_μ
HTD_A	1e−7	10.00	172	9e8	26	N/A	2.0	1e5	13	6e−6	2e−3	2e−3
HTD_M		10.00	179	9e8	24	N/A	1.7	8e4	12	7e−6	2e−3	3e−3
HTD_F		10.00	179	9e8	128	N/A	1.7	8e5	48	7e−6	2e−3	3e−3
QTT		10.00	151	4e12	32	11	2.4	6e6	422	5e−6	2e−5	6e−5
HTD_A	1e−8	10.00	220	1e9	46	N/A	1.8	3e5	42	8e−7	7e−5	1e−4
HTD_M		10.00	227	1e9	41	N/A	1.6	2e5	35	9e−7	1e−4	2e−4
HTD_F		6.60	208	8e8	481	N/A	1.5	1e7	1,024	8e−7	6e−5	–
QTT		10.00	212	5e11	71	11	2.4	8e6	746	8e−7	1e−6	5e−5
HTD_A	1e−9	10.00	294	2e9	49	N/A	1.6	5e5	93	1e−7	3e−5	8e−5
HTD_M		10.00	295	2e9	59	N/A	1.4	5e5	91	1e−7	3e−5	8e−5
HTD_F		5.05	263	7e8	394	N/A	1.4	1e7	1,295	1e−7	1e−5	–
QTT		6.19	274	2e12	38	12	2.2	8e6	1,003	1e−7	1e−7	–

Note. HTD = hierarchical Tucker decomposition; QTT = quantized TT.

protein that enhances the expression of the succeeding gene.[23,66] The number of molecules in the cascade model may vary. We consider a cascade model with six molecules. Transition classes for the six-dimensional cascade model are given in Table 7. Here, $d = 6$, $\mathbf{s} = (s_1, s_2, s_3, s_4, s_5, s_6)$, $J = 12$, and $a, b, c, \mu \in \mathbb{R}_{>0}$.

In the numerical experiments, we let $(a, b, c, \mu) = (0.7, 1.0, 5.0, 0.07)$[23] for initial state $\mathbf{s}_0 = (10, 10, 10, 10, 10, 10)$ and final time $t_f = 10$. Simulation yields mean values with an accuracy of 5e−5 in 1,317 s. Table 8 presents the numerical results for the cascade model.

HTD_F fails to compute the solution at time t_f for accuracy tolerances of 1e−8 and 1e−9. Adaptive rank control in the Newton–Schulz method decreases time and memory requirements significantly compared with fixed rank control in HTD-based solvers. Adaptive rank control in the Jacobi method slightly decreases the maximum rank of solution vectors for $tol = 1e-9$; however, this has a relatively limited effect on time and memory requirements. Figure 5 shows that the maximum rank of solution vectors changes smoothly over time with the HTD_A, HTD_M, and QTT formats when $tol = 1e-9$.

The QTT-based solver requires more time and memory than HTD-based solvers with adaptive rank control in the Newton–Schulz method to compute the solution as accurately as simulation computes mean values, although the QTT-based solver is again more accurate for the same value of *tol*. However, the QTT-based solver is not able to reach t_f for $tol = 1e-9$ within 1,000 s in this model. The number of nonzeros allocated for flat vectors would roughly be 30,000 and 36,000 times that of compact vectors allocated in HTD_A for accuracy tolerances of 1e−8 and 1e−9, respectively.

5 | CONCLUSION

This study considers the implicit time stepping BDF5 method to solve the CME with compact vectors in HTD and QTT formats. At each time step, the countably infinite state space is truncated using a prediction vector without having to compute the solution at that step. The HTD-based solver works with the truncated generated matrix that can be expressed exactly in Kronecker form, thereby circumventing its low-rank approximation that takes place in the QTT-based solver. The linear system associated with the truncated generator matrix in Kronecker form is solved in the HTD-based solver with the Jacobi iteration, which utilizes the Newton–Schulz iteration to compute the elementwise reciprocated diagonal of the coefficient matrix. The QTT-based solver uses the DMRG method to handle the linear system. When a fixed rank control strategy is employed in the iterative methods of Jacobi and Newton–Schulz methods, the solution vectors in HTD format may end up being computed to be unnecessarily accurate. Adaptive rank control strategies that aim to avoid this when possible at the expense of additional iterations are investigated.

Numerical experiments on four benchmark models indicate that adaptive rank control strategies prevent the ranks of HTD solution vectors from becoming very large compared with fixed rank control strategies. This leads to considerable improvement in time and memory requirements in three of the four models considered. The QTT-based solver is more accurate than the HTD-based solver for the same accuracy tolerance. However, HTT-based solvers with adaptive rank control can achieve an accuracy comparable with that of the QTT-based solver in less time and smaller memory by choosing a smaller accuracy tolerance in three of the four models. When the results computed by the proposed BDF5 solvers using compact solution vectors are compared with those of simulation, it can be said that it is possible to achieve comparable or better accuracy in less time using the numerical approach.

ACKNOWLEDGEMENTS

Part of this work is supported by the Alexander von Humboldt Foundation through the Research Group Linkage Programme. The research of M. Can Orhan is carried out during his PhD studies at Bilkent University and supported by The Scientific and Technological Research Council of Turkey under Grant 2211-A. The authors thank the referee for the constructive report that led to an improved manuscript.

ORCID

Tuğrul Dayar http://orcid.org/0000-0001-6906-8458

REFERENCES

1. Goutsias J, Jenkinson G. Markovian dynamics on complex reaction networks. Phys Rep. 2013;529(2):199–264.
2. Gadgil C, Lee CH, Othmer HG. A stochastic analysis of first-order reaction networks. B Math Biol. 2005;67(5):901–946.
3. Heuett WJ, Qian H. Grand canonical Markov model: A stochastic theory for open nonequilibrium biochemical networks. J Chem Phys. 2006;124(4):044110.
4. Jahnke T, Huisinga W. Solving the chemical master equation for monomolecular reaction systems analytically. J Math Biol. 2007;54(1):1–26.
5. Zhang X, Cock KDe, Bugallo MF, Djurić PM. A general method for the computation of probabilities in systems of first order chemical reactions. J Chem Phys. 2005;122(10):104101.
6. Gillespie DT. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. J Chem Phys. 1976;22(4):403–434.
7. Gillespie DT. Exact stochastic simulation of coupled chemical reactions. J Chem Phys. 1977;81(25):2340–2361.
8. Gillespie DT. Approximate accelerated stochastic simulation of chemically reacting systems. J Chem Phys. 2001;115(4):1716–1716.
9. Rathinam M, Petzold LR, Cao Y, Gillespie DT. Stiffness in stochastic chemically reacting systems: The implicit tau-leaping method. J Chem Phys. 2003;119(24):12784–12794.
10. Cao Y, Gillespie DT, Petzold LR. The slow-scale stochastic simulation algorithm. J Chem Phys. 2005;122(1):014116.
11. Goutsias J. Quasiequilibrium approximation of fast reaction kinetics in stochastic biochemical systems. J Chem Phys. 2005;122(18):184102.
12. Rao CV, Arkin AP. Stochastic chemical kinetics and the quasi-steady-state assumption: Application to the Gillespie algorithm. J Chem Phys. 2003;118(11):4999–5010.
13. Munsky B, Khammash M. The finite state projection algorithm for the solution of the chemical master equation. J Chem Phys. 2006;124(14):044104.
14. Stewart WJ. Introduction to the numerical solution of Markov chains. Princeton, NJ: Princeton University Press; 1994.

15. Munsky B, Khammash M. A multiple time interval finite state projection algorithm for the solution to the chemical master equation. J Comput Phys. 2007;226(1):818–835.
16. Burrage K, Hegland M, Macnamara S, Sidje R. A Krylov-based finite state projection algorithm for solving the chemical master equation arising in the discrete modelling of biological systems. In: Langville AN, Stewart WJ, editors. Proceedings of the AA Markov 150th Anniversary Meeting; 2006 Jun 12–14; Charleston, SC, USA. Raleigh, NC: Boson Books; 2006.
17. MacNamara S, Burrage K, Sidje RB. Multiscale modeling of chemical kinetics via the master equation. Multiscale Model Sim. 2008;6(4):1146–1168.
18. Sunkara V, Hegland M. An optimal finite state projection method. Procedia Comput Sci. 2010;1(1):1579–1586.
19. Wolf V, Goel R, Mateescu M, Henzinger TA. Solving the chemical master equation using sliding windows. BMC Syst Biol. 2010;4:42.
20. Dinh KN, Sidje RB. Understanding the finite state projection and related methods for solving the chemical master equation. Phys Biol. 2016;13(3): 035003.
21. Dayar T, Mikeev L, Wolf V. On the numerical analysis of stochastic Lotka–Volterra models. Paper presented at: Proceedings of the 2010 International Multiconference on Computer Science and Information Technology (IMCSIT); 2010 Oct 18–20; Wisla, Poland. Piscataway, NJ: IEEE; 2010.
22. Hellander A, Lötstedt P. Hybrid method for the chemical master equation. J Comput Phys. 2007;227(1):100–122.
23. Hegland M, Burden C, Santoso L, MacNamara S, Booth H. A solver for the stochastic master equation applied to gene regulatory networks. J Comput Appl Math. 2007;205(2):708–724.
24. Hegland M, Hellander A, Lötstedt P. Sparse grids and hybrid methods for the chemical master equation. BIT. 2008;48:265–283.
25. Deuflhard P, Huisinga W, Jahnke T, Wulkow M. Adaptive discrete Galerkin methods applied to the chemical master equation. SIAM J Sci Comput. 2008;30(6):2990–3011.
26. Engblom S. Spectral approximation of solutions to the chemical master equation. J Comput Appl Math. 2009;229(1):208–221.
27. Jahnke T. An adaptive wavelet method for the chemical master equation. SIAM J Sci Comput. 2010;31(6):4373–4394.
28. Jahnke T, Udrescu T. Solving chemical master equations by adaptive wavelet compression. J Comput Phys. 2010;229(16):5724–5741.
29. Grasedyck L, Kressner D, Tobler C. A literature survey of low-rank tensor approximation techniques. GAMM Mitt. 2013;36(1):53–78.
30. Carroll JD, Chang JJ. Analysis of individual differences in multidimensional scaling via an n-way generalization of "Eckart–Young" decomposition. Psychometrika. 1970;35(3):283–319.
31. Harshman RA. Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multi-modal factor analysis. UCLA Working Papers in Phonetics. 1970;16:1–84.
32. Jahnke T, Huisinga W. A dynamical low-rank approach to the chemical master equation. B Math Biol. 2008;70(8):2283–2302.
33. Hegland M, Garcke J. On the numerical solution of the chemical master equation with sums of rank one tensors. ANZIAM J. 2011;52:C628–C643.
34. Kolda TG, Bader BW. Tensor decompositions and applications. SIAM Rev. 2009;51(3):455–500.
35. Tucker L. Some mathematical notes on three-mode factor analysis. Psychometrika. 1966;31(3):279–311.
36. De Lathauwer L, De Moor B, Vandewalle J. A multilinear singular value decomposition. SIAM J Matrix Anal A. 2000;21(4):1253–1278.
37. Oseledets IV. Tensor-train decomposition. SIAM J Sci Comput. 2011;33(5):2295–2317.
38. Khoromskij BN. $O(d\log N)$-Quantics approximation of $N - d$ tensors in high-dimensional numerical modeling. Constr Approx. 2011;34(2):257–280.
39. Khoromskij BN. Tensor numerical methods for high-dimensional PDEs: Basic theory and initial applications. ESAIM. 2015;48:1–28.
40. Dolgov SV, Khoromskij BN, Oseledets IV. Fast solution of multi-dimensional parabolic problems in the TT/QTT formats with initial approximation to the Fokker–Planck equation. SIAM J Sci Comput. 2012;34(6):A3016–A3038.
41. Dolgov SV, Khoromskij B. Two-level QTT-Tucker format for optimized tensor calculus. SIAM J Matrix Anal Appl. 2013;34(2):593–623.
42. Dolgov SV, Khoromskij B. Simultaneous state-time approximation of the chemical master equation using tensor product formats. Numer Linear Algebra. 2015;22(2):197–219.
43. Kazeev V, Khammash M, Nip M, Schwab C. Direct solution of the chemical master equation using quantized tensor trains. PLoS Comput Biol. 2014;10(3):e1003359.
44. Oseledets IV, Dolgov SV. Solution of linear systems and matrix inversion in the TT-format. SIAM J Sci Comput. 2012;34(5):A2718–A2739.
45. Grasedyck L. Hierarchical singular value decomposition of tensors. SIAM J Matrix Anal A. 2010;31(4):2029–2054.
46. Hackbusch W, Kühn S. A new scheme for the tensor representation. J Fourier Anal Appl. 2009;15(5):706–722.
47. Kressner D, Tobler C. htucker — A Matlab toolbox for tensors in hierarchical Tucker format. Lausanne, Switzerland: Mathematics Institute of Computational Science and Engineering; 2012. Available from: http://anchp.epfl.ch/htucker
48. Kressner D, Tobler C. Algorithm 941: htucker—A Matlab toolbox for tensors in hierarchical Tucker format. ACM T Math Software. 2014;40(3):Article 22.
49. Buchholz P, Dayar T, Kriege J, Orhan MC. Compact representation of solution vectors in Kronecker-based Markovian analysis. In: Agha G, Van Houdt B, editors. Quantitative Evaluation of Systems. QEST 2016. Lecture Notes in Computer Science. vol. 9826. Cham: Springer; 2006:260–276.
50. Buchholz P, Dayar T, Kriege J, Orhan MC. On compact solution vectors in Kronecker-based Markovian analysis. Perf Eval. 2017;115:132–149.

51. Mateescu M, Wolf V, Didier F, Henzinger TA. Fast adaptive uniformisation of the chemical master equation. IET Syst Biol. 2010;4(6):441–452.
52. Sandmann W, Wolf V. Computational probability for systems biology. In: Fisher J, editor. Formal Methods in Systems Biology. Lecture Notes in Computer Science. vol. 5054. Berlin, Heidelberg: Springer; 2008:33–47.
53. Ascher UM, Petzold LR. Computer methods for ordinary differential equations and differential-algebraic equations. Philadelphia: SIAM; 1998.
54. Shampine LF, Reichelt MW. The MATLAB ODE suite. SIAM J Sci Comput. 1997;18(1):1–22.
55. Dayar T, Orhan MC. Kronecker-based infinite level-dependent QBD processes. J Appl Probab. 2012;49(4):1166–1187.
56. Dayar T. Analyzing Markov chains using Kronecker products: Theory and applications. New York: Springer; 2012.
57. TT-Toolbox. Tensor Train Toolbox. 2014. Available from: https://github.com/oseledets/TT-Toolbox
58. CompactTransientSolver Software. 2017. Available from: http://www.cs.bilkent.edu.tr/~tugrul/software.html
59. APNN-Toolbox. Abstract Petri Net Notation Toolbox. 2004. Available from: http://www4.cs.uni-dortmund.de/APNN-TOOLBOX
60. Bause F, Buchholz P, Kemper P. In: Puigjaner R, Savino NN, Serra B, editors. Computer Performance Evaluation. TOOLS 1998. Lecture Notes in Computer Science. vol. 1469. Berlin, Heidelberg: Springer; 1998:356–359.
61. Netlib. A collection of mathematical software, papers, and databases. 2017. Available from: http://www.netlib.org
62. 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.
63. Golub GH, Van Loan CF. Matrix computations. 4th ed. Baltimore, MD: John Hopkins University Press; 2012.
64. Hackbusch W. Tensor spaces and numerical tensor calculus. Heidelberg, Germany: Springer; 2012.
65. Dayar T, Orhan MC. Cartesian product partitioning of multi-dimensional reachable state spaces. Probab Eng Inf Sci. 2016;30(3):413–430.
66. Dolgov SV, Savosyanov DV. Alternating minimal energy methods for linear systems in higher dimensions. SIAM J Sci Comput. 2014;36(5):A2248–A2271.
67. Asmussen S, Glynn PW. Stochastic simulation: Algorithms and analysis. Vol. 57. New York: Springer-Verlag; 2007.
68. Sjöberg PL, Lötstedt P, Elf J. Fokker–Planck approximation of the master equation in molecular biology. Comput Vis Sci. 2009;12(1):37–50.
69. Shea MA, Ackers GK. The OR control system of bacteriophage lambda: A physical-chemical model for gene regulation. J Mol Biol. 1985;181(2):211–230.

How to cite this article: Dayar T, Orhan MC. On compact vector formats in the solution of the chemical master equation with backward differentiation. *Numer Linear Algebra Appl*. 2018;e2158. https://doi.org/10.1002/nla.2158