# 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 [3,4,5], 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]Institute of Physics, University of Freiburg, Freiburg, Germany,
[4]BIOSS Centre for Biological Signaling Studies, University of Freiburg, Freiburg, Germany,
[5]Merrimack Pharmaceuticals, Inc., Cambridge, Massachusetts, United States of America

August 4, 2015

[*]to whom correspondence should be addressed.

# Contents

# Chapter 1

# Introduction to stochastic biochemical reactions

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

## 1.1 Biochemical reaction networks

Consider a biological system comprising a chemical reaction network. The kinetics of such a network is usually modeled/envisioned as 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 (to build a mathematical model), 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 $j^{\text{th}}$ chemical reaction, $R_j$, is a process as below:

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

where $c_j$ 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. The overall stoichiometry of a reaction $R_j$ is then given by a vector $\nu_j = (\nu_{1,j}, \nu_{2,j}, \cdots, \nu_{n_s,j}) \in \mathbb{N}_0^{n_s}$.

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 a reaction $R_j$, however, the state transition $\mathbf{X_t} \to \mathbf{X_t} + \nu_j$ occurs, where $\nu_j = \nu_j^{+} - \nu_j^{-}$ is the overall stoichiometry of reaction $R_j$. The statistics of the time until the firing of next reaction, as well as

Table 1.1: Reaction propensities given by the law of mass action.

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

Table 1.2: Relationship between the microscopic and macroscopic parameters.

| Reaction Order | Reaction Type | Microscopic Parameter | Macroscopic parameter |
|---|---|---|---|
| 0 | $\emptyset \xrightarrow{c_j}$ product | $c_j$ | $c'_j = \frac{c_j}{\Omega}$ |
| 1 | $S_i \xrightarrow{c_j}$ product | $c_j$ | $c'_j = c_j$ |
| 2 | $S_i + S_j \xrightarrow{c_j}$ product | $c_j$ | $c'_j = c_j \Omega$ |

the index of the next reaction, are determined by the *reaction propensities*. The *propensity* of reaction $R_j$, $a_j(X_t) : \mathbb{N}_0^{n_s} \rightarrow \mathbb{R}_+$ is a function of the state of the system. If we assume 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, $c_j$ 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 relations 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 $c_j$, 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 $\mathbf{X_t}$. However, 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 $\Omega$ 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, $c_j X_i X_j$. the corresponding propensity in terms of concentrations is obtained as

$$\frac{1}{\Omega} c_j (\Omega C_i)(\Omega C_j) = c_j \Omega C_i C_j.$$

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

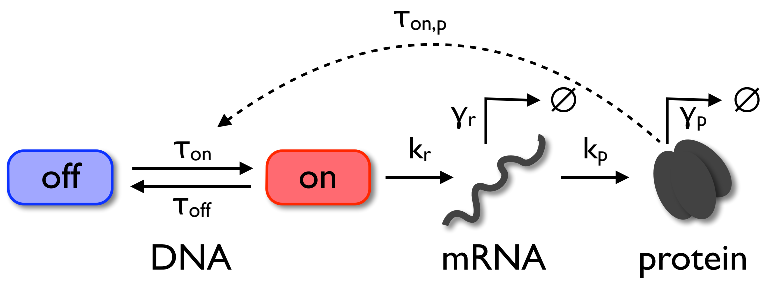

Figure 1.1: Schematic of the three-stage model of gene expression.

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

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

## 1.2 The three-stage gene expression example

In this section, we consider the generalized three-stage model of gene expression (Shahrezaei and Swain, 2008), depicted in Figure 1.1, as an example system to further illustrate the aforementioned notions. This system will be used in later sections to demonstrate the usage of ACME functions.

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:

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

The state vector of the system, $\mathbf{X} = (X_{\text{DNA}_{\text{off}}}, X_{\text{DNA}_{\text{on}}}, X_{\text{mRNA}}, X_{\text{protein}})$, 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 stoichiojmety matrix of the system is given as:

$$S = \begin{array}{c} \\ X_{\text{DNA}_{\text{off}}} \\ X_{\text{DNA}_{\text{on}}} \\ X_{\text{mRNA}} \\ X_{\text{protein}} \end{array} \begin{array}{c} \begin{array}{ccccccc} R_1 & R_2 & R_3 & R_4 & R_5 & R_6 & R_7 \end{array} \\ \begin{pmatrix} -1 & 1 & 0 & 0 & 0 & 0 & -1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 \end{pmatrix} \end{array},$$

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

## 1.3 Modeling 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). The state of this CTMC is represented by a vector $\mathbf{X} = (X_1, \ldots, X_{n_s})$, where $X_i$ denotes the count of species $S_i$. A reaction $R_j$ with stoichiometric coefficients $\nu_{i,j}^-$ and $\nu_{i,j}^+$,

$$R_j : \sum_{i=1}^{n_s} \nu_{i,j}^- s_i \overset{a_j}{\to} \sum_{i=1}^{n_s} \nu_{i,j}^+ s_i, \tag{1.1}$$

fires with propensity $a_j(\mathbf{x})$, resulting in the state transition $\mathbf{x} \to \mathbf{x} + \nu_j$, with $\nu_j = \nu_j^+ - \nu_j^-$. The probability of observing the CTMC at state $\mathbf{x}$ 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):

$$\frac{\partial}{\partial t} p(\mathbf{x}|t) = \sum_{j=1}^{n_r} \left( a_j(\mathbf{x} - \nu_j) p(\mathbf{x} - \nu_j|t) - a_j(\mathbf{x}) p(\mathbf{x}|t) \right). \tag{1.2}$$

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.,

$$\begin{aligned} \text{mean} \quad m &= \sum_{\mathbf{x}} \mathbf{x} p(\mathbf{x}|t), \\ \text{covariance} \quad C &= \sum_{\mathbf{x}} (\mathbf{x} - \mu)(\mathbf{x} - \mu)^T p(\mathbf{x}|t), \end{aligned} \tag{1.3}$$

and higher-order moments. The microscopic, mesoscopic 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). An estimate to the probability distribution $p(\mathbf{x}|t)$ is 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. 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.

### 1.3.2 Finite State Projection.

To enable a direct approximation of $p(\mathbf{x}|t)$, FSP (Munsky and Khammash, 2006) reduces the number of state variables of the CME by only considering the states of non-negligible probabilities. To achieve this, all states with sufficiently low probabilities are lumped into a single sink state, and the remaining set of ODEs yield a lower bound for $p(\mathbf{x}|t)$. Growing the state-space of FSP decreases the approximation error at the cost of increased computational complexity.

### 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 $m$ in the macroscopic limit. Thus, as copy-numbers increase, the solution of RRE tends towards the true mean of the stochastic process. Neglecting the stochastic effects, RRE prediction can be considerably different from the true mean of the system for finite sizes.

### 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).

### 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. The equations can be derived for moments up to any moment order $m$. The resulting equations for moments of order $m$ generally depend on moments of order $m + 1$ and higher. Therefore, moment closures are applied yielding a closed set of approximative moment equations. Commonly used closure techniques include low dispersion closure, mean field closure, zero cumulants closure, and derivative matching closure (Singh and Hespanha, 2011). 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. The evolution equations for marginal probabilities and conditional moments are derived from the CME, and form a system of differential algebraic equations (DAEs). Similar to moment equations, the evolution of lower-order conditional moments can depend on higher-order conditional moments, rendering moment closure necessary. The conditional moments and marginal probabilities can be used to calculate the overall moments. 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 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. Assuming that the model possesses $n$ state-variables and $n_\theta$ parameters, roughly a system of $n(1 + n_\theta)$ differential equations is solved to compute the state sensitivities with respect to all parameters. The sensitivity of measured quantities and objective functions can then be computed based on state sensitivities.

### 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 an ODE system backward in time. They 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 $n_p$ parameters roughly requires forward integration of a system of $n$ ODEs, backward integration of a system of $n$ ODEs and calculating $n_\theta$ one-dimensional integrals. Thus, in applications with high-dimensional parameter spaces, calculating adjoint sensitivities tends to be more computationally advantageous. In parameter estimation, the likelihood function can be defined as the sole output functional of the system, providing the perfect platform for the application of adjoint sensitivities.

# Chapter 2

# ACME startup

## 2.1 Installation

Downloading `ACME.tar` from webpage?, the file can be unzipped by any unzip utilities, or by executing the following command in a shell script:

```
tar xvf ACME.tar
```

The unzipped ACME folder then can be stored in the desired directory. 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 generation 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 FSP and SSA simulations.
  - `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 this folder.
- `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.

## 2.2 Requirements

ACME can be successfully used on MAC OS and Linux. However, on Windows machines, the compilation of simulation files may be problematic. In such cases, the user may try to fix the problem with their own responsibility. No guarantee or support for using ACME on Windows OS is provided.

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. However, if the user is merely interested in numerical simulation using MATLAB ODE solvers (see Section 4.1.6), the installation of these packages is not required.
- **SBML Toolbox.** The usage of `importSBML` (see Section 3.2) requires the installation of SBML Toolbox (**?**).

# Chapter 3

# Setting up a simulation problem in ACME

To carry out simulations in ACME, first the biochemical reaction 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.
- `system.compartments` is a string array containing the names of the compartments of the chemical reaction network.

```
system.compartments = {'comp_1','comp_2', ···}
```

- `system.volumes` is an 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 array 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 array 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 *molecule numbers* and not in concentrations.

- `system.state.xmin` specifies the lower bound on the count of species. This field is only necessary in FSP simulations.

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

- `system.state.xmax` specifies the upper bound on the count of species. This field is only necessary in FSP simulations. The specified lower and upper bounds are used to construct the state space of FSP simulations.

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

- `system.state.constraint` is a function that defines any constraints that should be imposed on the state of the systems. The output of this function should be a logical value. This filed is used in FSP simulations to ensure that every state in the state-space of FSP fulfills the constraints.

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

- `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 $j^{\text{th}}$ 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 $j^{\text{th}}$ reaction.

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

  - `system.reaction(j).product` is a symbolic vector of the products of the $j^{\text{th}}$ 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 $j^{\text{th}}$ reaction.

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

  - `system.reaction(j).propensity` is the symbolic expression of the microscopic propensity of the $j^{\text{th}}$ reaction.

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

  - `system.reaction(j).MacroscopicPropensity` is the symbolic expression of the macroscopic propensity of the $j^{\text{th}}$ 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.

```
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?. 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 `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 totMP mRNATimesProtein % observables
syms tau_on tau_off k_m gamma_m k_p gamma_p tau_on_p % kinetic rates
syms Omega % volume
syms time % time
```

```
% Definition of general fields:
system.time = time;
system.compartments = {'cyt'};  % 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    = {'cyt'; 'cyt'; 'cyt'; 'cyt'};
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     ;    0       ;    0   ];
system.state.C0         = 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;Omega];
system.parameter.name   = {'\\tau_{on}'; '\\tau_{off}'; 'k_m'; '\\gamma_m'; 'k_p'; '\\gamma_p';
                           '\\tau_{on,p}'; '\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  = [totMP; mRNATimesProtein ];
system.output.name      = {'sum of mRNA and Protein'; 'product of mRNA and Protein'};
system.output.function  = [mRNA+Protein; mRNA*Protein];

system.input.variable   = [];
system.input.name       = {};
system.input.function   = [];
```

## 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` 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.

### 3.2.1 Example

The the three-stage gene expression example 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 how to compile simulation MEX-files. After creating the `modelDef.m` file, one of the following commands can be used to generate the system of equations and compile simulation files:

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

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

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

where

- `modelName` stands for the name of the model. This name will be used in naming the simulation files.
- `modelDefName` stands for the name of the `modelDef.m` file.
- `method` stands for the selected modeling approach. The possible options for `method` are presented in Section 4.1.

`genSimFile` consists 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 is relied on the SUNDIALS CVODES package, and therefore the corresponding CVODES options can be specified as inputs (see Chapter **??** 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 Chapter **??**).

```
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 Chapter **??** 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 Chapter **??**).

```
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 is relies on the SUNDIALS IDAS package, and therefore the corresponding IDAS options can be specified as inputs (see Chapter **??** 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 Chapter **??**).

```
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`, and therefore the user only needs to use call this function to generate all the necessary simulation files.

## 4.1 Selection of modeling approach

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

### 4.1.1 SSE

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 states 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 is 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 for molecule numbers. The conversion of the initial conditions to concentrations is automatically done when generating SSE equations.

### 4.1.2 MM

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` stands for the truncation order for the moment equations of the state variables.
- `MC` stands for the moment closure scheme (see Section ??). 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` stands 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 moment equations and merely determines which moments of the output variables should be calculated based on the moments of the state variables.
- `SC` stands for 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.
- `R` determines whether a reduction in covariance matrix should be performed to increase computational efficiency (see Section ??):
  - `f` specifies that the full covariance matrix should be used in moment equations.
  - `c` specifies that the the reduced covariance matrix should be used in moment equations.

The system of MM 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 is 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.

### 4.1.3 MCM

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` stands for the truncation order for the conditional moment equations.
- `MC` stands for the moment closure scheme (see Section ??). 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` stands 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` stands for 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 MCM DAEs are derived for the marginal probabilities of `'stochastic'` species and the first `XO` 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 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.

### 4.1.4 FSP

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 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 molecules of species.

### 4.1.5 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 used can consult this file to easily access the equations and initial conditions.

### 4.1.6 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,
- $\theta$ 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 the user does not with to compile MEX-files. However, as far 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.

### 4.1.7 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. For more details see Section **??**.

## 4.2 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 **??**).

Similarly, in 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 **??**). 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 (see Section 3.1). However, when carrying out numerical simulations, the user can specify which sensitivity equations should be solved (see Section **??**).

## 4.3 The three-stage gene expression example

In the following an excerpt of code to generate and compile the simulation files for the three-stage gene expression exampleusing 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 specyinf the `method` needs to change. For example, to use the 3$^{\mathrm{rd}}$-order MM (without reduction) with low dispersion closure and 2$^{\mathrm{nd}}$-order outputs for the calculation of the concentration of species, the following should be specified:

```
method = 'MEC_3_LD_2_c_f'; % Specifying the modeling approach
```

Similarly, to use the 2$^{\mathrm{rd}}$-order MCM with zero cumulants closure and 2$^{\mathrm{nd}}$-order outputs for the calculation of the molecule numbers of species, 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 for arbitrary parameter values.

## 5.1 Numerical simulation of the system and corresponding options

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,
- $\theta$ 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 input and output variables (`x` and `y`) is specific to the selected modeling approach (see Section 4.1).

This command will carry out the numerical simulation with the default solver options (can be looked up in where?). User-specified solver options can be given as additional input arguments:

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

Basically, any CVODES/IDAS option 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 CVODES/IDAS documentations (Hindmarsh *et al.*, 2005).

### The three-stage gene expression example

Using the MEX-file compiled in Section 4.3, 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]; % Parameter values
t = linspace(0,100,500); % Time vector
[status_MCM,tout_MCM,x_MCM,y_MCM] = simulate_geneExpressionMCM(t,theta);
```

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 4.3 with specific solver accuracy 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.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 acquire the calculation of 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 input and output variables (x and y) is specific to the selected modeling approach (see Section 4.1).

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 filed 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 `help`? section of `simulate_modelName.m`.

#### The three-stage gene expression example

To solve the forward sensitivity equations for the three-stage gene expression example using IOS simulation (MEX-file compiled in Section 4.3) 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 $500 \times 48$ matrix containing the simulation results for the IOS states.
- `y_IOS` is a $500 \times 2$ matrix containing the simulation results for the $1^{\text{st}}$ and $2^{\text{nd}}$-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 filed 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 4.1) with respect to variables. 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 `help`? section of `simulate_modelName.m`.

**The three-stage gene expression example**

To calculate the adjoint sensitivity equations for the three-stage gene expression example using IOS simulation (MEX-file compiled in Section 4.3) 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.

## 5.3 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:

### 5.3.1 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 $1-\sigma$ intervals. 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` 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` moments of species. Similarly, `yo` figures will be generated displaying the time courses of the first `yo` moments of output variables.

### 5.3.2 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 4.1).
- `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 course of the mean of output variables, including the $1 - \sigma$ intervals, 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.

### 5.3.3 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.

### 5.3.4 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.

### 5.3.5 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,s 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.

### 5.3.6 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.

### 5.3.7 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.

#### 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. Do we want this?

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`.

**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 contain 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!!!.

### 5.3.8 The three-stage gene expression example

The simulation results obtained by MM simulation (see Section 5.1) are plotted as below:

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

Figure!

To plot the 2nd and 3rd-moments of the species, the following options are set:

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

Figure!

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

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

Figure!

In addition, the moments of species can be plotted using the following:

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

Figure!

# Chapter 6

# SSA simulation

In addition to all the modeling approaches that yield a system of differential equations (see Section 4.1), 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.
- `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.

- `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.
- `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.

**Constant propensities**

If the propensities are constant (i.e., time-independent), 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? (?)!) is performed to generate the SSA trajectories.

**Time-dependent propensities**

If the propensities are time-dependent, 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.

## 6.1 The three-stage gene expression example

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 $500 \times 10000$ 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 $500 \times 10000$ matrix which contains the value of scaled `Protein` count at the time points specified by `t` vector.

- `mX_SSA` is a 500 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 matrix which contains the mean of the scaled `Protein` count at the time points specified by `t` vector.
- `CX_SSA` is a 500 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 matrix which contains the variance of the scaled `Protein` count at the time points specified by `t` vector.

# Chapter 7

# Further examples

## 7.1 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 7.1, 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.

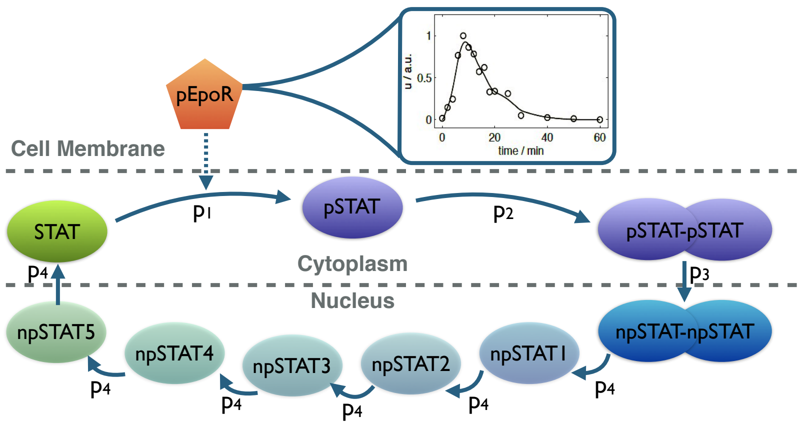

Figure 7.1: Schematic of the simplified JAK-STAT signaling pathway.

# Bibliography

Anderson, D. F. (2007). A modified next reaction method for simulating chemical systems with time dependent propensities and delays. *Journal of Chemical Physics*, **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. *Journal of Mathematical Biology*, **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.

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.

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.