UNIVERSITÀ DEGLI STUDI DI TORINO

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI

Corso di Laurea Magistrale in Metodologie e Sistemi Informatici

# Simulation and Analysis of Chemical Reactions using Stochastic Differential Equations

Relatore:

Prof.ssa Susanna Donatelli

Correlatore:

Dr. Stephen Gilmore

Tesi di Laurea di:

Luca Cacchiani

Anno Accademico 2006/2007

# Contents

**1 Introduction** **3**

1.1 Background to the research . . . . . 5

1.2 Overview of the Thesis . . . . . 5

**2 Related works** **7**

**3 Simulation methods for solving chemical reaction systems** **9**

3.1 Deterministic Solution . . . . . 11

3.2 Stochastic Solution . . . . . 13

3.2.1 The Master Equation Approach . . . . . 14

3.2.2 Stochastic Simulation Approach . . . . . 16

3.3 Stochastic Differential Equations Solution . . . . . 18

3.4 Relationships between deterministic and stochastic models . . . . . 19

**4 Dizzy: a chemical kinetics simulation software** **22**

4.1 Package structure . . . . . 24

4.2 Stochastic and deterministic simulators . . . . . 25

4.3 Graphical user and command line interface . . . . . 29

**5 Statistical analysis of results** **33**

5.1 Confidence Intervals . . . . . 34

5.2 Candlestick chart . . . . . 36

5.3 Profiling . . . . . 38

5.3.1 The GAL model . . . . . 41

**6 Working on Stochastic Differential Equations** **48**

6.1 Numerical Solution . . . . . 48

6.2 Experiments . . . . . 51

6.2.1 The Michaelis-Menten model . . . . . 52

6.2.2 The Schlögl model . . . . . 52

6.2.3 The Lotka-Volterra model . . . . . 56

6.2.4 Final results . . . . . 57

6.2.5 Concluding remarks . . . . . 58

**7 Other works on Dizzy** **59**

7.1 Sensitivity Analysis . . . . . 60

7.2 Deadlock Analysis . . . . . 60

**8 Concluding Remarks and Future Works** **63**

**A Yeast model** **66**

A.1 Template definitions . . . . . 66

A.2 Galactose pathway . . . . . 69

**Bibliography** **74**

# List of Figures

3.1 Map of the main stochastic simulators developed from 1977 in order to solve chemical reactions. All the "improved" SSA implementations do not imply that there is anything wrong with the original SSA procedure by Gillespie: on the contrary, they either optimise or approximate Gillespie's Direct Method. . . . . 10

4.1 Simulator structure in Dizzy: from the common abstract class `Simulator`, four abstract classes define the common simulation methods for each main algorithm. This enables a very simple method to create new simulators, allowing the developer to focus only the core of simulators. . . . . 26

4.2 Dizzy is able to simulate models written in the Systems Biology Markup Language (SBML), in addition to the Chemical Model Description Language (CMDL). This is the user-friendly interface which allows the user to define models using the languages specified before. . . . . 30

4.3 The menu-driven graphical interface provides an easy and efficient method to select a model simulator, its parameters and finally one or more plotting simulation charts. Here I show two views of the simulator interface: in the first Picture (a), the simulator is waiting for the user to fill all the information regarding the simulation. Finally, in Picture (b), we can see the simulator performing the trajectory calculations. . . . . 32

5.1 Description of a candlestick used to show confidence intervals of a set of simulations: the shadow of a candlestick illustrates the highest and the lowest value of a set of simulations, and the body describes the endpoints of the confidence interval . . . . . 37

5.2 Candlestick charts: this is an example of the Michaelis-Menten model solved with a stochastic simulator Gibson-Bruck and shown by using a candlestick chart. As we can see, the bottom chart is slightly different from the top chart, due to the change of the alpha value. . . . . 39

5.3 Profiling of the GAL model: it is immediately clear that some reactions do not occur during the process of chemical simulation, on the other hand, others, like the *G3_transcription* reaction, have been fired 529 times. . . . . 40

5.4 Solution of the GAL model, solved with the Gibson-Bruck method. 42

5.5 Comparison of the galactose value in the GAL model between the Gibson-Bruck method and the just developed Stochastic Differential Equation algorithm . . . . . 43

5.6 Galactose chart: changing the ensemble size, we can fire more reactions and obtain almost the same results in the GAL model . . . 44

5.7 GAL model simulated with different time intervals: we can notice that, increasing the interval of the simulation, we obtain very different kind of Profile charts . . . . . 47

6.1 Michaelis-Menten Charts . . . . . 53

6.2 Schlögl Charts . . . . . 55

6.3 Lotka-Volterra Charts . . . . . 57

7.1 Three-dimensional chart of the Michaelis-Menten model, which represents the behaviour of the species $P$. The model has been simulated five times with a different initial concentration of molecules $E$, from 100 (the standard condition) to 80. . . . . 61

# Abstract

Stochastic and analytical approaches are widely used for the analysis of chemical reactions systems. Dizzy, the chemical kinetic simulation software developed by the Institute for System Biology, provides a solution for such chemical reaction models either by using different stochastic simulation algorithms or by solving a set of ordinary differential equations. This thesis provides a method to analyse this data by employing confidence intervals and showing candle sticks for the representation of those results. Moreover we have developed a profile analysis of simulated models. The thesis also deals with a new simulation method which establishes a solution developed by solving stochastic differential equations. This method combines differential equations with Brownian motion, and it turns out to be faster than traditional stochastic algorithms when applied to specific chemical models. Trajectories of well-known chemical models, like the Lotka-Volterra and the Schlögl models, are computed by solving stochastic differential equations, and compared with the solution obtained using other simulation methods. In the end the thesis explains other works related to Dizzy regarding sensitivity analysis and deadlock analysis of chemical reaction models.

# Acknowledgements

I would like to express my gratitude to all those who gave me the possibility to complete this thesis.

First of all the completion of this thesis would not have been possible without the constant support of Dr. Stephen Gilmore, who has helped me throughout my research project and has given me the chance to participate in the PASTA Workshop.

The thesis at the University of Edinburgh was a unique experience and I would also like to express my sincere thanks to Prof.ssa Susanna Donatelli for this great opportunity.

Finally, I wish to thank my parents for their continuous support and encouragement.

# Chapter 1

# Introduction

THIS thesis gives an account of my investigations into the methods for solving chemical reaction systems, namely by approaching the issue with stochastic differential equations.

Models of chemically reacting systems have traditionally been simulated either by solving a set of ordinary differential equations (ODEs) or by using the stochastic simulation algorithm (SSA) of Gillespie [10]. Traditional ODE-based approaches are adequate when dealing with large numbers of molecules, when discreteness and noise have no macroscopic effects, but in general they are not able to provide a fully physically accurate representation of the noise amplification caused by the essential stochastic processes in living cells. At the same time SSA approaches can simulate accurately all those dynamics by using a discrete-space Markov process, but their drawback remains the great amount of computation time that is often required to simulate a model.

From this setting we come to stochastic differential equations (SDEs), a combination between a strictly deterministic approach and a stochastic one. SDEs represent a position on the border between different branches of science and engineering, from mathematical analysis to probability calculus. In particular regarding stochastic processes: the application of

SDEs is of interest to different subjects like physics, financial mathematics and biology.

In this paper, I present an extended analysis of this methodology applied to Dizzy [22], the chemical kinetics stochastic simulation software package written in Java. This software was developed by Stephen Ramsey in the Institute for Systems Biology since 2002, is licensed under the GNU Lesser General Public License (LGPL), which is a standard "free software" and "open source" license[1]. Dizzy provides a model definition environment and a set of simulation engines, both deterministic, like the ODEs, and stochastic, like the Gillespie's direct method, Gibson-Bruck algorithm [9] and Gillespie's $\tau$-leap [5]. Results of the trajectories of the simulated dynamical models are shown either by way of numerical tables of values or a chart of the average simulations.

To validate the correctness of the numerical solution of the SDE simulator, I compare this new methodology implemented with more traditional ODEs and SSA simulators across three models: the Michaelis-Menten model, the Schlögl model and the Lotka-Volterra model.

Besides providing a new stochastic simulator for Dizzy based on solving a set of SDEs, the aim of my thesis is also to help Dizzy's users to have a more accurate comprehension of the final results. To reach this target, I have developed a statistical analysis of the computed trajectories supported by confidence intervals. Such analysis allows a user to set a specific confidence interval in which a number of simulation results are considered "good" and others, due to the stochastic nature of the simulator, can be rejected. To show properly these two new categories of trajectories I have implemented a candlestick chart, like those used in the financial field. These charts make very clear the two classes of results, and represent a useful improvement for Dizzy. Moreover, because some chemical reaction models like the Schlögl exhibit a bistable behaviour, I decided to introduce a chart which represents all the simulation runs. This dynamic

[1] A copy of the license is available online at
`http://magnet.systembiology.net/software/License.html`

behaviour cannot be represented using charts of mean values, but only by showing all the trajectory paths.

During my enquiry period I also worked on two other different little projects on Dizzy: sensitivity and deadlock analysis. Sensitivity analysis allows Dizzy's user to set a range of values in which a single species of a chemical reaction model can start. Results of each trajectory computed with these different values are shown in a three-dimensional chart. This particular graph is one of the best ways to represent changes in the initial conditions of the model, and how these conditions influence trajectories. Deadlock analysis instead involves stochastic simulators: I can reveal the moment in which the probability to step to the next reaction becomes void, and terminate the simulation.

## 1.1 Background to the research

I led my thesis enquiry as visiting student to the Laboratory for Foundation of Computer Science, University of Edinburgh, under the supervision of Doctor Stephen Gilmore. That was a great opportunity for me to complete my Laurea Specialistica in Metodologie e Sistemi Informatici course abroad, and I found at the University of Edinburgh, namely the King's Buildings campus, an amazing place to study and settle the long research on the field of chemical reactions. Moreover Dizzy, the chemical kinetic software, has been developed through the years at the University of Edinburgh, and I am very proud to have given my little contribution to this great project.

## 1.2 Overview of the Thesis

This paper is organised as follows. First in Chapter 2 I show some projects in the chemical kinetics field related to this thesis, like some chemical kinetics simulators already developed. Then in Chapter 3 I introduce the

theory of the three most important simulation methods to solve chemical reactions: Ordinary Differential Equations, Stochastic Simulation Algorithms and Stochastic Differential Equations; furthermore I analyse some features of both deterministic and stochastic models. Then in Chapter 4 I present Dizzy, the chemical kinetics simulation software developed by the Institute for System Biology: focusing the attention upon this software, I discuss the main properties of Dizzy in order to solve and analyse chemical reaction models by using different algorithms and attributes. In Chapter 5 I give a deep account of the statistical analysis software I have added to Dizzy, supplying a confidence interval analysis and a brand new candlestick chart in order to show results. Moreover in Chapter 6 I introduce the stochastic differential equation algorithm I have developed for Dizzy: I discuss the numerical solution approach with the Euler-Maruyama method, then I show some well-known models, like Michaelis-Menten and Lotka-Volterra models, solved with this simulator and compared to other simulation methods. In Chapter 7 I describe other pieces of software I have developed during my enquiry on Dizzy for sensitivity and the deadlock analysis. Finally in Chapter 8 I point the reader to future works, then I provide conclusions and a summary of the whole thesis.

# Chapter 2

# Related works

OTHER simulators have been developed in order to solve chemical kinetics reaction systems, either by university consortium or by companies.

One of the most important software in this field is Biochemical Network Stochastic Simulator (BioNetS)[1], developed by the BioMed Central. This software works in particular with biochemical networks, and allows the user to specify the type of random variable (discrete or continuous) for each chemical species in the network. For the continuous random variables, BioNetS constructs and numerically solves the appropriate Chemical Langevin Equations (CLE). Basically one of the peculiarities of this software is the ability to handle hybrid models that consist of both continuous and discrete random variables and its ability to model cell growth and division. This framework has been developed by David Adalsteinsson, David McMillen and Timothy C. Elston [1].

Another important stochastic simulator software is StochSim, developed by Nicolas Le Novère and Thomas Simon Shimizu [19]. The biggest difference between this software and Dizzy can be found in the ability of StochSim to perform only stochastic simulations of chemical kinetics

[1] A copy of the BioNetS software is available at `http://x.amath.unc.edu:16080/BioNetS/`

models, rather than Dizzy is able to simulate a model either with a deterministic and with a stochastic method. The benefit of StochSim is that each molecule exists as an independent software object, and this allows the representation of molecules that have specific internal states called *multistate molecules*.

One of the most recent simulator for biochemical processes, COPASI, has been developed by Stefan Hoops and Sven Sahle [17]. COPASI combines traditional stochastic simulations of reaction networks and flux analysis with some unique methods for the simulation of chemical reaction models, such as hybrid method which combines the stochastic simulation algorithm of Gibson-Bruck (Section 4.2) with different algorithms for the numerical integration of ODEs. So the chemical model is dynamically partitioned into a deterministic and stochastic subnet depending on the current particle numbers in the system. The two subnets are then simulated in parallel using the stochastic and deterministic solver. A description of this hybrid method can be found in a diploma thesis of one of the authors [21].

Stochastic Differential Equations (SDEs) have been recently studied and analysed by K. Burrage and T. Tian [3, 4]. After an interesting discussion comparing ODE and SDE approach, they have defined a strong solution for SDEs, either with explicit and implicit methods. K. Burrage [2], in his Ph.D. thesis, gave an overview of extant methods of Runge-Kutta type for solving SDEs, discussing how the theories developed for ODEs may be useful in developing efficient numerical methods in the stochastic case.

D. J. Higham [15, 16] proposed a Matlab implementation of SDEs, in order to compare the Michaelis-Menten model with Reaction Rate Equations and Gillespie's Direct Method algorithm.

# Chapter 3

# Simulation methods for solving chemical reaction systems

In a fixed volume $V$, containing a spatially uniform mixture of $N$ chemical species interacting through $M$ specific chemical reaction channels, there are two well-known methods to predict the number of molecules of a particular species after some amount of time. These two methods have two very different natures: one is a deterministic approach, the other is a stochastic one.

The deterministic method is the first approach realised in the middle of the twentieth century. This method involves Ordinary Differential Equations (ODEs), using such equations to solve a set of chemical reactions. Since this method does not consider any stochastic process during the evolution of the chemical model through time, it is now judged to miss interesting behaviour. However, this method is appreciated when we would like to solve chemical kinetic models in a small amount of time, with modest computational effort.

Gillespie [10, 13] between 1976 and 1977 defined the first stochastic approach to chemical reactions, developing the Direct Method, a simulator

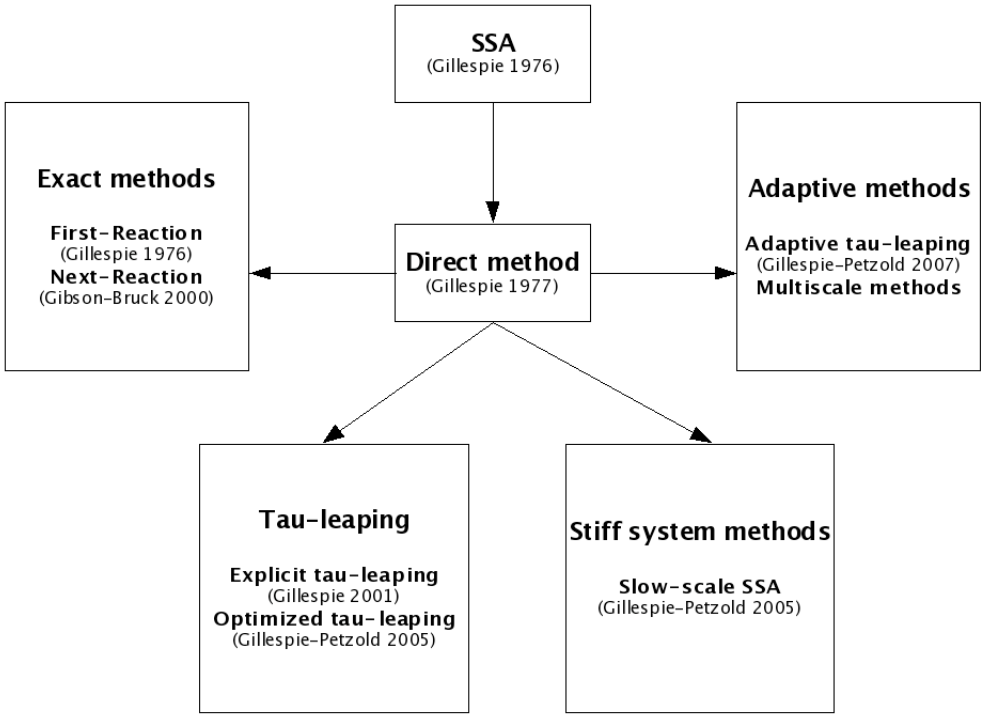



*Figure 3.1:* Map of the main stochastic simulators developed from 1977 in order to solve chemical reactions. All the "improved" SSA implementations do not imply that there is anything wrong with the original SSA procedure by Gillespie: on the contrary, they either optimise or approximate Gillespie's Direct Method.

which could exactly predict the molecular population level after different time steps. This simulator was innovative because it focuses the attention not on the chemical species, as the deterministic approach, but on the reactions which fire from time to time in the chemical model. Since this simulator was very computationally expensive, due to the amount of data to compute, a lot of improvements have been made: a map of some stochastic simulators developed since 1977 is shown in Figure 3.1:

A particular note about the $\tau$-leap simulator, developed by Gillespie in 2001. This new simulator is a stochastic process that approximately solves the Chemical Master Equation (CME), based on a controllable,

dimensionless error parameter. A quantity known as the "maximum allowed leap time" $\tau$ is periodically computed, according to the Gillespie-Petzold formula [5]. If the time scale $\tau$ is less than a few times the inverse aggregate reaction probability density (where the exact threshold is configurable and only affects efficiency), a Gillespie Direct discrete event is carried out. In the case where $\tau$ exceeds the threshold time scale, a "leap" is performed. The number of times each reaction occurs during the time interval $\tau$ is generated using the Poisson distribution based on the reaction's probability density per unit time.

Finally the theory behind Stochastic Differential Equations (SDEs) is explained in order to give the reader a full knowledge of the basis on which the SDEs simulator has been developed. SDEs are a combination between a strictly deterministic approach and a stochastic one; their solution comes from mixing a portion of ordinary differential equation with a Brownian process, which enables the stochastic part to the simulator.

In this chapter I am going to introduce a detailed explanation of the two approaches described above: in particular Section 3.1 gives an overview of the Deterministic Solution, Section 3.2 of the Stochastic approach and Section 3.3 of the SDEs solution. Finally a relationship between the deterministic and the stochastic approach is provided, in order to understand better the strong connections between all these simulation approaches.

## 3.1 Deterministic Solution

Chemical reactions are usually modelled by a set of Ordinary Differential Equations (ODEs). This is the simplest method to solve a chemical model, evolving the concentration of a particular species according to a probability law: indeed this approach focuses the attention solving a single differential equation per species of the model, rather than looking at the single states as in the Stochastic Solution (Section 3.2).

Suppose we have four chemical species A, B, C and D. We can write a chemical equation related to these species like the relation below

$$A + B \xrightarrow{k_1} C + D \tag{3.1}$$

and another equation that involves only three of the four species described before

$$2A + C \xrightarrow{k_2} B \tag{3.2}$$

Both chemical equations will not take place instantaneously, but they will react respectively at the rate $k_1$ and $k_2$ according to the concentration of the species A and B for the Equation 3.1 and A and C for the Equation 3.2.

Defining these two chemical equations as the full specification of the reactions taking place in our system, we can create four ODEs corresponding to the concentration of the four chemical species according to the model above:

$$\begin{aligned}
\tfrac{d[A]}{dt} &= -k_1[A][B] - 2k_2[A][A][C] \\
\tfrac{d[B]}{dt} &= -k_1[A][B] + 2k_2[A][A][C] \\
\tfrac{d[C]}{dt} &= k_1[A][B] - 2k_2[A][A][C] \\
\tfrac{d[D]}{dt} &= k_1[A][B]
\end{aligned} \tag{3.3}$$

where the square brackets denote the concentration of the single species.

Looking only at the first part of the very first equation of the Model 3.3, the rate of reaction will depend proportionally on $[A]$, $[B]$ and the constant of proportionality given by $k_1$. So the rate of Reaction 3.1 will be given by $k_1[A][B]$. Each instance of Reaction 3.1 will use up one unit of $A$. So the rate of change of concentration of $A$ due to Reaction 3.1 will be equal to $-k_1[A][B]$. The minus signifies one unit being used up.

The rate of the second reaction will depend proportionally two times on $[A]$ and $[C]$; so the rate of reaction will be $k_2[A][A][C]$. Each reaction will use up two units of $A$, so the rate of change of concentration of $A$ due to Reaction 3.2 will be equal to $-2k_1[A][A][C]$.

There are similar terms in the other ODEs corresponding to using up the species $B$ and creating species $C$ and $D$, and in others consuming up one species $C$ creating species $D$.

One of the most common problem related to ODE simulators is stiffness. A stiff ODE is an ordinary differential equation that has a transient region whose behaviour is on a different scale from that outside this transient region. Very often a stiff system which involves chemical reaction rates can converge to a final solution quite rapidly. In order to solve this issue, new "stiff-free" solvers have been developed by the scientific community. Although this problem seems to be solved, these new simulators require a very high amount of computational resources, and due to a lack of efficiency they are rarely used to solve ODEs.

Two examples of chemical stiff models are the Schlögl model (Section 6.2.2) and Lotka-Volterra model (Section 6.2.3), described in Chapter 6.

This approach obviously assumes that the time evolution of a chemically reacting system is both continuous and deterministic. However, the time evolution of a chemically reacting system is not a continuous process, because molecular population levels can change only by discrete integer amounts. Moreover, the time evolution is not a discrete process either. Indeed it is impossible to predict the exact molecular population levels at some future times unless we take account of the precise positions and velocities of all molecules in the system.

## 3.2 Stochastic Solution

The stochastic approach of chemical kinetics takes its stand from the issue that collisions in a system of molecules in equilibrium occur in an

essentially random manner.

Let $S_i\,(1 \leq i \leq N)$ be the list of chemical species which comprise our model, and suppose these species can interact through *M* specified chemical reaction channels $R_\mu\,(1 \leq \mu \leq M)$. We can assert the existence of M constants $c_\mu dt$, which represent the average probability that a particular combination of $R_\mu$ reactant molecules will react accordingly in the next infinitesimal interval *dt* only depending on the physical properties of the molecules and the temperature of the system.

These *M* constants are the basis for developing the two most important approaches to a stochastic solution: the master equation approach, described in Section 3.2.1, and the stochastic simulation approach, described in Section 3.2.2. Since both these formulations are derived from the existence of $c_\mu dt$, they reach the same results in the limit as the volume *V* increase.

### 3.2.1 The Master Equation Approach

The Chemical Master Equation (CME) can be thought of as a huge system of coupled ordinary differential equations. The difference between the traditional reaction-rate approach explained in Section 3.1 and this method can be found in the number of differential equations to solve: in the CME there is one differential equation per state of the system, rather than ODEs approach where only one differential equation per species is required.

The complete characterisation of the "stochastic state" is defined by using the Grand Probability Function

$$P\,(X,t) = P\,(X_1, X_2, ..., X_n, t) \qquad n \in \mathbb{N} \tag{3.4}$$

which states the probability of having $X_1$ molecules of species $S_1$, $X_2$ molecules of species $S_2$, ... , $X_n$ molecules of species $S_n$, at time *t*. Let $S_j$ be the reactants or reagents and $S_k$ the products of the reaction, we

also assume $v_{jk} \in \mathbb{N}^n$ the stoichiometric matrix that corresponds to the state change which occurs whenever reaction $R_{jk}$ fires. In order to understand better the meaning of this matrix, we can assume each component $v_{jk}$ of the stoichiometric matrix as an element which represents the load of that particular chemical species in a reaction. The convention is to assign negative coefficients to "reactants" (which are consumed) and positive ones to "products". Defining in example Reaction 3.1 and Reaction 3.2 as the full specification of the reactions taking place in our system, we can obtain a stoichiometric matrix as described below:

| | A | B | C | D |
|---|---|---|---|---|
| R1 | -1 | -1 | 1 | 1 |
| R2 | -2 | 1 | -1 | 0 |

Finally let $a_{\text{jk}}(x,t)$ be the propensity function, which gives us the probability that this reaction will take place in the infinitesimal interval time $(t, t+dt)$ when the system is in the state $X(t) = x \in \mathbb{N}^n$. This infinitesimal time interval is chosen as the smallest time interval possible in which at most only a single reaction could take place. Sometimes it could also happen that in such a short interval, even a single reaction does not fire: usually that happens in those models which involve rare events. In order to avoid this issue, some simulators, like the Tau-Leap Complex algorithm (Section 4.2), generate automatically a shorter time interval *dt* as long as a reaction will take place.

In order to reach the solution of the Grand Probability Function, we can expand the Equation 3.4 as

$$P(X,t) = \sum_{j=0}^{n} \sum_{k=0}^{n} \left( a_{jk}(x - v_{jk}, t)\, P(x - v_{jk}, t) - a_{jk}(x,t)\, P(x,t) \right) \qquad (3.5)$$

Now we can split the propensity function $a_{\text{jk}}(x,t)$ in three different parts, according to the kind or reaction we have to deal with:

$$a_{jk}(x,t) = \begin{cases} c_{jk}(t)x_j & \text{for reaction } R_{jk} \\ c_{0k}(t) & \text{for reaction } R_{0k} \\ c_{j0}(t)x_j & \text{for reaction } R_{j0} \end{cases}$$

Having defined the propensity function in terms of $c(t)$, now we can rewrite the CME 3.6 as

$$\begin{aligned} P(X,t) = \; & \textstyle\sum_{k=1}^{n} c_{0k}(t)\left(P(x-v_{0k},t) - P(x,t)\right) + \\ & \textstyle\sum_{k=1}^{n} c_{j0}(t)\left((x_k+1)P(x+v_{0k},t) - x_k P(x,t)\right) + \\ & \textstyle\sum_{j=1}^{n}\sum_{k=1}^{n} c_{jk}(t)\left((x_j+1)P(x-v_{j0}-v_{0k},t) - x_j P(x,t)\right) \end{aligned} \tag{3.6}$$

The last sum of the CME 3.6 corresponds the normal reaction $R_{jk}$, instead the first and the second show respectively the inflow reaction $R_{0k}$ and the degradation reaction $R_{j0}$.

The number of problems for which the CME 3.6 can be solved analytically is even fewer than the number of problems for which the deterministic reaction-rate equation described in Section 3.1 can be solved analytically. In addition, unlike the reaction-rate equations, the master equation does not readily lend itself to the number and nature of its independent variables.

### 3.2.2 Stochastic Simulation Approach

We introduce now the Reaction Probability Density Function $P(\tau,\mu)$. This probability, accounted in $d\tau$, defines the probability that, given the state $(X_1,...,X_N)$ at time *t*, the next reaction will occur in the infinitesimal time interval $(t+\tau, t+\tau+dt)$, and will be an $R_\mu$ reaction. This function indeed defines the probability of when the next reaction will occur and what this kind of reaction will be. Furthermore we assume $P_0(\tau)$ as the probability that no reaction will occur in the time $(t, t+\tau)$, given the state $(X_1,...,X_N)$ at time *t*.

Having defined in Section 3.2.1 the propensity function $a_\mu$, we can assert that the probability that the next reaction will occur in the time interval $(t+\tau, t+\tau+dt)$ can be considered to be the probability that no reaction will occur in $(t, t+\tau)$ multiplied by the probability that a reaction will happen in $(t+\tau, t+\tau+dt)$:

$$P(\tau, \mu)\, d\tau = P_0(\tau) a_\mu d\tau \tag{3.7}$$

Using the definition of the propensity function, given the state $(X_1, ..., X_N)$ at time *t*, we can derive $P_0(\tau + d\tau)$ as

$$P_0(\tau + d\tau) = P_0(\tau) \left(1 - \sum_{k=1}^{M} a_k d\tau\right) \tag{3.8}$$

from which is deduced

$$\frac{P_0(\tau + d\tau) - P_0(\tau)}{d\tau} = -\sum_{k=1}^{M} a_k \tag{3.9}$$

Passing to the limit $d\tau \to 0$ we can solve $P_0(\tau)$ as

$$P_0(\tau) = e^{-\sum_{k=1}^{M} a_k \tau} \tag{3.10}$$

Recalling the Equation 3.7, now we can write the Reaction Probability Density Function $P(\tau, \mu)$ in terms of propensity functions. Hence, given the state $(X_1, ..., X_N)$ at time *t*, the probability that the next reaction will occur in the infinitesimal time interval $(t+\tau, t+\tau+dt)$ is

$$P(\tau, \mu) = a_\mu e^{-\sum_{k=1}^{M} a_k \tau} \tag{3.11}$$

This is a very important result: it should be emphasised that the second term of the Equation 3.11 answers the two requests made at the begin-

ning of this section: when the next reaction will occur and what kind of reaction it will be. Indeed $a_\mu e^{-\sum_{k=1}^{M} a_k \tau}$ may be written as $\frac{a_\mu + \sum_{k=1}^{M} a_k \tau}{\sum_{k=1}^{M} a_k \tau} e^{-\sum_{k=1}^{M} a_k \tau}$. In this form is easy to consider the next reaction index as $\frac{a_\mu}{\sum_{k=1}^{M} a_k \tau}$ , a discrete random variable, and the time until the next reaction will occur as $\left(\sum_{k=1}^{M} a_k \tau\right) e^{-\sum_{k=1}^{M} a_k \tau}$, a continuous random variable with an exponential distribution.

## 3.3 Stochastic Differential Equations Solution

SDEs, used to solve chemical reactions, are Markov processes described by the Chemical Langevin Equation (CLE) [11]. We assume $Y(t) \in \mathbb{R}^n$ the state vector of a continuous time, real-valued stochastic process at the time *t*: so $Y_i(t)$ is a real-valued random variable representing the number of molecules of the *i*th species. The stoichiometric or state-change vector is described by $v_\mathrm{j} \in \mathbb{R}^n$, whose *i*th component is the change in the number of $S_\mathrm{i}$ molecules due to the *j*th reaction. Finally let $a_\mathrm{j}\left(Y(t)\right)$ be the propensity function, which gives us the probability that this reaction will take place in the infinitesimal interval time $(t, t+dt)$. Having made this assumption, the CLE takes the Itô form

$$dY(t) = \sum_{j=1}^{M} v_j a_j\left(Y(t)\right) dt + \sum_{j=1}^{M} v_j \sqrt{a_j\left(Y(t)\right)} dW_j(t) \tag{3.12}$$

where the $W_j(t)$ are independent Wiener processes. A Wiener process is a stochastic process satisfying

$$E\left(W(t)\right) = 0, \qquad E\left(W(t)W(s)\right) = \min\left\{t, s\right\}$$

The Wiener increments are independent Gaussian processes with mean 0 and variance $|t-s|$. Specifically the Wiener increment $\Delta W(t) \equiv W\left(t+\tau\right) - W(t)$ is a Gaussian random variable $N\left(0,\tau\right) = \sqrt{\tau} N\left(0,1\right)$.

A numerical solution to solve these kinds of SDEs comes from the Euler-Maruyama method and, discretising the Brownian path and applying the Euler-Maruyama method to the linear SDE, it takes the form

$$Y(t+\tau) = Y(t) + \tau \sum_{j=1}^{M} v_j a_j(Y(t)) + \sqrt{\tau} \sum_{j=1}^{M} v_j \sqrt{a_j(Y(t))} Z_j$$

I have produced a Java implementation of the Euler-Maruyama method in Section 6.1 as an extension to a simulator for chemical reactions Dizzy, the chemical kinetics stochastic simulation software of the Institute for Systems Biology.

Although SDEs are capable of capturing the major aspects of a chemical reaction, we can find this approach unsuitable for multi-scale models in which we manage multi-scale problems with exponentially large (or small) population variables or we have to deal with exponentially unlikely events. The Schlögl model, in Section 6.2.2, is one of these models for which SDEs do not give satisfactory results.

## 3.4 Relationships between deterministic and stochastic models

The relationship between Markov chain models for chemical reactions and classical deterministic ordinary differential equation models is strong and deep.

Recalling the chemical equation 3.1 and considering this equation to be the complete specification of our new chemical model, we can easily derive the common deterministic approach using the law of mass action, where the population of $A$ molecules is defined by the following nonlinear differential equation:

$$\frac{d[A]}{dt} = -k_1[A][B] \tag{3.13}$$

The same equation can be reached by using the stochastic approach, using the Master Equation Approach (Section 3.2.1). We declare *A(t)*, *B(t)*, *C(t)* and *D(t)* to be the number of molecules of the species *A*, *B*, *C* and *D* at time *t*. Moreover, we assume *P(a,b,c,d,t)* the probability that *A(t)*, *B(t)*, *C(t)* and *D(t)* have respectively values of *a*, *b*, *c* and d, and finally, let $\alpha$ and $\beta$ be the initial concentration of *A* and *B*, or *A(t)* and *B(t)* with $t = 0$.

The form of *P(a,b,c,d,t)* can be determined by the stochastic master equation, which describes the means of transition from and to the state *(a,b,c,d)* during the stochastic process of chemical reaction. It may be defined by the following axiom: the probability of a reaction event in the interval $(t, t + \Delta t)$ whereby $(a, b, c, d) \rightarrow (a - 1, b - 1, c + 1, d + 1)$ is $k_1 ab\Delta t + O(\Delta t)$, where $k_1$ is the stochastic constant rate for the reaction.

A detailed balance equation can be written in these terms

$$\begin{aligned} P(a,b,c,d,t+\Delta t) = k_1(a+1)(b+1)\Delta t P(a+1,b+1,c-1,d-1,t) \\ + (1 - k_1 ab\Delta t) P(a,b,c,d,t) + O(\Delta t) \end{aligned} \tag{3.14}$$

Now the stoichiometry of the reaction, *a*, *b*, *c* and *d* are related as follows:

$$\alpha - a = \beta - b = c = d \tag{3.15}$$

Hence, the probability density function *P(a,b,c,d,t)* can be expressed as a function of just one species population. In order to prove the relation between the deterministic and the stochastic solution, I will take the species *A* as reference, such that $P_a(t) = P(A(t) = a) = P(a, \beta - \alpha + a, \alpha - a, \alpha - a, t)$. Now the Equation 3.14 can be simplified to the following equations, letting $\Delta t \rightarrow 0$:

$$\begin{aligned} \frac{dP_0(t)}{\Delta t} &= k_1(\beta - \alpha + 1) P_1(t) \\ \frac{dP_\alpha(t)}{\Delta t} &= -k_1 \alpha\beta P_\alpha(t) \end{aligned} \tag{3.16}$$

The obtained result of the Equation 3.16 can be easily brought back to Equation 3.13, without considering the volume issue. Indeed, As Oppen-

heim, Shuler and Weiss [20] have shown in their article, the deterministic model of certain special cases is the infinitive volume limit of the Markov chain models, and the same can be shown for all the chemical models.

# Chapter 4

# Dizzy: a chemical kinetics simulation software

In this chapter, I give an overview of the most important features of Dizzy, a chemical kinetics simulation tool for analysing the dynamics of complex biochemical models.

This software framework is capable of simulating the trajectory behaviour of a chemical kinetics model by using different simulators, either deterministic or stochastic. Dizzy uses an intuitive user interface to define the model, and to set the simulation parameters. Finally, using charts and tables, it is able to show graphs with the results of the simulation.

Dizzy has been developed and is maintained by the CompBio Group, Institute for System Biology with the collaboration of the Laboratory for Foundations of Computer Science, a research institute inside the School of Informatics at the University of Edinburgh. The first stable release of the tool (version 1.0.0) dates back to December 2003; today a copy of the latest Dizzy software (version 2.4.4) is available for download in the web site `http://magnet.systembiology.net/dizzy`. Dizzy is a free and open-source software, distributed under the GNU Lesser General Public License (LGPL) [26].

Dizzy enables the creation of reduced stochastic models containing reactions whose propensities may be expressions of arbitrary complexity, representing the average effect of underlying reaction steps that are in quasi-steady-state (QSS). This permits efficient approximate modelling of enzyme-catalysed reactions and other processes for which the overall kinetic rate is more complicated than mass-action kinetics.

Dizzy provides a feature for estimating or calculating the steady-state stochastic fluctuations of the species in a biochemical model, requiring only the solution of the deterministic dynamics. Dizzy also has several important software features including integration with external software tools, a graphical user interface GUI, and a high level of portability.

To the best of our knowledge, Dizzy is the first software tool available that includes all of the features enumerated above. In addition, it includes novel implementations of the Gibson-Bruck and Gillespie Tau-Leap algorithms that are applicable to models with complex kinetic rate laws. At present, Dizzy is notable for explicitly modelling spatially in homogeneous chemical species concentrations and transport phenomena such as diffusion. However, Dizzy permits partitioning of a model into distinct spatial compartments. Each compartment volume is treated as a spatially homogeneous, continuously well-stirred system.

The performance of the deterministic and stochastic simulation algorithms described above has been benchmarked using a variant of the heat-shock response model for Escherichia coli proposed by Srivastava and adapted by Takahashi for benchmarking the performance of the E-Cell simulator. This model includes a large separation of dynamical time scales, which is typical of complex biochemical networks. The results show the efficiency of the Gibson-Bruck algorithm relative to the Gillespie Direct algorithm, and the significant speed improvement of (approximate) Tau-Leap algorithms over the (exact) Gibson-Bruck and Gillespie algorithms. It should be emphasised that no modifications of the model definition file were necessary in order to switch between the various simulation algorithms shown above. This is made possible because our model definition

language is simulation algorithm-agnostic. Furthermore, the Tau-Leap method does not require an ad hoc partitioning of the model into stochastic and deterministic reaction channels. This is a potential advantage in analysing a complex model for which the "fast" and "slow" degrees of freedom are not known a priori.

The structure of the software is explained in Section 4.1, whereas all the simulators included in it are shown in Section 4.2. Finally an overview of the graphical and the command line user interface can be found in Section 4.3.

## 4.1 Package structure

Dizzy has been designed using a modular organisation in which each simulator is a software unit that conforms to a simple, well-defined interface specification, as Ramsey [22] shows in his article regarding this framework. Dizzy is implemented in the Java programming language, which can be executed on any computer platform in which a Java Runtime Environment is available. That makes Dizzy one of the most versatile tools capable to compute chemical kinetics trajectories.

The biochemical modelling semantics are separated from the description of how the dynamics is to be solved: this architecture facilitates an iterative model development cycle in which the model is analysed using various simulation algorithms. Moreover Dizzy's model definition language permits the definition of reusable, parametrised model elements called templates: this enables the construction of a prepackaged library of templates that can simplify the task of constructing a complex model.

Dizzy is structured in six main packages: the most important packages are listed below

- *chem* package, which provides all the classes able to perform a simulation with one of the simulators described in Section 4.2.

- *gui* package, which gathers all the graphical user interface classes of the main window of the simulator, namely the GUI to describe the model in the formal language. GUI classes to draw charts and to manage simulators are located in the *chem* package.
- *math* package, which collects all the basic types of the formal language to describe chemical kinetics models. It also provides a list of math functions and other classes useful to deal with very accurate numbers.

## 4.2 Stochastic and deterministic simulators

Dizzy allows the user to perform simulations of chemical reaction models by using several different algorithms, both stochastic and deterministic. The stochastic simulators are discrete-event or multiple-event Monte Carlo algorithms, instead the deterministic simulators model the dynamics as a set of ordinary differential equations (ODEs) which are solved numerically.

Stochastic simulators use a Colt Project[1] random number generator and distribution to produce customised sets of random numbers.

One benefit of this modular design is that one may use a deterministic ODE-based solver for optimisation and parameter fitting, and switch to a stochastic simulation technique for exploring the stochastic dynamics, once the model parameters have been established. This modularity also simplifies the task of implementing a new simulator and integrating it into the system. In this section we describe the simulators available in our software system.

- Deterministic Simulators: Runge-Kutta simulator, OdeToJava

[1]Colt Project provides a set of Open Source Libraries for High Performance Scientific and Technical Computing in Java. A complete description of this package can be found at the web site `http://dsd.lbl.gov/~hoscheck/colt/`

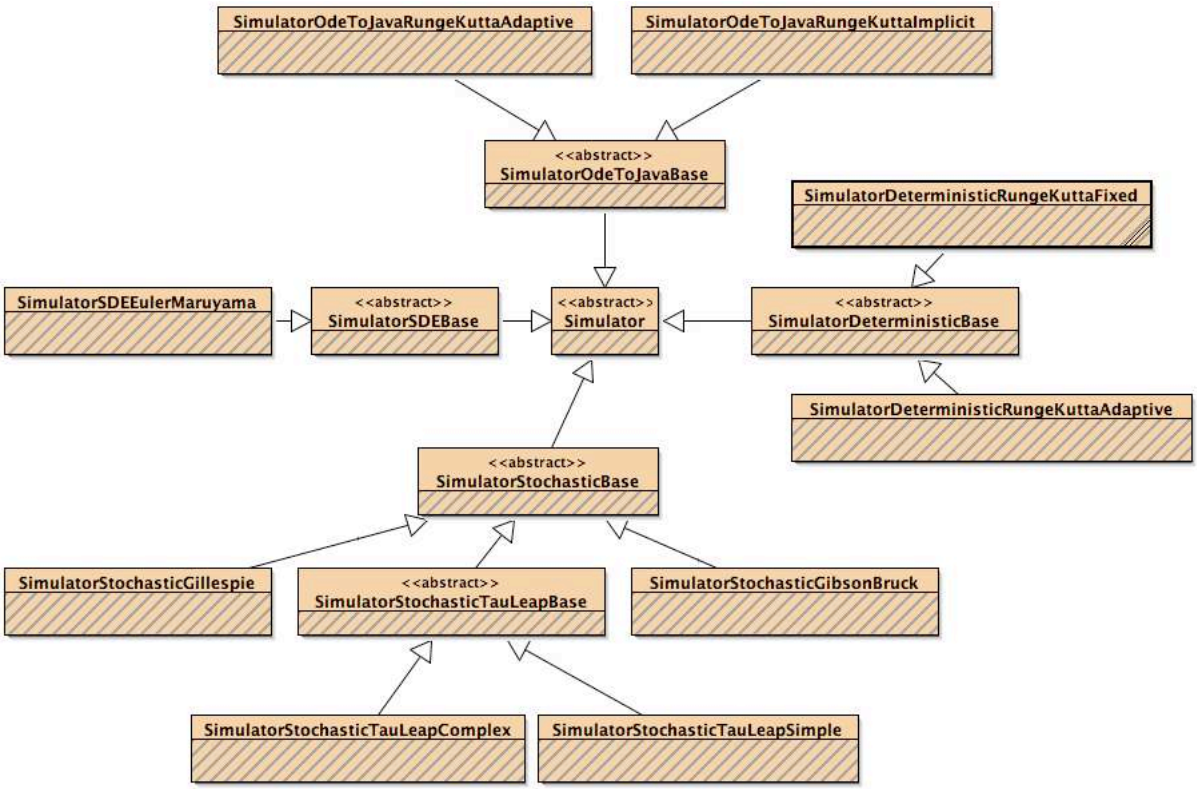



*Figure 4.1:* Simulator structure in Dizzy: from the common abstract class `Simulator`, four abstract classes define the common simulation methods for each main algorithm. This enables a very simple method to create new simulators, allowing the developer to focus only the core of simulators.

- Stochastic Simulator: Gillespie's Direct Method, Gibson-Bruck Method, Gillespie's Accelerated Approximate Method ("Tau-Leap" Method)
- Stochastic Differential Equation Simulator: Euler-Maruyama

Dizzy, as specified before, includes a deterministic simulator based on a fifth order Runge-Kutta ODE solver. Step size is adaptively controlled, based on a fourth order error estimation. Both relative and absolute error tolerances may be independently specified, as well as the initial step

size. Although Runge-Kutta is not state-of-the-art for high-accuracy integration, it is particularly useful for models in which a derivative function is discontinuous.

In order to get an accurate deterministic solution, two additional deterministic simulators have been implemented based upon the OdeToJava ODE solver package by Patterson and Spiteri [25]. This package includes a Dormand-Prince fourth/fifth order solver [7] with adaptive step size control. It also contains a Runge-Kutta implicit-explicit ODE solver that is useful for systems with a high degree of stiffness.

Considering exact solutions using stochastic algorithms, Dizzy includes an efficient implementation of a stochastic simulator based on Gillespie's Direct Method and Gibson-Bruck.

Gillespie's Direct Method uses the Monte Carlo technique to generate an approximate solution of the master equation for chemical kinetics. In this method, simulation time is advanced in discrete steps, with precisely one reaction occurring at the end of each discrete time-step. Both the time steps and the reaction that occurs are random variables. The Direct Method requires recomputing all reaction probability densities after each iteration. The computational complexity of the method therefore increases linearly with the number of reactions. Furthermore, for sufficiently large simulation time, the total number of iterations can be prohibitively large for some systems. Nevertheless, for simple systems with small numbers of species and reactions, the Direct Method can be useful.

As stated before, Dizzy also implements a stochastic simulator based on Gibson and Bruck's Next Reaction method [9]. The computational cost of this Monte Carlo-type method scales logarithmically with the number $M$ of reaction channels, in contrast with the Gillespie algorithm which scales linearly with $M$. A tree traversal technique to analyse a rate expression for a chemical reaction that has a complex kinetic rate law has been implemented in order to ascertain the dependence of the rate expression upon the various chemical species in the model. This permits

applying the Gibson-Bruck method to models that implement complex kinetic rate laws.

Two stochastic simulators based on Gillespie's Accelerated Approximate Method (here referred to as the "Tau-Leap" Method) have been implemented in Dizzy. The Tau-Leap Method is a stochastic process that approximately solves the chemical master equation, based on a controllable, dimensionless error parameter. A quantity known as the "maximum allowed leap time" $\tau$ is periodically computed according to the Gillespie-Petzold formula already described in Chapter 3. If the time scale $\tau$ is less than a few times the inverse aggregate reaction probability density (where the exact threshold is configurable and only affects efficiency), a Gillespie Direct discrete event is carried out. In the case where $\tau$ exceeds the threshold time scale, a "leap" is performed. The number of times each reaction occurs during the time interval $\tau$ is generated using the Poisson distribution based on the reaction's probability density per unit time. The state of the system is updated to reflect the predicted number of times each reaction occurs during the time interval $\tau$. After each "leap" iteration, the maximum allowed leap time $\tau$ is recomputed.

Two versions of the Tau-Leap algorithm have been implemented, Tau-Leap Complex and Tau-Leap Simple.

The Tau-Leap Simple algorithm is intended for simple models entirely composed of reaction channels with mass-action kinetics. The Tau-Leap Complex algorithm is a novel adaptation of Tau-Leap that is intended for use with models with complicated (e.g., enzymatic) rate expressions whose partial derivatives are very expensive to evaluate symbolically. In this method, the full symbolic Jacobian is stored and used at each iteration, in order to exploit the caching of evaluated, algebraically complex sub-expressions in the computation of the Gillespie-Petzold formula. Using the Poisson distribution to model the number of times a given reaction can occur during a time interval $\tau$, has the disadvantage that the exponential tail allows for rare events in which the realised number of times a reaction occurs (generated from the distribution) is too great for the

numbers of reactant species available; we call this “reactant exhaustion”. This is not indicative of a failure of the algorithm per se, but of a need to decrease the time scale $\tau$.

Finally, the last stochastic simulator has been implemented by using SDEs. A full detailed overview of this simulator is shown in Chapter 6.

## 4.3 Graphical user and command line interface

The Dizzy graphical user interface (GUI) is a Java 2 application embedded in the Dizzy package. It allows users to deal with the simulation of chemical reactions with the help of a menu-driven interface. This user interface includes screens for simulation control, model editing, plotting simulation results and browsing/searching the hypertext user manual.

Trajectories computed by Dizzy are shown in plot charts. Dizzy is capable of drawing up to four different kind of charts: *average*, *all runs*, *candlestick* and *profile* chart.

- Average chart: each trajectory is computed a user-specified number of times, then Dizzy computes the average and plots the result in the chart.
- All Runs chart: each trajectory simulation calculated is drawn in the chart. This can be helpful when dealing with bistable systems (Section 6.2.2), in which the average of trajectories becomes useless due to the nature of the model.

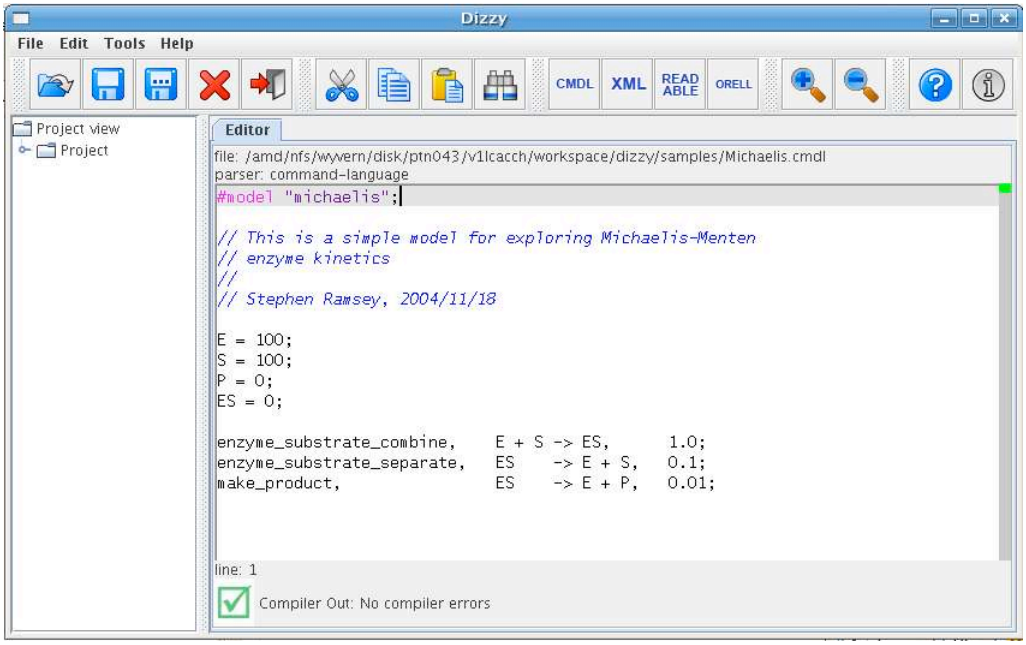
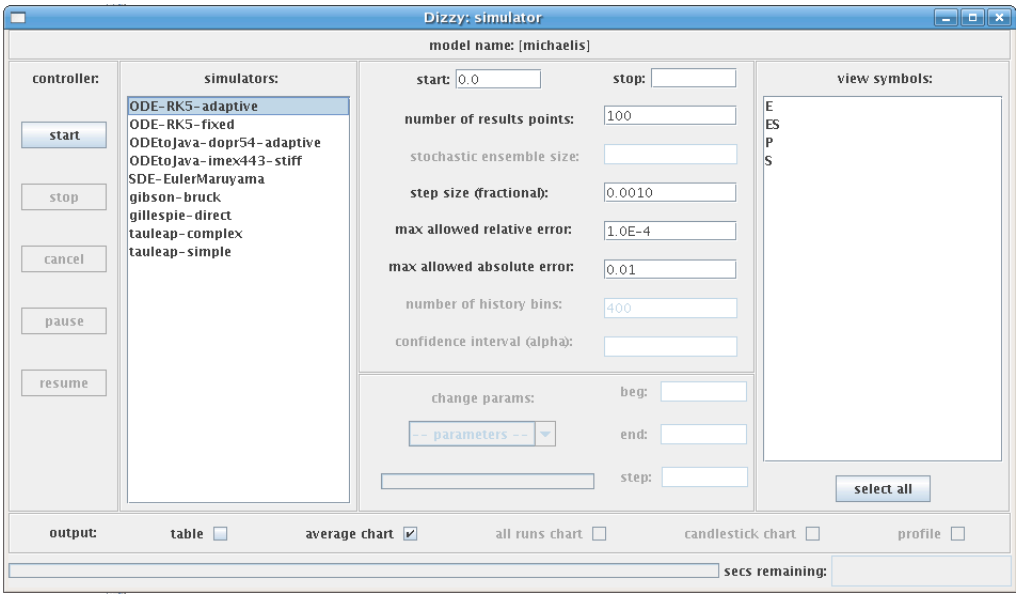



*Figure 4.2:* Dizzy is able to simulate models written in the Systems Biology Markup Language (SBML), in addition to the Chemical Model Description Language (CMDL). This is the user-friendly interface which allows the user to define models using the languages specified before.

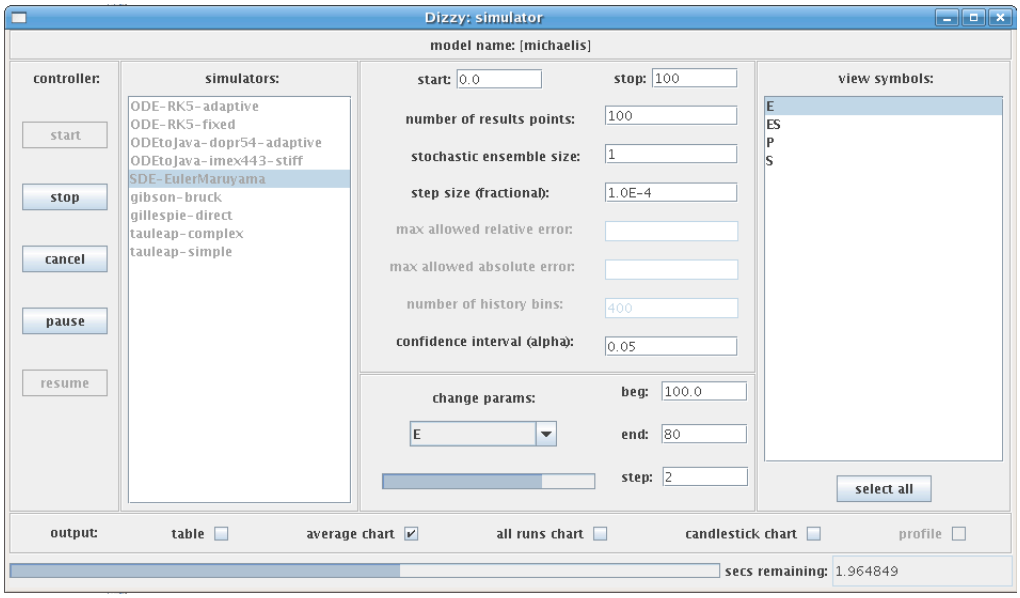
- Candlestick chart: introducing the statistical analysis of the results (Chapter 5), the candlestick chart is a convenient way to show a number of simulations and a confidence interval associated with them.
- Profile chart: this bar chart shows the number of time that a single reaction of the chemical model occurs. It is important to look at this chart to understand the effective reliability of the trajectories computed. For example, from this chart we can determine that some reactions never fired on this simulation run, or that some events occur only very rarely.

Dizzy is also able to produce three-dimensional charts to show the Sensitivity Analysis of chemical kinetics models: a detailed description of this feature can be found in Section 7.1

Moreover, Dizzy provides a command line interface. The command line interface is capable of performing all the features as in the graphical interface, and can be useful when working on Dizzy on remote machine, or when doing batch-mode replications on a computer cluster.




(a) Choose the simulator..




(b) .. the simulation is running

*Figure 4.3:* The menu-driven graphical interface provides an easy and efficient method to select a model simulator, its parameters and finally one or more plotting simulation charts. Here I show two views of the simulator interface: in the first Picture (a), the simulator is waiting for the user to fill all the information regarding the simulation. Finally, in Picture (b), we can see the simulator performing the trajectory calculations.

# Chapter 5

# Statistical analysis of results

THE first software package I am going to show is the statistical analysis of the results obtained by solving chemical reaction models with a stochastic simulator in Dizzy.

Before this package was developed, Dizzy was able to perform and show a simple average of different stochastic simulations of a single kinetic model. This kind of output was useful for some chemical models, like the Michaelis-Menten model (Section 6.2.1), when the behaviour of the trajectories through different simulations is almost the same, so the variance does not reach high values. Problems come when we attempt to simulate multiple times a model like the Schlögl model (Section 6.2.2), in which the main species has a bistable behaviour. Indeed showing the average of different simulations of this model can be almost useless, due to the high variance of each trajectory simulated.

In order to reach a better understanding of the obtained results, I have developed Java code to analyse the calculated trajectories, introducing, apart from the mean, also a variance and a confidence interval estimation. A theoretical introduction of the confidence interval is proposed in Section 5.1, followed by a representation of these confidence intervals by using candlestick charts in Section 5.2.

Another noticed problem was the fact that in some models, like the Grid-Averaged Lagrangian (GAL) model[1], during a single round of simulation sometimes not every reactions occurred. Due to this problem, the result of each simulation could be very different and totally unpredictable. A solution of this problem is therefore described in Section 5.3.

## 5.1 Confidence Intervals

Let $X_j\,(0 \leq j \leq M)$ be the list of values $X_j = x$ that the *j*th trajectory simulated assumes in a particular time step, and let $T_i\,(0 \leq i \leq N)$ be the list of instanced *X* lists in each single time steps *i*. The list *X* $(X_1, X_2, ..., X_m)$ is composed by an independent sample from a normally distributed population with mean $\mu$ and variance $\sigma^2$

Computing the average as $\overline{X} = \frac{X_1+X_2+...+X_m}{m}$ and the variance as $S^2 = \frac{1}{m-1}\sum_{i=1}^{m}\left(X_i - \overline{X}\right)^2$ we can derive the theoretical confidence interval for the mean $\mu$ using the Student's T distribution with *n-1* degrees of freedom. Indeed, defining the Student's T distribution as $T = \frac{\overline{X}-\mu}{S/\sqrt{m}}$, and *t* as the Student's T constant value for *m* values and confidence interval $1-\alpha$, with probability $1-\alpha$ we will find the parameter $\mu$ between the two endpoints

$$P\left(\overline{X} - t \cdot \frac{S}{\sqrt{m}} < \mu < \overline{X} + t \cdot \frac{S}{\sqrt{m}}\right) = 1 - \alpha \tag{5.1}$$

Consequently we can compute the confidence interval as

$$\left[\overline{X} - t \cdot \frac{S}{\sqrt{m}}, \overline{X} + t \cdot \frac{S}{\sqrt{m}}\right] \tag{5.2}$$

We now introduce the package Stochastic Simulation in Java[8] (SSJ)[2].

[1]Dizzy comes with the simple stochastic model of the GAL4 system of yeast, taking into account the proteins GAL4, GAL80 and GAL3, as well as galactose. For a full specification of the model, please refer to the GAL.cmdl document inside the Dizzy package

[2]The latest version of the SSJ package can be downloaded at the web site `http://www.iro.umontreal.ca/~simardr/ssj`

SSJ is a Java library for stochastic simulation, developed under the direction of Pierre L'écuyer, in the Département d'Informatique et Recherche Opérationnelle (DIRO) at the Université de Montréal. It provides facilities for generating uniform and non-uniform random variables, computing different measures related to probability distributions, performing goodness-of-fit tests, applying quasi-Monte Carlo methods, collecting (elementary) statistics and programming discrete-event simulations with both events and processes.

SSJ provides different packages. One of these, the `stat` package, allows the user to develop a statistical analysis storing all his data in the `TallyStore` object. This type of statistical collector takes a sequence of real-valued observations $X_1, X_2, ..., X_n$ and can return the average, the variance, a confidence interval for the theoretical mean, and other statistical analysis. All the individual observations are stored in a list implemented as a `DoubleArrayList`. Such class is imported from the Colt Project[3] library and provides an efficient way of storing and manipulating a list of real-valued numbers in a dynamic array. Each value can be accessed via the `getArray` method.

I have embedded `TallyStore` in the classes `SimulatorStochasticBase` and `SimulatorSDEBase`, which provide respectively the main methods for the simulation of stochastic and SDE algorithms. Namely `TallyStore` has been combined with the previous system of data storage, to maintain the retro-compatibility with some methods that continue using the old system to store data results, such as the `SimulatorDeterministicBase`. Obviously registering only the average of the simulation results is faster than storing all the data inside the `TallyStore`: this is the price we must pay in order to get more specific and detailed results.

Once we have computed $N$ different `TallyStore`, one for each time step of the simulation, we can send these to the class `SimulationResultsPlot`, the main class which provides the construction of the result charts.

[3]Colt Project provides a set of Open Source Libraries for High Performance Scientific and Technical Computing in Java. A complete description of this package can be found at the web site `http://dsd.lbl.gov/~hoscheck/colt/`

`TallyStore`, as described before, allows the user to get the confidence interval of the stored values, using the method `confidenceIntervalStudent (double level, double[] centerAndRadius)`. This method returns, in element 0 and 1 of the array object `centerAndRadius[]`, the centre and the half-length (radius) of a confidence interval on the true mean of the random variable $X$, with confidence interval level `level`, assuming that the observations given to this collector are independent and identically distributed copies of $X$, and that $X$ has the normal distribution.

In the next section I am going to show how we can illustrate in charts these confidence intervals.

## 5.2 Candlestick chart

One of the best methods to show to the user the confidence interval of a distribution comes from the finance field, and is defined with the name of "candlestick chart". A candlestick chart is a style of bar-chart used primarily to describe price movements of a stock over time. Each bar is composed of the body and an upper and a lower shadow. In finance, the shadow shows the highest and the lowest traded price of a stock, instead the body illustrates the opening and closing trades. Furthermore, the body assumes a white colour when the opening price is lower than the closing price (hence the price of the equity is grown), otherwise it assumes a black colour when the opening price is greater than the closing price.

Apart from the colour of the body, which is the field of chemical kinetics is quite useless, I decided to use the shadow of a candlestick to illustrate the highest and the lowest value of a set of simulations, and the body to describe the endpoints of the confidence interval. A graphical explanation of the solution is shown in Figure 5.1.

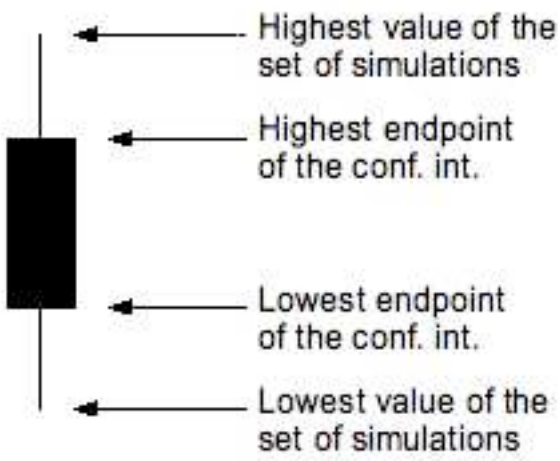



*Figure 5.1:* Description of a candlestick used to show confidence intervals of a set of simulations: the shadow of a candlestick illustrates the highest and the lowest value of a set of simulations, and the body describes the endpoints of the confidence interval

In order to get a candlestick chart, I have used the package JFreeChart[4], which had been already embedded in Dizzy because of displaying the average trajectories of a set of simulations. This functionality was studied in the previous versions of Dizzy. JFreeChart provides the APIs to build and show a candlestick chart: according to these APIs, I have developed three classes, `CandleStickDataItem`, `CandleStickSeries` and `CandleStickDataset`. These three classes supply the management of the simulation results and make such data ready for the manipulation inside the chart:

- `CandleStickDataItem`: this class provides all the methods to create a single candle object, with the highest and lowest values and with both endpoints. A candle here describes the behaviour of all the simulations in a single frame step of a single chemical species.
- `CandleStickSeries`: this class works as a collector of all the candles, described in `CandleStickDataItem`, in different time steps

[4]JFreeChart is a free software: the latest version of JFreeChart can be downloaded at the web site `http://www.jfree.org/jfreechart/`

of a particular chemical species. That is very useful because we can show more than a single species in our candlestick chart, and this class allows us to disambiguate each trajectory.

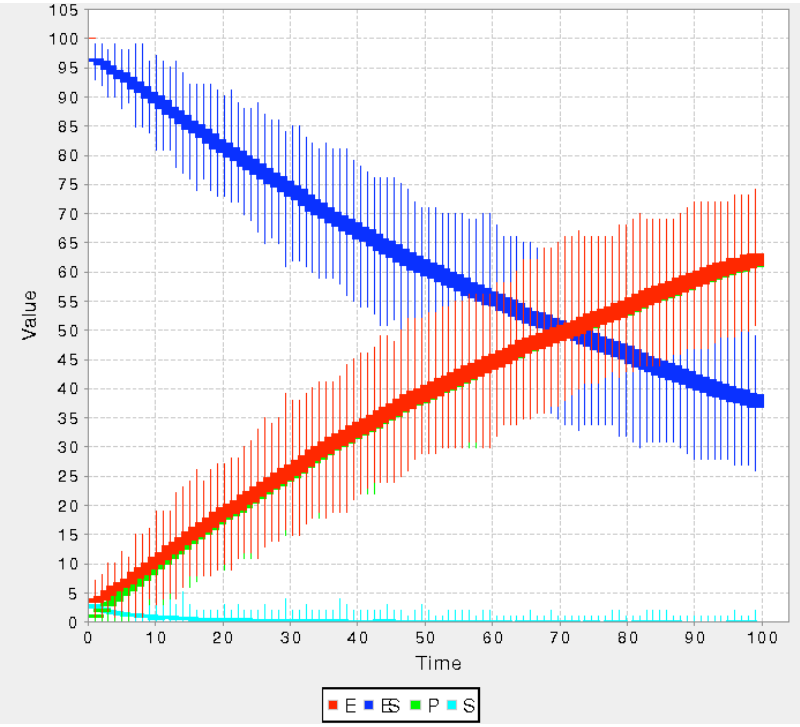
- `CandleStickDataset`: this is the final collector of all the objects of `CandleStickSeries`, and is developed according to the APIs of the JFreeChart package.

An example of an obtained candlestick chart is shown in Figure 5.2.

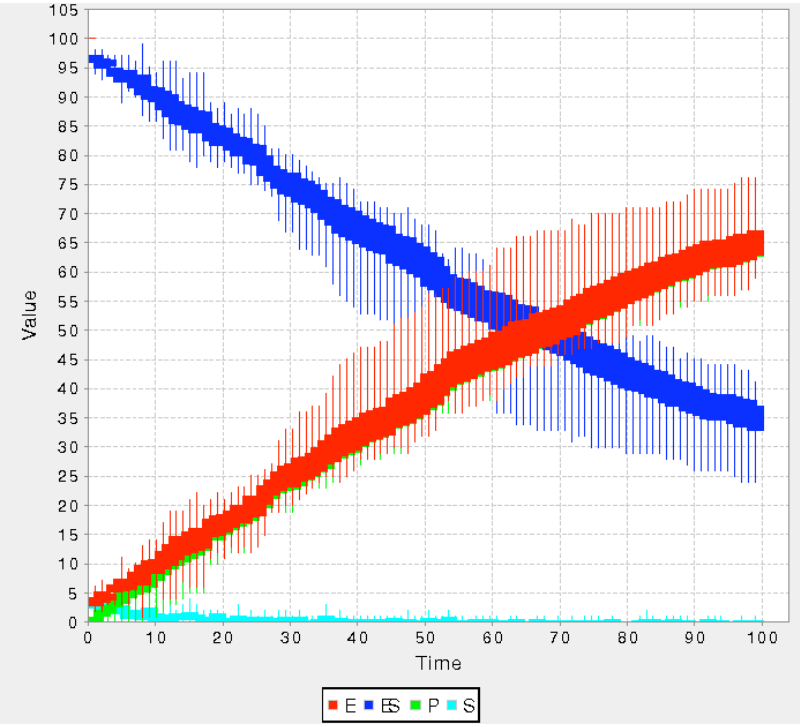
## 5.3 Profiling

The last issue to solve in order to improve the quality and knowledge of the computed results of a solved chemical reaction system is reaction profiling. Profiling answers at the questions: "Have all the reactions been fired during the simulation? And how many times?". This problem turns out to be very interesting, namely in some models, like the GAL model mentioned before. To complete a list of the number of reactions which have been fired during a simulation, it is necessarily to keep a counter of each reaction fired during a single simulation.

I have developed a simple object `FireCounter`, that registers the number of times each reaction has been fired. At the end of a simulation, the data of this object is converted to a bar-chart by using the package JFreeChart described before. An example of an obtained bar-chart for the GAL method is shown in Figure 5.3.




(a) alpha = 0.05




(b) alpha = 0.2

*Figure 5.2:* Candlestick charts: this is an example of the Michaelis-Menten model solved with a stochastic simulator Gibson-Bruck and shown by using a candlestick chart. As we can see, the bottom chart is slightly different from the top chart, due to the change of the alpha value.

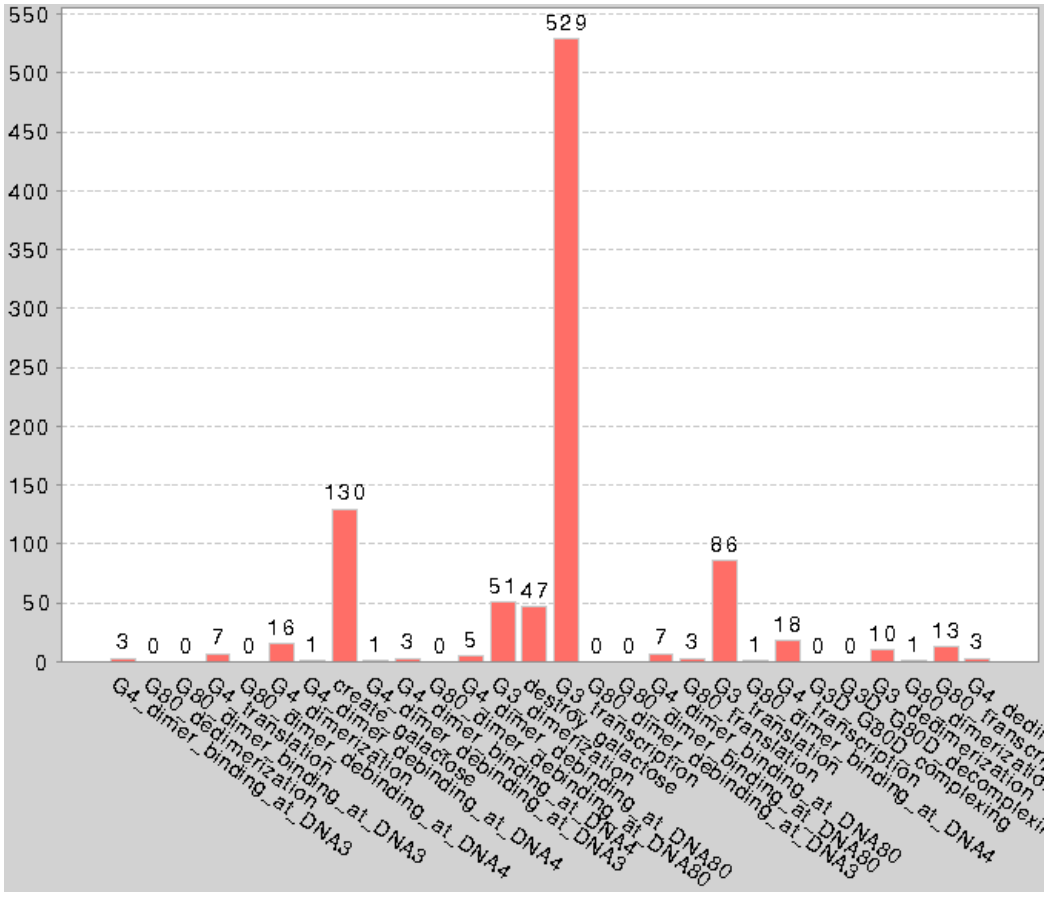



*Figure 5.3:* Profiling of the GAL model: it is immediately clear that some reactions do not occur during the process of chemical simulation, on the other hand, others, like the *G3_transcription* reaction, have been fired 529 times.

### 5.3.1 The GAL model

Ramsey [22, 23] showed in his article the yeast galactose utilisation pathway, described in the model Grid-Averaged Lagrangian (GAL), and namely the complex interactions among the regulatory genes *GAL3*, *GAL4* and *GAL80* which control the synthesis of a handful of enzymes that regulate galactose metabolism. In particular, we can observe two different situations of the model: a metabolic part, in which galactose is transported into the cell and various enzymes worked in concert to transform the internal galactose into glucose-1-phosphate; and a genetic part, in which enzymes are produced via transcription and translation processes that are controlled by the amounts of transcription factor and repressor molecules present in the cell. The yeast model used for the simulation is described in Appendix A, which includes the model definition and the galactose pathway, according to the Atauri, Orrell and Ramsey article [6].

One of the most relevant issues in this model is that the small numbers of molecules involved result in high levels of stochastic noise. Although the computational cost of these simulations has been extremely high, Ramsey has analysed such variability with a stochastic simulation, using the Gibson-Bruck simulator, and he has compared these results with the Dormand-Prince fourth/fifth order ODE solver. Both methods have been described in Section 4.2.

The solution of the GAL model, solved with the Gibson-Bruck method by Ramsey, is shown in Figure 5.4. Instead in Figure 5.5 is presented a comparison between the galactose chart obtained with the Gibson-Bruck method and the just developed Stochastic Differential Equation algorithm.

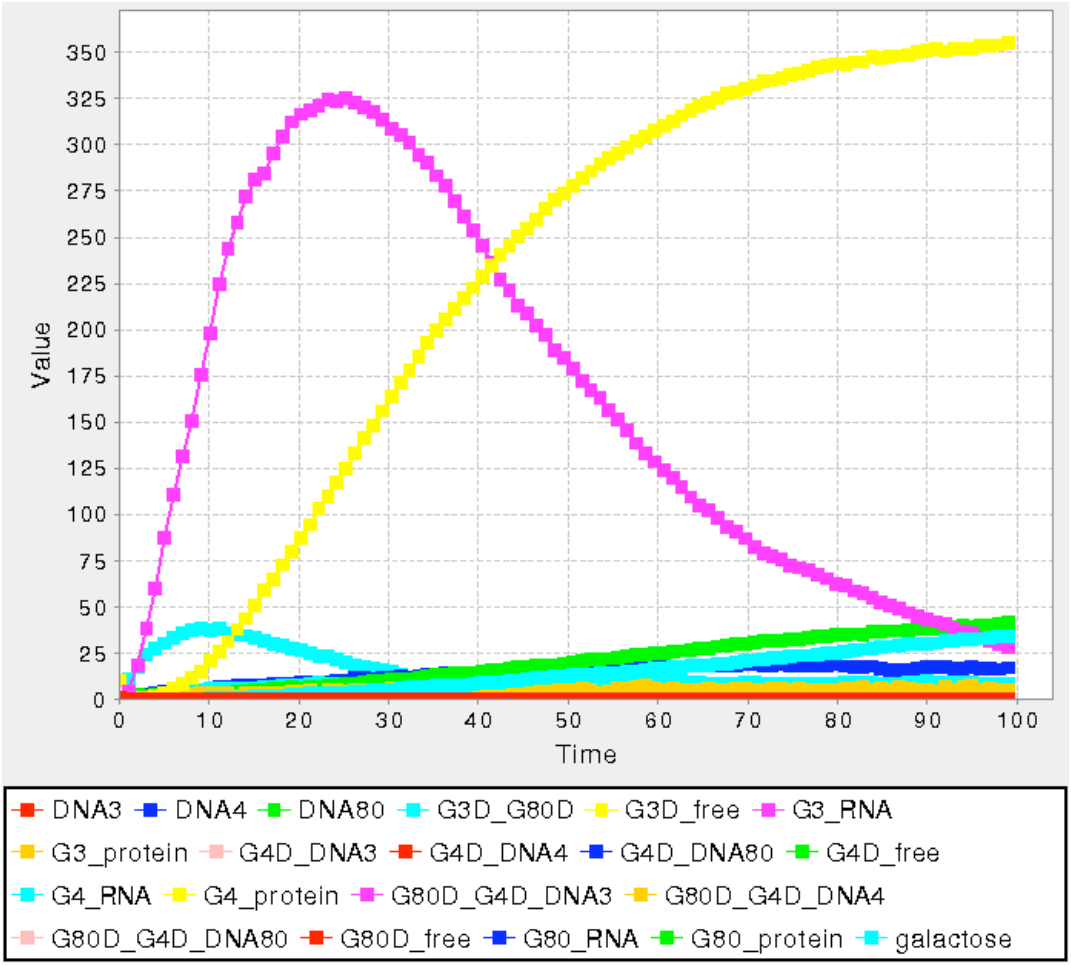
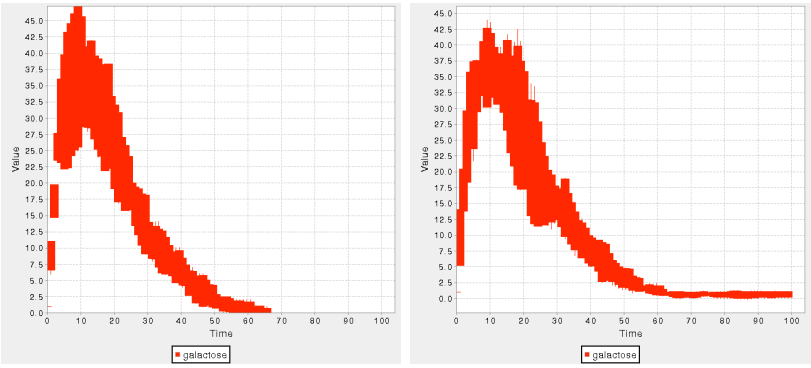



*Figure 5.4:* Solution of the GAL model, solved with the Gibson-Bruck method.

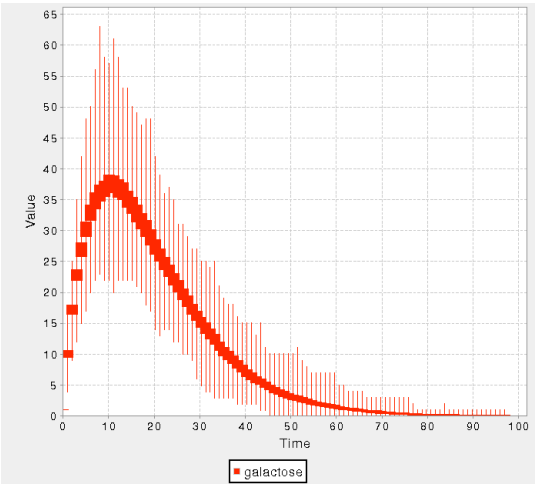



(a) Gibson-Bruck simulator (b) SDE algorithm

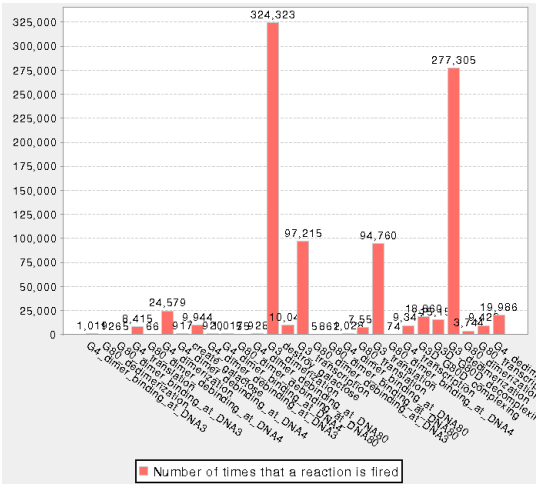
*Figure 5.5:* Comparison of the galactose value in the GAL model between the Gibson-Bruck method and the just developed Stochastic Differential Equation algorithm

In his exposition of the model results, Ramsey has not considered the fact that only some reactions have been fired during the simulation, as previously shown in Figure 5.3. This situation can be now easily noticed by looking at the Profile chart: I can assert that, even though only few reactions have been intensively consumed, this problem does not influence the final solution of the chemical model. I have tried to perform different kinds of simulations of the GAL model, still using the Gibson-Bruck simulator but enlarging the ensemble size. As shown in Figure 5.6, we can notice that galactose tends toward zero after almost the same amount of time, independently to the ensemble size of the simulation. This result has been also achieved by Ramsey, and the charts showed below prove that, in this model, we can obtain the same trajectories of the chemical species with a small ensemble size.

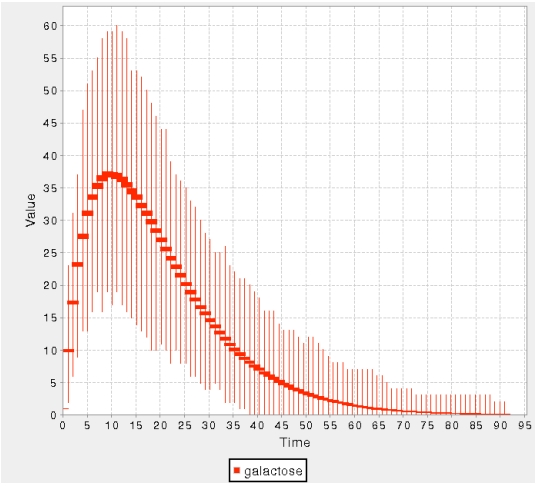
An interesting result comes out from performing the GAL model for more than 200 time steps. According to Ramsey's article, all performed results have been studied within an interval of 200 time steps. In this range, as shown in Figure 5.7, only four reactions have been usually fired. So



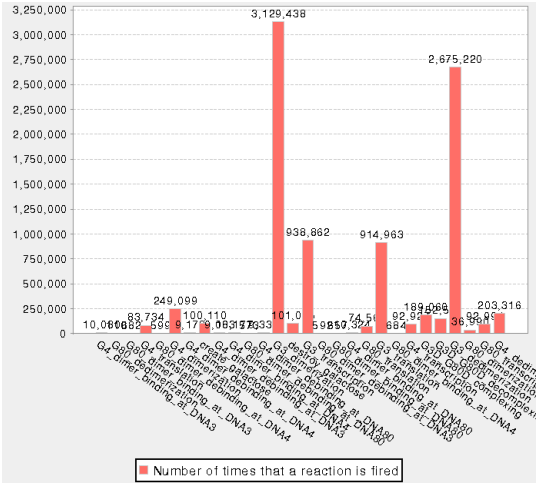

(a) Candlestick with ensemble size = 100



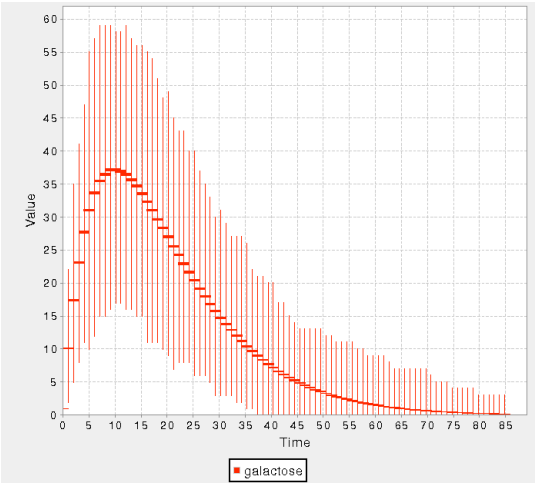

(b) Profile with ensemble size = 100



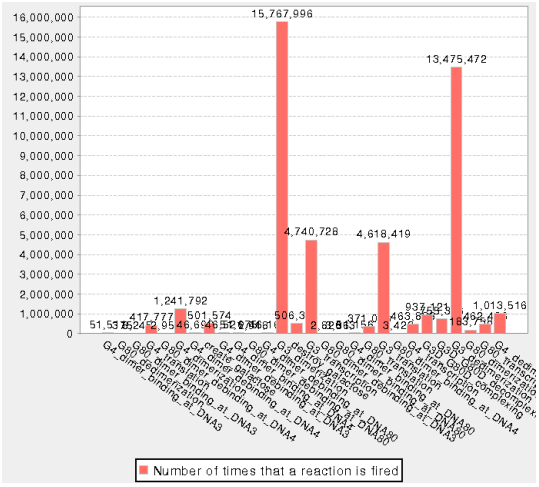

(c) Candlestick with ensemble size = 1000




(d) Profile with ensemble size = 1000




(e) Candlestick with ensemble size = 5000




(f) Profile with ensemble size = 5000

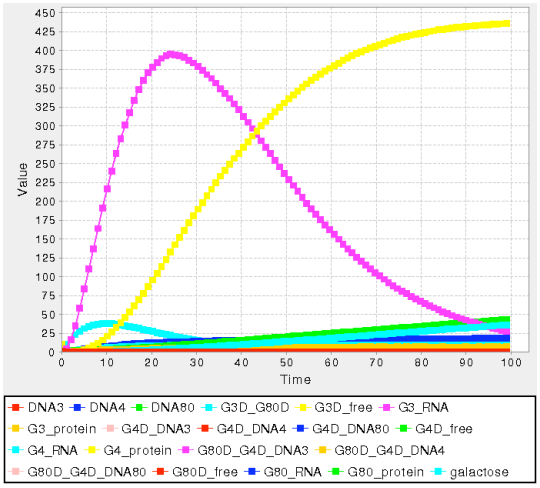
*Figure 5.6:* Galactose chart: changing the ensemble size, we can fire more reactions and obtain almost the same results in the GAL model

other reactions can be considered as “rare events”. This consideration changes when we perform simulations for more than 100 time steps. In this case we can observe that some reactions, like *G4_dimerization* and *G4_dedimerization*, respectively the sixth and the last reaction described in the Profile chart, overtake those reactions which could be considered “rare events” by looking at Ramsey’s results.

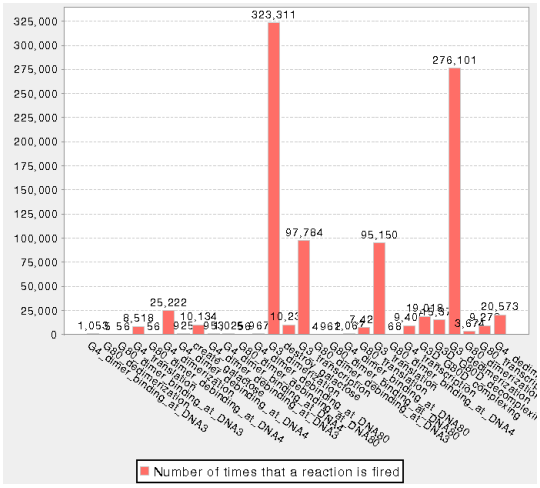
This result demonstrates the importance of the Profile chart, in order to analyse those models in which rare events are a considerable portion of the simulated reactions. In particular the Profile chart, together with the length of the simulation and the ensemble size, improves in a substantial manner the quality of the obtained results. Indeed now we can observe that *G4D_free* reaches steady-state after 4500 time steps, according also to the highlight reactions in the Profile Chart, *G4_dimerization* and *G4_dedimerization*.

We conclude that Ramsey will not have seen many activities of interest in the reaction kinetics. We gained additional insights into the reaction behaviour because our added reaction profiling highlighted that some reactions were very rarely seen over short time scale simulations.

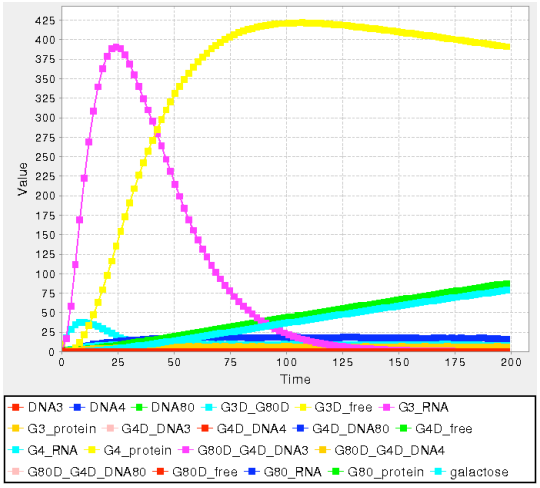
A complete overview of the yeast pathway results can be also found in the Zimmermann’s book “Yeast Sugar Metabolism” [28].



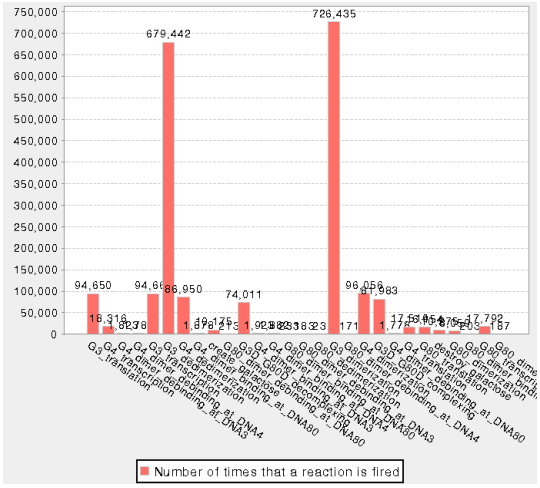

(a) Average chart with time 0 - 100



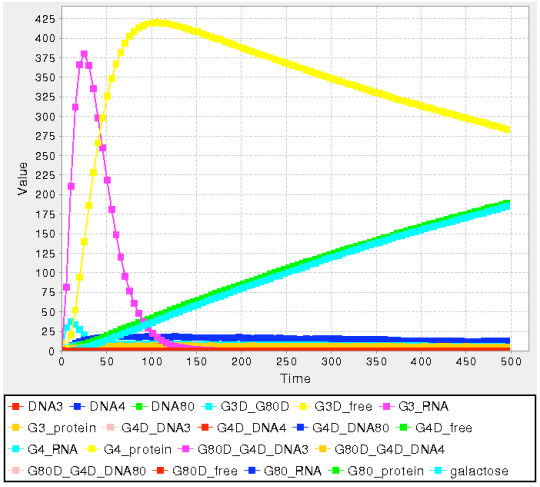

(b) Profile with time 0 - 100



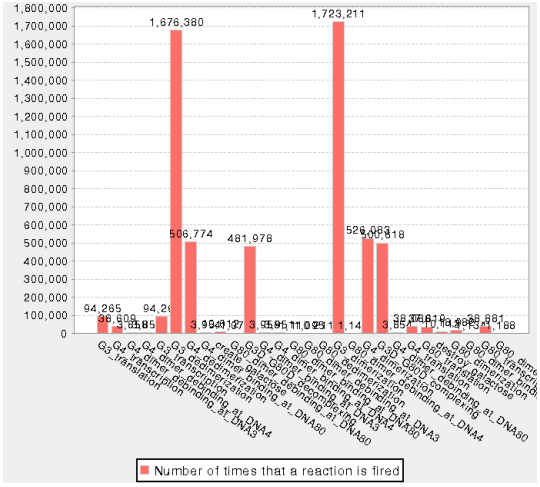

(c) Average chart with time 0 - 200



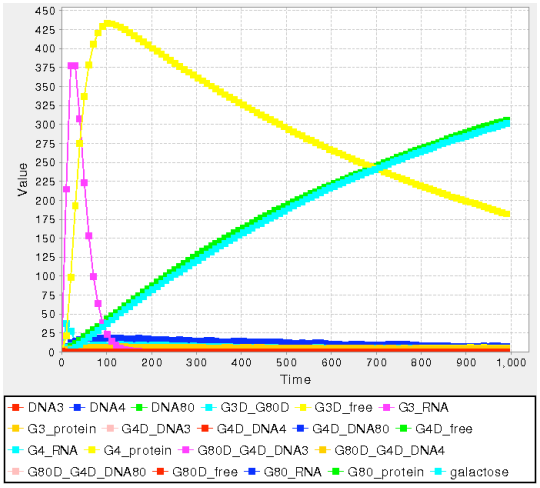

(d) Profile with time 0 - 200



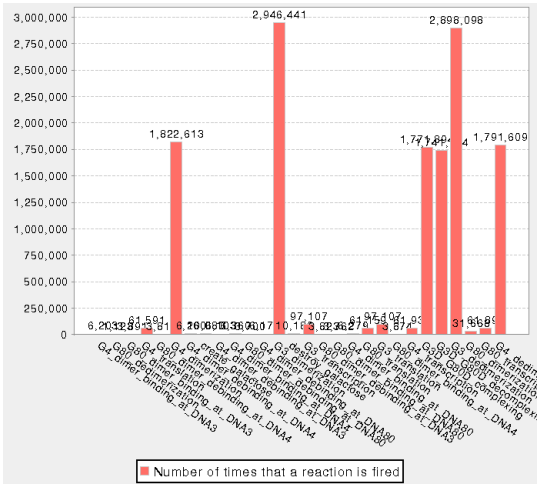

(e) Average chart with time 0 - 500



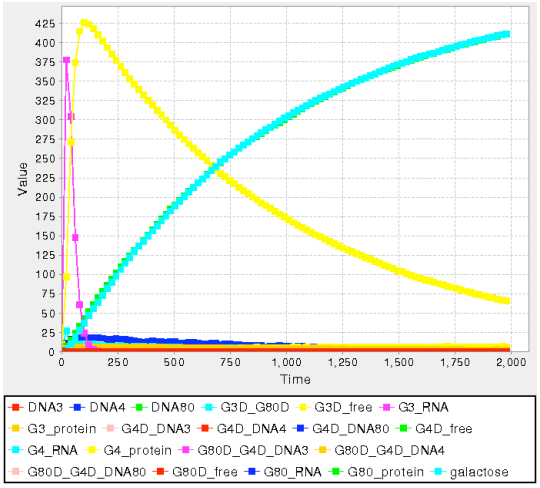

(f) Profile with time 0 - 500



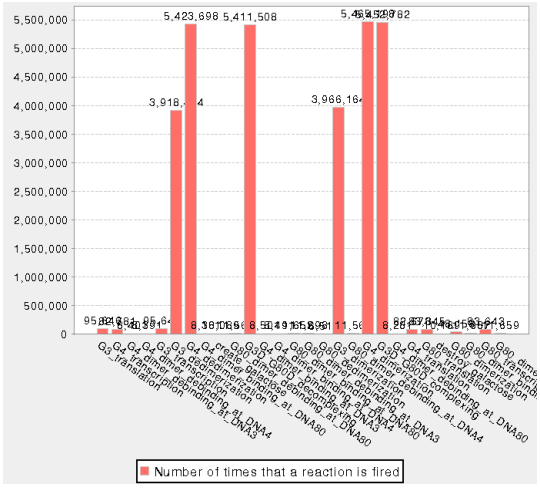

(g) Average chart with time 0 - 1000



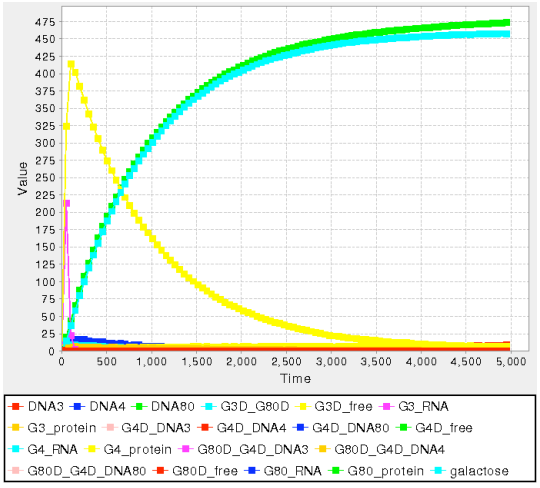

(h) Profile with time 0 - 1000



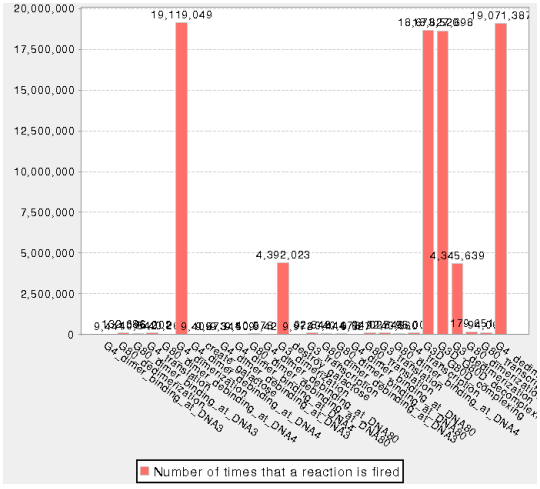

(i) Average chart with time 0 - 2000




(j) Profile with time 0 - 2000




(k) Average chart with time 0 - 5000




(l) Profile with time 0 - 5000

*Figure 5.7:* GAL model simulated with different time intervals: we can notice that, increasing the interval of the simulation, we obtain very different kind of Profile charts

# Chapter 6

# Working on Stochastic Differential Equations

In Section 3.3 I have given an overview of the theory behind Stochastic Differential Equations (SDEs). Now I am going to show how to implement a SDEs simulator in order to solve chemical kinetic reaction systems, introducing a numerical solution which develops the Euler-Maruyama algorithm (Section 6.1). Such a simulator is now embedded into the Dizzy package, and can be used as the other previous simulators.

Furthermore I present an experiment made with three well-known models: the Michaelis-Menten model, the Schlögl model and the Lotka-Volterra model. These models are analysed using different simulators, in order to show differences of time spent to simulate each model. As we can see in Section 6.2.4, some simulators are more efficient when used on particular models.

## 6.1 Numerical Solution

The aim of this section is to give the reader a concrete understanding of the Euler-Maruyama algorithm [18]. To achieve this target, I present

the Java code I have written for the stochastic simulator Dizzy. The code is structured in two classes, the `SimulatorSDEBase` class and the `SimulatorSDEEulerMaruyama` class.

The first class is a basic class for possible further SDEs simulators, extends `Simulator` and gathers all information regarding model parameters and input data. Moreover `SimulatorSDEBase` provides the main cycle wherein a specific SDEs simulator computes each iteration, supplying all the essential environment.

`SimulatorSDEEulerMaruyama` implements the interface `ISimulator` and extends `SimulatorSDEBase`, which provides all the mandatory methods for every simulator of chemical reactions in Dizzy. The most important method in this class is `iterate()`, called directly from the main cycle in `SimulatorSDEBase` in every iteration. The code below is an extract of the method `iterate()`:

```
/* SimulatorSDEEulerMaruyama.java
 * Implementation of Euler−Maruyama algorithm for Dizzy */

/* The probability that a reaction fires in this particular time step
 * is computed, and the result is stored in the array
 * mReactionProbabilities */
computeReactionProbabilities();

int numReactions = mReactions.length;
int numSymbols = pNewDynamicSymbolValues.length;
double reactionRate = 0.0;

/* Initialise all the elements of the vector 'derivative' to the value 0 */
DoubleVector.zeroElements(derivative);

/* Define a new vector linked to mDynamicSymbolAdjustmentVectors, the stoichiometric matrix
 * of the considered chemical kinetics model */
Object []dynamicSymbolAdjustmentVectors = mDynamicSymbolAdjustmentVectors;

/* For each reaction of the model, compute the drift and the diffusion term, in order to
 * obtain the result of the Euler−Maruyama algorithm */
for(int i = 0; i < numReactions; i++) {
```

```
    reactionRate = mReactionProbabilities[i];
    drift = reactionRate * stepSize;
    diffusion = Math.sqrt(Math.abs(drift)) * mScratchPad.nextRand();
26  diffusion = drift + diffusion;
    double []symbolAdjustmentVector =
      (double[])dynamicSymbolAdjustmentVectors[i];
    DoubleVector.scalarMultiply(symbolAdjustmentVector, diffusion, k);
    DoubleVector.add(k, derivative, derivative);
31 }

   /* Compute the scale of the evaluated step, in order to get accuracy analysis
    * of the just obtained result. Now the SDE simulator is unable to use this feature,
    * and it will be maybe provided in a further release */
36 double sumScale = 0;
   for(int i = 0; i < numSymbols; i++) {
     scale[i] = Math.abs(pNewDynamicSymbolValues[i]) + Math.abs(derivative[i]);
     sumScale += scale[i];
   }
41 if(sumScale == 0)
     throw new AccuracyException("unable to determine any scale at time: " +
       mSymbolEvaluator.getTime());

   DoubleVector.add(pNewDynamicSymbolValues,derivative,pNewDynamicSymbolValues);
46 mSymbolEvaluator.setTime(mSymbolEvaluator.getTime() + stepSize);

   return(mSymbolEvaluator.getTime());
```

*Listing 6.1:* SimulatorSDEEulerMaruyama.java

First of all I recall the stochastic differential equation with the discretised Brownian path.

$$Y(t+\tau) = Y(t) + \tau \sum_{j=1}^{M} v_j a_j\left(Y(t)\right) + \sqrt{\tau} \sum_{j=1}^{M} v_j \sqrt{a_j\left(Y(t)\right)} Z_j \tag{6.1}$$

Such an equation can be divided in four main components: the previous result, the drift and the diffusion term, and the white noise.

$$Y(t+\tau) = \overbrace{Y(t)}^{prev} + \overbrace{\tau \sum_{j=1}^{M} v_j a_j\left(Y(t)\right)}^{drift} + \overbrace{\sqrt{\tau} \sum_{j=1}^{M} v_j \sqrt{a_j\left(Y(t)\right)}}^{diffusion} \cdot \overbrace{Z_j}^{noise} \qquad (6.2)$$

In line 7, I compute the probability rate for each reaction in the model: `computeReactionProbabilities()` is a method inherited from the class `Simulator`. This probability is described in (6.2) as the $a_j\left(Y(t)\right)$ term.

Once probabilities are evaluated, for each simulation (line 22) I begin to determine the drift (line 24) and the diffusion term (line 25), referring to $\tau$ as the `stepsize`. The diffusion term is furthermore multiplied by the white noise: such latter term, which in the code is described as `mScratchPad.nextRand()`, is a double random number generated by a Normal distribution. I have used the package provided by Colt Project[1] embedded in Dizzy to generate those random values.

Finally in line 29 I multiply the drift and the diffusion term for the stoichiometric matrix $v_j$ described in the array `symbolAdjustmentVector` and I add the value to the other values obtained with other reactions.

## 6.2 Experiments

I apply the methodology implemented in the previous section to three models: the Michaelis-Menten model, the Schlögl model and the Lotka-Volterra model. All these models are solved[2] in three different ways: with Ordinary Differential Equations (ODEtoJava adaptive [25]), with

[1] Colt Project provides a set of Open Source Libraries for High Performance Scientific and Technical Computing in Java. A complete description of this package can be found at the web site `http://dsd.lbl.gov/~hoscheck/colt/`

[2] All the experiments were performed using P4 Dell 512Mb RAM using a Fedora Linux with GUI Gnome 2.14

Stochastic Simulation Algorithms (Gillespie's direct method, Gibson-Bruck algorithm and Gillespie's $\tau$-leap [12]) and with Stochastic Differential Equations proposed in Section 6.1.

### 6.2.1 The Michaelis-Menten model

The Michaelis-Menten model is widely used in biology for studying the kinetics of many non-allosteric enzymes: enzyme (E) and substrate (S) can combine themselves in complex (ES), which then eventually dissociates to free enzyme and product (P).

```
E  = 100;
S  = 100;
P  = 0;
ES = 0;
enzyme_substrate_combine, E + S –> ES, 1.0;
enzyme_substrate_separate, ES –> E + S, 0.1;
make_product,          ES –> E + P, 0.01;
```

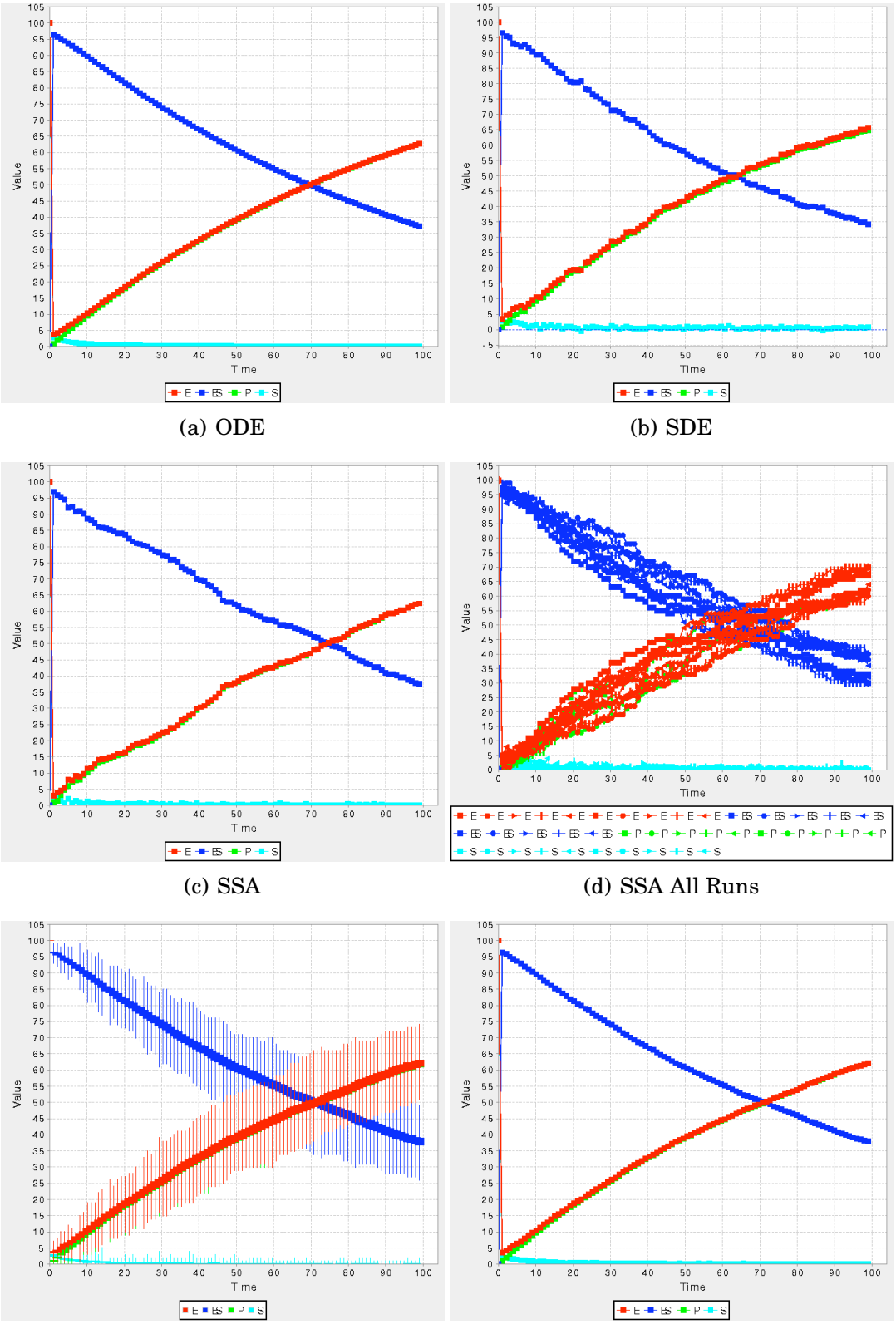
*Listing 6.2:* Michaelis-Menten model

As described in the model, the initial concentration of the enzymes and substrates is 100; the rate to combine enzymes and substrates is 1.0; to separate them is 0.1 and to create products is 0.01.

Comparing these three charts, we can observe a very similar behaviour of all four species trends. In particular, around the time 60 - 80, the concentration of P (and E) exceeds the number of ES independent of the solution algorithm.

### 6.2.2 The Schlögl model

The Schlögl model is a famous artificial chemical system used to describe bistable behaviour in the state variable for certain parameters.




(a) ODE (b) SDE

(c) SSA (d) SSA All Runs

(e) Candlesticks (f) SSA Average

*Figure 6.1:* Michaelis-Menten Charts

```
X = 250;
c1 = 3E−7;
c2 = 1E−4;
c3 = 1E−3;
c4 = 3.5;
N1 = 1E5;
N2 = 2E5;
R1,    −> X, [(c1/2)∗N1∗X∗(X−1)];
R2, X −> ,  [(c2/6)∗X∗(X−1)∗(X−2)];
R3,    −> X, [c3∗N2];
R4, X −> ,  [c4∗X];
```

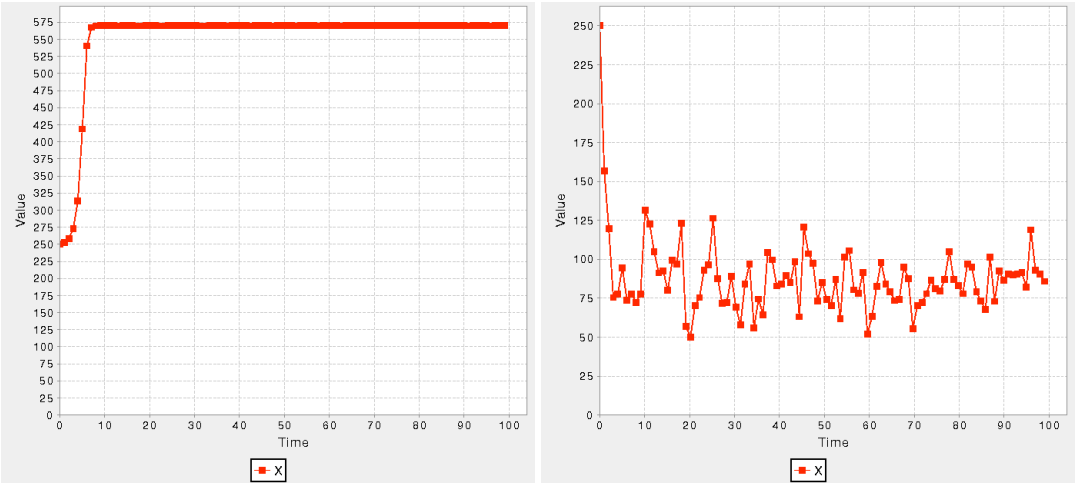
*Listing 6.3:* Schlögl model

As the charts below demonstrate, the trajectory of X can be accurately simulated only with stochastic algorithms. During these simulations the state variable has always taken values above the initial condition (250), either using the SDEs and the SSA simulator. Assuming bistable states, the mean and the variance do not have a physically significant meaning: so both SDEs and SSA simulation were computed only a single time.

Unlike the Michaelis-Menten model, SSA Gibson Bruck simulator is slower than SDEs computing the Schlögl model.

The Schlögl model, due to its rare events that arise in multi-scale systems when we calculate exponentially large or small observables, has a poor description if we simulate it with the diffusion approximation. This issue resides in the diffusion approximation of the SDE process (drift term + diffusion term), which leads to a wrong result simulating exponential values. In order to understand the problem, first of all we can give a deterministic description of the problem, defining the concentration $x = X/V$, assuming V the system volume, with the ODE $\frac{dx}{dt} = u(x) - d(x)$, with $u(x) = c1 * x^2 + c4$ and $d(x) = c2 * x^3 + c4 * x$. In this form 3.12 can be written as

$$dx = [u(x) - d(x)]\, dt + \sqrt{u(x) + d(x)} dW \tag{6.3}$$




(a) ODE

(b) SDE

(c) SSA

(d) SSA All Runs

*Figure 6.2:* Schlögl Charts

Passing to an intensive quantity $x = n/V = \varepsilon n$ in order to analyse large system size limits, we can rewrite 6.3 in this form

$$dx = [u(x) - d(x) + \varepsilon (u_1(x) - d_1(x))]\, dt + \sqrt{\varepsilon (u(x) + d(x))} dW \qquad (6.4)$$

that corresponds to the original SDE only with the rates rescaled by $\varepsilon$. Applying to 6.4 the large deviations principle leads us to a wrong Partial Differential Equation (PDE).

A satisfactory description of the issue can be found in [24].

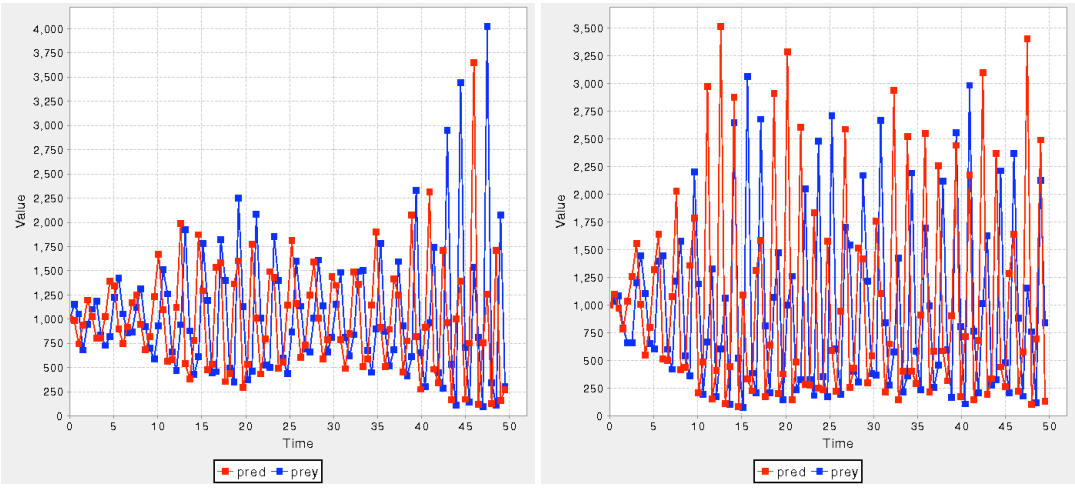
### 6.2.3 The Lotka-Volterra model

The Lotka-Volterra model is one of the simplest models in which predators and prey interact together, in a one-prey-group and one-predator-group scenario. In this example, I set the quantity of food at 1, the number of predators (pred) and prey (prey) at 1000.

```
food = 1;
prey = 1000;
pred = 1000;
natc = 0;
r1, food + prey −> prey + prey + food, 10;
r2, prey + pred −> pred + pred,     0.01;
r3,         pred −> natc,           10;
```

*Listing 6.4:* Lotka-Volterra model

Three simple reactions describe the model: food and prey generate two prey and release a food with rate 10; a prey and a predator react with rate 0.01 and become two predators; finally a predator dies of natural causes (natc) with rate 10.




(a) SDE (b) SSA

*Figure 6.3:* Lotka-Volterra Charts

Both charts are characterised by oscillations in the population size of both preys and predators, as predicted: using the SDEs simulator we can save a large amount of time computing these oscillations.

### 6.2.4 Final results

In the table below I show the average time (in seconds) spent by each algorithm for simulating the three models described above. Each model was simulated ten times with all the algorithms:

| Algorithm/Model | Michaelis-Menten | Schlögl | Lotka-Volterra |
|---|---|---|---|
| ODEs | 0.93 | 0.18[3] | 0.25[3] |
| SDEs | 2.81 | 4.78 | 2.83 |
| Gillespie's SSA | 0.15 | 3.72 | 7.38 |
| Gibson-Bruck Next React. | 0.16 | 4.01 | 7.65 |
| Gillespie's $\tau$-leap | 0.12 | 3.69 | 7.12 |

As we can see, Gillespie's $\tau$-leap method is the fastest way to solve either Michaelis-Menten model and Schlögl model. In these examples, both in the Schlögl model and in the Lotka-Volterra model, ODEs are not good

[3]ODEs became stable after a very short amount of time, so this method is not relevant

enough to describe the behaviour of the species trends. Instead SDEs and SSAs algorithms, because of their stochastic nature, are suitable for all three models: moreover SDEs are faster in the Lotka-Volterra model, and a little bit slower in the Michaelis-Menten model.

### 6.2.5 Concluding remarks

I presented a simulator which uses stochastic differential equations to solve chemical reactions, and I compared this simulator with other two standard ways to solve chemical reaction systems. After making several tests, a few shown in Section 6.2, I can assert this simulator is good enough for models when we treat chemical reaction systems with non-exponential size variables (i.e. without multi-scale problems). This simulator has also a better performance than a traditional stochastic simulator algorithm in some models, and can represent a valid alternative when these become too slow to compute the given model.

# Chapter 7

# Other works on Dizzy

DURING my period at the University of Edinburgh, besides developing the statistical analysis of the results and the SDE Simulator, I focused my attention also to other different issues of the Dizzy framework: the Sensitivity (Section 7.1) and the Deadlock Analysis (Section 7.2).

Sensitivity Analysis involves the study of different behaviour that a chemical kinetic model can exhibit when we change the values of some initial parameters. In particular I am showing how the evolution of a simulated trajectory of one or two species changes with a different number of initial molecules: this can be very useful for the model enhancement: all the data are represented in a three-dimensional chart, so trajectories of each new-defined model can be viewed within a single instance of simulation.

Deadlock Analysis instead regards a particular behaviour of Stochastic Simulation Algorithms. These algorithms indeed work, as I have detailed in Chapter 3, calculating the probability to get a fired reaction in each time interval (typically such time step is defined as $\tau$). When all the molecules per species in our model are consumed, the probability that a reaction can fire is null. So I have developed a control system in order to detect when the probability to get a new reaction becomes null.

## 7.1 Sensitivity Analysis

Sensitivity Analysis of chemical kinetics model has been developed with the intention to show how slight changes in a chemical system can cause great differences in the final result. In biological systems exact parameter values are difficult or impossible to obtain, so modellers are motivated to find out which parameter values have greatest impact. I have developed and embedded in Dizzy a system which allows the user to set a range of values of a single parameter of a chemical reaction model, either a species or a reaction rate. Once the user has defined a range, he must also defines the number of steps: defining in example a number of 4 steps, the range will be divided in four parts, and five different instances of the considered variable are created.

These new variables are used also to create new instances of the model, hence to perform different simulations, one for each instance of the model. The result of the simulation is shown in Figure 7.1.

Three-dimensional charts are provided by a deep proofreading of Surface Plotter 1.30B2[1], originally written by Yanto Suryono [27]. Indeed the provided code is a stand-alone Java applet application that shows three-dimensional charts having in input a defined algebraical formula. I have worked to extract the graphical part from the applet application, and embedded it into Dizzy by creating a new JAR package.

## 7.2 Deadlock Analysis

As already described at the beginning of Chapter 7, deadlock analysis involves the detection of a null probability in Stochastic Simulation Algorithms. Gillespie [14] has given a detailed description in his paper about

[1]The original version of this package can be found at the Yanto Suryono's web site `http://www.fedu.uec.ac.jp/~yanto/java/surface/`

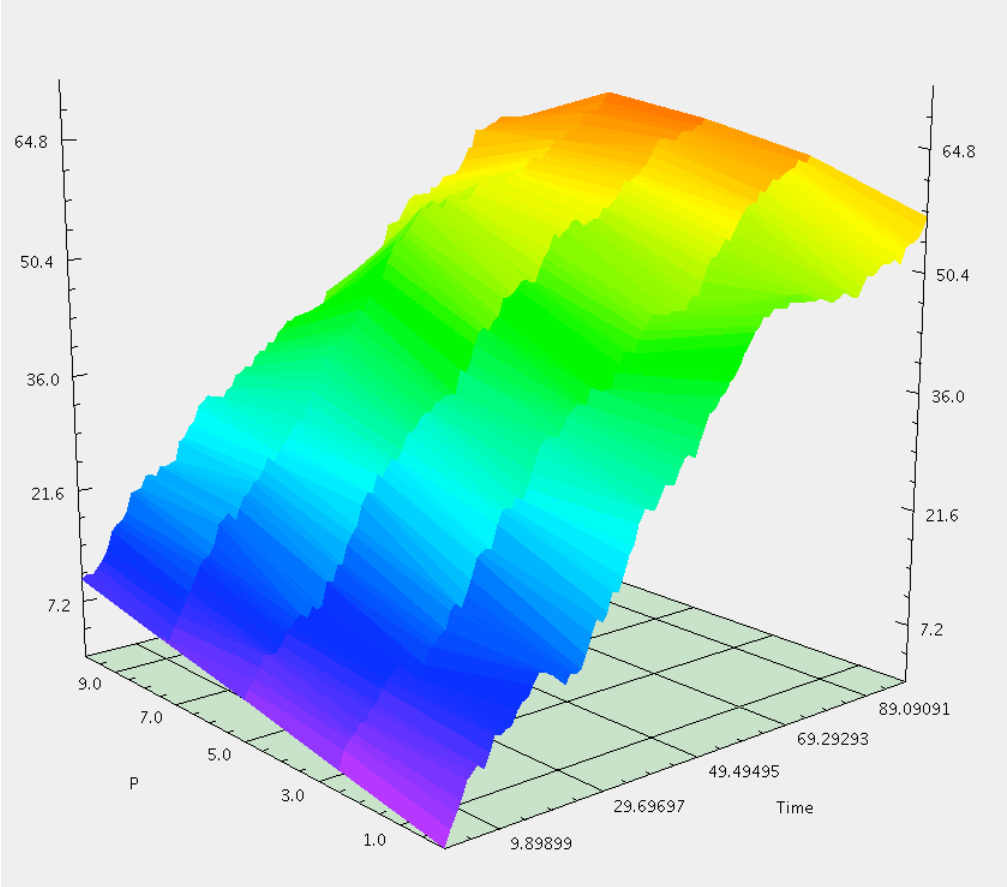



*Figure 7.1:* Three-dimensional chart of the Michaelis-Menten model, which represents the behaviour of the species $P$. The model has been simulated five times with a different initial concentration of molecules $E$, from 100 (the standard condition) to 80.

this problem, comparing the chemical oscillations between deterministic reaction-rate equations and the first moment of the solution of the stochastic master equation. These oscillations cause the fast consumption of molecules in some kinetics models, hence the probability that a reaction occurs in our model becomes null.

In order to alert the user that our model has reached a deadlock state, I have added a control in each stochastic simulator, to check if the probability to fire a reaction is null or not. That is, the simulator reports then it is impossible for any reaction to fire.

# Chapter 8

# Concluding Remarks and Future Works

THIS paper has described and illustrated the use of three different methods for calculating the trajectories of chemical reaction models, the Java framework Dizzy and the work I have produced on that specific tool.

After a large theoretical overview on Ordinary Differential Equations (ODEs) and Stochastic Simulation Algorithms (SSAs), focusing on the main features of these two different classes of simulation, I have introduced Dizzy, the kinetic simulation software developed by the University of Edinburgh for solving chemical reaction models. Indeed these two chapters have defined the theoretical and the practical side of chemical kinetics, giving an overview of the main issues faced during the solution of these chemical kinetics models.

I have shown the Statistical Analysis I have developed and embedded in Dizzy, coupled with the brand new profiling feature, added in order to understand which reactions fire during a simulation process. Furthermore I have studied how Stochastic Differential Equations (SDEs) can deal with chemical reactions in order to compute a well-defined solution. In order to demonstrate its efficiency and accuracy, the SDE algorithm

has been tested on different well-known models: the Michaelis-Menten, the Schlögl and the Lotka-Volterra model. Also two minor works have been presented, in order to get sensitivity and deadlock analysis.

Comparing Dizzy with other software, I can say that Dizzy is the most complete software that allows users to solve chemical kinetics reaction models. It allows users to perform simulations with three different families of simulation methods, and that is a unique peculiarity in the world of chemical reaction kinetics simulation software.

I have improved the analysis of the results, increasing the level of knowledge and insight obtained from the data produced by the simulators. In particular the Profile analysis is an important feature to evaluate models which involve "rare events", like the GAL yeast pathway: indeed analysing the Profile chart of this model, we have found some differences compared to Ramsey's article [23], as described in Section 5.3.1.

The SDE simulator in particular appears to be a valid alternative to the most common methods developed to solve some chemical kinetics models, like Lotka-Volterra. Since I have shown the relation between the SDE and the traditional SSA algorithms, I can assert that a chemical kinetics model can be solved using either with SDE or SSA methods, obtaining the same results. Having defined a sort of equivalence between these two methods, users can now choose the fastest way to solve any chemical kinetics models, according to the peculiarities and the distinctive features of the model.

Several improvements can still be made to the Dizzy framework, namely in the Stochastic Differential Equation algorithm. Indeed the simulator, which solves chemical kinetics reactions using this method, is not able to calculate either a relative or an absolute tolerance yet, in contrast to the ODE algorithm OdeToJava. This feature can be very useful in the future to increase the reliability of the algorithm, and also to improve the efficiency of the simulation method according to the accuracy required.

Another interesting improvement in Dizzy can be obtained by developing a method to predict the number of simulations to perform in order to

reach a pre-defined level of accuracy of the final results. As I have previously described in Section 4.2, Dizzy allows users to define an ensemble size: this number defines how many simulations of the same chemical kinetics model Dizzy has to perform before showing the final result. Due to the fact that some models are hard to solve, because of either the complexity of the chemical system or the chosen algorithm, the ensemble size should be set with regard by the user, trying to specify the lowest number possible. It would be very useful if Dizzy could choose the best ensemble size, according to a desired accuracy requested by the user.

Finally, I can suggest Dizzy should be able to read different kinds of SBML languages. Currently Dizzy understands only SMBL Level 1 models, which now has become a quite old standard for chemical models[1]. Today Level 2 and brand new Level 3 standards are often used to define complex chemical kinetics models, and the majority of the chemical simulators enable users to perform models written with these standards.

Despite this drawback, Dizzy remains a powerful and efficient tool offering sound analysis methods to users, easing the burden of analysing the complex chemical models of today and tomorrow.

[1]Standard Biology Model Language (SBML) specifications can be found at the website `http://sbml.org/index.psp`

# Appendix A

# Yeast model

## A.1 Template definitions

```
#model gal4_indirect_combined_fracsat;
// ================================
// % One Binding Site, Three States
// ================================
#define fracSatThreeStatesOneSite( kfp, krp, kfr, krr, f0,
 f1, fracsat )
{
 kp = kfp/krp;
 kr = kfr/krr;

 kpf0 = [ kp*f0 ];
 krf1 = [ kr*f1 ];

 fracsat = [ kpf0 / (1.0 + kpf0 + kpf0*krf1) ];
}

// ================================
// % Two Binding Sites, Two States
// ================================
#define fracSatThreeStatesTwoSites( kfp, krp, kfr, krr, qr, f0,
 f1, fracsat )
{
```

```
 kp = kfp/krp;
 kr = kfr/krr;

 kpf0 = [ kp*f0 ];
 kpf0_2 = [ kpf0*kpf0 ];

 krf1 = [ kr*f1 ];
 krf1_2 = [ qr*krf1*krf1 ];

 numerator = [ (kpf0_2) +
               (2.0 * kpf0) +
               (2.0 * kpf0_2 * krf1) ];

 fracsat = [ numerator / (numerator +
               1.0 +
               (kpf0_2 * krf1_2) +
               (2.0 * kpf0 * krf1)) ];
}

// ==================================
// % Four Binding Sites, Three States
// ==================================
#define fracSatThreeStatesFourSites( kfp, krp, kfr, krr, qr, f0,
 f1, fracsat )
{
 kp = kfp/krp;
 kr = kfr/krr;

 kpf0 = [ kp*f0 ];
 kpf0_2 = [ kpf0*kpf0 ];
 kpf0_3 = [ kpf0_2*kpf0 ];
 kpf0_4 = [ kpf0_3*kpf0 ];

 krf1 = [ kr*f1 ];
 krf1_2 = [ qr*krf1*krf1 ];
 krf1_3 = [ qr*krf1_2*krf1 ];
 krf1_4 = [ qr*krf1_3*krf1 ];

 numerator = [ (4.0 * kpf0 ) +
               (12.0 * kpf0_2 * krf1 ) +
               (6.0 * kpf0_2 ) +
```

```
                (4.0 * kpf0_3 ) +
                (12.0 * kpf0_3 * krf1 ) +
                (12.0 * kpf0_3 * krf1_2 ) +
                ( kpf0_4 ) +
                (4.0 * kpf0_4 * krf1 ) +
                (4.0 * kpf0_4 * krf1_3 ) +
                (6.0 * kpf0_4 * krf1_2 ) ];
 fracsat = [ numerator / (numerator +
                1.0 +
                (4.0 * kpf0 * krf1) +
                (6.0 * kpf0_2 * krf1_2) +
                (4.0 * kpf0_3 * krf1_3) +
                (kpf0_4 * krf1_4) )];
}
// ====================================
// % Five Binding Sites, Three States
// ====================================
#define fracSatThreeStatesFiveSites( kfp, krp, kfr, krr, qr, f0,
 f1, fracsat )
{
 kp = kfp/krp;
 kr = kfr/krr;
 kpf0 = [ kp*f0 ];
 kpf0_2 = [ kpf0*kpf0 ];
 kpf0_3 = [ kpf0_2*kpf0 ];
 kpf0_4 = [ kpf0_3*kpf0 ];
 kpf0_5 = [ kpf0_4*kpf0 ];

 krf1 = [ kr*f1 ];
 krf1_2 = [ qr*krf1*krf1 ];
 krf1_3 = [ qr*krf1_2*krf1 ];
 krf1_4 = [ qr*krf1_3*krf1 ];
 krf1_5 = [ qr*krf1_4*krf1 ];

 numerator = [ (5.0 * kpf0 ) +
                (10.0 * kpf0_2 ) +
                (20.0 * kpf0_2 * krf1 ) +
                (30.0 * kpf0_3 * krf1_2 ) +
                (30.0 * kpf0_3 * krf1 ) +
                (10.0 * kpf0_3 ) +
                (5.0  * kpf0_4) +
```

```
                (30.0 * kpf0_4 * krf1_2 )+
                (20.0 * kpf0_4 * krf1 ) +
                (20.0 * kpf0_4 * krf1_3 ) +
                (       kpf0_5 ) +
                (5.0  * kpf0_5 * krf1 ) +
                (5.0 * kpf0_5 * krf1_4 ) +
                (10.0 * kpf0_5 * krf1_3 ) +
                (10.0 * kpf0_5 * krf1_2 ) ];

 fracsat = [ numerator / (numerator +
                1.0 +
                (5.0 *    kpf0   * krf1) +
                (10.0 *  kpf0_2 * krf1_2 ) +
                (10.0 *  kpf0_3 * krf1_3 ) +
                (5.0 *    kpf0_4 * krf1_4 ) +
                (       kpf0_5 * krf1_5 ) ) ];
}

// ============================================
// % Simple Transcription and Translation model
// ============================================
#define yeastGeneFast( fracsat,
                       mRNA,
                       ki_mRNA,
                       kd_mRNA,
                       prot,
                       ki_prot,
                       kd_prot)
{
make_mrna, -> mRNA, [ fracsat * ki_mRNA ];
degrade_mrna, mRNA -> , kd_mRNA;

make_prot, $mRNA -> prot, ki_prot;
degrade_prot, prot ->, kd_prot;
}
```

---

## A.2 Galactose pathway

```
molecTomM = 4.65 * 10^(-8); // mM/molecules
```

```
// zero−galactose initial conditions:
GAI = 0.0;                    // internal galactose
GAE = 0.5 / molecTomM;        // external galactose
GA1P = 0.0;                   // galactose−1−phosphate
UGA = 1490322.0;              // UDP−galactose
UGL = 5217204.0;              // UDP−glucose
G1 = 1027.0;                  // Gal1p
G2 = 922.0;                   // Gal2p
G4 = 7.4;                     // Gal4p
G7 = 395.0;                   // Gal7p
G10 = 602.0;                  // Gal10p
G3 = 3580.0;                  // Gal3p
G80 = 809.0;                  // Gal80p
G7d = 92.0;                   // Gal7p(dimer)
G10d = 213.0;                 // Gal10p(dimer)
G4d = 5.0;                    // Gal4p(dimer)
G80d = 384.0;                 // Gal80p(dimer)
G3i = 0.00;                   // Gal3p−Galactose complex
G4dG80d = 106.0;              // Gal4p−Gal80p complex
G80G3i = 0.0;                 // Gal80p−Gal3p−galactose

//raffinose condition, steady−state:
R1 = 0.2;                     // GAL1 mRNA
R2 = 0.26;                    // GAL2 mRNA
R7 = 0.11;                    // GAL7 mRNA
R10 = 0.2;                    // GAL10 mRNA
R3 = 0.75;                    // GAL3 mRNA
R80 = 0.37;                   // GAL80 mRNA

// Kinetic Parameters:

alpha_TR = 1.0;               // dimensionless
Km_TR = 1.0 / molecTomM;      // molecules
k_TR = 4350.0;                // min^−1

kcat_GK = 3350.0;             // min^−1
Km_GK = 0.6 / molecTomM;      // molecules

kcat_TF = 59200.0;            // min^−1
Km_ga1p_TF = 4.0 / molecTomM; // molecules
Km_ugl_TF = 0.26 / molecTomM; // molecules
```

```
kcat_EP = 38900.0;              // min^-1
Km_uga_EP = 0.22 / molecTomM; // molecules
Keq_EP = 3.5;                   // dimensionless
Km_ugl_EP = 0.25 / molecTomM; // molecules

q = 30.0;  // cooperativity for multiple repressor (G80d) binding

kfp = 0.1;
kfr = 0.1;
krp = 1.14676;
krr = 1.82588;

kdr_struct = 0.0336;
kdr_reg = 0.159193;
kdr_2 = 0.0156;

kir_struct = 1.09;
kir_reg = 1.44385;
kir_2 = 0.52;

kdp_struct = 0.00197;
kdp_reg = 0.003747;
kdp_2 = 0.00197;

kip_struct = 9.92;
kip_reg = 17.9844;
kip_2 = 6.94;

// Metabolic Pathway Velocities:

v_TR = [ k_TR * G2 * (GAE - GAI) /
       (Km_TR + GAE + GAI + (alpha_TR*GAE*GAI/Km_TR)) ];
                                // v_TR : molecules/min

v_GK = [ kcat_GK * G1 * GAI / (Km_GK + GAI) ];
                                // v_GK : molecules/min

v_TF = [ kcat_TF * G7d * GA1P * UGL /
          (Km_ga1p_TF*UGL + Km_ugl_TF*GA1P + GA1P*UGL) ];
                                // v_TF : molecules/min
```

```
v_EP = [ kcat_EP * G10d * (UGA - UGL/Keq_EP)/
          (Km_uga_EP + UGA + (UGL*Km_uga_EP/Km_ugl_EP)) ];

// Reactions:
-> GAI, [ v_TR ];
GAI ->, [ v_GK ];
-> GA1P, [ v_GK ];
GA1P ->, [ v_TF ];

-> UGA, [ v_TF ];
UGA ->, [ v_EP ];
-> UGL, [ v_EP ];
UGL ->, [ v_TF ];

G4 + G4 -> G4d , 0.1;
G4d -> G4 + G4, 1.0;
G7 + G7 -> G7d , 0.1;
G7d -> G7 + G7, 170.2;
G10 + G10 -> G10d , 0.1;
G10d -> G10 + G10, 170.2;
G80 + G80 -> G80d , 0.1;
G80d -> G80 + G80, 170.2;
G3 + GAI -> G3i , 0.00000040;
G3i -> G3 + GAI, 890.0;
G4d + G80d -> G4dG80d , kfr;
G4dG80d -> G4d + G80d, krr;
G80 + G3i -> G80G3i , 0.1;
G80G3i -> G80 + G3i, 0.03023;

#ref fracSatThreeStatesOneSite "oneSite"
    (kfp, krp, kfr, krr,    G4d, G80d, fracsat_onesite);
#ref fracSatThreeStatesTwoSites "twoSites"
    (kfp, krp, kfr, krr, q, G4d, G80d, fracsat_twosites);
#ref fracSatThreeStatesFourSites "fourSites"
    (kfp, krp, kfr, krr, q, G4d, G80d, fracsat_foursites);
#ref fracSatThreeStatesFiveSites "fiveSites"
    (kfp, krp, kfr, krr, q, G4d, G80d, fracsat_fivesites);

#ref yeastGeneFast "gal7" ( fracsat_twosites, R7,
    kir_struct, kdr_struct, G7, kip_struct, kdp_struct );
```

```
#ref yeastGeneFast "gal10" ( fracsat_foursites, R10,
    kir_struct, kdr_struct, G10, kip_struct, kdp_struct );
#ref yeastGeneFast "gal1" ( fracsat_foursites, R1,
    kir_struct, kdr_struct, G1, kip_struct, kdp_struct );
#ref yeastGeneFast "gal2" ( fracsat_fivesites, R2,
    kir_2,      kdr_2,      G2, kip_2,      kdp_2     );
#ref yeastGeneFast "gal3" ( fracsat_onesite, R3,
    2.0*kir_reg,   kdr_reg,   G3, kip_reg,    kdp_reg );
#ref yeastGeneFast "gal80" ( fracsat_onesite, R80,
    kir_reg,    kdr_reg,   G80, kip_reg, kdp_reg );

// G4 creation and decay processees
G4 -> , kdp_reg;
-> G4 , 0.86181;

// dimer decay
G4d -> , kdp_reg;
G7d -> , kdp_struct;
G10d -> , kdp_struct;
G80d -> , kdp_reg;

// other molecular decay processes
G3i -> , kdp_reg;
G4dG80d -> , kdp_reg;
G80G3i -> , kdp_reg;
```

# Bibliography


[1] D. Adalsteinsson, D. McMillen, and T. C. Elston, "Biochemical network stochastic simulator (BioNetS): software for stochastic modeling of biochemical networks," *BMC Bioinformatics*, vol. 24, no. 5, 2004.

[2] K. Burrage and E. Platen, "Runge – Kutta methods for stochastic differential equations," *Ph.D. thesis, University of Queensland*, 1999.

[3] K. Burrage, T. Tian, and P. Burrage, "A multi-scaled approach for simulating chemical reaction systems," *Progress in Biophysics and Molecular Biology*, vol. 85, 2004.

[4] K. Burrage, T. Tian, and P. Burrage, "Numerical methods for strong solutions of stochastic differential equations: an overview," *Proceedings: Mathematical, Physical and Engineering, Royal Society of London*, vol. 460, no. 2041, 2004.

[5] Y. Cao, D. Gillespie, and L. Petzold, "Efficient step size selection for the tau-leaping simulation method," *The Journal of Chemical Physics*, vol. 124, no. 044109, 2006.

[6] P. de Atauri, S. Ramsey, D. Orrell, and H. Bolouri, "Evolution of 'design' principles in biochemical networks," *IEEE System Biology*, 2004.

[7] J. R. Dormand and P. J. Prince, "A family of embedded Runge-Kutta formulae," *Journal of Computational and Applied Mathematics*, vol. 6, 1980.

[8] P. Ecuyer, L. Meliani, and J. Vaucher, "SSJ: A framework for stochastic simulation in Java," *Proceedings of the 34th conference on Winter simulation: exploring new frontiers*, 2006.

[9] M. A. Gibson and J. Bruck, "Efficient exact stochastic simulation of chemical systems with many species and many channels," *Caltech Parallel and Distributed Systems Group*, no. 26, 1999.

[10] D. Gillespie, "Exact stochastic simulation of coupled chemical reactions," *Journal of Physical Chemistry*, vol. 81, no. 25, 1977.

[11] D. Gillespie, "The chemical langevin equation," *Journal of Chemical Physics*, vol. 113, no. 1, 2000.

[12] D. Gillespie, "Approximate accelerated stochastic simulation of chemically reacting systems," *Journal of Chemical Physics*, vol. 115, no. 4, 2001.

[13] D. Gillespie, "Stochastic simulation of chemical kinetics," *Review in Advance*, 2006.

[14] D. Gillespie and M. Mangel, "Conditioned averages in chemical kinetics," *The Journal of Chemical Physics*, 1981.

[15] D. J. Higham, "An algorithmic introduction to numerical simulation of stochastic differential equations," *SIAM Review, Society for Industrial and Applied Mathematics Philadelphia, PA.*, vol. 43, no. 525, 2001.

[16] D. J. Higham, "Modeling and simulating chemical reactions," *University of Strathclyde Research Report*, vol. 2, 2007.

[17] S. Hoops, S. Shale, R. Gauges, C. Lee, J. Pahle, N. Simus, M. Singhal, L. Xu, P. Mendes, and U. Kummer, "COPASI - a COmplex PAthway SImulator," *Bioinformatics and Computational Biochemistry*, vol. 22, no. 24, 2006.

[18] H. Lamba, J. C. Mattingly, and A. M. Stuart, "An adaptive Euler-Maruyama scheme for SDEs: convergence and stability.

preprint," 2006. [Online]. Available: http://front.math.ucdavis.edu/math.NA/0601029

[19] N. L. Novere and T. S. Shimizu, "STOCHSIM: modelling of stochastic biomolecular processes," *Bioinformatics Application Note*, vol. 17, no. 6, 2001.

[20] I. Oppenheim, K. E. Schuler, and G. H. Weiss, "Stochastic and deterministic formulation of chemical rate equations," *Journal of Chemical Physics*, vol. 460, no. 50, 1969.

[21] J. Pahle, "Eine Hybridmethode zur Simulation biochemischer Prozesse (in German)," *Diploma thesis Univesitat Karlsruhe*, 2002.

[22] S. Ramsey, D. Orrell, and H. Bolouri, "Dizzy: stochastic simulation of large-scale genetic regulatory networks," *Journal of Bioinformatics and Computational Biology*, 2005.

[23] S. Ramsey, D. Orrell, and H. Bolouri, "Dizzy: stochastic simulation of large-scale genetic regulatory networks (supplementary material)," *Journal of Bioinformatics and Computational Biology*, 2005.

[24] K. Sargsyan, "Fluctuations in chemical reactions in a large volume," *GFD Program Proceedings*, 2005.

[25] R. Spiteri, "Introducing odeToJava: A problem-solving environment for initial-value problems," *Abstract at SciCADE*, 2005.

[26] R. Stallman, "GNU lesser general public license," *Computer software license agreement, Free Software Foundation, http://www.gnu.org/copyleft*, 1999.

[27] Y. Suryono, "Surface plotter," 1996. [Online]. Available: http://www.fedu.uec.ac.jp/~yanto/java/surface/

[28] F. K. Zimmermann and K. D. Enitan, *Yeast Sugar Metabolism*. Lancaster, PA, USA: North Holland, 1997.