The URDME manual
Version 1.0

Josef Cullhed   Stefan Engblom[1]   Andreas Hellander[1]

December 15, 2008

[1]*Div of Scientific Computing, Dept of Information Technology*
Uppsala University, P. O. Box 337, SE-75105 Uppsala, Sweden
email: `stefane@it.uu.se, andreas.hellander@it.uu.se`

Abstract

We have developed URDME, a general software for simulation of stochastic reaction-diffusion processes on unstructured meshes. This allows for a more flexible handling of complicated geometries and curved boundaries compared to simulations on structured, Cartesian meshes. The underlying algorithm is the next subvolume method (NSM), extended to unstructured meshes by obtaining diffusion jump coefficients from the finite element formulation of the corresponding macroscopic equation.

In this manual, we describe how to use the software together with `Comsol Multiphysics 3.4` and `Matlab` to set up simulations. We provide a detailed account of the code structure and of the available interfaces. This makes modifications and extensions of the code possible. We also give two detailed examples, in which we describe the process of simulating and visualizing two models from the systems biology literature in a step-by-step manner.

Keywords: URDME, reaction-diffusion master equation, stochastic chemical kinetics, stochastic spatial simulation, unstructured meshes.

1   Introduction

Stochastic simulation methods are frequently used to study the behavior of cellular control systems modeled as continuous-time discrete-space Markov processes (CTMC). Compared to the most frequently used deterministic

model, the reaction rate equations, the mesoscopic stochastic description can capture effects from intrinsic noise on the behavior of the networks [1, 6, 23, 24, 27].

In the discrete mesoscopic model the state of the system is the copy number of the different chemical species and the reactions are usually assumed to take place in a well-stirred reaction volume. The chemical master equation is the governing equation for the probability density, and for small to medium sized systems it can be solved by direct, deterministic methods [8, 9, 13, 19, 22, 26]. For larger models however, exact or approximate kinetic Monte Carlo methods [15, 16] are frequently used to generate realizations of the stochastic process. Many different hybrid and multiscale methods have also emerged that deal with the typical stiffness of biochemical reactions networks in different ways, see e.g. [2, 4, 17, 20, 21, 25].

Many processes inside the living cell can not be expected to be explained in a well-stirred context. The natural macroscopic model is the reaction-diffusion equation and it has the same limitations as the reaction rate equations. By discretizing the space with small subvolumes it is possible to model the reaction-diffusion process by a CTMC in the same formalism as for the well-stirred case. A diffusion event is now modeled as a first order reaction from a subvolume to an adjacent one and the state of the system is the number of molecules of each species in each subvolume. The corresponding master equation is called the reaction-diffusion master equation (RDME) and due to the very high dimensionality it cannot be solved by deterministic methods for realistic problem sizes.

The RDME has been used to study biochemical systems in [5, 12]. Here the next subvolume method (NSM) [5], an extension of Gibson and Bruck's next reaction method (NRM) [14], was suggested as an efficient method for realizing sample trajectories. An implementation on a structured Cartesian grid is freely available in the software MesoRD [18].

The method was extended to unstructured meshes in [10] by making connections to the finite element method (FEM). This has several advantages, but the most notable one is the ability to handle complicated geometries in a flexible way which is particularly important when internal structures of the cell must be taken into account.

This manual describes the software URDME which implements the unstructured extension of NSM as suggested in [10]. The purpose with the code is to provide an efficient, modular implementation that is easy to use for simulating and studying a particular model in an applied context, but also for developing and testing new approximate methods. We achieve this by relying on commercial software for the geometry definition, meshing, preprocessing and visualization and use a highly efficient computational core written in

`Ansi C`. This keeps the implementation of the actual Monte Carlo simulation small and easily extendible, while the user benefits from the advanced pre- and postprocessing capabilities of modern FEM software. In URDME, we have chosen to provide an interface to `Comsol Multiphysics 3.4` [3].

The rest of this manual is organized as follows. In Section 2 we recapitulate the mesoscopic reaction-diffusion model and show how the stochastic diffusion intensities are obtained from a FEM discretization of the diffusion equation. An overview of the code structure is also offered in Section 3. The details concerning the input to the code, the provided interface to `Comsol Multiphysics 3.4` and the way models should be specified are found in Section 4. Finally, two models are set up and simulated in a step-by-step manner in Section 5, which is followed in Section 6 by a short discussion of performance.

2 Background

In this section we briefly describe how reaction and diffusion events are modeled and how we obtain the diffusion rate constants when the domain is discretized using an unstructured mesh. For a fuller introduction to the subject along with many additional references, consult [7].

The computational core of URDME is based on the next subvolume method (NSM) [5], but it has been adapted to simulation on unstructured meshes by supporting a more general input format. Details concerning the actual simulation algorithms can be found in Appendix A.

2.1 Mesoscopic chemical kinetics

In a well-stirred chemical environment reactions are understood as transitions between the states of the integer-valued state space counting the number of molecules of each of D different species. The intensity of a transition is described by a *reaction propensity* defining the transition probability per unit of time for moving from the state x to $x - N_r$;

$$x \xrightarrow{\omega_r(x)} x - N_r, \tag{2.1}$$

where $N_r \in \mathbf{Z}^D$ is the transition step and is the rth column in the *stoichiometric matrix* N. Eq. (2.1) defines a continuous-time Markov chain over the positive D-dimensional integer lattice.

When the reactions take place in a container of volume Ω, it is sometimes useful to know that the propensities often satisfy the simple scaling law

$$\omega_r(x) = \Omega u_r(x/\Omega) \tag{2.2}$$

for some function u_r which does not involve Ω. Intensities of this form are called *density dependent* and arise naturally in a variety of situations [11, Ch. 11].

2.2 Mesoscopic diffusion

In the mesoscale model, a diffusion event is modeled as a first order reaction taking species S_l in subvolume ζ_i from its present subvolume to an adjacent subvolume ζ_j,

$$S_{li} \xrightarrow{a_{ij}\mathbf{x}_{li}} S_{lj}, \tag{2.3}$$

where $\mathbf{x}_{li}$ is the number of molecules of species l in subvolume i. On a uniform Cartesian mesh such as those used in MesoRD [18], the rate constant takes the value $a_{ij} = \gamma/h^2$ where h is the side length of the subvolumes and γ is the diffusion constant. In URDME we use an unstructured mesh made up of tetrahedra and the rate constants are taken such that the expected value of the number of molecules divided by the volume (the concentration) converges to the solution obtained from a consistent FEM discretization of the diffusion equation

$$u_t = \gamma \Delta u. \tag{2.4}$$

Using piecewise linear Lagrange elements and mass lumping, we obtain the discrete problem

$$u_t = M^{-1}Ku \tag{2.5}$$

where M is the lumped mass matrix and $K = \{k_{ij}\}$ is the stiffness matrix. The rate constants on the unstructured mesh are then given by

$$a_{ij} = \frac{1}{\Omega_i}k_{ij}, \tag{2.6}$$

where Ω_i is the diagonal entry of M and can be interpreted as the volume of the dual element associated with mesh node i (see Figure 2.1). For more details, consult [10].

The assumption made in the mesoscopic model is that molecules are well-stirred within a dual cell. These dual cells correspond to the cubes of the staggered grid in a Cartesian mesh.

3 Code overview

In this section we give an overview of the structure of the code. The computational core of URDME is an efficient implementation of NSM and additionally consists of a small set of routines written in `Ansi C`, `Matlab` and `Comsol`
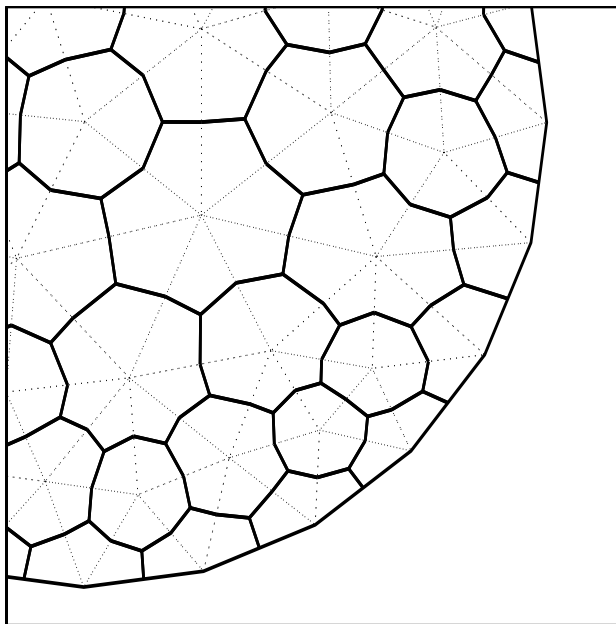
Figure 2.1: A 2D example of an unstructured triangular mesh. The primal mesh is shown in dashed and the dual in solid. Within each dual element the system is assumed to be well-stirred, and molecules can jump from each dual cell to the neighboring ones.

`Script`. Table 3.1 shows the directory structure of URDME together with a short description of each routine.

The software logically consists of three major parts. From bottom to top those are the `Ansi C` kernel, a `Matlab/Mex`-interface to the kernel and an interface to the FEM software. By design, the computational core is stand-alone, and in principle, for each FEM software a separate interface can be provided. This interface needs to export the internal representation of the problem and the mesh and convert it to the input format of `rdme_solve` which is detailed in Section 4. The actual call to `rdme_solve` is made via a gateway routine which could be a C `main()` function or as in our case, through a `Matlab/Mex` construction.

We have provided an interface to `Comsol Multiphysics 3.4`, which is used to specify the problem, create the mesh and provides for postprocessing. Since `Comsol Script` is very similar to the `Matlab` language, it is convenient to supply the gateway routine as a `Mex`-function using the `Matlab` C API. The `Comsol Script` routine `fem2rdme` assembles the matrix with diffusion rate constants D and the lumped mass matrix M. They are written to a `.mat` file together with information of the mesh and the different subdomains. This

5

Directory	File(s)	Description
src	rdme.c	The computational core `rdme_solve`.
	binheap.c	The binary heap used in `rdme_solve`.
	report.c	Report function used in `rdme_solve`.
	mexrdme.c	`Mex`-interface to `rdme_solve`.
include	rdme.h	Header for rdme.c.
	propensities.h.	Definition of the propensity function datatype.
	binheap.h	Header for binheap.c.
	report.h	Header for report.c.
comsol	fem2rdme.m	`Comsol`-script converting `Comsol`'s FEM-struct to a valid `rdme_solve`-input.
	rdme2fem.m	`Comsol`-script for conversion of the output of rdme.m to the solution format in `Comsol`. The purpose is to obtain a valid FEM-struct so that e.g. `Comsol`'s postprocessing can be used.
msrc	rdme.m	`Matlab`-wrapper for mexrdme. Handles error checking of the input.
	make.m	Makefile for building the `Mex`-file mexrdme.
	startup.m	Initializes URDME.
examples	(various)	Contains files specifying the two examples studied in detail in Section 5.
doc	manual.pdf	The most recent version of this manual.

Table 3.1: Overview of the files and routines that make up URDME.

.mat file can subsequently be loaded into `Matlab` where some of the other inputs are specified and the simulation is made via `rdme`.

The chemical reactions in the model can be specified in two different ways. Either as *inline propensities* specified from within `Matlab` or as *compiled propensities* where the rate functions are defined in a separate model file written in `Ansi C`. Inline propensities can be used to define basic polynomial

rate laws, while the compiled propensities are completely general. The details of this are found in Section 4.

When the model has been completely specified, the call to `rdme_solve` is made via the interface pair rdme.m/mexrdme.c.

At first, this construction may appear both cumbersome and inconvenient. However, the structure is transparent and makes it possible to separate the computational core from the actual FEM software used and development and maintenance of the core is made easier. Also, there are inputs to the stochastic simulation that are hard to standardize such as initial conditions or "exotic" diffusion over manifolds. Using `Matlab` as an interface facilitates handling of input and non-standard pre- and postprocessing since both the input and output data are available in `Matlab`. Hence the user can provide postprocessing routines which is particularly valuable in experimental stages of model development.

It is of course possible to eliminate the dependence on `Matlab` by providing another gateway in the form of for example a C `main()` routine, that directly reads the `.mat` file provided by the `Comsol` interface. This might be included in a later release.

4 Details and specifications

In this section we give a detailed description of the input to `rdme_solve` and we also give some details concerning the provided interface routines. Any gateway routine (see for example 'rdme.m' and 'mexrdme.c') should carefully ensure that the input has the correct format since the computational kernel puts very little effort in error checking.

4.1 Input to `rdme_solve`

Tables 4.1 and 4.2 summarize the input to `rdme_solve`. For precise type definitions, consult the preamble of the source file 'rdme.c'.

The diffusion matrix `D` and the (diagonal of) the lumped mass matrix `vol` as well as the vector giving the (generalized) subdomain number of each subvolume, `sd`, are generated by the `Comsol` interface routine `fem2rdme`. How `sd` can be specified in the `Comsol` GUI is explained in Section 5. The generalized data vector `data` may be used to pass any additional arguments to the *compiled* propensity functions.

The stoichiometry matrix `N`, the dependency graph `G`, the initial condition `u0`, the vector of output times `tspan` as well as the definition of the inline

propensities (if they are used) are explicitly specified in `Matlab` and passed on to `rdme_solve` via the `Mex`-interface in `rdme`, cf. Section 4.2.1.

If compiled propensities are used, they are specified as a vector of function pointers, `prop`. This is slightly more involved and is explained in Section 4.2.2.

The above mentioned data is passed to `sd` via the `Matlab` interface `rdme`. The other inputs (see 'rdme.c') are implicitly defined by the data structures, and are obtained automatically in `mexrdme`, where also the report function `report` is defined. The functionality of the report function is stated in the help section of 'rdme.m' and can easily be modified to suit the need of the user.

4.2   Specifying propensities for chemical reactions

We have provided two separate methods to specify the reaction propensities. Simple polynomial rate laws can be provided as inline propensities and can be specified on the gateway level (e.g. in `Matlab`).

The other option is to specify the rate laws in a model file written in C and recompile the `Mex`-interace. To simplify this process, we have supplied a makefile in '/msrc/make.m' using the `mex` command in `Matlab` with make options for two platforms (Linux and OS X). We have also provided an example GNU Makefile in '/src/Makefile' that allows full control of the make process. The example shows how to compile URDME on an Intel based MacBook using `gcc` and the Intel compiler `icc`. Both the `Matlab` and GNU Makefile will require some local editing, see Section 5.1.

For other configurations, it should be fairly straightforward to modify the Makefiles in order to compile the code. When using `Mex` the user needs to make sure that the configuration is set up correctly and that a proper compiler is installed. In Section 5 we show examples of using both the inline propensities and compiled propensities.

Note that one can easily use *both* inline and compiled propensities simultaneously. This is convenient when only a few propensities are complicated and has to be compiled.

4.2.1   Inline propensities

An "inline propensity" is a compact data format for specifying basic chemical reactions with polynomial rate laws. An inline propensity P can be defined

Name	Type	Description
Ncells	scalar (int)	Number of subvolumes.
Mspecies	scalar (int)	Number of different species. This also defines `Ndofs:=` `Mspecies`×`Ncells`.
Mreactions	scalar (int)	Number of reactions.
M1	scalar (int)	Number of inline propensities.
dsize	scalar (int)	Size of the data vector used in the propensity function.
u0	Matrix (`Mspecies`×`Ncells`) (int)	$u0(i,j)$ gives the initial number of species i in subvolume j.
tspan	vector (double)	An increasing sequence of points in time where the state of the system is to be returned.
prop	Vector[`Mreactions`-`M1`] (`PropensityFun`)	Propensity function pointers. See Section 4.2 for details.
report	ReportFun	Pointer to a report function. This function is called every time the chain reaches a value in `tspan`.
vol	Vector[`Ncells`] (double)	The volume of the macroelements, i.e. the diagonal elements of the lumped mass-matrix M.
sd	Vector[`Ncells`] (int)	The subdomain numbers of all subvolumes. See Section 5.2 for more details.
data	Matrix (`dsize`×`Ncells`) (double)	Generalized data vector. A pointer to column j is passed as an additional argument to the propensities in subvolume j.

Table 4.1: The first few input arguments to `rdme_solve`. See also Table 4.2.

9

Name	Type	Description
D	Sparse matrix (Ndofs× Ndofs) (double)	The *transpose* of the diffusion matrix $M^{-1}K$ obtained from the FEM discretization of the macroscopic diffusion equation, cf. (2.5). Each column in D (i.e. each row in $M^{-1}K$) corresponds to a subvolume, and the non-zero coefficients $D(i,j)$ give the diffusion rate constant from subvolume i to subvolume j.
N	Sparse matrix (Mspecies× Mreactions) (int)	The stoichiometry matrix. Each column corresponds to a reaction, and execution of reaction j amounts to *subtracting* the jth column from the state vector.
G	Sparse matrix (Mreactions× [Mspecies+Mreactions]) (int)	Dependency graph. The first Mspecies columns correspond to diffusion events and the following Mreactions columns to reactions. A non-zeros entry in element i of column j indicates that propensity i needs to be recalculated if the event j occurs. See Section 5 for examples.
K	Matrix (3 × M1) (double)	The rate constants of the inline propensities.
I	Matrix (3 × M1) (int)	Specifies which species are involved in each inline propensity.
S	Sparse matrix (M_S× M1) (int) where $M_S \in [0, \text{Ncells}]$	Column j contains a list of all subdomains in which the jth inline propensity is *off*. For more details concerning the inline propensities, see Section 4.2.

Table 4.2: Input arguments to `rdme_solve` (continued from Table 4.1). For more details, see the source file 'rdme.c'. For sparse matrices, the compressed column sparse (CCS) format is used. This is the same format `Matlab` uses and online documentation is available.

as

$$P(x) = \begin{cases} \frac{k_1 x_i x_j}{\Omega} + k_2 x_k + k_3 \Omega & \text{if } i \neq j, \\ \frac{k_1 x_i (x_i - 1)}{2\Omega} + k_2 x_k + k_3 \Omega & \text{if } i = j. \end{cases}$$

Here x is the column in $\mathbf{x}$ which contains the state of the subvolume considered and Ω is the corresponding volume. The coefficients and indices are specified in matrices K and I where $K(:, r) = [k_1; k_2; k_3]$ and $I(:, r) = [i; j; k]$ are the constants corresponding to the rth inline propensity. The matrix S is a (possibly empty) sparse matrix such that $S(:, r)$ lists all subdomains in which the rth inline propensity is turned off. *Note that no inline propensities are active in subdomain zero!*

4.2.2   Compiled propensities

The rate functions can also be supplied as an Ansi C model file. This is necessary if the rate functions are not simple polynomials or if they depend on additional input. Changing models in this setting amounts to recompiling the Mex gateway with the desired model file.

Any model file *must* implement the following routines:

- PropensityFun *ALLOC_propensities(void)

- void FREE_propensities(PropensityFun *ptr)

The first function should allocate and initialize an array of function pointers to the propensity functions and return a pointer to this array. The second function should deallocate the pointer ptr but sometimes additional actions need to be implemented.

An important point is that neither Ansi C's malloc/free nor Matlab's mxMalloc/mxFree should be invoked explicitly. Instead it is preferred to use the defines MALLOC/FREE which can be modified at the stage of compilation (see for example '/msrc/make.m')

The datatype PropensityFun is defined in the header 'propensities.h' (found in the 'include' directory) as

```
typedef double (*PropensityFun)(const int *, double,
                                const double *, int);
```

Below is a commented example of a model file defining a simple chemical system composed of a single species X undergoing a dimerization reaction.

```c
/* Propensity definition of a simple dimerization reaction. */
#include <stdlib.h>
#include <stdio.h>
/* Type definition of propensity functions: */
#include "propensities.h"

/* Rate constant (in units M^{-1}s^{-1}). */
const double k = 1e-3;

double rFun1(const int *x, double vol,
             const double *data, int sd)
/* Propensity for the reaction X + X -> 0. */
{
  return k*x[0]*(x[0]-1)/vol;
}


PropensityFun *ALLOC_propensities(void)
/* Allocation. */
{
  PropensityFun *ptr = MALLOC(sizeof(PropensityFun));
  ptr[0] = rFun1;

  return ptr;
}

void FREE_propensities(PropensityFun *ptr)
/* Deallocation. */
{
  FREE(ptr);
}
```

More advanced examples can be found in the model files found in the 'examples' directory; '/bistab/bistab.c' and '/Min/Min.c' corresponding to the examples discussed in Section 5. There we also explain how to compile and link the propensity functions.

5   Examples

In this section we provide a step-by-step description of how to set up and simulate two different models from the systems biology literature. The first

is a model of a bistable system displaying phase separation when the species are diffusing slowly. This model was first studied in [5] for simple geometries. In this example, we will simulate the system in a model of a *S. cerevisiae* cell containing a nucleus and a large vacuole and we will see the ease with which we handle inner (curved) boundaries.

The second example is a model of protein oscillations in a model of an *E. coli* cell proposed in [12]. Here we will illustrate how to use different subdomains in order to incorporate membrane diffusion in the model.

5.1 Installing URDME

There is no automated installation procedure. Simply download the source and decompress it in a directory of your choice (it will result in the path to the code being '/yourdir/urdme/'). There are a few path dependencies that need local editing. In the following, all paths are given relative to '/yourdir/urdme/'.

1. If you plan to use the GNU Makefile, open it ('/src/Makefile') and edit the variables `PREFIX`, `MATLAB_EXT` and `MATLAB_LIB`. In addition, you may need to modify other variables, depending on your configuration.

2. Open the file '/msrc/make.m'. In `Matlab`, use the function `mexext` to find out the extension of `Mex`-files on your platform. If it is something different from the provided examples ('mexa64' and 'mexmaci'), you will have to add an option similar to the existing ones. The compile and link flags of the first example ('mexa64') are likely to work for most Linux flavors and for Solaris (using gcc). The second option will likely work for a wide range of OS X configurations. More information on how to build a valid `Mex`-file on your platform can be obtained from the file 'mexopts.sh' (or 'mexopts.bat' in a Windows environment) in your `Matlab` installation directory.

Before proceeding, make sure that URDME compiles. In `Matlab`, change the directory to '/msrc' and type

```
>> make ../examples/bistab/bistab.c
```

or using the GNU Makefile, from a bash-shell, change the directory to '/src' and type

```
$ make MODEL=../examples/bistab/bistab
```

If `mexrdme` was built successfully, you can proceed to the examples in the two following sections.

Currently, there is no support for Windows. To use URDME in a Windows environment we recommend installing `gcc` for Windows (such as in

Mingw) and using for example the freely available tool `gnumex` to configure `mex` to use `gcc`.

5.2   General workflow

In this section we describe the general workflow involved in setting up and simulating a model using the `Comsol` and `Matlab` interfaces. In the following sections, we then consider two specific examples in some detail.

1. *Specify the model.* In order to do this, you need to define the geometry as well as the chemical reactions in the model. In `Comsol Multiphysics 3.4` a geometry can be created and a discretization of the diffusion equation with Neumann boundary conditions is readily obtained. In `Matlab`, a model file that defines the stoichiometry matrix N and the dependency graph G can easily be specified. Depending on how the reactions are to be given, this file may also define the inline propensities.

2. *Convert the FEM structure to valid input.* After meshing your model, export the FEM structure to `Comsol Script` and create a model `.mat` file for use by the `Matlab` interface. This can be achieved by calling the interface routine `fem2rdme`.

3. *Run the simulation.* Load the model file in `Matlab`, specify initial conditions and (if you are using compiled propensities) compile the gateway `mexrdme` using either 'make.m' or the GNU makefile, cf. Section 5.1. Then run the simulation by calling `rdme_solve` via the interface `rdme`.

4. *Postprocessing.* Save the result to a result `.mat` file. In `Comsol Script`, load this file and add the solution to the FEM struct previously exported by calling `rdme2fem`. At this point, you can use all postprocessing options available in `Comsol`. If you have other needs not covered by the built-in routines, you can implement your own routines in e.g. `Matlab`.

5.3   A bistable system in *S. cerevisiae*

In this section we will simulate the model from [5] in a spherical volume modeling a Brewer's yeast cell. It contains a nucleus and a large vacuole, which will be treated as solid objects into which no molecules can enter. The set of chemical reactions in the system can be found in Table 5.1. All the files needed to complete this example can be found in the directory '/examples/bistab/'.

$$E_A \xrightarrow{k_1} E_A + A \qquad E_B \xrightarrow{k_1} E_B + B$$
$$E_A + B \underset{k_d}{\overset{k_a}{\rightleftharpoons}} E_A B \qquad E_B + A \underset{k_d}{\overset{k_a}{\rightleftharpoons}} E_B A$$
$$E_A B + B \underset{k_d}{\overset{k_a}{\rightleftharpoons}} E_A B_2 \qquad E_B A + A \underset{k_d}{\overset{k_a}{\rightleftharpoons}} E_B A_2$$
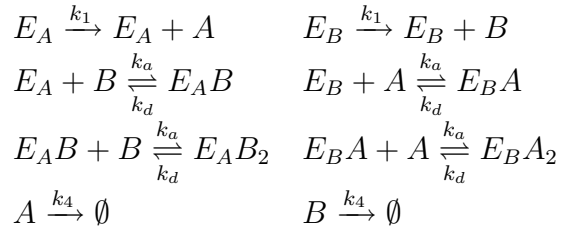$$A \xrightarrow{k_4} \emptyset \qquad\qquad B \xrightarrow{k_4} \emptyset$$

Table 5.1: The chemical reactions of the bistable model. The constants take the values $k_1 = 150 s^{-1}$, $k_a = 1.2 \times 10^8 s^{-1} M^{-1}$, $k_d = 10 s^{-1}$ and $k_4 = 6 s^{-1}$.

1. Start `Comsol Multiphysics 3.4`. In the Model navigator, select 'Open' and open the file 'bistab.mph'. This model file specifies the yeast geometry and in the Draw mode you can study the different subdomains.

2. The model is composed of three subdomains. We will treat the inner structures as solid objects that molecules cannot enter. This can be achieved in different ways. One of them is to set the diffusion coefficient identically to zero in these subdomains. Here we will remove these inner structures and create holes in the geometry which reduces the number of degrees of freedom. In the Draw mode, select all three subdomains by shift-clicking on them in turn. In the 'Draw' menu, select 'Create Composite Object' and choose 'Difference' (this can also be done using the action buttons in the draw mode action bar). Next, again in the 'Draw' menu, select 'Delete interior boundaries'.

3. Next we will initialize the mesh. In the Mesh menu, choose 'Free Mesh Parameters'. Set the maximum element size to $0.1 \times 10^{-6} m$ and click on the 'Remesh' button.

4. The parameters in the model (in this case the diffusion coefficients of the different species) can be edited in the menu 'Physics – Subdomain Settings'. Choose subdomain one and check that the species all have the diffusion constant $\gamma = 0.5 \times 10^{-13} m^2/s$.

5. We are now ready to export the FEM-structure to `Comsol Script` in order to prepare for the simulation. In the 'File' menu, choose 'Export – FEM Structure...' and choose the name of the struct, e.g. 'fem'. This opens the `Comsol Script` window and you can display the contents in 'fem' by simply typing
`C>> fem`

6. We have now specified the geometry and will next create the input to

`rdme_solve`. First, change the working directory to '/msrc' and set the paths by calling `startup`.

```
C>> cd /yourdir/urdme/msrc;
C>> startup;
```

In order to assemble the input to `rdme_solve`, call the interface routine `fem2rdme`,
```
C>> fem2rdme(fem,'bistab');
```
This will generate the file 'bistab.mat' that contains the diffusion matrix, the subvolume sizes as well as information regarding the different subdomains (in this example there is only one).

7. Start `Matlab`, change the working directory to '/msrc' and set the paths as above. Then load the model file we just created
```
>> load bistab.mat;
```
and examine the variables. We will now specify the set of chemical reactions. There are two different ways of doing this; using inline propensities or using compiled propensities, see Section 4. Here we will use the compiled propensities specified in the model file 'bistab.c'. Open the file and look at the reaction definitions. Compile the code with these propensities
```
>> make ../examples/bistab.c;
```

8. Before we can run the simulation, we need to specify the stoichiometry matrix `N` and the dependency graph `G`. They are both defined in the script 'bistab.m'. Run it by invoking
```
>> bistab;
```

9. We have yet to specify the initial conditions. Let the enzymes E_A and E_B be present in a total of 100 molecules, randomly distributed in the whole domain. The other species are set to zero everywhere. Something like this can be achieved by

```
>> ind1 = randperm(Ncells);
>> ind2 = randperm(Ncells);
>> u0(3,ind1(1:100))=1;
>> u0(4,ind2(1:100))=1;
```

10. At this point, we are ready to run the simulation by invoking `rdme_solve` through `rdme`;
```
>> bistab_sol = rdme(0:5:1000,u0,D,N,G,vol,sd,[2 123]);
```

16

When the simulation has finished, save the solution and the volume vector (it is needed for the postprocessing in order to scale the solution to local concentrations).
```
>> save('bistab_results', 'bistab_sol', 'vol');
```

11. In `Comsol Script`, load the solution
   ```
   C>> load bistab_results.mat
   ```
   and add the solution to the FEM-struct using the interface routine `rdme2fem`;
   ```
   C>> fem = rdme2fem(bistab_sol, vol, 0:5:1000);
   ```

12. In the main `Comsol` window, import the FEM structure that now contains the solution. This is done in the File menu; 'File – Import – FEM Structure...'. Change mode to 'Plot mode' and look at the solution. You can change variables and the type of plot in the menu 'Postprocessing – Plot Parameters...', where you also have the possibility to make animations. This can also be done directly in `Comsol Script` using the functions `postplot` and `postmovie`, see the documentation for more details.

Figure 5.3 shows an example of the output from a simulation of the model. For more details concerning the interpretation of the results, see [5, 10].

5.4   MinD/MinE oscillations in *E. coli*

In this section we will reproduce simulations of the MinD/MinE system studied in [12] in a model of an *E. coli* bacterium. It is rod shaped with length 3.5µm and diameter 0.5µm. The reactions and parameters of the model can be found in Table 5.2.

$$\text{MinD\_c\_atp} \xrightarrow{k_d} \text{MinD\_m} \qquad \text{MinD\_c\_atp} + \text{MinD\_m} \xrightarrow{k_{dD}} 2\text{MinD\_m}$$
$$\text{Min\_e} + \text{MinD\_m} \xrightarrow{k_{de}} \text{MinDE} \quad \text{MinDE} \xrightarrow{k_e} \text{MinD\_c\_adp} + \text{Min\_e}$$
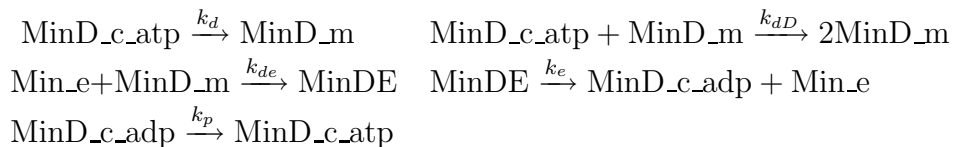$$\text{MinD\_c\_adp} \xrightarrow{k_p} \text{MinD\_c\_atp}$$

Table 5.2: The chemical reactions of the MinD/MinE model. The constants take the values $k_d = 0.0125 \mu m^{-1}s^{-1}$, $k_{dD} = 9 \times 10^6 M^{-1}s^{-1}$, $k_{de} = 5.56 \times 10^7 M^{-1}s^{-1}$, $k_e = 0.7s^{-1}$ and $k_p = 0.5s^{-1}$.

Two of the species, `MinD_m` and `MinDE` are bound to the membrane and can not diffuse in the cytoplasm. In order to incorporate this in the model we need to define two different subdomains, the membrane and the cytosol,
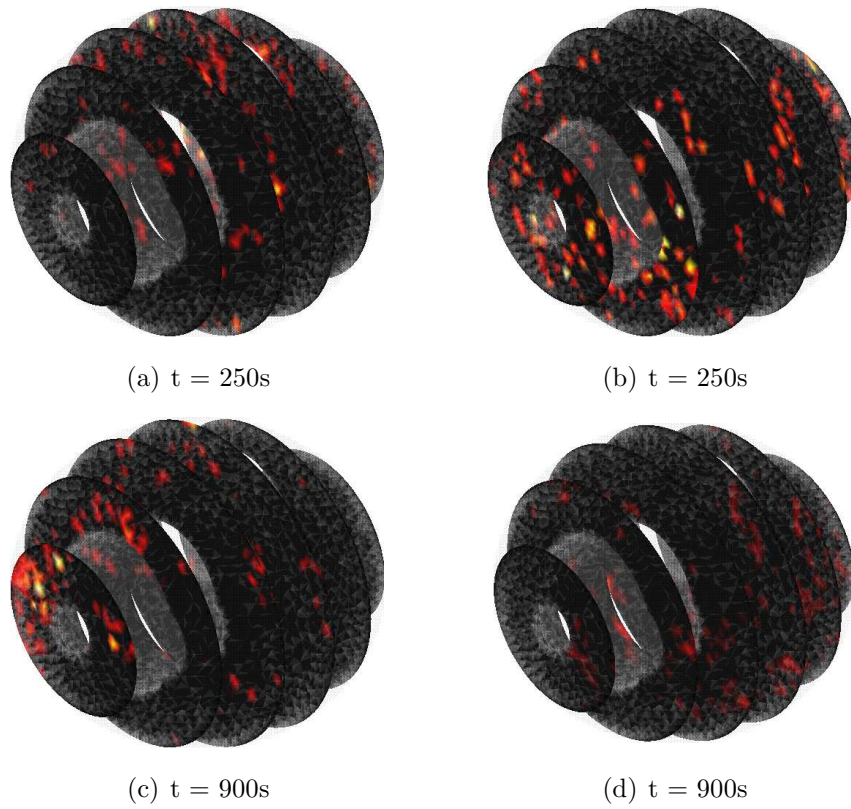
(a) t = 250s   (b) t = 250s

(c) t = 900s   (d) t = 900s

Figure 5.1: Snapshots of the system simulated with $\gamma = 0.5 \times 10^{-13} m^2/s$. The left column shows species A and the right species B. They are present in different parts of the geometries, and the global behavior does not display bistability.

and specify which reactions take place in each subdomain. We also need to modify the diffusion matrix in order to inactivate diffusion events from the membrane into the interior of the cell. This is achieved by using the subdomain data vector `sd`.

Note that the subdomain data vector `sd` is a completely general piece of information. In particular, it need not be explicitly related to the 'subdomain' numbers in `Comsol`.

One of the reactions taking place on the membrane has a scaling with a local length scale. In this example, we will use inline propensities to which we pass the average of the side length of the tetrahedra on the boundary. If compiled propensities were used, it would instead be possible to pass the actual length of each individual subvolume to the propensity functions using the generalized data matrix `data`. All the files used in this example can be found in the directory '/examples/Min'.

1. Open `Comsol` and select 'Chemical engineering module – Mass transport – Diffusion – Transient analysis' (3D). In the 'Dependent variables' field write `MinD_c_atp, MinD_m, Min_e, MinDE, MinD_c_adp`. These are the names of the variables that we will use. Select *Lagrange – Linear* elements and press 'OK'.

2. Next we create the geometry. We will build the rod shaped domain from two spheres and one cylinder. Press the 'Cylinder' button and in the radius and height field enter `0.5e-6` and `3.5e-6` and press 'OK'. You should now see a cylinder in your workspace. In the 'Draw mode' action bar, press 'Zoom extents' in order to get a better view of the domain. Press the 'sphere' button and enter `0.5e-6` in the radius field and press 'OK'. Create another identical sphere but enter `3.5e-6` as the z-coordinate. Select all three figures and press the 'union' button and then the 'Delete interior boundaries' button.

3. Having defined the geometry, the next step is to specify the parameters in the model. In the menu 'Physics – Subdomain settings', choose subdomain 1 and set the diffusion constants to `2.5e-12` for `MinD_c_adp`, `MinD_c_atp` and `Min_e`. For `MinDE` and `MinD_m` the diffusion constant should be `1e-14`.

4. We must also create two domains. One interior domain that represent the cytoplasm and one boundary domain that represent the membrane. This is done by defining `rdme_sd` as an expression with different value in the different subdomains. It can then be used to find the nodes on

19

the boundary and in the interior. Select 'Options – Subdomain expressions' and enter `rdme_sd` with value 1 and click 'OK'. Select 'Options – Boundary expressions' and select all boundaries (there should be 12 of them). Enter `rdme_sd` with value 2. Finally select 'Options - Global expressions' and enter `rdme_sdlevel` with value 2 indicating that the *lowest* dimension where `rdme_sd` is defined was on the surfaces.

5. In the 'Mesh' menu, select 'Mesh – Free mesh parameters' and choose 'Custom mesh size'. Set the maximum element size to `0.7e-7` and press 'Initialize mesh'. Now select 'Solve – Update model' and export the FEM structure from the 'File – export' menu. This will open `Comsol Script` and the FEM structure will be available in the workspace.

6. The information we need regarding the discretization of the geometry is stored in the FEM structure (call it `fem`). At this point, we need to transform the information in `fem` into the input format of `rdme_solve`. This is done by calling the interface routine `fem2rdme`. First, initialize URDME by calling `startup` (from the '/msrc' directory). Then, assemble the input and store it in the file 'min.mat' by
`C>> fem2rdme(fem,'min',1);` This will save the variables `u0`, `D`, `vol`, `data` and `sd` to the file `min.mat`. In order to make sure that the diffusion of the species `MinD_m` and `MinDE` are active only on the boundary, type the following:

```
C>> load min;
C>> icells = find(sd==1);
C>> D = min_membrane(D,icells,[2 4],5);
C>> save('min','tspan','u0','D','vol','data','sd','-mat');
```

This will set the diffusion rate constants to zero in the interior. The first command loads the variables that `fem2rdme` saved. The second command gives the indices to the cells in subdomain one, which we defined to be the interior of the domain by the expression `rdme_sd`. The third one calls the script `min_membrane` which turns off the diffusion for species 2 and 4 (`MinD_m` and `MinDE`). Finally, we rewrite the file `min.mat` with the modified matrix `D`.

7. We are now ready to specify the part of the model related to the chemical reactions. Open `Matlab` and initialize URDME by calling `startup`. Load the file `min.mat`. Next, in a new `Matlab`-script create the stoichiometric matrix and the dependency graph which are used to update the state due to chemical reactions and to minimize the calculation of

20

reaction rates respectively. If you want to skip this step, just use the script `Min.m`, where they are already defined.

$$
\mathbf{N} = \begin{bmatrix} 1 & 1 & 0 & 0 & -1 \\ -1 & -1 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & -1 & 1 \end{bmatrix},
$$

$$
\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}.
$$

We will use inline propensities so we also need to create `K`, `I`, and `S`

$$
\mathbf{K} = \begin{bmatrix} 0 & k_{dD} & k_{de} & 0 & 0 \\ k_d & 0 & 0 & k_e & k_p \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.
$$

Note the unit of k_d; it involves the local length scale of the computational cells at the boundary. Using inline propensities, we cannot pass this to the propensity functions. Instead, we use the average length scale and modify k_d accordingly. The local length scale can be obtained using the `Comsol` routine `posteval` and for this mesh size the average at the boundary takes the value 0.041×10^{-6}.

$$
\mathbf{I} = \begin{bmatrix} 1 & 1 & 3 & 1 & 1 \\ 1 & 2 & 2 & 1 & 1 \\ 1 & 1 & 1 & 4 & 5 \end{bmatrix},
$$

$$
\mathbf{S} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 \end{bmatrix}.
$$

8. We have now specified both the geometry, the chemical reactions and the dependency graphs. Before we run the simulation, we need to set the initial condition. Set the initial number of molecules to 2001 `MinD_c_atp`, 2001 `MinD_c_adp` and 1040 `MinE`, randomly distributed.

```
>> mind=[ceil(rand(2001,1)*Ncells)];
>> mind2=[ceil(rand(2001,1)*Ncells)];
>> mine=[ceil(rand(1040,1)*Ncells)];
>> a=sparse(mind,1,1,Ncells,1);
```

```
>> b=sparse(mine,1,1,Ncells,1);
>> c=sparse(mind2,1,1,Ncells,1);
>> u0=[a zeros(Ncells,1) b zeros(Ncells,1) c]';
>> u0=full(u0);
```

9. Set `tspan=[0:30]` and start the simulation by:
   ```
   >> U = rdme(tspan,u0,D,N,G,vol,data,sd, [2 123], K, I, S);
   ```
   After the simulation has finished, save the solution on a `.mat` file, e.g.
   ```
   >> save('min_results','U');
   ```

10. To visualize the trajectory, load the result in the same `Comsol Script` window used before. Make sure that the variable `vol` is in your workspace and convert the result to `Comsol`'s internal representation using the interface routine `rdme2fem`;
    ```
    >> fem = rdme2fem(fem,U,vol,[0:30]);
    ```

11. Import the FEM structure into the `Comsol` GUI from the 'File - Import' menu and plot the solution in the postprocessing mode. For example of output generated by this model, see Figure 5.2, where we have plotted the variable `MinD_m` on the membrane using 'Boundary Plot' from the 'Plot Parameters' menu.

6 Performance

The purpose of this section is to give an indication of the performance of the code by simulating the model in Section 5.3 on increasingly fine meshes. The time required to simulate a particular model will obviously depend on the parameters since the average timestep is the inverse of the sum of all reaction- and diffusion rates. For example, the fraction of diffusion to reaction events increases with the resolution and the number of events increase linearly with the diffusion constant. Consequently, it is hard to tell in advance how many events that are required to reach a given final time. In Table 6.1 we show for fixed parameters in the model how the number of events simulated per hour depends on the number of subvolumes.

The geometry was taken to be a cube with side length 6×10^{-6}m. At $t = 0s$, there were 300 molecules each of E_A and E_B, randomly distributed and all other species were set to zero. The final time was chosen to be $T = 300s$, and the state was saved every three seconds. The timings in Table 6.1 does not include the time taken to generate the mesh and the input to URDME, which was much less than the simulation time.
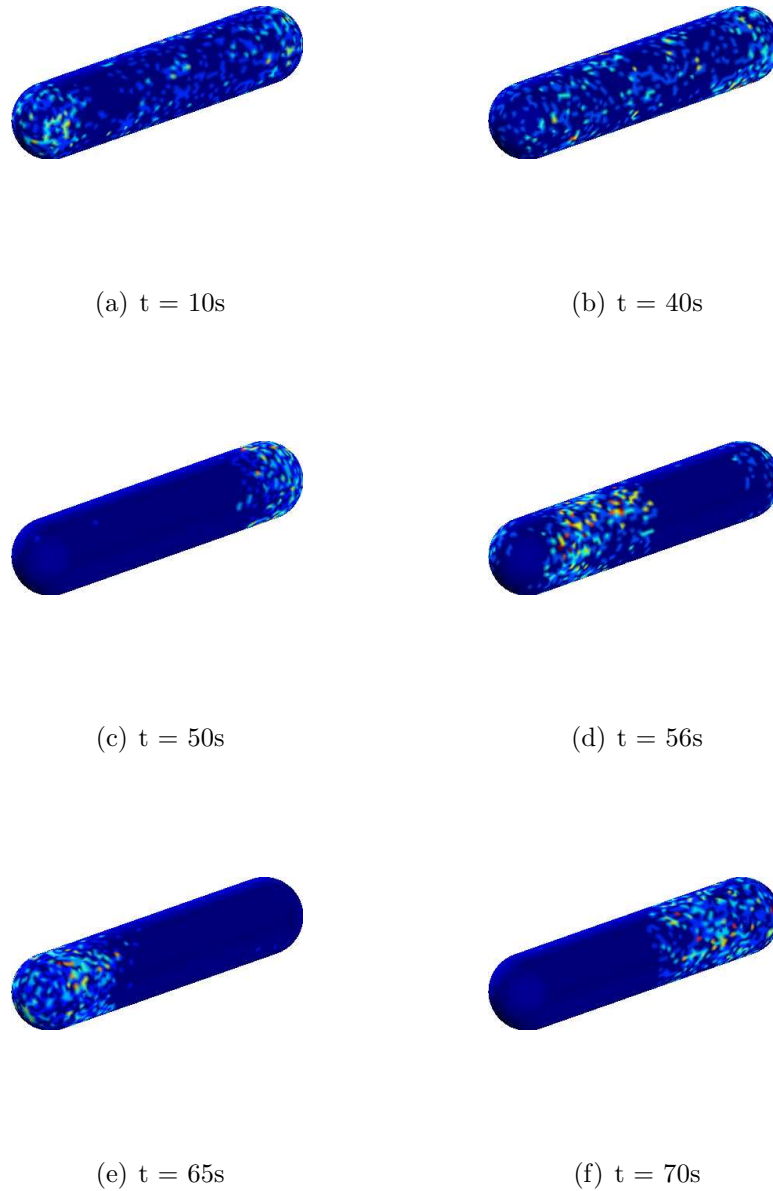
(a) t = 10s

(b) t = 40s

(c) t = 50s

(d) t = 56s

(e) t = 65s

(f) t = 70s

Figure 5.2: Simulation of the MinD model in *E. coli*. At $t = 0s$, no MinD is present on the membrane. After an initial period when the number of molecules are increasing over the whole membrane in (a)–(b), the membrane bound MinD oscillates from pole to pole in (c)–(f).

Ncells	11,107	25,608	83,843
CPU time [s]	155	230	667
% diffusion events	55.7	69.1	83.3
events/h [$\times 10^9$]	2.04	1.95	1.24

Table 6.1: Performance when the spatial resolution is increased. Note that the fraction of diffusion events increases with the resolution of the mesh.

The simulations were made on a Macbook Core Duo 2.0 Ghz with 2GB RAM, using compiled propensities and compiled using `gcc 4.3.2` with optimization flags `-O3 -ftree-vectorize`.

Acknowledgment

References

[1] Naama Barkai and Stanislav Leibler. Circadian clocks limited my noise. *Nature*, 403:267–268, 2000.

[2] Yang Cao, Dan T. Gillespie, and Linda Petzold. Multiscale stochastic simulation algorithm with partial equilibrium assumption for chemically reacting systems. *J. Comput. Phys.*, 206:395–411, 2005.

[3] Comsol Inc., Stockholm, Sweden. *Comsol Multiphysics Reference Guide Version 3.4*, 2008. http://www.comsol.com.

[4] Weinan E, Di Liu, and Eric Vanden-Eijnden. Nested stochastic simulation algorithm for chemical kinetic systems with disparate rates. *J. Chem. Phys.*, 123, 194107, 2005.

[5] Johan Elf and Måns Ehrenberg. Spontaneous separation of bi–stable biochemical systems into spatial domains of opposite phases. *Syst. Biol.*, 1(2), 2004.

[6] Michael B. Elowitz, Arnold J. Levine, Eric D. Siggia, and Peter S. Swain. Stochastic gene expression in a single cell. *Science*, 297(5584):1183–1186, 2002.

[7] Stefan Engblom. *Numerical Solution Methods in Stochastic Chemical Kinetics*. PhD thesis, Uppsala University, 2008.

[8] Stefan Engblom. Spectral approximation of solutions to the chemical master equation. *J. Comput. Appl. Math.*, 2008 *(to appear)*.

[9] Stefan Engblom. Galerkin spectral method applied to the chemical master equation. *Commun. Comput. Phys.*, 5(5):871–896, 2009 *(to appear)*.

[10] Stefan Engblom, Lars Ferm, Andreas Hellander, and Per Lötstedt. Simulation of stochastic reaction–diffusion processes on unstructured meshes. *SIAM J. Scientific. Comp.*, 2008 *(to appear)*.

[11] Stewart N. Ethier and Thomas G. Kurtz. *Markov Processes: Characterization and Convergence*. Wiley series in Probability and Mathematical Statistics. John Wiley & Sons, New York, 1986.

[12] David Fange and Johan Elf. Noise-induced Min phenotypes in *E. coli. PLOSB*, 2(6):0637–0647, 2006.

[13] Lars Ferm and Per Lötstedt. Adaptive solution of the master equation in low dimensions. *Appl. Numer. Math.*, 59(1):265–284, 2009.

[14] Michael A. Gibson and Jehoshua Bruck. Efficient exact stochastic simulation of chemical systems with many species and many channels. *J. Phys. Chem.*, 104:1876–1889, 2000.

[15] Daniel T. Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reacting systems. *J. Comput. Phys.*, 22:403–434, 1976.

[16] Daniel T. Gillespie. Approximate accelerated stochastic simulation of chemically reacting systems. *J. Chem. Phys.*, 115(4):1716–1733, 2001.

[17] Eric L. Haseltine and James B. Rawlings. Approximate simulation of coupled fast and slow reactions for stochastic chemical kinetics. *J. Chem. Phys.*, 117(15):6959–6969, 2002.

[18] Johan Hattne, David Fange, and Johan Elf. Stochastic reaction–diffusion simulation with MesoRD. *Bioinformatics*, 21(12):2923–2924, 2005.

[19] Markus Hegland, Conrad Burden, Lucia Santoso, Shev MacNamara, and Hilary Booth. A solver for the stochastic master equation applied to gene regulatory networks. *J. Comput. Appl. Math.*, 205(2):708–724, 2007.

[20] Andreas Hellander. Numerical simulation of well stirred biochemical reaction networks governed by the master equation. *Licentiate thesis, Department of Information Technology, Uppsala University*, 2008.

[21] Andreas Hellander and Per Lötstedt. Hybrid method for the chemical master equation. *J. Comput. Phys.*, 227(1):127–151, 2008.

[22] Shev F. MacNamara. *Krylov and Finite State Projection Methods for Simulating Stochastic Biochemical Kinetics via the Chemical Master Equation.* PhD thesis, The University of Queensland, Australia, 2008.

[23] Harley H. McAdams and Adam Arkin. It's a noisy business! Genetic regulation at the nanomolar scale. *Trends in Genetics*, 15(2):65–69, 1999.

[24] Johan Paulsson, Otto G. Berg, and Mans Ehrenberg. Stochastic focusing: Fluctuation-enhanced sensitivity of intracellular regulation. *Proc. Nat. Acad. Sci. USA*, 97(13):7148–7153, 2000.

[25] Christopher V. Rao and Adam P. Arkin. Stochastic chemical kinetics and the quasi-steady-state assumption: Application to the Gillespie algorithm. *J. Chem. Phys.*, 118(11):4999–5010, 2003.

[26] Paul Sjöberg. *Numerical Methods for Stochastic Modeling of Genes and Proteins.* PhD thesis, Uppsala University, 2007.

[27] Mukund Thattai and Alexander van Oudenaarden. Intrinsic noise in gene regulatory networks. *Proc. Nat. Acad. Sci. USA*, 98:8614–8619, 2001.

A   Algorithms

One of the most popular algorithms to generate realizations of the CTMC in the well-stirred case is Gillespie's direct method (DM) [15]. Several algorithmic improvements of this method exist, one of them being the next reaction method (NRM) due to Gibson and Bruck [14].

The underlying algorithm in URDME is the next subvolume method (NSM) [5]. The NSM essentially combines ideas from NRM and DM in order to tailor the algorithm to reaction-diffusion processes.

For reference, we first state below both DM and NRM and then outline NSM.

Algorithm 1 Gillespie's direct method (DM)

Initialize: Set the initial state $\mathbf{x}$ and compute all propensities $\omega_r(\mathbf{x}), r = 1, \ldots, M_{\text{reactions}}$. Also set $t = 0$.

while $t < T$ **do**

   Compute the sum λ of all the propensities.

   Sample the next reaction time (by inversion), $\tau = -\log(\text{rand})/\lambda$. Here and in what follows, 'rand' conveniently denotes a uniformly distributed random number in $(0,1)$ which is different for each occurrence.

   Sample the next reaction event (by inversion); find n such that
   $\sum_{j=1}^{n-1} \omega_j(\mathbf{x}) < \lambda \, \text{rand} \leq \sum_{j=1}^{n} \omega_j(\mathbf{x})$
   Update the state vector, $\mathbf{x} = \mathbf{x} - N_n$ and set $t = t + \tau$.

end while

Algorithm 2 Gibson and Bruck's next reaction method (NRM)

Initialize: Set $t = 0$ and assign the initial number of molecules. Generate the dependency graph G. Compute the propensities $\omega_r(\mathbf{x})$ and generate the corresponding *absolute* waiting times τ_r for all reactions r. Store those values in a heap H.

while $t < T$ **do**

    Remove the smallest time $\tau_n = H_0$ from the top of H, execute the nth reaction $\mathbf{x} := \mathbf{x} - N_n$ and set $t := \tau_n$.

    for all edges $n \rightarrow j$ in G **do**

      if $j \neq n$ **then**

        Recompute the propensity ω_j and update the corresponding waiting time according to

$$\tau_j^{\text{new}} = t + \left(\tau_j^{\text{old}} - t\right) \frac{\omega_j^{\text{old}}}{\omega_j^{\text{new}}}.$$

      else $\{j = n\}$

        Recompute the propensity ω_n and generate a new absolute time τ_n^{new}. Adjust the contents of H by replacing the old value of τ_n with the new one.

      end if

    end for

end while

Algorithm 3 The next subvolume method (NSM)

Initialize: Compute the sum σ_i^r of all reaction rates ω_{ri} and the sum σ_i^d of all diffusion rates $a_{ij}\mathbf{x}_{si}$ in all subvolumes $i = 1, \ldots, N_{\text{cells}}$. Compute the time until the next event in each subvolume, $\tau_i = -\log(\text{rand})/(\sigma_i^r + \sigma_i^d)$, and store all times in a heap H.

while $t < T$ **do**

  Select the next subvolume ζ_n where an event takes place by extracting the minimum τ_n from the top of H.

  Set $t = \tau_n$.

  Determine if the event in ζ_n is a reaction or a diffusion event. Let it be a reaction if $(\sigma_n^r + \sigma_n^d)\,\text{rand} < \sigma_n^r$, otherwise it is a diffusion event.

  if Reaction event **then**

    Determine the reaction channel that fires. This is done by inversion of the distribution for the next reaction given τ_n in the same manner as in Gillespie's direct method in Algorithm 1.

    Update the state matrix using the (sparse) stoichiometry matrix N.

    Update σ_n^r and σ_n^d using the dependency graph G to recalculate only affected reaction- and diffusion rates.

  else {Diffusion event}

    Determine which species S_{ln} diffuses and subsequently, determine to which neighboring subvolume $\zeta_{n'}$. This is again done by inversion using a linear search in the corresponding column of D.

    Update the state: $S_{nl} = S_{nl} - 1$, $S_{n'l} = S_{n'l} + 1$.

    Update the reaction- and diffusion rates of subvolumes ζ_n and $\zeta_{n'}$ using G.

  end if

  Compute a new waiting time τ_n by drawing a new random number and add it to the heap H.

end while