User Documentation for ACME: A MATLAB-based toolbox for the simulation of biochemical reaction networks using approximations to the solution of the chemical master equation

Atefeh Kazeroonian [1,2], Fabian Fröhlich [1,2], Andreas Raue [3], Fabian J. Theis [1,2] and Jan Hasenauer [1,2*]

[1]Institute of Computational Biology, Helmholtz Zentrum München - German Research Center for Environmental Health, Neuherberg, Germany,
[2]Department of Mathematics, Chair of Mathematical Modeling of Biological Systems, Technische Universität München, Garching, Germany,
[3]Merrimack Pharmaceuticals Inc., Discovery Devision, Cambridge, MA 02139, USA

September 15, 2015

[*]to whom correspondence should be addressed.

Contents

Chapter 1

Introduction to stochastic biochemical kinetics

In this chapter, we briefly explain the fundamental concepts of the stochastic reaction kinetics and the modeling approaches used for the simulation of such kinetics.

1.1 Biochemical reaction networks

Consider a biological system comprising a chemical reaction network. The kinetics of such a network is usually modeled by a system of *chemical species* which undergo a set of *chemical reactions*. A *chemical species* represents an ensemble of molecules that are chemically identical, such as RNA molecules. A *chemical reaction* is a process in which chemical species are produced, degraded or converted into other chemical species. Accordingly, a chemical reaction network can be described as a system of:

- n_s chemical species, $S_1, S_2, \cdots, S_{n_s}$, and
- n_r chemical reactions, $R_1, R_2, \cdots, R_{n_r}$,

where the j^{th} chemical reaction, R_j, is a process as below:

$$R_j : \sum_{i=1}^{n_s} \nu_{i,j}^{-} S_i \xrightarrow{k_j} \sum_{i=1}^{n_s} \nu_{i,j}^{+} S_i$$

where k_j denotes the kinetic constant of reaction R_j. The coefficient $\nu_{i,j}^{-} \in \mathbb{N}_0$ denotes the stoichiometric coefficients of species S_i as a reactant in reaction R_j, and is defined as the number of S_i molecules consumed in reaction R_j. Similarly, the coefficient $\nu_{i,j}^{+} \in \mathbb{N}_0$ denotes the stoichiometric coefficients of species S_i as a product in reaction R_j, and is defined as the number of S_i molecules produced in reaction R_j. The overall stoichiometric coefficient of species S_i in reaction R_j, $\nu_{i,j} = \nu_{i,j}^{+} - \nu_{i,j}^{-} \in \mathbb{Z}$ is defined as the net change in the count of S_i when reaction R_j takes place.s The overall stoichiometry of a reaction R_j is then given by a vector $\nu_j = (\nu_{1,j}, \nu_{2,j}, \cdots, \nu_{n_s,j})^T \in \mathbb{N}_0^{n_s}$.

The state of the chemical reaction network at time t is represented by a vector $\mathbf{X_t} = (X_{1,t}, X_{2,t}, \ldots, X_{n_s,t}) \in \mathbb{N}_0^{n_s}$, where $X_{i,t}$ denotes the count of species S_i at time t, and remains constant as long as no reactions occur. Upon firing of the reaction R_j the state transition $\mathbf{X_t} \to \mathbf{X_t} + \nu_j$ occurs. The statistics of the time until the firing of next reaction, as well as the index of the next reaction, are determined by the *reaction propensities*. The *propensity* of reaction R_j, $a_j(X_t) : \mathbb{N}_0^{n_s} \to \mathbb{R}_+$ is a function of the state of the system. If we assume

Table 1.1: Reaction propensities according to the law of mass action.

Reaction Order	Reaction Type	Propensity
0	$\emptyset \xrightarrow{k_j}$ product	k_j
1	$S_i \xrightarrow{k_j}$ product	$k_j X_i$
2	$S_i + S_j \xrightarrow{k_j}$ product	$k_j X_i X_j$
2	$S_i + S_i \xrightarrow{k_j}$ product	$\frac{1}{2} k_j X_i (X_i - 1)$

Table 1.2: Relationship between the microscopic and macroscopic parameters.

Reaction Order	Reaction Type	Microscopic Kinetic Constant	Macroscopic Kinetic Constant
0	$\emptyset \xrightarrow{k_j}$ product	k_j	$k'_j = \frac{k_j}{\Omega}$
1	$S_i \xrightarrow{k_j}$ product	k_j	$k'_j = k_j$
2	$S_i + S_j \xrightarrow{k_j}$ product	k_j	$k'_j = k_j \Omega$

that the kinetics follow the law of mass action, the reaction propensities are determined by the order and the type of reaction as given in Table 1.1. In this Table, k_j denotes the kinetic constant of reaction R_j.

The reaction kinetics need not necessarily follow the law of mass action, and can be of more complicated forms. For example, the Mechaelis-Menten kinetics describes the rate of the enzymatic reactions as $\frac{V_{\max}[X]}{K_M + [X]}$, where $V_{\max}$ and K_M are constants, and $[X]$ denotes the concentration of a substrate.

Microscopic and macroscopic parameters.

The kinetic constants k_j, as given in Table 1.1, are used in describing the changes in the count of species, i.e., the changes in the state vector $\mathbf{X_t}$. They are, therefore, referred to as the *microscopic* parameters. Similarly, the propensities in Table 1.1 are referred to as *microscopic* propensities.

In some cases one may be interested in the *concentration* of species, instead of the number of molecules. The concentration of species S_i, c_i, is related to the count of S_i by $c_i = \frac{X_i}{\Omega}$, where Ω is the volume of the compartment in which the reactions take place. The propensities can be written in terms of concentrations by dividing the microscopic propensities by the volume of the compartment, and replacing all molecule numbers by the corresponding concentrations. For example, consider the propensity of the second-order reaction in Table 1.1, $k_j X_i X_j$. The corresponding propensity in terms of concentrations is obtained as

$$\frac{1}{\Omega} k_j (\Omega c_i)(\Omega c_j) = k_j \Omega c_i c_j.$$

The product $k'_j = k_j \Omega$ is called the rate constant, or the *macroscopic parameter*. The macroscopic parameters are usually used in Reaction Rate Equations (see Section 1.3.3), instead of the microscopic rates. The conversion of the microscopic parameters to macroscopic parameters depends on the order of the reaction, and can be done as explained above. Table 1.2 provides the relationship between the macroscopic and microscopic parameters for several reaction types.

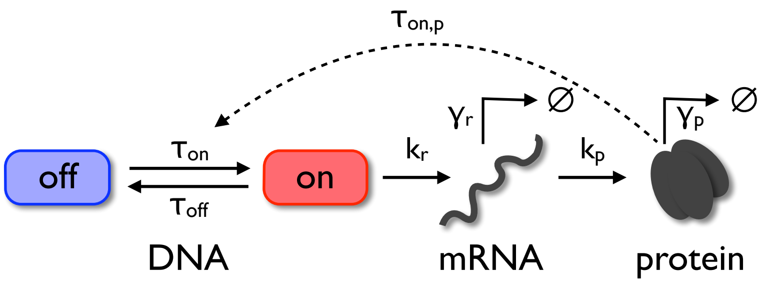

Figure 1.1: Schematic of the three-stage model of gene expression.

Table 1.3: Chemical reactions of the three-stage gene expression example.

Reactions	Microscopic Propensity	Macroscopic Propensity
$R_1:\ \mathrm{DNA_{off}} \overset{\tau_{\mathrm{on}}}{\to} \mathrm{DNA_{on}}$	$\tau_{\mathrm{on}}\ X_{\mathrm{DNA_{off}}}$	$\tau_{\mathrm{on}}\ X_{\mathrm{DNA_{off}}}$
$R_2:\ \mathrm{DNA_{on}} \overset{\tau_{\mathrm{off}}}{\to} \mathrm{DNA_{off}}$	$\tau_{\mathrm{off}}\ X_{\mathrm{DNA_{on}}}$	$\tau_{\mathrm{off}}\ X_{\mathrm{DNA_{on}}}$
$R_3:\ \mathrm{DNA_{on}} \overset{k_r}{\to} \mathrm{DNA_{on}} + \mathrm{mRNA}$	$k_r\ X_{\mathrm{DNA_{on}}}$	$k_r\ X_{\mathrm{DNA_{on}}}$
$R_4:\ \mathrm{mRNA} \overset{\gamma_r}{\to} \emptyset$	$\gamma_r\ X_{\mathrm{mRNA}}$	$\gamma_r\ X_{\mathrm{mRNA}}$
$R_5:\ \mathrm{mRNA} \overset{k_p}{\to} \mathrm{mRNA} + \mathrm{protein}$	$k_p\ X_{\mathrm{mRNA}}$	$k_p\ X_{\mathrm{mRNA}}$
$R_6:\ \mathrm{protein} \overset{\gamma_p}{\to} \emptyset$	$\gamma_p\ X_{\mathrm{protein}}$	$\gamma_p\ X_{\mathrm{protein}}$
$R_7:\ \mathrm{protein} + \mathrm{DNA_{off}} \overset{\tau_{\mathrm{on}}^{\mathrm{p}}}{\to} \mathrm{protein} + \mathrm{DNA_{on}}$	$\tau_{\mathrm{on}}^{\mathrm{p}}\ X_{\mathrm{DNA_{off}}}\ X_{\mathrm{protein}}$	$\Omega\ \tau_{\mathrm{on}}^{\mathrm{p}}\ X_{\mathrm{DNA_{off}}}\ X_{\mathrm{protein}}$

1.2 Example: The three-stage gene expression

In the following, we illustrate the aforementioned notions using an example. We choose the three-stage gene expression example (Shahrezaei and Swain, 2008) and extend it by a feedback loop to introduce a nonlinearity. A schematic of the model is depicted in Figure 1.1. This system will be used in later sections to demonstrate the capabilities of ACME.

This model includes a gene with a promotor switching between on- and off-states. Transcription of mRNA takes place if the promotor is in the on-state, and the transcribed mRNA can be translated into protein. The model also incorporates a protein-induced activation of the promoter which establishes a positive feedback loop. Protein and mRNA are subject to degradation.

The chemical species of this model are:

$$S_1 : \mathrm{DNA_{on}}\ (\text{promoter on-state}), \quad S_2 : \mathrm{DNA_{off}}\ (\text{promoter off-state}), \quad S_3 : \mathrm{mRNA}, \quad S_4 : \mathrm{protein}.$$

The state vector of the system, $\mathbf{X} = (X_{\mathrm{DNA_{off}}}, X_{\mathrm{DNA_{on}}}, X_{\mathrm{mRNA}}, X_{\mathrm{protein}})$, represents the counts of the species. Table 1.3 presents the list of the chemical reactions of this model, together with their propensities.

According to the reactions in Table 1.3, the stoichiometry matrix of the system is given as:

$$\begin{array}{c} \\ X_{\mathrm{DNA_{off}}} \\ X_{\mathrm{DNA_{on}}} \\ X_{\mathrm{mRNA}} \\ X_{\mathrm{protein}} \end{array} \begin{array}{c} \begin{array}{ccccccc} R_1 & R_2 & R_3 & R_4 & R_5 & R_6 & R_7 \end{array} \\ \begin{pmatrix} -1 & 1 & 0 & 0 & 0 & 0 & -1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 \end{pmatrix} \end{array},$$

Table 1.4: Parameter values and initial conditions of the three-stage gene expression example.

τ_{on}	τ_{off}	k_r	γ_r	k_p	γ_p	$\tau_{\text{on}}^{\text{p}}$	Ω	$X_{\text{DNA}_{\text{off}}}(0)$	$X_{\text{DNA}_{\text{on}}}(0)$	$X_{\text{mRNA}}(0)$	$X_{\text{protein}}(0)$
0.3	0.3	10	1	4	1	0.015	1	1	0	0	0

where each row represents the count of a species, and each column represents a reaction.

In this documentation, we use the parameter values and initial conditions given in Table 1.4 to simulate the three-stage gene expression example using different modeling approaches.

1.3 Modeling and simulation approaches

A chemical reaction network, comprising of n_s chemical species, $S_1, \ldots, S_{n_s}$, and n_r chemical reactions, $R_1, \ldots, R_{n_r}$, is described using a continuous-time Markov chain (CTMC) (Norris, 1997). The state of this CTMC is represented by the state vector $\mathbf{X} = (X_1, \ldots, X_{n_s})^T$, where X_i denotes the count of species S_i. As explained in Section 1.1, the state $\mathbf{X}$ changes by the firing of chemical reactions.

The probability of observing the CTMC at a particular state $\mathbf{x} = (x_1, \ldots, x_{n_s})^T$ at time t is denoted by $p(\mathbf{x}|t)$. The time evolution of the probability distribution $p(\mathbf{x}|t)$ is governed by the CME, which is a system of ordinary differential equations (ODEs) for every possible state $\mathbf{x}$:

$$\frac{\partial}{\partial t} p(\mathbf{x}|t) = \sum_{j=1}^{n_r} \Big(a_j(\mathbf{x} - \nu_j) p(\mathbf{x} - \nu_j|t) - a_j(\mathbf{x}) p(\mathbf{x}|t) \Big). \tag{1.1}$$

As solving the CME is mostly infeasible due to the large or infinite number of states $\mathbf{x}$, various approximative methods have been developed. Several methods concentrate on the full distribution $p(\mathbf{x}|t)$ to provide a microscopic description. For mesoscopic and macroscopic descriptions, there exist several methods that focus on representing the solution of the CME in terms of its statistical moments, i.e.,

$$\begin{aligned} \text{mean} \quad & \mathbf{m} = (m_1, \ldots, m_{n_s})^T = \sum_{\mathbf{x}} \mathbf{x} p(\mathbf{x}|t), \\ \text{covariance} \quad & \mathbf{C} = \sum_{\mathbf{x}} (\mathbf{x} - \mathbf{m})(\mathbf{x} - \mathbf{m})^T p(\mathbf{x}|t), \\ \text{higher-order moments} \quad & \mathbf{C}_I = \sum_{\mathbf{x}} (\mathbf{x} - \mathbf{m})^I p(\mathbf{x}|t), \end{aligned} \tag{1.2}$$

with the following product notation:

$$(\mathbf{x} - \mathbf{m})^I := \prod_{i=1}^{n_s} (x_i - m_i)^{I_i}, \tag{1.3}$$

where $I = (I_1, \ldots, I_{n_s})$ is a vector of non-negative integers. The order of the moment $\mathbf{C}_I$ is given by $M = \sum_{i=1}^{n_s} I_i$.

The microscopic, mesoscopic, macroscopic and hybrid methods implemented in ACME, are briefly introduced in the following.

1.3.1 Stochastic Simulation Algorithm

SSAs generate statistically representative sample paths of the CTMC (Gillespie, 1977). A next-reaction SSA method is implemented in ACME. In this method, starting from an initial condition $\mathbf{X_0} = (X_{1,0}, \ldots, X_{n_s,0})^T$

randomly sampled from an initial distribution, the time until the next reaction, and the index of the next reaction to fire are sampled from their corresponding distributions. The state vector $\mathbf{X}$ is then updated by $\mathbf{X}_\tau \to \mathbf{X}_{\tau-} + \nu_j$, where j is the index of the next reaction to fire, and τ is the time of the firing. In this way, sample paths of the stochastic process are generated for the time interval of interest. An estimate to the probability distribution $p(\mathbf{x}|t)$ is then given by the frequency of sample paths that occupy state $\mathbf{x}$ at time t. To estimate the moments of the process, Monte-Carlo integration can be performed. For example, the mean and variance of the process are approximated by

$$\begin{aligned} \mathbf{m}(\mathbf{t}) &= \frac{1}{N}\sum_{k=1}^{N} \mathbf{x_k}(t), \\ \mathbf{C}(\mathbf{t}) &= \frac{1}{N-1}\sum_{k=1}^{N} \Big(\mathbf{x_k}(t) - \mathbf{m}(t)\Big)\Big(\mathbf{x_k}(t) - \mathbf{m}(t)\Big)^T, \end{aligned} \tag{1.4}$$

where N is the number of generated sample paths, and $\mathbf{x_k}(t)$ is the state of the CTMC at time t in the k^{th} sample path. While estimators for probability distribution and moments are unbiased and converge, the sample-sizes required to obtain low-variance estimates are generally large, rendering SSA-based methods computationally demanding.

In the case of time-dependent propensities, ACME offers a modified next-reaction method (Anderson, 2007), which takes into account the exact evolution of the time-dependent propensities in between the reaction firings.

1.3.2 Finite State Projection

To enable a direct approximation of $p(\mathbf{x}|t)$, FSP (Munsky and Khammash, 2006) considers a finite subspace of CME, $\Psi \subset \mathbb{N}_0^{n_s}$, which only contains states with non-negligible probabilities $p(\mathbf{x}|t)$. All other states $\mathbf{x} \notin \Psi$ are lumped into a single sink state which only absorbs probability from states $\mathbf{x} \in \Psi$, but does not return any probability back to the system. In this way, the set of ODEs for the probabilities of states $\mathbf{x} \in \Psi$, derived from the CME, yield a lower bound for $p(\mathbf{x}|t)$. This lower bound, denoted by $\underline{p}(\mathbf{x}|t)$ is the solution of the following ODE system:

$$\forall \mathbf{x} \in \Psi : \frac{\partial}{\partial t}\underline{p}(\mathbf{x}|t) = \sum_{\substack{j=1 \\ \mathbf{x}-\nu_j \in \Psi}}^{n_r} a_j(\mathbf{x}-\nu_j)\underline{p}(\mathbf{x}-\nu_j|t) - \sum_{j=1}^{n_r} a_j(\mathbf{x})\underline{p}(\mathbf{x}|t). \tag{1.5}$$

Growing the state-space of FSP, Ψ, decreases the approximation error at the cost of increased computational complexity. In FSP simulations, the sum of the probabilities remained in the system, $\sum_{\mathbf{x}\in\Psi}\underline{p}(\mathbf{x}|t)$, is an indicator of the approximation error by truncating the state space. For details we refer to Munsky and Khammash (2006).

1.3.3 Reaction Rate Equations

The RRE is the most commonly used modeling approach for biochemical reaction networks. It constitutes a system of ODEs for the time evolution of the mean of the stochastic process in the macroscopic limit. Thus, as copy-numbers increase, the solution of RRE tends towards the true mean of the stochastic process. However for finite system sizes, RRE prediction can be considerably different from the true mean of the process since RRE neglects the stochastic effects.

The RRE equations for the time evolution of the mean of species can be derived by multiplying the stoichiometric matrix by the vector of reaction propensities (see Section 1.1):

$$\frac{\partial \mathbf{m}(t)}{\partial t} = \begin{pmatrix} \frac{\partial m_1(t)}{\partial t} \\ \frac{\partial m_2(t)}{\partial t} \\ \vdots \\ \frac{\partial m_{n_s}(t)}{\partial t} \end{pmatrix} = \begin{pmatrix} \nu_{1,1} & \nu_{1,2} & \cdots & \nu_{1,n_r} \\ \nu_{2,1} & \nu_{2,2} & \cdots & \nu_{2,n_r} \\ \vdots & \vdots & \ddots & \vdots \\ \nu_{n_s,1} & \nu_{n_s,2} & \cdots & \nu_{n_s,n_r} \end{pmatrix} \begin{pmatrix} a_1(\mathbf{m}(t)) \\ a_2(\mathbf{m}(t)) \\ \vdots \\ a_{n_s}(\mathbf{m}(t)) \end{pmatrix}.$$

Here, $m_i(t)$ denotes the mean of species S_i at time t, $\nu_{i,j}$ denotes the stoichiometric coefficient of species S_i in reaction R_j, and $a_j(\mathbf{m}(t))$ denotes the propensity of reaction R_j evaluated at the mean $\mathbf{m}(t)$.

1.3.4 System Size Expansion

For a systematic approximation of the dynamics of mesoscopic systems, the SSE has been introduced (van Kampen, 2007). The SSE is a power series expansion of the CME in the inverse volume of the system. Depending on the order of the SSE, different ODE systems approximating the moments of the CME solution are obtained. The lowest-order approximation for the mean reproduces the aforementioned RRE. For the covariance, the lowest-order approximation yields the well-known linear noise approximation (LNA) (van Kampen, 2007). Higher-order corrections for the mean and covariance yield the effective mesoscopic rate equation (EMRE) (Grima, 2010) and the inverse omega square (IOS) method (Thomas *et al.*, 2013). These methods tend to be more accurate for systems of small and medium volumes (Ramaswamy *et al.*, 2012). For SSE equations, we refer to Grima (2010) and Thomas *et al.* (2013).

1.3.5 Method of Moments

The method of moments (MM) (Engblom, 2006) is conceptually similar to SSE in that it also sets a framework for describing the moments of the solution of CME. Following the definition of moments, a system of ODEs for the exact time evolution of the moments is derived from the CME which constitutes the moment equations.

Given that the reactions are at most bimolecular, the MM equations for the mean and (co)variance can be written as

$$
\begin{aligned}
\frac{\partial m_i(t)}{\partial t} =& \sum_{j=1}^{n_r} \nu_{i,j} \left(a_j(\mathbf{m}(t)) + \frac{1}{2} \sum_{k_1,k_2} \frac{\partial^2 a_j(\mathbf{m}(t))}{\partial x_{k_1} \partial x_{k_2}} C_{k_1k_2}(t) \right), \\
\frac{\partial C_{i_1i_2}(t)}{\partial t} =& \sum_{j=1}^{n_r} \left(\nu_{i_1,j} \sum_k \frac{\partial a_j(\mathbf{m}(t))}{\partial x_k} C_{i_1k} + \nu_{i_2,j} \sum_k \frac{\partial a_j(\mathbf{m}(t))}{\partial x_k} C_{i_2k} \right. \\
& \left. + \nu_{i_1j}\nu_{i_2j} \left(a_j(\mathbf{m}(t)) + \frac{1}{2} \sum_{k_1,k_2} \frac{\partial^2 a_j(\mathbf{m}(t))}{\partial x_{k_1} \partial x_{k_2}} C_{k_1k_2}(t) \right) \right) \\
& + \sum_{j=1}^{n_r} \left(\nu_{i_1j} \sum_{k_1,k_2} \frac{\partial^2 a_j(\mathbf{m}(t))}{\partial x_{k_1} \partial x_{k_2}} C_{i_1k_1k_2}(t) + \nu_{i_2j} \sum_{k_1,k_2} \frac{\partial^2 a_j(\mathbf{m}(t))}{\partial x_{k_1} \partial x_{k_2}} C_{i_2k_1k_2}(t) \right), \\
\frac{\partial C_{i_1,\cdots,i_{n_s}}(t)}{\partial t} =& \sum_{j=1}^{n_r} a_j(\mathbf{m}(t)) \sum_{\substack{l_1,l_2,\cdots,l_{n_s} \\ l_1+l_2+\cdots+l_{n_s} \neq M}} \binom{i_1}{l_1} \cdots \binom{i_{n_s}}{l_{n_s}} \nu_{1,j}^{i_1-l_1} \cdots \nu_{n_s,j}^{i_{n_s}-l_{n_s}} C_{l_1,\cdots,l_{n_s}}(t) \\
& + \sum_{j=1}^{n_r} \sum_k \frac{\partial a_j(\mathbf{m}(t))}{\partial x_k} \sum_{\substack{l_1,l_2,\cdots,l_{n_s} \\ l_1+l_2+\cdots+l_{n_s} \neq M}} \binom{i_1}{l_1} \cdots \binom{i_{n_s}}{l_{n_s}} \nu_{1,j}^{i_1-l_1} \cdots \nu_{n_s,j}^{i_{n_s}-l_{n_s}} C_{l_1,\cdots,l_k+1,\cdots,l_{n_s}}(t) \\
& + \frac{1}{2} \sum_{j=1}^{n_r} \sum_{k_1,k_2} \frac{\partial^2 a_j(\mathbf{m}(t))}{\partial x_{k_1} \partial x_{k_2}} \sum_{\substack{l_1,l_2,\cdots,l_{n_s} \\ l_1+l_2+\cdots+l_{n_s} \neq M}} \binom{i_1}{l_1} \cdots \binom{i_{n_s}}{l_{n_s}} \nu_{1,j}^{i_1-l_1} \cdots \nu_{n_s,j}^{i_{n_s}-l_{n_s}} C_{l_1,\cdots,l_{k_1}+1,\cdots,l_{k_2}+1,\cdots,l_{n_s}}(t) \\
& - \sum_{k=1}^{n_s} i_k \frac{\partial m_k(t)}{\partial t} C_{i_1,\cdots,i_k-1,\cdots,i_{n_s}}(t),
\end{aligned}
\tag{1.6}
$$

where $M = i_1 + i_2 + \cdots + i_{n_s}$ denotes the order of the moment, $C_{ij}(t)$ denotes the covariance of species S_i and S_j, and $C_{i_1k_1k_2}(t)$ denotes the third-order moment of species S_{i_1}, S_{k_1} and S_{k_2}. For details of derivation, we refer to Lee *et al.* (2009). The equations for moments of order M generally depend on moments of order $M+1$ and higher. For example in Equation (1.6), the time evolution of the mean depends on the covariances, and the time evolution of the covariance depends on the third-order moments. Therefore, moment closures are applied yielding a closed set of approximative moment equations.

Moment closure schemes approximate the higher-order moments as functions of lower-order moments by making specific assumptions about the distribution. Commonly used closure techniques include (Singh and Hespanha, 2011)

- Low dispersion closure: Assuming higher-order central moments are zero.
- Mean field closure: Assuming independence of variables (i.e. species concentrations).
- Zero cumulants closure: Assuming the distribution is close to a normal distribution.
- Derivative matching closure: Assuming the distribution is similar to a log-normal distribution.

The accuracy of these closures is problem-specific.

1.3.6 Method of Conditional Moments

A hybrid approach for the approximation of the CME solution is provided by the method of conditional moments (MCM) (Hasenauer *et al.*, 2014). The MCM combines a microscopic description of low copy-number species with a moment-based description of high copy-number species. Since stochastic fluctuations are more dominant for low copy-number species, marginal probability densities for these species are determined. The distributions of high-copy number species are merely described in terms of their moments, conditioned on the state of low-copy number species. If we decompose the state vector $\mathbf{X}_t$ into two vectors $\mathbf{Y}_t$ and $\mathbf{Z}_t$ such that $\mathbf{Y}_t$ represents the state of low copy-number species and $\mathbf{Z}_t$ represents the state of high copy-number species, then the state probability $p(\mathbf{x}|t)$ is written as the joint probability

$$p(\mathbf{x}|t) = p(\mathbf{y}, \mathbf{z}|t) = p(\mathbf{z}|\mathbf{y}, t)p(\mathbf{y}|t), \tag{1.7}$$

where $p(\mathbf{y}|t)$ denotes the probability of $\mathbf{Y}_t = \mathbf{y}$, and $p(\mathbf{z}|\mathbf{y}, t)$ denotes the conditional probability of $\mathbf{Z}_t = \mathbf{z}$ given that $\mathbf{Y}_t = \mathbf{y}$.

Accordingly, the state variables of the MCM are defined as

$$\begin{aligned}
\text{Marginal probabilities of low copy-number species}: \quad & p(\mathbf{y}|t) = \sum_{\mathbf{z}\geq\mathbf{0}} p(\mathbf{y}, \mathbf{z}|t), \\
\text{Conditional means of high copy-number species}: \quad & \mathbf{m}_z(\mathbf{y}, t) = \mathbb{E}_z[\mathbf{Z}|\mathbf{y}, t] = \sum_{\mathbf{z}\geq\mathbf{0}} \mathbf{z}p(\mathbf{z}|\mathbf{y}, t), \\
\text{Higher-order central conditional moments} \quad & \mathbf{C}_{I,z}(\mathbf{y}, t) = \mathbb{E}_z\left[\left(\mathbf{Z} - \mathbf{m}_z(\mathbf{y}, t)\right)^I |\mathbf{y}, t\right] \\
\text{of high copy-number species}: \quad & \qquad = \sum_{\mathbf{z}\geq\mathbf{0}} \left(\mathbf{z} - \mathbf{m}_z(\mathbf{y}, t)\right)^I p(\mathbf{z}|\mathbf{y}, t).
\end{aligned} \tag{1.8}$$

The evolution equations for marginal probabilities and conditional moments are derived from the CME, and form a system of differential algebraic equations (DAEs):

$$
\begin{aligned}
\frac{\partial}{\partial t}p(\mathbf{y}|t) = &\sum_{j=1}^{n_r}\mathbb{E}_z[a_j(\mathbf{y}-\nu_{jy},\mathbf{Z})|\mathbf{y}-\nu_{jy},t]p(\mathbf{y}-\nu_{jy}|t)\\
&-\sum_{j=1}^{n_r}\mathbb{E}_z[a_j(\mathbf{y},\mathbf{Z})|\mathbf{y},t]p(\mathbf{y}|t),\\
p(\mathbf{y}|t)\frac{\partial}{\partial t}m_{i,z}(\mathbf{y},t) = &\sum_{j=1}^{n_r}\mathbb{E}_z[a_j(\mathbf{y}-\nu_{jy},\mathbf{Z})(Z_i-m_{i,z}(\mathbf{y}-\nu_{jy},t))|\mathbf{y}-\nu_{jy},t]p(\mathbf{y}-\nu_{j,y}|t)\\
&+\sum_{j=1}^{n_r}(m_{i,z}(\mathbf{y}-\nu_{jy},t)+\nu_{ij,z})\,\mathbb{E}_z[a_j(\mathbf{y}-\nu_{jy},\mathbf{Z})|\mathbf{y}-\nu_{j,y},t]p(\mathbf{y}-\nu_{j,y}|t)\\
&-\sum_{j=1}^{n_r}\left(\mathbb{E}_z[a_j(\mathbf{y},\mathbf{Z})(Z_i-m_{i,z}(\mathbf{y},t))|\mathbf{y},t]+m_{i,z}(\mathbf{y},t)\mathbb{E}_z[a_j(\mathbf{y},\mathbf{Z})|\mathbf{y},t]\right)p(\mathbf{y}|t)\\
&-m_{i,z}(\mathbf{y},t)\frac{\partial}{\partial t}p(\mathbf{y}|t),\\
p(\mathbf{y}|t)\frac{\partial}{\partial t}\mathbf{C}_{I,z}(\mathbf{y}|t) = &\sum_{j=1}^{n_r}\Bigg(\sum_{0\le k\le I}\binom{I}{k}(\mathbf{m}_z(\mathbf{y}-\nu_{jy},t)-\mathbf{m}_z(\mathbf{y}|t)+\nu_{jz})^{I-k}\\
&\mathbb{E}_z[a_j(\mathbf{y}-\nu_{jy},\mathbf{Z})(\mathbf{Z}-\mathbf{m}_z(\mathbf{y}-\nu_{jy},t))^k|\mathbf{y}-\nu_{jy},t]p(\mathbf{y}-\nu_{jy}|t)\Bigg)\\
&-\sum_{j=1}^{n_r}\mathbb{E}_z[a_j(\mathbf{y},\mathbf{Z})(\mathbf{Z}-\mathbf{m}_z(\mathbf{y},t))^I|\mathbf{y},t]p(\mathbf{y}|t)\\
&-\sum_{\substack{i=1\\ I_i\ge 1}}^{n_{s,z}} I_i\mathbf{C}_{I-e_i,z}(\mathbf{y},t)p(\mathbf{y}|t)\frac{\partial}{\partial t}m_{i,z}(\mathbf{y},t)-\mathbf{C}_{I,z}(\mathbf{y},t)\frac{\partial}{\partial t}p(\mathbf{y}|t).
\end{aligned}
\tag{1.9}
$$

Similar to moment equations, the evolution of lower-order conditional moments can depend on higher-order conditional moments, rendering moment closure necessary. The moment closure techniques introduced in Section 1.3.5 can be used to approximate the higher-order central conditional moments of the high copy-number species. The conditional moments and marginal probabilities can be used to calculate the overall moments as below:

$$
\begin{aligned}
&\text{Mean of low copy-number species}: \bar{m}_{i,y}(t) = \sum_{\mathbf{y}\ge 0} y_i p(\mathbf{y}|t),\\
&\text{Mean of high copy-number species}: \bar{m}_{i,z}(t) = \sum_{\mathbf{y}\ge 0} m_{i,z}(\mathbf{y}|t)p(\mathbf{y}|t),\\
&\text{Higher-order central moments}: \bar{\mathbf{C}}_I(t) = \sum_{y\ge 0}(\mathbf{y}-\bar{\mathbf{m}}_y(t))^{I_y}\sum_{0\le k\le I_z}\binom{I_z}{k}(\mathbf{m}_z(\mathbf{y}|t)-\bar{\mathbf{m}}_z(t))^{I_z-k}\mathbf{C}_{k,z}(\mathbf{y},t)p(\mathbf{y}|t),
\end{aligned}
\tag{1.10}
$$

where $I=(I_y,I_z)$ and $\bar{\mathbf{m}}_y(t)=\left(\bar{m}_{1,y}(t),\ldots,\bar{m}_{n_{s,y},y}(t)\right)$.

This hybrid description can yield an improved approximation accuracy (Hasenauer *et al.*, 2014).

1.4 Sensitivity analysis

FSP, RRE, SSE, MM and MCM yield systems of differential equations. The parameters of differential equations can efficiently be inferred using gradient-based optimization methods (Raue *et al.*, 2013). While gradients can be approximated using finite differences, methods based on sensitivity equations are known to be more robust and computationally more efficient (Raue *et al.*, 2013). Sensitivity equations describe how much an aspect/functional of the process changes in response to changes in parameter values. ACME implements forward and adjoint sensitivity analyses (Hindmarsh *et al.*, 2005) for FSP, RRE, SSE, MM, and forward sensitivity analysis for MCM.

1.4.1 Forward sensitivity equation

Forward sensitivity equation provides the time-dependent sensitivity of the state-variables of the differential equations with respect to the parameters. Consider an n-dimensional ODE system

$$\begin{aligned} &\dot{\mathbf{x}} = f(\mathbf{x},\theta,t), \quad \mathbf{x}(t_0) = \mathbf{x}_0(\theta) \\ &\mathbf{y} = h(\mathbf{x},\theta,t), \end{aligned} \tag{1.11}$$

or an n-dimensional DAE system

$$\begin{aligned} &F(\dot{\mathbf{x}},\mathbf{x},\theta,t) = 0, \quad \mathbf{x}(t_0) = \mathbf{x}_0(\theta), \quad \dot{\mathbf{x}}(t_0) = \dot{\mathbf{x}}_0(\theta) \\ &\mathbf{y} = h(\mathbf{x},\theta,t), \end{aligned} \tag{1.12}$$

where $\mathbf{x} \in \mathbb{R}^n$ denotes the state variables of the ODE/DAE system, $\mathbf{y} \in \mathbb{R}^{n_o}$ denotes the vector of output variables, and θ denotes the set of parameters. The forward sensitivities for these systems are defined as

$$\begin{aligned} \mathbf{S}^x(t) &= \left(\mathbf{s}_1^x(t), \mathbf{s}_2^x(t), \cdots, \mathbf{s}_{n_\theta}^x(t)\right) \in \mathbb{R}^{n \times n_\theta}, \\ \mathbf{s}_i^x(t) &= \frac{\partial \mathbf{x}(t)}{\partial \theta_i}, \quad \text{for } i = 1,2,\cdots,n_\theta, \end{aligned} \tag{1.13}$$

where $\mathbf{s}_i^x(t) \in \mathbb{R}^n$ denotes the sensitivity of the state variables $\mathbf{x}$ with respect to the i^{th} parameter, θ_i.

In the case of the ODE system, the forward sensitivity equations are

$$\begin{aligned} \dot{\mathbf{s}}_i &= \frac{\partial f}{\partial x}\mathbf{s}_i + \frac{\partial f}{\partial \theta_i}, \quad \text{for } i = 1,\cdots,n_\theta, \\ \mathbf{s}_i(t_0) &= \frac{\partial \mathbf{x}_0(\theta)}{\partial \theta_i}. \end{aligned} \tag{1.14}$$

In the case of a DAE system, the forward sensitivity equations are

$$\begin{aligned} &\frac{\partial F}{\partial x}\mathbf{s}_i + \frac{\partial F}{\partial \dot{x}}\dot{\mathbf{s}}_i + \frac{\partial F}{\partial \theta_i} = 0, \quad \text{for } i = 1,\cdots,n_\theta, \\ &\mathbf{s}_i(t_0) = \frac{\partial \mathbf{x}_0(\theta)}{\partial \theta_i}, \quad \dot{\mathbf{s}}_i(t_0) = \frac{\partial \dot{\mathbf{x}}_0(\theta)}{\partial \theta_i}. \end{aligned} \tag{1.15}$$

Assuming that the model possesses n state-variables and n_θ parameters, a system of $n(1+n_\theta)$ differential equations is solved to compute the state sensitivities with respect to all parameters.

The sensitivity of output variables $\mathbf{y}$ is defined as

$$\begin{aligned} \mathbf{S}^y(t) &= \left(\mathbf{s}_1^y(t), \mathbf{s}_2^y(t), \cdots, \mathbf{s}_{n_\theta}^y(t)\right) \in \mathbb{R}^{n_o \times n_\theta}, \\ \mathbf{s}_i^y(t) &= \frac{\partial \mathbf{y}(t)}{\partial \theta_i}, \quad \text{for } i = 1,2,\cdots,n_\theta, \end{aligned} \tag{1.16}$$

where $\mathbf{s}_i^y(t) \in \mathbb{R}^{n_o}$ denotes the sensitivity of the output variables $\mathbf{y}$ with respect to the i^{th} parameter θ_i. Having solved the forward sensitivity equations, the output sensitivities can be computed via

$$\mathbf{s}_i^y = \frac{\partial h}{\partial x}\,\mathbf{s}_i^x + \frac{\partial h}{\partial \theta_i}, \tag{1.17}$$

where $\frac{\partial h}{\partial x} = \left(\frac{\partial h_j}{\partial x_k}\right)_{jk} \in \mathbb{R}^{n_o \times n}$.

1.4.2 Adjoint sensitivity equation

If the sensitivity of few output functionals with respect to many parameters is required, computing the state sensitivities is unnecessarily demanding, and solving adjoint sensitivity equation is more practical (Hindmarsh *et al.*, 2005). In adjoint sensitivity analysis, an adjoint state of size n is defined which is independent of the parameters. The adjoint state is obtained by solving the adjoint ODE system backward in time. The adjoint states can then be used to calculate the sensitivity with respect to any parameter of interest by computing a one-dimensional integral. Therefore, the adjoint sensitivity of each output functional with respect to n_θ parameters roughly requires forward integration of a system of n ODEs, backward integration of a system of n ODEs and calculating n_θ one-dimensional integrals. Thus, in applications with high-dimensional parameter spaces and/or few output functionals, calculating adjoint sensitivities tends to be more computationally advantageous. In parameter estimation, the likelihood function can be defined as the sole output of the system, which enables efficient exploitation of adjoint sensitivity analysis.

Chapter 2

ACME startup

2.1 Installation

To install the toolbox, go to { address of webpage } and download `ACME.tar`. Copy the downloaded tar file to the desired installation path and extract it using the following command:

```
tar xvf ACME.tar
```

ACME folder contains the following directories:

- `ACME/lib`: Includes all the scripts of ACME , structured as follows:
  - `ACME/lib/compilation_tools` contains the scripts for generating MATLAB-based and SUNDIALS-based simulation files.
  - `ACME/lib/moment_equations` contains the scripts for the derivation of moment equations and conditional moment equations.
  - `ACME/lib/CME_tools` contains the scripts for the finite state projection and stochastic simulation algorithms.
  - `ACME/lib/auxiliary` contains the auxiliary scripts used in other routines.
  - `ACME/lib/visualization_tools` contains the visualization routines.
- `ACME/examples`: Includes systems implemented in ACME. The three-stage gene expression example used throughout this documentation can also be found in `ACME/examples/geneExpression`.
- `ACME/doc`: Includes the ACME documentation.

To use ACME, `install_acme.m` must be run at the beginning of each MATLAB session. This step adds the ACME path to the top of MATLAB search path. Alternatively, the ACME path can be permanently added to MATLAB search path, in which case the execution of `install_acme.m` in the beginning of each session will not be necessary.

2.2 Requirements

ACME is installed and tested on Mac OS X 10.9 and 10.10 (?) and Linux (?). However, configuration of the toolbox on Windows is yet to be completed. Therefore, depending on the Windows version, users might experience difficulties compiling the simulation files. While we keep working on this issue, users are encouraged to send us their feedback on configuring ACME.

ACME requires the installation of following software/toolboxes:

- **MATLAB.** ACME is implemented in MATLAB, and therefore all functionalities of ACME require MATLAB software.
- **MATLAB Symbolic Math Toolbox.** ACME extensively uses MATLAB Symbolic Math Toolbox for the derivation of system equations and the compilation of simulation files.
- **SUNDIALS CVODES and IDAS packages.** The compilation of simulation MEX-files in ACME relies on CVODES and IDAS packages. These packages are included in the corresponding wrappers, i.e. `ACME/compilation_tools/cvodewrap` and `ACME/compilation_tools/idawrap`. Therefore, the user does not need to install CVODES and IDAS packages separately.
- **SBML Toolbox.** The usage of `importSBML` (see Section 3.2) requires the installation of SBML Toolbox (Keating *et al.*, 2006).

Chapter 3

Setting up a model in ACME

To simulate a biochemical process in ACME, first the biochemical reaction network has to be specified. The network specification is provided by the user in the form of a *model definition file*. Alternatively, networks specified in the Systems Biology Markup Language (SBML) format can be imported.

3.1 Model definition file

A model definition file is a `.m` file that contains all the necessary specifications of the biochemical reaction network. The model definition file will be referred to as `modelDef.m` in this documentation. `modelDef.m` assembles a `System` structure with the following fields:

- `System.time` is the symbolic variable used to denote time. This symbolic variable is required in handling of time-dependent expressions, e.g., in input functions or reaction propensities.
- `System.compartments` is a string cell array containing the names of the compartments of the chemical reaction network.

```
System.compartments = {'comp_1';'comp_2'; ···}
```

- `System.volumes` is a symbolic or numeric array containing the compartment volumes in the same order as `System.compartments`.

```
System.volumes = [vol_1; vol_2; ···]
```

- `System.state` is a field containing the following information about the chemical species or the *states* of the system:
  - `System.state.variable` is an array of the symbolic variables denoting the chemical species.

```
System.state.variable = [x_1; x_2; x_3; ···]
```

  - `System.state.compartment` is an array specifying the compartment to which each species belongs. This array is defined in the same order as `System.state.variable`.

```
System.state.compartment = {'comp_1'; 'comp_1'; 'comp_2'; ···}
```

– `System.state.name` (optional) is a string array containing the names of species in the same order as `System.state.variable`. By default, the names will be set to the string version of `System.state.variable`.

```
System.state.name = {'species_1'; 'species_2'; 'species_3'; ···}
```

– `System.state.type` (optional) is an array in the order of `System.state.variable` that determines whether a microscopic or a mesoscopic description for each species should be used. This field is only necessary for MCM simulations. For microscopic descriptions, `stochastic` type and for mesoscopic descriptions `moment` type should be specified. If unspecified, all types will be set to `moment` by default.

```
System.state.type = {'stochastic'; 'moment'; 'moment'; ···}
```

– `System.state.mu0` is a symbolic or numeric vector of the initial values of the mean of species in the same order as `System.state.variable`.

`System.state.mu0 = [`m_{0_1}`;` m_{0_2}`;` m_{0_3}`; ···]`

Attention! The initial conditions should be provided in *molecule numbers* and not in concentrations.

– `System.state.C0` is a symbolic or numeric vector of the initial values of the covariances of species. If unspecified, all covariances will be set to zero by default.

`System.state.C0 = [`$C_{0_{11}}$`;` $C_{0_{12}}$`;` $C_{0_{13}}$`; ···]`

Attention! For the correct ordering of the covariances in `System.state.C0` see chapter?.

Attention! The initial conditions should be provided in $(\textit{molecule numbers})^2$ and not in $(\text{concentrations})^2$.

– The following fields are only required for FSP and MCM simulations, and are used to construct the state space of FSP or MCM:

  ∗ `System.state.xmin` specifies the lower bound on the count of species.

`System.state.xmin = [`$x_{\texttt{min}_1}$`;` $x_{\texttt{min}_2}$`;` $x_{\texttt{min}_3}$`; ···]`

  ∗ `System.state.xmax` specifies the upper bound on the count of species.

`System.state.xmax = [`$x_{\texttt{max}_1}$`;` $x_{\texttt{max}_2}$`;` $x_{\texttt{max}_3}$`; ···]`

  ∗ `System.state.constraint` is a function that defines any constraints that should be imposed on the state of the system. The output of this function should be a logical value (true or false).

```
System.state.constraint = @(x) f(x)
```

– The specified lower and upper bounds and the constraint are used to construct the state space of FSP/MCM simulations. More specifically, the state space of FSP is defined as:

$$\forall \mathbf{x} \in \mathbb{N}_0^{n_s} : \mathbf{x}_{\min} \leq \mathbf{x} \leq \mathbf{x}_{\max} \ \wedge \ f(x) \qquad (3.1)$$

Similarly, the state space of the *stochastic states* of MCM simulation are defined in this way (see Section 1.3.6).

- `System.parameter` contains the following information about the parameters of the system, e.g., kinetic rates. KAPPA!!!!

  Attention! It is assumed that all the parameters are macroscopic. All microscopic parameters should be manually converted to macroscopic parameters by rescaling with respect to the compartment volumes. For more information see Section 1.1.

  - `System.parameter.variable` is a symbolic array including the variables that denote the parameters of the system, e.g., the kinetic rates.

```
System.parameter.variable = [k_1; k_2; k_3; ···]
```

  - `System.parameter.name` (optional) is a string array of the parameter names in the same order as `System.parameter.variable`. By default, the parameter names are set to the string version of `System.parameter.variable`.

```
System.parameter.name = {'par_1'; 'par_2'; 'par_3'; ···}
```

- `System.reaction` is a field specifying the chemical reactions of the system, containing the following subfields:

  - `System.reaction(j).educt` is a symbolic vector of the reactants of the j^{th} reaction. Each species, S_i, should be repeated $\nu^-_{i,j}$ times where $\nu^-_{i,j}$ is the reactant stoichiometric coefficient of S_i in the j^{th} reaction.

```
System.reaction(j).educt = [x_1, ···]
```

  - `System.reaction(j).product` is a symbolic vector of the products of the j^{th} reaction. Each species, S_i, should be repeated $\nu^+_{i,j}$ times where $\nu^+_{i,j}$ is the product stoichiometric coefficient of S_i in the j^{th} reaction.

```
System.reaction(j).product = [x_2, ···]
```

  - `System.reaction(j).propensity` is the symbolic expression of the microscopic propensity of the j^{th} reaction.

```
System.reaction(j).propensity = g(k,x)
```

  - `System.reaction(j).MacroscopicPropensity` is the symbolic expression of the macroscopic propensity of the j^{th} reaction.

```
System.reaction(j).MacroscopicPropensity = h(k,x)
```

  Attention! Some of the simulation methods implemented in ACME rely on the microscopic propensity and some rely on the macroscopic rate. However, only one of the `propensity` and `MacroscopicPropensity` fields should be provided by the user. The unspecified field will then be automatically added. More information on the definition of microscopic and macroscopic rates, and the automatic conversion of the two into each other can be found in Section 1.1.

- `System.output` is a field that includes information about the observables of the system, or any functions of the states of the system.
  - `System.output.variable` is an array of the symbolic variables used to denote the output variables.

```
System.output.variable = [y_1; y_2; ···]
```

  - `System.output.name` (optional) is a string array containing the names of outputs in the same order as `System.output.variable`. By default, the names will be set to the string version of `System.output.variable`.

```
System.output.name = {'observable_1'; 'observable_2'; '···}
```

  - `System.output.function` is an array containing the symbolic expressions of the output functions that correspond to the output variables. The output functions can depend on the abundance of species, **x**, and the parameters **k**.

```
System.output.function = [fy_1(k,x); fy_2(k,x); ···]
```

- `System.input` is a field that includes information about the inputs to the chemical reaction network such as stimuli.
  - `System.input.variable` is an array of the symbolic variables used to denote the input variables.

```
System.input.variable = [u_1; ···]
```

  - `System.input.name` (optional) is a string array containing the names of inputs in the same order as `System.input.variable`. By default, the names will be set to the string version of `System.input.variable`.

```
System.input.name = {'stimulus_1'; ···}
```

  - `System.input.function` is an array containing the symbolic expressions of the input functions that correspond to the input variables.

```
System.input.function = [fu_1(t); ···]
```

The unspecified fields and default settings will then be added to the `System` structure using `completeSystem.m` function. This step is automatically done in ACME as explained in Chapter 4. The `System` structure defined in this way will have all the necessary information for simulating the reaction network in ACME.

3.1.1 Example: The three-stage gene expression

The `modelDef.m` file for the three-stage gene expression example is shown below. This file (`modelDef_geneExpression.m`) can be found in the examples provided in the toolbox .

```
%% MODEL DEFINITION
% Definition of symbolic variables:
syms DNA_off DNA_on mRNA Protein %  species
syms scaledProtein % observables
syms tau_on tau_off k_m gamma_m k_p gamma_p tau_on_p % kinetic rates
syms Omega % volume
syms scaleP offsetP % scaling parameters for the observable
syms r0 % initial count of mRNA
syms time % time

% Definition of general fields:
System.time = time;
System.compartments = {'cell'};  % The name of the compartment, e.g., cytoplasm
System.volumes = [Omega];      % The volume of the compartment

% Definition of state field:
System.state.variable       = [DNA_off; DNA_on;  mRNA; Protein];
System.state.compartment    = { 'cell'; 'cell';'cell';  'cell'};
System.state.name           = {'DNA_{off}' ; 'DNA_{on}' ;  'mRNA';'Protein'};
System.state.type           = {'stochastic';'stochastic';'moment'; 'moment'};
System.state.xmin           = [      0     ;      0     ;   0    ;    0    ];
System.state.xmax           = [      1     ;      1     ;   40   ;    150  ];
System.state.mu0            = [      1     ;      0     ;   r0   ;     0   ];
System.state.C0         = sym(zeros(System.state.number*(System.state.number+1)/2,1));
System.state.constraint         = @(x) ((x(1)+x(2)) == 1);

% Definition of parameters field:
System.parameter.variable   = [ tau_on; tau_off; k_m ; gamma_m; k_p ; gamma_p; tau_on_p;
                                scaleP; offsetP; r0];
System.parameter.name   = {'\\tau_{on}'; '\\tau_{off}'; 'k_m'; '\\gamma_m'; 'k_p'; '\\gamma_p';
                           '\\tau_{on,p}'; 'scale_p'; 'offset_p'; 'r_0'};

% Definition of constant parameters field:
System.kappa.variable = [Omega];

% Definition of reactions:
% (R1)
System.reaction(1).educt        = DNA_off;
System.reaction(1).product      = DNA_on;
System.reaction(1).propensity   = tau_on*DNA_off;
% (R2)
System.reaction(2).educt        = DNA_on;
System.reaction(2).product      = DNA_off;
System.reaction(2).propensity   = tau_off*DNA_on;
% (R3)
System.reaction(3).educt        = DNA_on;
System.reaction(3).product      = [mRNA,DNA_on];
System.reaction(3).propensity   = k_m*DNA_on;
% (R4)
System.reaction(4).educt        = mRNA;
System.reaction(4).product      = [];
System.reaction(4).propensity   = gamma_m*mRNA;
% (R5)
System.reaction(5).educt        = mRNA;
System.reaction(5).product      = [Protein,mRNA];
System.reaction(5).propensity   = k_p*mRNA;
% (R6)
System.reaction(6).educt        = Protein;
System.reaction(6).product      = [];
System.reaction(6).propensity   = gamma_p*Protein;
% (R7)
System.reaction(7).educt        = [DNA_off,Protein];
System.reaction(7).product      = [DNA_on ,Protein];
System.reaction(7).propensity   = tau_on_p*Protein*DNA_off;

System.output.variable  = [scaledProtein ];
System.output.name      = {'scaledProtein'};
System.output.function  = [offsetP + scaleP * Protein];
```

3.2 SBML Import

If the SBML specification of the reaction network is available, it is not necessary to manually create a `modelDef.m` file. Instead, ACME creates the `modelDef.m` by using an automatic SBML import, `importSBML.m`:

```
System = importSBML('model_name')
```

where `model_name` is a string standing for the name of the corresponding SBML file.The generated `System` structure array can be altered in MATLAB by overwriting individual fields.

3.2.1 Example: The three-stage gene expression

The three-stage gene expression model has been specified in SBML, and stored in `GeneExpressionCopasi.xml`. The following command then automatically generates a `modelDef.m` file for this example, named `modelDef_GeneExpressionCopasi.m`:

```
System = importSBML('GeneExpressionCopasi')
```

It also returns the `System` structure as the output. The `modelDef.m` generated in this way can be found in the examples provided in the ACME toolbox.

Chapter 4

Compiling Simulation Files

In this chapter, we demonstrate how ACME can be used to derive the governing equations for the method of interest, and to compile simulation MEX-files.

4.1 Compilers for CVODES and IDAS

After creating the `modelDef.m` file, one of the following commands can be used to generate the system of equations and compile simulation files:

- For generating the C code for the simulation of ODE systems, e.g., FSP, RRE, SSE, and MM the `genSimFile` command is used:

```
genSimFile(modelName,modelDefName,method)
```

- For generating the C code for the simulation of DAE systems, e.g., MCM the `genSimFileIDA` command is used:

```
genSimFileIDA(modelName,modelDefName,method)
```

- For generating the C code for the simulation of ODE systems, e.g., FSP, RRE, SSE, and MM, together with the evaluation of an objective function, the `genSimFileLLH` command is used:

```
genSimFileLLH(modelName,modelDefName,method)
```

where

- `modelName` is a string providing the name of the model. This name will be used in naming the simulation files.
- `modelDefName` is the name of the `modelDef.m` file.
- `method` is the selected modeling approach. The possible options for `method` are presented in Section 4.2.

The three commands above consist of three main functions, `completeSystem()`, `genmexp()` and one of the wrappers for SUNDIALS solvers: `cvodewrap`, `idaswrap`, and `llhwrap`.

completeSystem()

This function is used to add the unspecified and default fields to the `System` structure. Most importantly, the interconversion of microscopic propensities and macroscopic rates is performed in this step.

```
System = completeSystem(System)
```

genmexp()

This function is used to derive the system of governing equations for the specified modeling approach. Executing this command also creates an m-file named `method_modelName_syms`. This file contains the symbolic representation of the system, including the equations and corresponding initial conditions.

Additionally, an m-file is created for the numerical simulation of the system using MATLAB `ode15s` solver, named `method_modelName_matlab`. The generation of this intermediate simulation file is optional, and is meant for simplicity in cases where the user wishes to perform numerical simulations using MATLAB. However, the ACME simulations do not rely on this m-file.

```
genmexp(modelName,modelDefName,method)
```

where the input arguments are the same as those in `genSimFile()`.

cvodewrap()

In case the governing equations are a system of ODEs, this function is used to compile the MEX-files used for numerical simulation, named `modelName`. The compilation of this MEX-file relies on the SUNDIALS CVODES package, and therefore the corresponding CVODES options can be specified as inputs (see Section 5.1 for more details). Also, a MATLAB interface for calling this MEX-file with the appropriate options and input arguments is generated, named `simulate_modelName`. This m-file is the main function that should be used for numerical simulation (see Section 5.1).

```
cvodewrap(modelName,method_modelName_syms)
```

where `method_modelName_syms` stands for the name of the m-file containing the symbolic representation of the governing equations.

llhwrap()

In case the governing equations are a system of ODEs, and the user wishes to calculate an objective function with additive normally distributed measurement noise `llhwrap` can be used to compile the corresponding MEX-file, named `modelName`. The compilation of this MEX-file relies on the SUNDIALS CVODES package, and therefore the corresponding CVODES options can be specified as inputs (see Section 5.1 for more details). Also, a MATLAB interface for calling this MEX-file with the appropriate options and input arguments is generated, named `simulate_modelName`. This m-file should be used to run the numerical simulation using MEX files (see Section 5.1).

```
llhwrap(modelName,method_modelName_syms)
```

where `method_modelName_syms` stands for the name of the m-file containing the symbolic representation of the governing equations.

idawrap()

In case the governing equations are a system of DAEs, this function is used to compile the MEX-files used for numerical simulation, named `modelName`. The compilation of this MEX-file relies on the SUNDIALS IDAS package, and therefore the corresponding IDAS options can be specified as inputs (see Section 5.1 for more details). Also, a MATLAB interface for calling this MEX-file with the appropriate options and input arguments is generated, named `simulate_modelName`. This m-file is the main function that should be used for numerical simulation (see Section 5.1).

```
idawrap(modelName,method_modelName_syms)
```

where `method_modelName_syms` stands for the name of the m-file containing the symbolic representation of the governing equations.

All the above three steps are contained in `genSimFile`, `genSimFileIDA` and `genSimFileLLH`, and therefore the user only needs to call one of these functions to generate all the necessary simulation files.

4.2 Selection of modeling approach

The possible options for modeling approaches include different orders of SSE (Section 1.3.4), different orders of MM (Section 1.3.5), different orders of MCM (Section 1.3.6), and FSP (Section 1.3.2).

4.2.1 System Size Expansion

To specify different orders of SSE as the modeling approach, the `method` is set to the following values:

Modeling Approach	`method`
SSE 1st-order (RRE)	`'RRE'`
SSE 2nd-order (LNA)	`'LNA'`
SSE 3rd-order (EMRE)	`'EMRE'`
SSE 4th-order (IOS)	`'IOS'`

The system of SSE ODEs are then derived for the state variables. In addition, equations for the calculation of the statistical moments based on the state variables are derived. Also, the equations for the mean and the variance of the output variables are derived and presented in the `method_modelName_syms` file. The initial conditions `System.state.mu0` provided in the `modelDef.m` file are used as the initial conditions for the mean of species. The initial conditions for the rest of the state variables are set to zero.

Attention! The SSE equations can only be derived for the concentration of species. However, the initial conditions in `modelDef.m` should be given in molecule numbers. The conversion of the initial conditions to concentrations is automatically done when generating SSE equations.

4.2.2 Method of Moments

To specify different orders of MM as the modeling approach, the `method` should be given in the following form:

```
'MEC_XO_MC_YO_SC_R'
```

where

- `MEC` stands for the "central moment equations", and should be kept unchanged.

- `XO` specifies the truncation order for the moment equations of the state variables.
- `MC` specifies the moment closure scheme (see Section 1.3.5). Any of the following options can be chosen for the closure scheme:
  - `LD` for low dispersion closure technique.
  - `ZC` for zero cumulants closure technique.
  - `MF` for mean field closure technique.
  - `DM` for derivative matching closure technique.
- `YO` specifies the order of the output moments, and can be smaller or equal to the truncation order `XO`. This value does not change the truncation order of the moment equations and merely determines which moments of the output variables should be calculated based on the moments of the state variables.
- `SC` specifies the scale of the equations, which can be either of the following:
  - `a` specifies that the moment equations for the molecule numbers of species should be derived.
  - `c` specifies that the moment equations for the concentration of species should be derived.

The system of ODEs are derived for the first `XO` moments of the species. In addition, equations for the calculation of the first `YO` moments of the outputs are derived and presented in the `MEC_XO_MC_YO_SC_R_modelName_syms` file. The initial conditions `System.state.mu0` provided in the `modelDef.m` file are used as the initial conditions for the mean of species. The initial conditions `System.state.C0` provided in the `modelDef.m` file are used as the initial conditions for the (co)variances of species. The initial conditions for the rest of the state variables are set to zero.

Attention! Regardless of the scale of the moment equations `SC`, the initial conditions should be provided in molecule numbers. If necessary, the conversion to concentration scale is automatically performed when generating MM equations.

4.2.3 Method of Conditional Moments

To specify different orders of MCM as the modeling approach, the `method` should be given in the following form:

```
'CMEC_XO_MC_YO_SC'
```

where

- `CMEC` stands for the "central conditional moment equations", and should be kept unchanged.
- `XO` specifies the truncation order for the conditional moment equations.
- `MC` specifies the moment closure scheme (see Section 1.3.6). Any of the following options can be chosen for the closure scheme:
  - `LD` for low dispersion closure technique.
  - `ZC` for zero cumulants closure technique.
  - `MF` for mean field closure technique.
  - `DM` for derivative matching closure technique.
- `YO` specifies for the order of the output moments, and can be smaller or equal to the truncation order `XO`. This value does not change the truncation order of the conditional moment equations and merely determines which moments of the output variables should be calculated based on the state variables.

- SC specifies the scale of the equations, which can be either of the following:
  - a specifies that the conditional moment equations for the molecule numbers of species should be derived.
  - c specifies that the conditional moment equations for the concentration of species should be derived.

The system of DAEs are derived for the marginal probabilities of 'stochastic' species and the first XO conditional moments of the 'moment' species. In addition, equations for the calculation of the first YO moments of the outputs are derived and presented in the CMEC_XO_MC_YO_SC_modelName_syms file.

Attention! Regardless of whether the output functions include 'stochastic' species, the outputs are always given in terms of their moments, ad no marginal probabilities for the outputs are calculated.

The initial conditions System.state.mu0 provided in the modelDef.m file are used as the initial conditions for

- (i) the number of molecules for the 'stochastic' species, and
- (ii) the conditional mean of species for the 'moment' species.

Therefore, the *stochastic state* that corresponds to the initial molecule numbers of the 'stochastic' species is given an initial marginal probability of 1. The rest of the *stochastic states* will have a zero initial marginal probability.

The initial conditions System.state.C0 provided in the modelDef.m file are used as the initial conditions for the (co)variances of 'moment' species conditioned on the *stochastic state* with initial probability of 1. The initial conditions for the rest of the state variables is set to zero.

Attention! Ensure that System.state.C0 is of the same size as the (co)variances of 'moment' species, and *not* all species.

Attention! Regardless of the scale of the conditional moment equations SC, the initial conditions should be provided in molecule numbers. If necessary, the conversion to concentration scale is automatically performed when generating MCM equations.

4.2.4 Finite State Projection

To specify FSP as the modeling approach, the method is simply set to 'FSP'. The state space of FSP is constructed by considering all combinations of the species counts that are

- in the interval [System.state.xmin, System.state.xmax], and
- satisfy the constraints defined by System.state.constraint.

The system of FSP ODEs for the probabilities of the state variables is derived. In addition, the equations for the calculation of the 1st and 2nd-order moments of the output variables are derived and presented in the FSP_modelName_syms file.

The initial conditions System.state.mu0 provided in the modelDef.m file are used as the initial conditions for the initial molecule numbers of species. Therefore, the FSP state that corresponds to the provided initial molecule numbers is given an initial probability of 1. The rest of the states will have a zero initial probability. The initial conditions System.state.C0 provided in the modelDef.m file are not used in FSP simulation.

Attention! The FSP equations can only be derived for the number of molecules of species.

4.2.5 Symbolic representation

As explained in the beginning of this Chapter, a symbolic representation of the derived system of equations, together with the initial conditions is generated entitled `method_modelName_syms.m`. This file is the basis for compiling the simulation files in later steps. The user can consult this file to easily access the equations and initial conditions.

4.2.6 MATLAB-based simulation file

In addition, a simulation file entitled `method_modelName_matlab.m` is generated that uses MATLAB `ode15s` solver for the numerical simulation of the system:

```
[tout, x, y] = method_modelName_matlab(t, θ)
```

with the input arguments:

- `t` denoting the time vector for which the simulation is to be performed,
- θ denoting the vector of parameter values.

And output arguments:

- `tout` denoting the time vector for which the simulation results are given,
- `x` denoting the simulation results for the state variables. It is a matrix in which each column is a state variable and each row is a time point.
- `y` denoting the simulation results for the output variables. It is a matrix in which each column is an output variable and each row is a time point.

The solver options for `ode15s` can be directly set by the user in the `method_modelName_matlab.m`. (should be made as an input arg?)

This m-file can be used to perform numerical simulations if SUNDIAL solvers CVODES and IDAS are not available, or if the user does not wish to compile MEX-files. However, as far as the computational efficiency is concerned, the usage of this m-file for numerical simulation is not recommended. Unlike the simulation based on MEX-files, the simulation based on m-files does not offer the calculation of sensitivity equations.

4.2.7 Other formats

In addition to the aforementioned files, a model definition file in the format required by Data2Dynamics software (Raue *et al.*, 2015) is generated.

4.3 Compilation of simulation MEX-files

ACME uses wrappers for SUNDIALS CVODES and IDAS packages to compile MEX-files for the numerical simulation of the system and the sensitivity equations. In case of a system of ODEs, the following command

```
cvodewrap(modelName,method_modelName_syms)
```

compiles a MEX-file (`modelName`) and the corresponding MATLAB interface (`simulate_modelName.m`) that can be called by the user (see Section 5.1).

Similarly, in the case of a system of DAEs, the following command

```
idawrap(modelName,method_modelName_syms)
```

compiles a MEX-file and the corresponding MATLAB interface that can be called by the user (see Section 5.1). Using these commands, the forward sensitivity equations (`cvodewrap`, `llhwrap` and `idawrap`) and adjoint sensitivity equations (`cvodewrap` and `llhwrap`) with respect to all the non-constant parameters are derived (see Section 3.1). However, when carrying out numerical simulations, the user can specify which sensitivity equations should be solved (see Section 5.2).

4.4 Example: The three-stage gene expression

In the following an excerpt of code to generate and compile the simulation files for the three-stage gene expression example using IOS is given:

```
modelDefName = 'modelDef_geneExpression'; % The name of the model definition file
modelName = 'geneExpressionIOS'; % The name that will be used for naming the simulation files
method = 'IOS'; % Specifying the modeling approach
genSimFile(modelName,modelDefName,method)
```

The generated files, i.e., `IOS_geneExpressionIOS_syms`, `IOS_geneExpressionIOS_matlab`, `geneExpressionIOS`, and `simulate_geneExpressionIOS` can be looked up in the examples provided in the toolbox.

In order to compile simulation files using other modeling approaches, only the line specifying the `method` needs to change. For example, to use the 3rd-order MM (`MEC_3`) with low dispersion closure (_LD) and 2nd-order outputs (_2) for the calculation of the concentration of species ((_c)), the following should be specified:

```
method = 'MEC_3_LD_2_c'; % Specifying the modeling approach
```

Similarly, to use the 2rd-order MCM (CMEC_2) with zero cumulants closure (_ZC) and 2nd-order outputs (_2) for the calculation of the molecule numbers of species (_a), the following should be specified:

```
method = 'CMEC_2_ZC_2_a'; % Specifying the modeling approach
```

Chapter 5

Numerical simulation and sensitivity analysis

The compiled simulation files can be used for numerical simulation and sensitivity analysis using arbitrary parameter values.

5.1 Numerical simulation of the system

The numerical simulation can be simply carried out for given parameter values and time vector (what about kappa?) by using the following command:

```
[status, tout, x, y] = simulate_modelName(t,θ)
```

with the input arguments:

- `t` denoting the time vector for which the simulation is to be performed,
- θ denoting the vector of parameter values.

And output arguments:

- `status` being a flag indicating whether the numerical simulation was successful or not. This flag is returned by CVODES and IDAS (check!) solvers, and can have the following values:

`status`	CVODES flag
0	CV_SUCCESS
1	CV_TSTOP_RETURN
2	CV_ROOT_RETURN
99	CV_WARNING
-1	CV_TOO_MUCH_WORK
-2	CV_TOO_MUCH_ACC
-3	CV_ERR_FAILURE
-4	CV_CONV_FAILURE
-5	CV_LINIT_FAIL
-6	CV_LSETUP_FAIL
-7	CV_LSOLVE_FAIL
-8	CV_RHSFUNC_FAIL
-9	CV_FIRST_RHSFUNC_ERR
-10	CV_REPTD_RHSFUNC_ERR
-11	CV_UNREC_RHSFUNC_ERR
-12	CV_RTFUNC_FAIL

Simply put, all status values smaller than zero indicate that the integration of the ODE/DAE system failed. Precise interpretation of the CVODES and IDAS flags can be found in the corresponding. documentations (Hindmarsh *et al.*, 2005).

- `tout` denoting the time vector for which the simulation results are given.
- `x` denoting the simulation results for the state variables. It is a matrix in which each column is a state variable and each row is a time point.
- `y` denoting the simulation results for the output variables. It is a matrix in which each column is an output variable and each row is a time point.

Attention! The meaning of state and output variables (`x` and `y`) is specific to the selected modeling approach (see Section 4.2).

This command will carry out the numerical simulation with the default solver options. The options are documented in the `simulate_modelName.m` file. User-specified solver options can be given as additional input arguments:

```
[status, tout, x, y] = simulate_modelName(t, θ, κ, options)
```

Most of the CVODES/IDAS options can be provided. For example, to specify the absolute and relative tolerances and the maximum number of integration steps the following fields in the `options` structure can be set:

```
options.cvodes_atol, options.cvodes_rtol, options.cvodes_maxsteps
```

For a complete list of solver options, we refer the user to the documentations in the `simulate_modelName.m` file.

Example: The three-stage gene expression

Using the MEX-file compiled in Section 4.4, the 2nd-order conditional moment equations for the three-stage gene expression example can be solved for given parameter values and time vector as follows:

```
theta = [0.3;0.3;10;1;4;1;0.015;1;0;0]; % Parameter values
kappa = 1; % Fixed parameters (Omega)
```

```
t = linspace(0,100,500); % Time vector
[status_MCM,tout_MCM,x_MCM,y_MCM] = simulate_geneExpressionMCM(t,theta,kappa);
```

where

- x_MCM is a matrix containing the simulation results for the marginal probabilities of DNA_off and DNA_on, and the 1st and 2nd-order conditional moments of mRNA and Protein molecule numbers.
- y_MM is a matrix containing the simulation results for the 1st and 2nd-order moments of the output variable, i.e. the sum of mRNA and Protein molecule numbers.

Similarly, to obtain the MM simulation results using the MEX-file compiled in Section 4.4 with specific solver accuracy the following code can be used:

```
theta = [0.3;0.3;10;1;4;1;0.015;1;0;0]; % Parameter values
t = linspace(0,100,500); % Time vector
kappa = 1; % Fixed parameters (Omega)
options.cvodes_atol = 1e-8; % Specifying the absolute tolerance of the numerical solver
options.cvodes_rtol = 1e-8; % Specifying the relative tolerance of the numerical solver
[status_MM,tout_MM,x_MM,y_MM] = simulate_geneExpressionMM(t,theta,kappa,options);
```

- x_MM is a matrix containing the simulation results for the 1st, 2nd and 3rd-order moments of species, i.e. DNA_off, DNA_on, mRNA and Protein concentrations.
- y_MM is a matrix containing the simulation results for the 1st and 2nd-order moments of the output variable, i.e. the sum of mRNA and Protein concentrations.

5.2 Sensitivity analysis

More output arguments can be appended to the command above to obtain the sensitivities:

```
[status, tout, x, y, sx, sy] = simulate_modelName(t,θ,...)
```

where

- sx denotes the sensitivity of state variables with respect to parameters. It is a 3-dimensional matrix in which the first dimension corresponds to time points, second dimension corresponds to state variables and third dimension corresponds to parameters.
- sy denotes the sensitivity of output variables with respect to parameters. It is a 3-dimensional matrix in which the first dimension corresponds to time points, second dimension corresponds to output variables and third dimension corresponds to parameters.

Attention! The meaning of state and output variables (x and y) is specific to the selected modeling approach (see Section 4.2).

Also, the following setting specifies that the sensitivity analysis should be performed:

```
options.sensi = 1
```

By default, ACME calculates the sensitivities with respect to all non-constant parameters (see Section 3.1). However, the user can specify the indices of parameters for which the sensitivity equations are to be solved:

```
options.sens_ind = par_ind
```

where `par_ind` is the specified list of parameter indices.

For sensitivity calculation, either forward or adjoint sensitivity equations can be used.

5.2.1 Forward sensitivity analysis

The simulation files generated using `cvodewrap` and `idawrap` by default use forward sensitivity analysis to calculate sensitivities of state and output variables. The sensitivity method can be set to forward sensitivity analysis by the following field in the `options` structure:

```
options.sensi_meth = 1
```

In this case, first the forward sensitivity equations for the state variables are solved, and then the sensitivity of the output variables are calculated based on the state sensitivities.

A list of options specific to the calculation of forward sensitivities can be looked up in the `header documentation` of `simulate_modelName.m`.

Example: The three-stage gene expression

To solve the forward sensitivity equations for the three-stage gene expression example using IOS simulation (MEX-file compiled in Section 4.4) with arbitrary settings, the following code can be used:

```
theta = [0.3;0.3;10;1;4;1;0.015;1]; % Parameter values
t = linspace(0,100,500); % Time vector
kappa = []; % Fixed parameters
options.sensi = 1; % To enable sensitivity analysis
% Specifying forward sensitivity analysis as the sensitivity calculation method
options.sensi_meth  = 1
% Indices of the parameters for which the sensitivities are to be calculated.
options.sens_ind = 1:7;
[status_IOS,tout_IOS,x_IOS,y_IOS,sx_IOS,sy_IOS] =
                                        simulate_geneExpressionIOS(t,theta,kappa,options);
```

where

- `x_IOS` is a 500×48 matrix containing the simulation results for the IOS states.
- `y_IOS` is a 500×2 matrix containing the simulation results for the 1st and 2nd-order moments of the output variable, i.e. the sum of `mRNA` and `Protein` concentrations.
- `sx_IOS` is a $500 \times 48 \times 7$ matrix containing the simulation results for the sensitivity of IOS states with respect to the first 7 parameters.
- `sy_IOS` is a $500 \times 2 \times 7$ matrix containing the simulation results for the sensitivity of output variables, i.e., the mean and the variance of the sum of `mRNA` and `Protein` concentrations, with respect to the first 7 parameters.

5.2.2 Adjoint sensitivity analysis

In `cvodewrap` and `llhwrap`, the sensitivity method can be set to adjoint sensitivity analysis by the following field in the `options` structure:

```
options.sensi_meth = 2
```

In this case, *merely* the adjoint sensitivity equations for the output variables are solved. Therefore the output argument `sx` is returned as a zero matrix. When using `cvodewrap`, `sy` will contain the sensitivity of output variables (see Section 4.2) with respect to the parameters. In `llhwrap`, however, `sy` will contain the sensitivity of the objective function with respect to parameters (see Section **??**).

A list of options specific to the calculation of adjoint sensitivities can be looked up in the `header documentation` of `simulate_modelName.m`.

Example: The three-stage gene expression

To calculate the adjoint sensitivity equations for the three-stage gene expression example using IOS simulation (MEX-file compiled in Section 4.4) with arbitrary settings, merely the following line in the aforementioned code should be changed:

```
% Specifying adjoint sensitivity analysis as the sensitivity calculation method
options.sensi_meth  = 2
```

In this case, `sx` will be a zero matrix, and `sy` will contain the adjoint sensitivities of the output variables, i.e., the mean and the variance of the sum of `mRNA` and `Protein` concentrations.

Chapter 6

SSA simulation

6.1 Description

In addition to all the modeling approaches that yield a system of differential equations (see Section 4.2), ACME offers SSA simulations (Gillespie, 1977). To perform the SSA simulation, firstly the `System` structure should be assembled by evaluating the `modelDef.m` file, and prepared for SSA simulations by the command below:

```
System = completeSystemSSA(System)
```

Using the following command, ACME generates statistically representative realizations of the state of the biochemical reaction network:

```
[X,Y,mX,mY,CX,CY] = simulate_SSA(t,θ,System,N,options)
```

where

- `t` stands for a vector of time points at which the SSA results are to be stored.
- `θ` stands for the parameter values used for the SSA simulation.
- `System` stands for the `System` structure returned by `completeSystemSSA`.
- `N` stands for the number of SSA realizations to be simulated. This input argument is optional with the default value equal to 10.
- `options` specifies the options for SSA simulation, as explained below.

The output arguments are as follow:

- `X` is a 3-dimensional matrix which contains the state of the system, i.e. the count of species, at the time points specified by `t` vector. The first dimension corresponds to time points, the second dimension corresponds to species, and the third dimension corresponds to individual realizations.
- `Y` is a 3-dimensional matrix which contains the state of output variables at the time points specified by `t` vector. The first dimension corresponds to time points, the second dimension corresponds to output variables, and the third dimension corresponds to individual realizations.
- `mX` is a 2-dimensional matrix which contains the mean of concentrations/counts of species at the time points specified by `t` vector. The first dimension corresponds to time point, and the second

dimension corresponds to species. The means are calculated by Monte-Carlo integration over individual realizations (see Section 1.3.1).

- `mY` is a 2-dimensional matrix which contains the mean concentrations/counts of output variables at the time points specified by `t` vector. The first dimension corresponds to time point, and the second dimension corresponds to output variables. The means are calculated by Monte-Carlo integration over individual realizations (see Section 1.3.1).
- `CX` is a 2-dimensional matrix which contains the variance of concentrations/counts of species at the time points specified by `t` vector. The first dimension corresponds to time point, and the second dimension corresponds to species. The variances are calculated by Monte-Carlo integration over individual realizations (see Section 1.3.1).
- `CY` is a 2-dimensional matrix which contains the variance of concentrations/counts of output variables at the time points specified by `t` vector. The first dimension corresponds to time point, and the second dimension corresponds to output variables. The variances are calculated by Monte-Carlo integration over individual realizations (see Section 1.3.1).

Constant propensities

If the propensities do not explicitly depend on time, i.e. $a_j = a_j(\mathbf{x})$, the following option should be provided when calling `simulate_SSA`:

```
options.mode = 'constant'
```

In this case, which is the default mode for SSA simulations, a next-reaction method (Gillespie, 1977) is performed to generate the SSA trajectories.

Time-dependent propensities

If the propensities are time-dependent, i.e. $a_j = a_j(\mathbf{x}, t)$, the following option should be provided when calling `simulate_SSA`:

```
options.mode = 'time-dependent'
```

In this case, ACME uses a variation of the next-reaction method for time-dependent propensities (Anderson, 2007) to simulate the trajectories of the system.

6.2 Example: The three-stage gene expression

The following code can be used to generate 10000 trajectories for the three-stage gene expression example with given parameter values and time vector:

```
modelDefName = 'modelDef_geneExpression';
theta = [0.3;0.3;10;1;4;1;0.015;1];
t = linspace(0,100,500);
eval(modelDefName);
System = completeSystemSSA(System);
options.mode = 'constant';
N = 10000;
[X_SSA,Y_SSA,mX_SSA,mY_SSA,CX_SSA,CY_SSA] = simulate_SSA(t,theta,System,N,options);
```

With the following output arguments:

- X_SSA is a $500 \times 4 \times 10000$ matrix which contains the counts of DNA_off, DNA_on, mRNA and Protein at the time points specified by t vector.
- Y_SSA is a $500 \times 1 \times 10000$ matrix which contains the value of scaled Protein count at the time points specified by t vector.
- mX_SSA is a 500×4 matrix which contains the mean of the counts of DNA_off, DNA_on, mRNA and Protein at the time points specified by t vector.
- mY_SSA is a 500×1 matrix which contains the mean of the scaled Protein count at the time points specified by t vector.
- CX_SSA is a 500×4 matrix which contains the variance of the counts of DNA_off, DNA_on, mRNA and Protein at the time points specified by t vector.
- CY_SSA is a 500×1 matrix which contains the variance of the scaled Protein count at the time points specified by t vector.

Chapter 7

Post-processing and visualization of simulation results

The results of the numerical simulation can be visualized using ACME plotting routines. For each of the modeling approaches, a specific plotting routine can be called as follows.

7.1 Visualization of trajectories

7.1.1 Visualization of MM simulation results

In MM simulations, the following command can be used to provide plots of the simulation results:

```
plotMM(t,x,y,options)
```

where

- `t` is the time vector used for numerical simulation.
- `x` is the output of the numerical simulation containing the moments of species.
- `y` is the output of the numerical simulation containing the moments of output variables.
- `options` contains specifications for plotting. This input argument is optional and can be left out.

If no options are specified, `plotMM` will generate two figures illustrating the time courses of the mean of species and output variables, including the 1-σ intervals, $[m - \sigma, m + \sigma]$. Here, σ denotes the standard deviation. To plot the higher-order moments of species and output variables, the following options can be set and passed to `plotMM`:

```
options.state_order = xo, options.output_order = yo
```

where

- `xo` specifies the moment order of interest for species.
- `yo` specifies the moment order of interest for output variables.

In this case, `xo` figures will be generated displaying the time courses of the first `xo` moments of species. Similarly, `yo` figures will be generated displaying the time courses of the first `yo` moments of output variables.

7.1.2 Visualization of SSE simulation results

In SSE simulations, the following command can be used to provide plots of the simulation results:

```
plotSSE(t,x,y,options)
```

where

- `t` is the time vector used for numerical simulation.
- `x` is the output of the numerical simulation containing the state variables of SSE (see Section 4.2).
- `y` is the output of the numerical simulation containing the mean and the variance of output variables.
- `options` contains specifications for plotting. This input argument is optional and can be left out.

If no options are specified, `plotSSE` will generate a figure illustrating the time courses of the SSE state variables. In addition, it generates a figure where the time courses of the mean of output variables, including the 1-σ intervals, $[m-\sigma, m+\sigma]$, are plotted. To plot the moments of species, the following option can be set and passed to `plotSSE`:

```
options.species_moments = 1
```

In this case, `plotSSE` calculates the mean and the variance of species based on the SSE state variables, and plots the time course of the mean of species including the $1-\sigma$ intervals.

7.1.3 Visualization of MCM simulation results

In MCM simulations, the following command can be used to provide plots of the simulation results:

```
plotMCM(t,x,y,options)
```

where

- `t` is the time vector used for numerical simulation.
- `x` is the output of the numerical simulation containing the marginal probabilities of `'stochastic'` species and the conditional moments of `'moment'` species.
- `y` is the output of the numerical simulation containing the mean and the variance of output variables.
- `options` contains specifications for plotting. This input argument is optional and can be left out.

If no options are specified, `plotMCM` will generate a figure illustrating the time courses of the marginal probabilities of the `'stochastic'` species and the conditional mean of `'moment'` species including the $1-\sigma$ intervals. In addition, a figure displaying the time courses of the mean of output variables including the $1-\sigma$ intervals is generated. To plot the higher-order conditional moments of `'moment'` species and output variables, the following options can be set and passed to `plotMCM`:

```
options.state_order = xo, options.output_order = yo
```

where

- `xo` stands for the moment order of interest for species.

- `yo` stands for the moment order of interest for output variables.

In this case, `xo` figures will be generated displaying the time courses of the first `xo` conditional moments of species, together with the corresponding `'stochastic'` states. Similarly, `yo` figures will be generated displaying the time courses of the first `yo` moments of output variables.

Attention! For the output variables, the overall moments, and *not* the conditional moments are plotted.

7.1.4 Visualization of FSP simulation results

In FSP simulations, the following command can be used to provide plots of the simulation results:

```
plotFSP(t,x,y,options)
```

where

- `t` is the time vector used for numerical simulation.
- `x` is the output of the numerical simulation containing probabilities of FSP states.
- `y` is the output of the numerical simulation containing the moments of output variables.
- `options` contains specifications for plotting. This input argument is optional and can be left out.

If no options are specified, `plotFSP` will generate a figure visualizing the probability distribution over the FSP states. In addition, the time courses of the mean of the output variables, including the $1-\sigma$ intervals are plotted in a figure. To plot the higher-order moments of the output variables, the following option can be set and passed to `plotFSP`:

```
options.output_order = yo
```

In this case, `yo` figures will be generated displaying the time courses of the first `yo` moments of output variables.

7.1.5 Visualization of SSA simulation results

The results of SSA simulation can be visualized using the following command:

```
plotSSA(t,X,Y,options)
```

where

- `t` is the time vector used for SSA simulation.
- `X` is the output of the SSA simulation containing the state of the system, i.e. the count of species, at the time points specified by t vector. It is a 3-dimensional matrix in which the first dimension corresponds to time points, the second dimension corresponds to species, and the third dimension corresponds to individual realizations.
- `Y` is the output of the SSA simulation containing the state of output variables at the time points specified by t vector. It is a 3-dimensional matrix in which the first dimension corresponds to time points, the second dimension corresponds to output variables, and the third dimension corresponds to individual realizations.
- `options` contains specifications for plotting. This input argument is optional and can be left out.

If no options are provided, `plotSSA` generates three figures: The first figure displays the counts of species and output variables in 10 randomly chosen realizations. To plot more realizations, the following option can be provided:

```
options.n_realization = nr
```

In this case, `nr` randomly chosen realizations are visualized. The second and third figure show the time courses of the mean of species and output variables including the $1-\sigma$ intervals. To plot higher-order moments of species or output variables the following options can be set and passed to `plotSSA`:

```
options.state_order = xo, options.output_order = yo
```

In this case, `xo` figures will be generated displaying the time courses of the first `xo` moments of species. Similarly, `yo` figures will be generated displaying the time courses of the first `yo` moments of output variables.

7.1.6 General options

In addition to the method-specific options, the user can specify general plotting options as below:

- `options.xlcolor` specifies the line color for plotting the state variables.
- `options.ylcolor` specifies the line color for plotting the output variables.
- `options.lwidth` specifies the line width.
- `options.fsize` specifies the font size.
- `options.save` specifies that the figures should be saved if set to 1.
- `options.save_path` specifies the path under which the figures are to be saved.

7.2 Visualization of correlation and partial correlation

In addition to directly plotting the simulation results, ACME offers the calculation and visualization of the correlation and partial correction maps for further investigations/interpretations of the simulation results.

7.2.1 Correlation map

The following command can be used to obtain the correlation map:

```
[corrMat,corrMatAll,covMat] = corrmat(x,options)
```

where

- `x` is a matrix for which the correlation map is to be obtained. Each row is a time point and each column is a variable.
- `options` contains specifications for plotting. This input argument is optional and can be left out.
- `corrMat` is a matrix consisting of the maximum correlation coefficients across all time points.
- `corrMatAll` is a 3-dimensional matrix of all correlation coefficients across all time points. The first two dimensions correspond to variables and the third dimension corresponds to time points.

- `covMat` is a covariance matrix corresponding to maximum correlation coefficients across all time points.

If `options.visualization = 'on'` (default), a correlation map consisting of the maximum correlation coefficients across all time points is plotted. Also, a movie of the correlation map over time is generated and entitled `options.movieName`.

7.2.2 Partial correlation map

The following command can be used to obtain the partial correlation map:

```
[pCorrMat,pCorrMatAll] = pcorrmat(x,covMat,corrMatAll,options)
```

where

- `x` is a matrix for which the partial correlation map is to be obtained. Each row is a time point and each column is a variable.
- `covMat` is a 3-dimensional matrix which contains the covariance matrix over all time points. The first two dimensions correspond to variables and the third dimension corresponds to time points.
- `corrMatAll` is given by `corrmat` and is a 3-dimensional matrix of all the correlation coefficients across all time points. The first two dimensions correspond to variables and the third dimension corresponds to time points.
- `options` contains specifications for plotting. This input argument is optional and can be left out.
- `pCorrMat` is a matrix consisting of the maximum partial correlation coefficients across all time points.
- `pCorrMatAll` is a 3-dimensional matrix of all the partial correlation coefficients across all time points. The first two dimensions correspond to variables and the third dimension corresponds to time points.

If `options.visualization = 'on'` (default), a partial correlation map consisting of the maximum partial correlation coefficients across all time points is plotted. Also, a movie of the partial correlation map over time is generated and entitled `options.movieName`.

Attention! If necessary, a shrinkage of the covariance matrix is automatically performed to yield a positive semi-definite covariance matrix to be used in the calculation of partial correlation coefficients reference and elaborate!!!.

7.3 Example: The three-stage gene expression

The simulation results obtained by MM simulation (see Section 5.1) are plotted as below, and shown in Figure 7.1:

```
plotMM(t,x_MM,y_MM)
```

To plot the 2nd and 3rd-order moments of the species (Figure 7.2), the following options are set:

```
options.state_order = 3;
plotMM(t,x_MM,y_MM,options)
```

Similarly, the states of the IOS simulation (see Section 5.1), together with the output moments, can be visualized as below (see Figure 7.3):

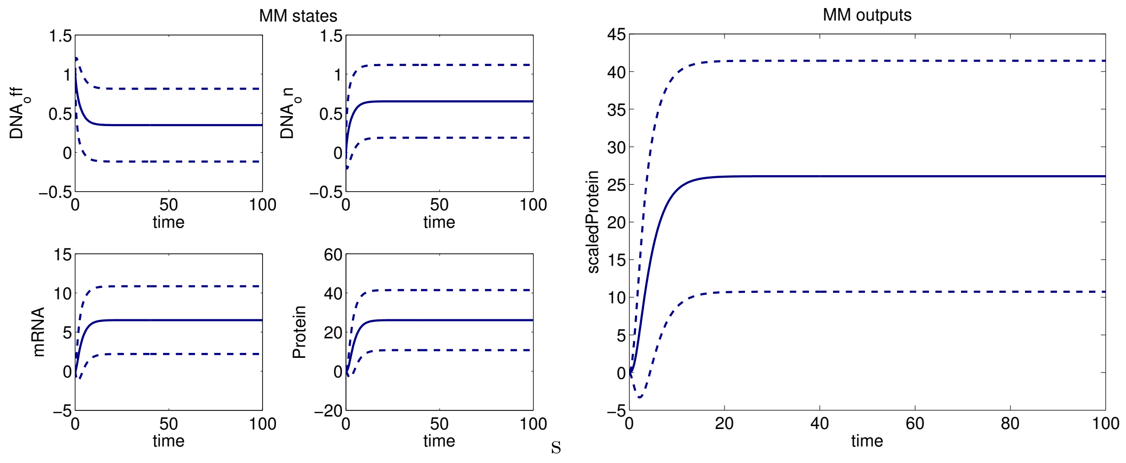

Figure 7.1: **Visualization of MM simulation results.** (a) Mean of species including the 1-σ intervals. (b) Mean of output variables including the 1-σ intervals.

```
plotSSE(t,x_IOS,y_IOS)
```

In addition, the moments of species can be plotted using the following (see Figure 7.4):

```
options.species_moments = 1;
plotSSE(t,x_IOS,y_IOS,options)
```

The results of MM, MCM, and IOS simulations are plotted together as below, and the resulting plots are shown in Figure 7.5:

```
SystemAll{1} = System_MM;
SystemAll{2} = System_MCM;
SystemAll{3} = System_IOS;
plotCompare(SystemAll)
```

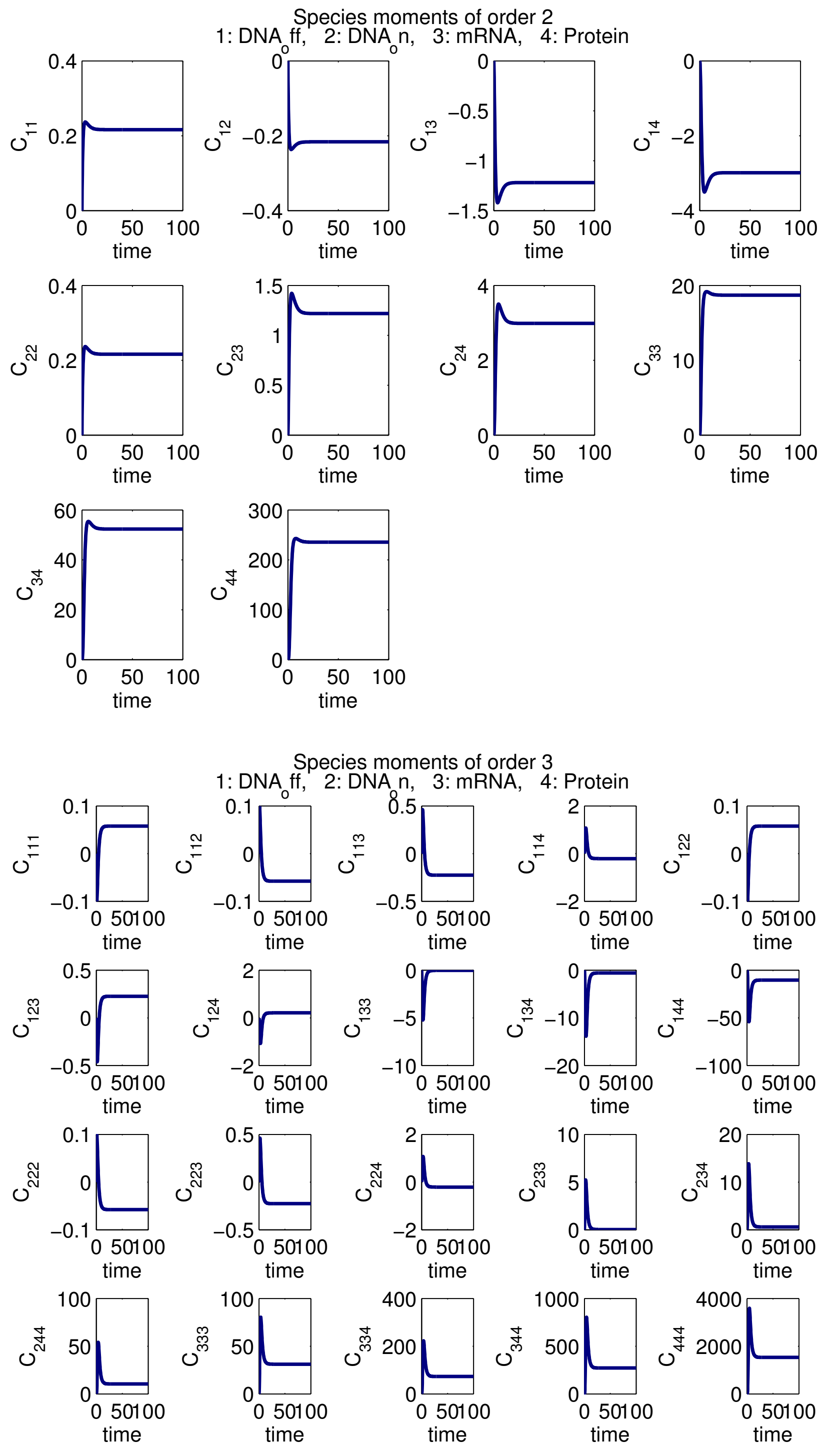

Figure 7.2: **Visualization of higher-order moments using MM simulation results.** (a) Second-order moments of species. (b) Third-order moments of species.

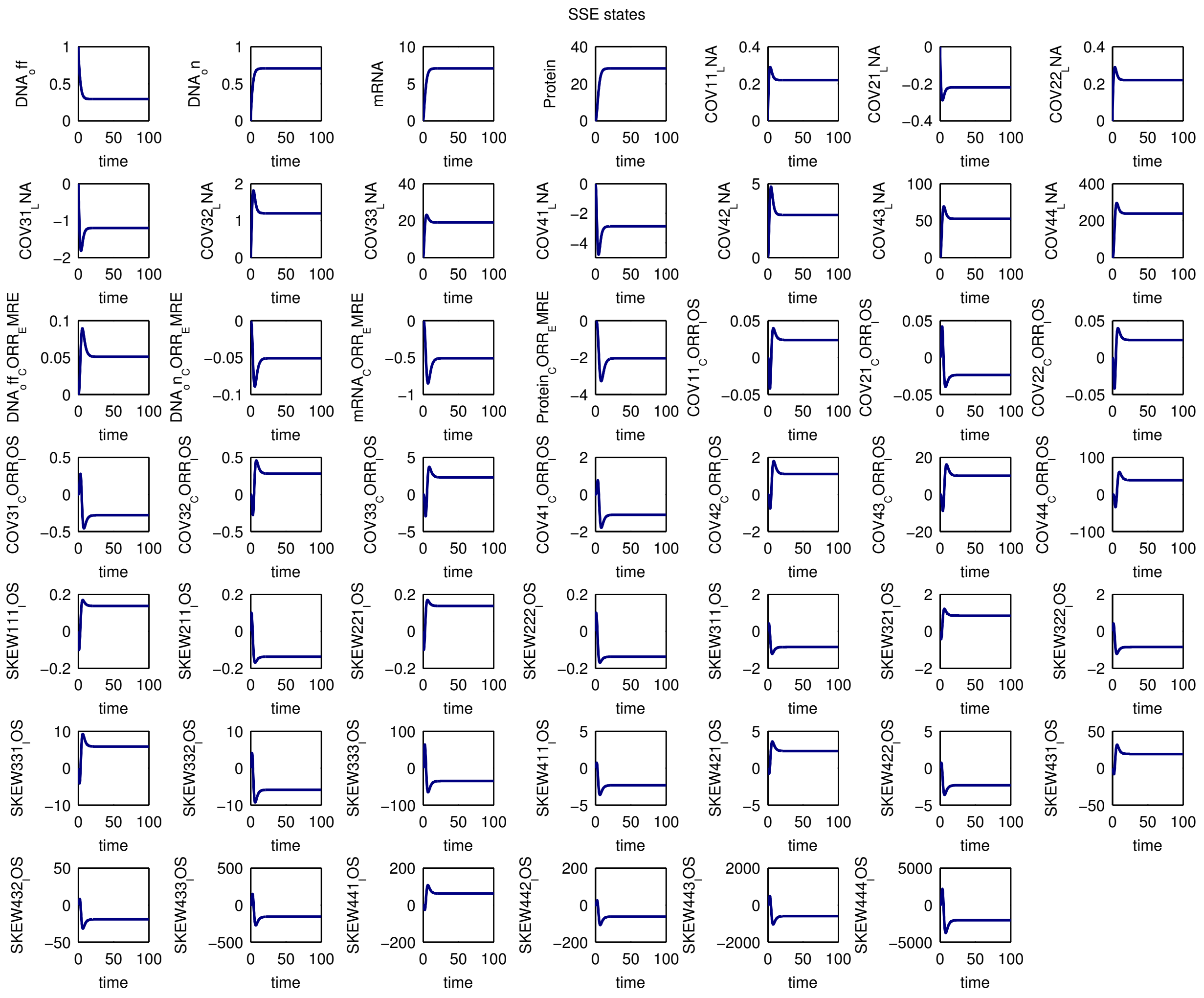

Figure 7.3: **Visualization of the SSE simulation results.**

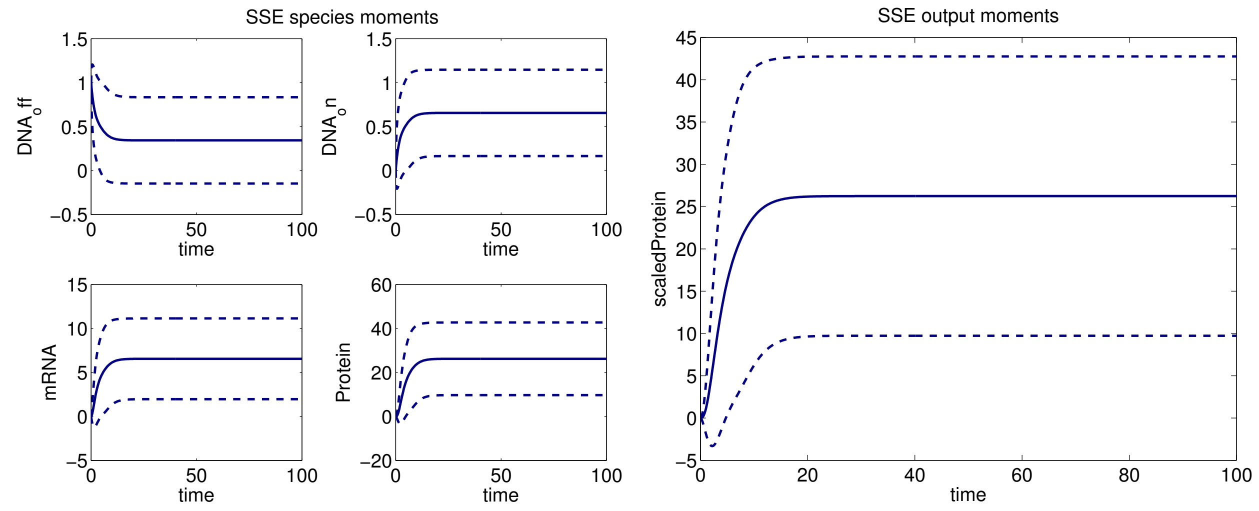

Figure 7.4: **Visualization of the moments of species and output variables obtained by SSE simulation.** (a) Mean of species including the 1-σ intervals. (b) Mean of output variables including the 1-σ intervals.

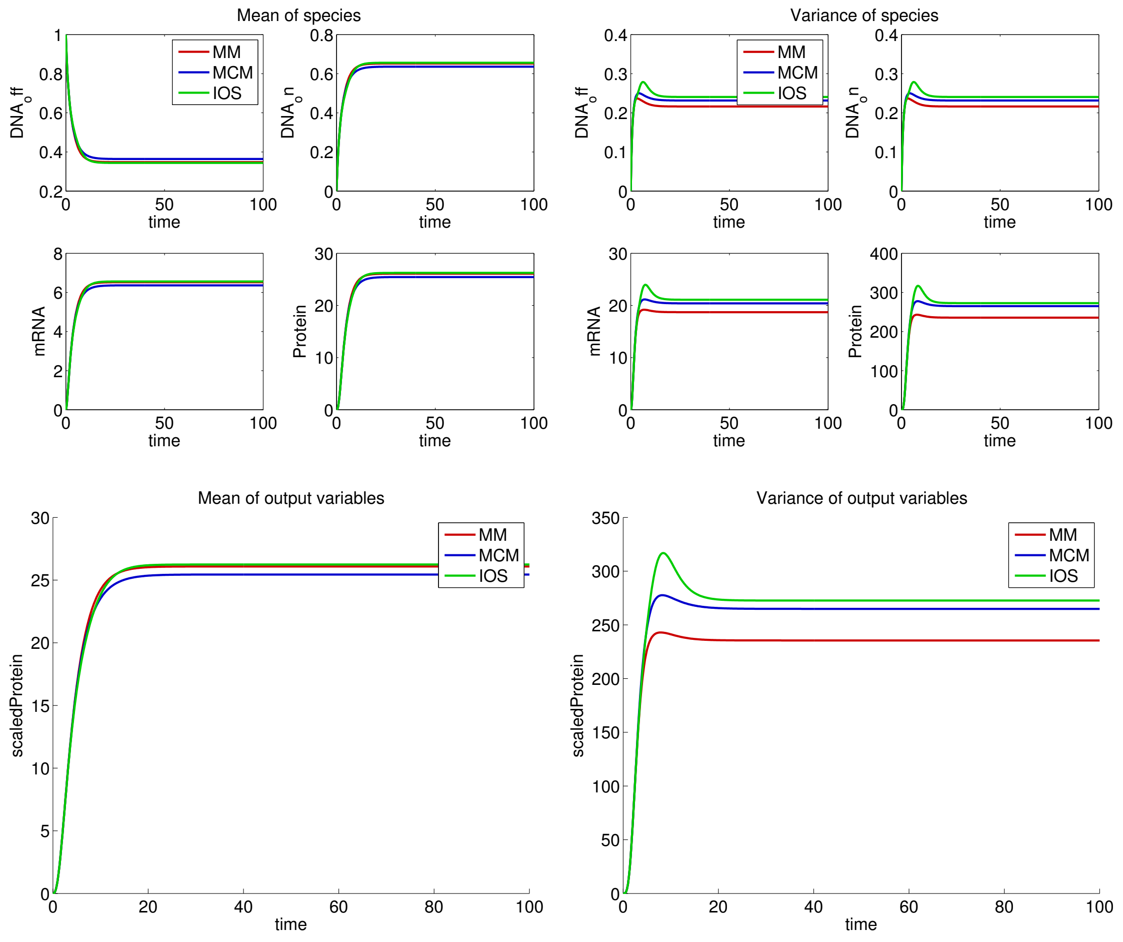

Figure 7.5: **Mean and variance of species and output variables obtained by MM, MCM and IOS simulations.**

Chapter 8

Further examples

8.1 JAK-STAT signaling pathway

The second example studied using ACME is a simplified model of the JAK-STAT signaling pathway introduced by Raue *et al.* (2009). The model, sketched in Figure 8.1, describes the signaling cascade of STAT protein. Upon activation, Epo receptor triggers the phosphorylation of cytoplasmic STAT. Dimerization and translocation of phosphorylated STAT into the nucleus, followed by a delayed export of STAT from the nucleus complete the pathway. The time-dependent concentration of phosphorylated Epo receptor, [pEpoR], functions as an input to the system. The experimental data for the concentration of phosphorylated Epo receptor, cytoplasmic STAT and phosphorylated cytoplasmic STAT are available from previous studies (Swameye *et al.*, 2003).

The JAK-STAT signaling pathway is an interesting application example as it (i) includes two compartments, namely cytoplasm and nucleus, and (ii) involves a time-dependent propensity. Given the parameter values and initial conditions in Table 8.1, this pathway is simulated using several modeling approaches, including SSA.

The SSA simulation results, are plotted using the command below, and are shown in Figures 8.2 and 8.3:

```
options.fs = 10;
plotSSA(System_SSA,options)
```

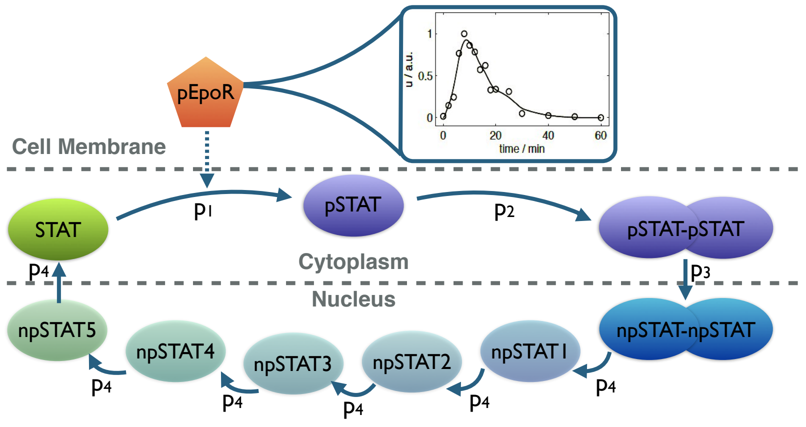

Figure 8.1: Schematic of the simplified JAK-STAT signaling pathway.

Table 8.1: Parameter values and initial conditions of the three-stage gene expression example.

p_1	3.9364
p_2	10
p_3	0.1125
p_4	0.9829
$X_{c,\text{STAT}}(0)$	1
$X_{c,\text{pSTAT}}(0)$	0
$X_{c,\text{pSTAT-pSTAT}}(0)$	0
$X_{c,\text{npSTAT-npSTAT}}(0)$	0
$X_{c,\text{npSTAT1}}(0)$	0
$X_{c,\text{npSTAT2}}(0)$	0
$X_{c,\text{npSTAT3}}(0)$	0
$X_{c,\text{npSTAT4}}(0)$	0
$X_{c,\text{npSTAT5}}(0)$	0

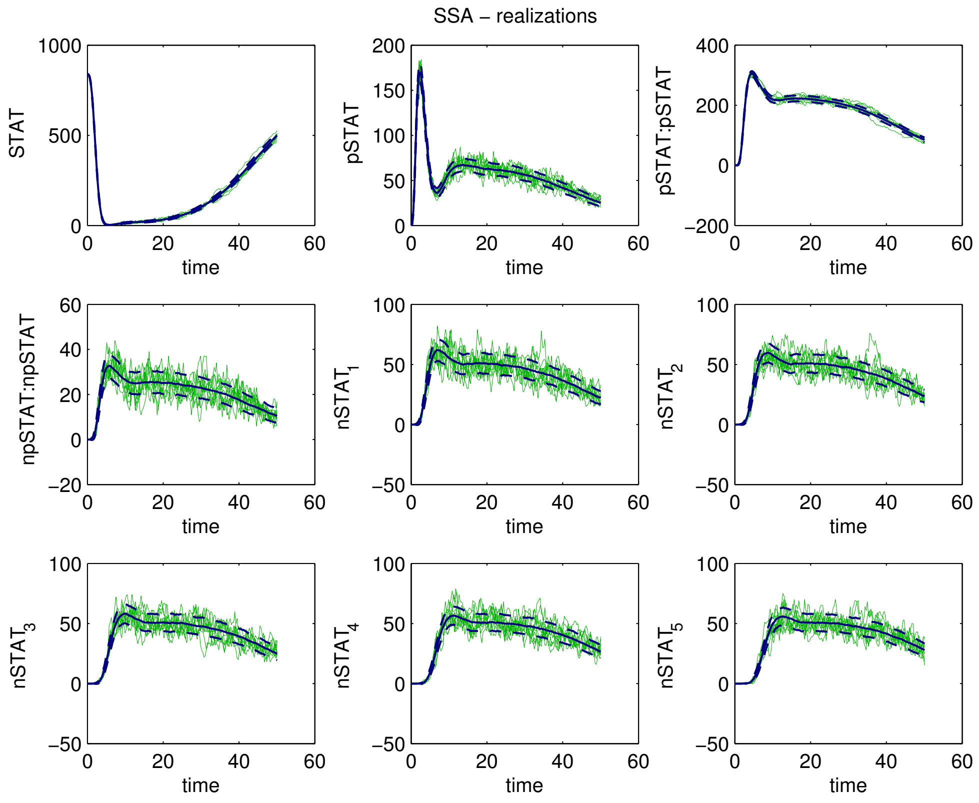

Figure 8.2: **SSA simulation results for the simplified JAK-STAT pathway.** Individual realization (green) and the mean of the counts of species including 1-σ intervals (blue) are shown.

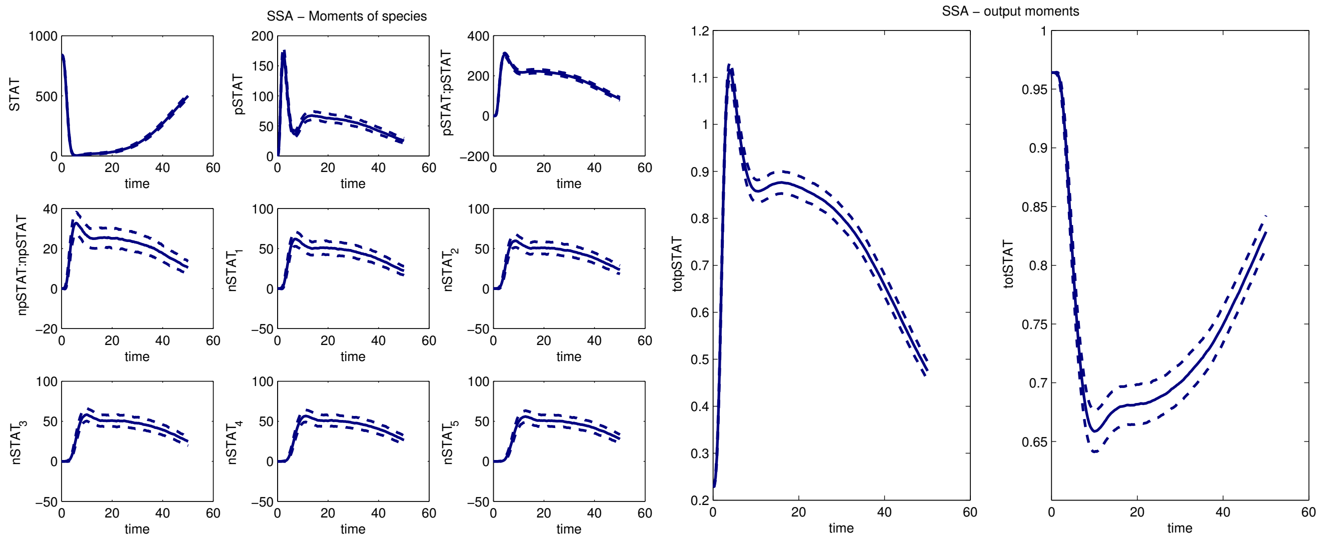

Figure 8.3: **The concentration of species and observable, i.e. total concentration of STAT, obtained by SSA simulation.** The means, including 1-σ intervals, are shown.

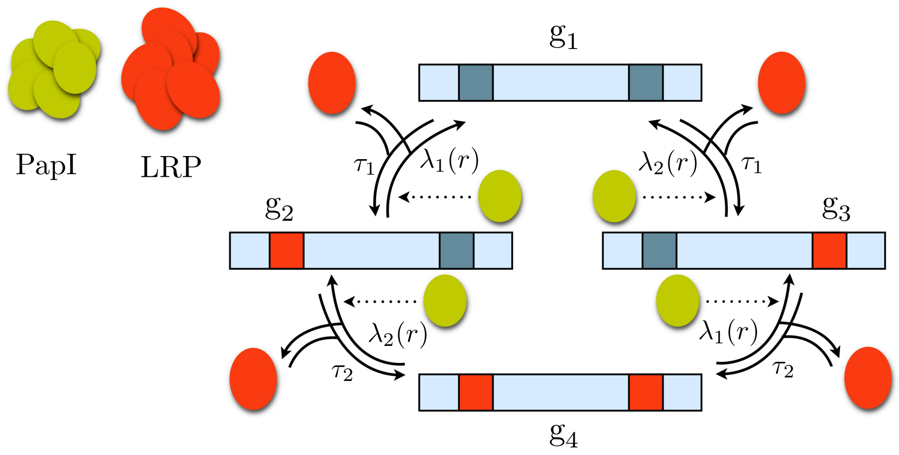

Figure 8.4: **Schematic of the PapI regulation model.** Arrows represent the binding and unbinding of LRP to/from the operon. Dotted arrows indicate the influence of PapI on the reaction rates.

8.2 PapI regulation model

This model, sketched in Figure 8.4, describes the regulation of Pap pili formation on the surface of *E. coli* (Kazeroonian *et al.*, 2014). This system includes a a *pap* operon and regulatory proteins LRP and PapI. Binding of LRP to either or both of the binding sites results in four configurations of the *pap* operon. Only if the operon is in state g_2, pili can be produced. PapI sets up a a positive feedback loop for the production of pili by reducing the unbinding rate of LRP from the operon.

This process is an interesting application example as it involves non-polynomial propensity functions. The model definition and simulation files for this system can be found in `ACME/examples/PapI`.

8.3 A gene cascade network

A gene cascade network, described in Singh and Hespanha (2011), is implemented in ACME. The model definition and simulation files can be looked up in `ACME/examples/Hespanha2011`.

Bibliography

Anderson, D. F. (2007). A modified next reaction method for simulating chemical systems with time dependent propensities and delays. *J. Chem. Phys.*, **127**(214107).

Engblom, S. (2006). Computing the moments of high dimensional solutions of the master equation. *Appl. Math. Comp.*, **180**, 498–515.

Gillespie, D. T. (1977). Exact stochastic simulation of coupled chemical reactions. *J. Phys. Chem.*, **81**(25), 2340–2361.

Grima, R. (2010). An effective rate equation approach to reaction kinetics in small volumes: Theory and application to biochemical reactions in nonequilibrium steady-state conditions. *J. Chem. Phys.*, **133**(035101).

Hasenauer, J., Wolf, V., Kazeroonian, A., and Theis, F. J. (2014). Method of conditional moments (MCM) for the chemical master equation. *J. Math. Biol.*, **69**(3), 687–735.

Hindmarsh, A. C., Brown, P. N., Grant, K. E., Lee, S. L., Serban, R., Shumaker, D. E., and Woodward, C. S. (2005). SUNDIALS: Suite of Nonlinear and Differential/Algebraic Equation Solvers. *ACM T. Math. Software*, **31**(3), 363–396.

Kazeroonian, A., Theis, F. J., and Hasenauer, J. (2014). Modeling of stochastic biological processes with non-polynomial propensities using non-central conditional moment equation. In *Proc. of the 19th IFAC World Congress*, volume 19, pages 1729–1735, Cape Town, South Africa.

Keating, S. M., Bornstein, B. J., Finney, A., and Hucka, M. (2006). Sbmltoolbox: an sbml toolbox for matlab users. *Bioinformatics*, **22**(10), 1275–1277.

Lee, C. H., Kim, K. H., and Kim, P. (2009). A moment closure method for stochastic reaction networks. *J. Chem. Phys.*, **130**(13), 134107.

Munsky, B. and Khammash, M. (2006). The finite state projection algorithm for the solution of the chemical master equation. *J. Chem. Phys.*, **124**(4), 044104.

Norris, J. R. (1997). Continuous-time markov chains i. In *Markov Chains*, pages 60–107. Cambridge University Press. Cambridge Books Online.

Ramaswamy, R., González-Segredo, N., Sbalzarini, I., and Grima, R. (2012). Discreteness-induced concentration inversion in mesoscopic chemical systems. *Nat. Comm.*, **3**(779).

Raue, A., Kreutz, C., Maiwald, T., Bachmann, J., Schilling, M., Klingmüller, U., and Timmer, J. (2009). Structural and practical identifiability analysis of partially observed dynamical models by exploiting the profile likelihood. *Bioinf.*, **25**(25), 1923–1929.

Raue, A., Schilling, M., Bachmann, J., Matteson, A., Schelke, M., Kaschek, D., Hug, S., Kreutz, C., Harms, B. D., Theis, F. J., Klingmüller, U., and Timmer, J. (2013). Lessons learned from quantitative dynamical modeling in systems biology. *PLoS ONE*, **8**(9), e74335.

Raue, A., Steiert, B., Schelker, M., Kreutz, C., Maiwald, T., Hass, H., Vanlier, J., Tönsing, C., Adlung, L., Engesser, R., Mader, W., Heinemann, T., Hasenauer, J., Schilling, M., Höfer, T., Klipp, E., Theis, F. J., Klingmüller, U., Schöberl, B., and J.Timmer (2015). Data2Dynamics: a modeling environment tailored to parameter estimation in dynamical systems. *in revision as Bioinformatics Software Note*.

Shahrezaei, V. and Swain, P. S. (2008). Analytical distributions for stochastic gene expression. *Proc. Natl. Acad. Sci. U S A*, **105**(45), 17256–17261.

Singh, A. and Hespanha, J. P. (2011). Approximate moment dynamics for chemically reacting systems. *IEEE Trans. Autom. Control*, **56**(2), 414–418.

Swameye, I., Müller, T. G., Timmer, J., Sandra, O., and Klingmüller, U. (2003). Identification of nucleocytoplasmic cycling as a remote sensor in cellular signaling by databased modeling. *Proc. Natl. Acad. Sci. U S A*, **100**(3), 1028–1033.

Thomas, P., Matuschek, H., and Grima, R. (2013). How reliable is the linear noise approximation of gene regulatory networks? *BMC Genomics*, **14(Suppl 4)**(S5).

van Kampen, N. G. (2007). *Stochastic processes in physics and chemistry*. North-Holland, Amsterdam, 3rd edition.

Acronyms

ACME A toolbox for the simulation and analysis of chemical reaction networks using **A**pproximations of the **C**hemical **M**aster **E**quation solution statistics. 5

CME Chemical Master Equation. 6–9

EMRE Effective Mesoscopic Rate Equation. 8, 23

FSP Finite State Projection. 2, 7, 11, 13, 16, 21, 23, 25, 38

IOS Inverse Omega Square. 8, 23, 27, 31, 32, 40, 41, 45

LNA Linear Noise Approximation. 8, 23

MCM Method of Conditional Moments. 2, 9, 11, 16, 21, 23–25, 27, 37, 41, 45

MM Method of Moments. 2, 8, 11, 21, 23, 24, 27, 30, 36, 40–42, 45

RRE Reaction Rate Equations. 7, 8, 11, 21, 23

SSA Stochastic Simulation Algorithms. 2, 6, 13, 33, 34, 38, 46, 48

SSE System Size Expansion. 2, 8, 11, 21, 23, 37, 44

Glossary

`System` is a MATLAB structure array containing the reaction network specifications. 15, 18, 20, 21, 33

`method` is the selected modeling approach. 21–25, 27

`modelDef.m` is the model definition file. 15, 18, 20, 21, 23–25, 33

`modelDefName` is the name of the `modelDef.m` file. 21, 22

`modelName` is the name of the reaction network. This name will be used in naming the simulation files. 21–23, 26, 27