Computational Methods for Stochastic Biological Systems

Thesis by
Michael Andrew Gibson

In Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy

California Institute of Technology
Pasadena, California

2000

(Submitted 18 April 2000)

Acknowledgements

I am particularly thankful to all my teachers, who gave me a broad grounding to pursue interdisciplinary science.

John Dell taught me everything I know about physics and a lot about problem solving; Michael Albert, Pat Gabriel, Victor Mizel and Juan Schäffer taught me math; John Liebermann taught me chemistry and introduced me to research; and Bruce Krogh and David Greve, my undergraduate advisors, let me be a part of some very exciting research.

Special thanks to Henry Lester for getting me interested in both biology in general and the Computation and Neural Systems program at Caltech in particular. CNS is a very exciting program where someone like me, who knows no biology coming in, can get a good grounding in the field and do interdisciplinary work, and I am very grateful to all the people who work so hard to make the program possible.

As to this particular branch of research, Paul Sternberg from Caltech's Division of Biology was my main biological contact, who helped me wade through the waters of molecular biology, and was willing to listen to many hopelessly naive models and ideas that I generated when I first started out. Hamid Bolouri and Eric Mjolsness worked with me on a review of the field of modeling gene regulation; that review pointed out several potential areas for further research. My work on stochastic simulation of biology grew out of discussions with Harley McAdams at Stanford and Adam Arkin at Berkeley. I have kept in relatively close contact with Adam, who has provided direction and encouragement over the last few years. Dan Gillespie at China Lake has been tremendously helpful, both by setting the stage for many of the results in this thesis, and for several incredibly stimulating discussions of stochastic processes.

The Caltech community has been very helpful in giving me feedback and providing a sounding board for ideas in development. Specifically, John Doyle has generated interest in biology among theorists, and is great at seeing implications and interesting areas for theory work. Several students and postdocs have also been particularly helpful: Reid Harrison, Anthony Leonardo, Fidel Santamaria Perez, and Sanza Kazadi, all CNS graduate students, struggled through many of the early biology classes with me. Ben Shapiro, Mark Borisuk, Tau-Mu Yi, and Andrey Levchenko have all given selflessly of their time and energy. Jim Primbs is the person I turned to with all my probability questions. My labmates in the parallel and distributed systems lab were always available for technical questions on the computer science side.

The funding for this work came from four sources: a National Science Foundation Graduate Research Fellowship, NSF Young Investigator Award CCR-9457811, ONR grant N00014-97-1-0293, and a JPL-CISM grant.

This thesis was created under the tutelage of Jehoshua “Shuki” Bruck, who gave me more advise and guidance than can adequately be expressed here. He has certainly answered the question I asked him the first time we met, more than five years ago, “what does a theorist do every day?”

Finally, thank you to my parents, my first and best teachers.

Abstract

The virtual completion of the genome project and prodigious amounts of work by different biologists throughout the world have elucidated many of the components of biological systems. The genes (and hence proteins) are largely known, and the tools of molecular biology allows one to manufacture and express them, so as to understand their function. Given this increased understanding of components, the next step in understanding complex biology will be understanding systems, which will almost certainly involve formal, detailed, and quantitative models.

One of the great challenges of modeling biological systems is that they tend to "break the math." Biological systems have small numbers of molecules, operate far from equilibrium, change shape and size, etc. This thesis develops mathematical and computational tools for biological systems with few molecules. Such systems are particularly problematic because the usual macroscopic view of chemistry, in which concentrations of molecules vary continuously, continually, and deterministically, does not work. Rather, one needs to use the mesoscopic view of chemistry: molecules undergo discrete reaction events, and the timing of these events is probabilistic. There are many standard numerical computational techniques for the macroscopic view, but far fewer for the mesoscopic view.

This thesis develops (1) an efficient, exact stochastic simulation algorithm, to generate trajectories of mesoscopic biological systems, (2) a sensitivity analysis algorithm, to quantify how a model's predictions depend on the exact values of parameters (e.g., rate constants) used, and (3) a parameter estimation algorithm, to estimate the values of model parameters from observed trajectories.

Contents

List of Figures

List of Tables

Part I

Introduction

Chapter 1 Overview

These are the times that excite men's souls.

The last 50 years have witnessed an amazing revolution in biology. The view that has emerged is that biological systems are very complex machines, which nonetheless obey the laws of physics and chemistry. These machines store genetic information in long linear molecules called DNA, which is passed from parents to children. In particular, DNA provides information for how to build proteins, complex macromolecules that are the workhorses of cells. A single protein may itself be a complex machine: hemoglobin transports oxygen, ion channels act as the "transistors" of the nervous system by conducting different amounts of current in response to applied voltages or other signals, molecular motors transform energy to work in complex ways. At a higher level, proteins can be grouped into pathways — biological circuits — where complex function is achieved not by individual molecules, but by combinations of molecules working in concert.

Human DNA consists of about 3 billion bases: A, C, T, or G. Simpler organisms have much less DNA. The first genome, that of a simple virus, was sequenced in 1977 (i.e., the complete sequence of As, Cs, Ts, and Gs was determined). Such a genome has tens of thousands of bases. Recently it has become possible to sequence much larger genomes beyond viruses: bacteria, eukaryotes such as yeast, and more recently multi-cellular animals: the nematode *Caenorhabditis elegans* (1998) and the fruit fly *Drosophila melanogaster* (2000). The human genome project is expected to determine the complete sequence of all 3 billion bases of human DNA within the next year.

1.1 Engineering and Biology

The human genome is thought to consist of about 100,000 genes, in other words, it contains recipes for making 100,000 different kinds of proteins. For comparison, the early viruses sequenced have 50 or so genes, and bacteria have 3,000-5,000. In the language of engineering, simple viruses are machines with 50 kinds of components, bacteria are machines with 3,000-5,000 types of components, and humans are complex machines with 100,000 types of components. While that number may seem frightening to a computer engineer used to dealing with complex systems built out of one kind of component — transistors — it is not unthinkably high; the Boeing 777 has 150,000 distinct subsystems.

Components of a system are important: much work was (and some Nobel prizes were) involved in developing transistors to replace vacuum tubes in early computers, then figuring out how to fabricate multiple transistors on a single chip. However, as time has passed on, many fundamental component problems have been solved, and the concentration has moved to how one combines components into systems. For example, the Intel P3 contains 28 million transistors. Although the individual components are well understood, there is still an enormous amount of work to put those components together into a useful system.

Years of unrelenting work by experimental biologists has brought molecular biology to a point where components are becoming understood. That is not to say that all components are understood, or that any component is understood in its entirety from first principles, but rather that components are known and enough is understood about them that it is now possible to understand simple biological systems in terms of their components.

In the same way that computers are built out of transistors and airplanes are built out of rivets, gears, etc., biological systems are built out of proteins. The human genome project is providing a "parts list" of proteins, much subsequent work will focus on determining the function of those individual components. *The* biology problem of the twenty-first century will be figuring out how those components work together to form systems, and understanding those systems at a low level in terms of their components. For lack of a better term, this thesis will refer to this burgeoning field as molecular systems biology.

Engineering techniques will be very important in molecular systems biology. Understanding systems is not the same as understanding components, and the properties of components one need understand depends greatly on the relationship of the components in a system. Very importantly, the sheer complexity of these systems dwarfs our ability to reason about them in ad-hoc or "back of the envelope" ways. Rather, one needs formalized methods for analysis and simulation of systems. As an analogy, consider electrical systems. Simple resistors, capacitors, inductors, power supplies, diodes, and transistors are well understood. However, one can combine them into arbitrarily complex systems, and the behavior of arbitrary systems is not well understood. Much of electrical engineering focuses on how to build systems that are understandable and achieve the desired performance. But more to the point, there are formalized reasoning techniques that let one simulate arbitrary electrical systems built out of these simple components.

1.2 Representing Biological Systems

To begin reasoning about biological systems using engineering thought processes, one needs to represent biological systems in a formal way — a standardized notation or terminology that different investigators agree upon and computers can operate on [6]. Such a representation is called a *model*. The term 'model' means different things to different people. For example:

- Biologists sometimes refer to a simple organism that displays a behavior of interest as a model: the virus lambda phage is a model of a genetic switch, the nematode *C. elegans* is a model of development, etc.
- Other times, biologists refer to a cartoon diagram as a model: a schematic of the key molecules involved and their interactions is a model.
- Engineers refer to a detailed, quantitative representation as a low-level model: a circuit simulator has a transistor model.
- Engineers typically do not refer to a higher-level representation as a model, but as a block diagram. Such a diagram might have an engine (or a cache for computer engineers) as a building block, rather than the detailed components that make up the engine.

In this thesis, a model is any formal representation of a system that leads to specific predictions. So the first usage is out, but the other three stand. A person in Los Angeles can write down a model of some system there, send that model to a person in New York, and the New Yorker will know exactly what is meant by the model and what predictions are made. (The New Yorker may not agree with the model; he need only agree what predictions the model makes, not whether those are correct.)

Models come in many shapes and sizes [14]. There are quantitative models, whose predictions are numeric, and qualitative models, whose predictions are not. (A quantitative model of a computer might contain voltages and currents of different transistors; a qualitative model of the same system might talk about 0s and 1s.) There are realistic models and phenomenological models. (A realistic model of turbulence over airplane wings might consider the Navier-Stokes equation applied to the air, a phenomenological model might just consider the turbulence to be Gaussian noise.) There are also model types geared to certain mathematical or computational assumptions: linear models, neural network models, etc. To the extent possible, this thesis will consider quantitative, realistic models, independent of computational assumptions, and will explicitly state computational or mathematical assumptions as they occur.

For a biological system, what does a quantitative, realistic, computation independent model look like?

Biological systems are not built out of transistors or rivets; they are built of molecules. Molecules move around, subject to physical laws and constraints: diffusion, temperature, electrical forces, etc. Molecules also react with each other, forming different types of molecules. To specify reactions, one writes chemical equations such as

$$A + B \to C$$

which says that one molecule of type A reacts with one molecule of type B to form one molecule of type C.

A complex biological system consists of many chemical reactions and possibly many physical constraints. Each individual chemical reaction is simple; each individual physical constraint is simple. The overwhelming complexity of biological systems comes from the sheer numbers of chemical reactions and physical processes, and their strong interconnectedness.

This thesis is concerned primarily with detailed, realistic, quantitative models consisting of chemical equations only, although certain physical effects — temperature and volume change — will be considered in Chapter 6 as they affect chemical reactions.

1.3 Computation with Models

Suppose a simple biological system boils down to the equations

$$\begin{aligned} A + B &\xrightarrow{k_1} C \\ 2C &\xrightarrow{k_2} A + D \\ D &\xrightarrow{k_3} B \end{aligned}$$

(For pedagogical purposes, most examples in this thesis will be simple, but the techniques used scale to larger systems as well, such as the model of the virus lambda phage in Chapter 4.)

1.3.1 Macroscopic Chemistry

In the usual, macroscopic view of chemistry, one deals with concentrations. The concentration of molecule type A is denoted [A], and is typically measured in moles per liter. (One mole $= 6.02 \times 10^{23}$ molecules, and the symbol M is used for 'moles per liter.') In this view, the model above leads to a set of ordinary differential equations:

$$\begin{aligned} \frac{d[A]}{dt} &= -k_1 \times [A] \times [B] + k_2 \times [C]^2 \\ \frac{d[B]}{dt} &= -k_1 \times [A] \times [B] + k_3 \times [D] \\ \frac{d[C]}{dt} &= k_1 \times [A] \times [B] - 2 \times k_2 \times [C]^2 \\ \frac{d[D]}{dt} &= k_2 \times [C]^2 - k_3 \times [D] \end{aligned}$$

which can be solved by standard numerical techniques, and so one can generate the time course of [A], [B], [C], and [D].

If one considers the model above augmented by the physical process of diffusion, then [A], [B], [C], and [D] are functions of position as well as time, and one may write partial differential equations which include diffusion terms. These too can be solved, using appropriate numerical techniques, and one gets [A], [B], [C], and [D] as a function of time and space.

Various analysis tools are available for the macroscopic chemistry:

- *Sensitivity analysis* to see how the time courses depend on k_1, k_2, and k_3.
- *Equilibrium analysis* to find the concentrations as $t \to \infty$.
- *Bifurcation analysis* to see how the equilibrium points change as k_1, k_2, and k_3 change.
- etc.

Furthermore, various model-building techniques and tools exist:

- *Parameter estimation* (sometimes called system identification or statistical learning theory) to find ks given time-courses.
- Various *approximation schemes*, such as linearization or neural networks to expedite parameter estimation.
- etc.

Finally, there are various standard approximation techniques useful for reasoning about systems and for improving calculations:

- *Linearization* about operating points.
- *Perturbation methods*, which separate fast and slow time scales.

These approximations are used, for example, in stiff solvers, which achieve efficiency by solving fast dynamics as fast temporal resolution and slow dynamics at slow resolution.

1.3.2 Limits of Macroscopic Chemistry

All is not well in the macroscopic view of chemistry. Specifically, the macroscopic view makes the following assumptions: (1) concentrations are well-defined quantities, (2) rate constants are well-defined quantities, (3) concentrations vary deterministically over time, and (4) concentrations vary continuously and continually. These assumptions are approximately satisfied if the number of molecules is sufficiently large. If the system is infinitely large (i.e., in the thermodynamic limit) they hold exactly.

Real systems are not infinitely large. Many real biological systems are very, very small, and some or all of the assumptions fail. For example, consider assumption (4). It says that if the concentration is now 1 M (one mole per liter), and 10 minutes from now, the concentration is 2 M, then at some point in between, it will be 1.5 M, at some point before that 1.25 M, etc. This seems reasonable enough in normal chemistry, and works well in large chemical systems (recall that 1 milliliter of water contains 3×10^{22} molecules of water, which is certainly a large system). However, real biological systems have amazingly low numbers of certain molecules. A cell will have one molecule of DNA, tens of molecules of certain RNAs and proteins, hundreds of molecules of other RNAs and proteins, etc. The statement that before seemed obvious now makes no sense. If there is 1 molecule now, and 10 minutes later there are 2 molecules, it is *not* true that at some intermediate time there were 1.5 molecules, or 1.25 molecules, etc. (This is similar to the interpretation of statistics such as "the average family has 2.4 children." While that statement may be true for the nation as a whole, there is certainly no individual family with 2.4 children.)

1.3.3 Mesoscopic Chemistry

A different chemistry holds at very small numbers of molecules [19]. In the mesoscopic view, one makes the assumption (1') the solution is well-mixed and at thermodynamic equilibrium (although not necessarily chemical equilibrium), which is essentially equivalent to (1) and (2). Assumptions (3) and (4) are replaced with: (4') concentrations change only by discrete numbers of molecules, corresponding to single reaction events, and (3') when such discrete changes occur is random or stochastic. (One typically phrases the mesoscopic chemistry in terms of number of molecules, rather than concentration, to keep things clearer.)

The reason to use mesoscopic chemistry is not that it is somehow more interesting mathematically, but rather, the macroscopic view breaks down (and makes incorrect predictions) for very small systems.

If one is forced to take the mesoscopic view, certain problems arise:

1. What is the formal transformation from chemical equations to mathematics?

2. How does one solve this math?

3. How does one deal with physical properties, such as diffusion, temperature change, etc.?

4. What analysis techniques can one use: sensitivity analysis, equilibrium analysis, bifurcation, etc.?

5. What model building techniques exist: parameter estimation, neural networks, etc.?

6. What approximation techniques exist: linearization, perturbation methods, etc.?

The answer to many of these question is “no techniques exist for the mesoscopic chemistry.” This thesis will develop some simple techniques to fill some of those holes:

1. There is a known way to translate to mathematics, which is covered in the background, Chapter 3.

2. To solve the mathematics, one does a Monte Carlo simulation of the type described by Gillespie [16]. One of the parts of this thesis contains improved algorithms for accomplishing such simulations.

3. Diffusion is a particularly hard problem, because it violates assumptions (1’) and (2’). (In the macroscopic chemistry, assumptions (1) and (2) are assumed to hold *locally.*) Changes such as temperature or volume can be handled by the Next Reaction Method, as described in Chapter 6.

4. Chapter 9 and 10 present a way to do mesoscopic sensitivity analysis. No other analysis techniques are developed here.

5. Chapter 11 presents a very simple parameter estimation algorithm.

6. There has been some interesting work on approximating mesoscopic chemical systems, which will not be covered in this thesis. (Some hint of this was reviewed in [14].)

1.4 Contributions of This Thesis

There are three main contributions of this thesis:

- Improved versions of two of Gillespie’s simulation algorithms. Both improved algorithms were previously described in [12]. Specifically:

  - The Next Reaction Method in Chapters 5 to 6 is more efficient than Gillespie's original algorithms [15, 16], taking time proportional to the logarithm of the number of reactions, rather than to the number of reactions itself. Further, the logarithm is an upper bound; the algorithm will typically do much better. Some previous work [27], focussing on surface processes, achieved the logarithmic result, but in that domain, the algorithm typically used the full logarithm.
  - The Next Reaction Method in Chapter 5 uses a single random number per simulation event, an improvement over the two previously needed for Gillespie's Direct Method.
  - The extended version of the Next Reaction Method, in Chapter 6, which uses a single random number for time-dependent Markov processes.
  - The extension of the Next Reaction Method to non-Markov processes in Chapter 6.

- The Multiple Next Reaction Method in Chapter 9, which runs multiple simulations, corresponding to different parameter values, in a much more efficient way than was previously possible. Also, applying the Multiple Next Reaction Method to the problem of sensitivity analysis, in Chapter 10.
- The Estimation of Stochastic Parameters algorithm in Chapter 11, which estimates rate constants from observed mesoscopic data. This algorithm is proved to be an inverse of the Next Reaction Method.

1.5 Organization

Much in the same way that all Gaul is divided into three parts, this thesis is divided into six parts.

Part I consists of this overview chapter.

Part II explains the mesoscopic chemistry and other sundry background for the thesis. Chapter 2 provides a more detailed description of representing biological systems and presents the key concepts of physical chemistry. Chapter 3 describes the mesoscopic physical chemistry, how to translate it to mathematics, and previous work on solving (analytically and numerically) that mathematics. Chapter 4 presents a simplified version of the Arkin et al. [3] model of lambda phage, a simple virus[1]. This model will serves as a running example for much of the rest of the text.

Part III deals with exact stochastic simulation of mesoscopic biological systems. Chapter 5 describes the basics of the Next Reaction Method, an improved version of Gillespie's First Reaction Method (detailed in Chapter 3). The Next Reaction Method takes time proportional to the logarithm of the number of reactions being simulated, and uses a single random number per simulation event. Chapter 6 extends the Next Reaction Method to time-dependent Markov processes and to non-Markov processes, which can be useful for (1) systems with additional physical properties, such as volume or temperature change, and (2) simplifying complex chemical processes where analytical solutions are available for some sub-process. Chapter 7 describes a different method for exact stochastic simulation: an improvement to Gillespie's Direct Method (also detailed in Chapter 3), which achieves the logarithmic speedup that the Next Reaction Method achieved. Chapter 8 applies the Next Reaction Method to the lambda phage model of Chapter 4.

Part IV deals with sensitivity analysis. Chapter 9 describes the Multiple Next Reaction Method (MNRM), an efficient algorithm for running multiple simulations with slightly different parameter sets. Chapter 10 uses MNRM as a building block for Mesoscopic Efficient Sensitivity Analysis, a numerical approximation algorithm for calculating "sensitivity gradients." The techniques developed are applied to the running lambda phage example.

Part V deals with parameter estimation. Chapter 11 presents the Estimation of Stochastic Parameters algorithm and applies it to test data from the running lambda phage model.

Part VI summarizes, pulls the pieces together, and provides directions for further research.

[1]This model is only presented as a motivating example. This thesis is not about lambda, not about modeling lambda, not about gene regulation, not about bacteriophages. Rather, this thesis is about general computational and mathematical techniques that can be used to deal with any mesoscopic chemical system. Lambda provides a good example of a mesoscopic chemical model of a molecular biological system, and hence is included as an illustration. The algorithms in the thesis apply to any mesoscopic system that can be described by a system of chemical reactions.

Having said that, specific parts any model will be very inefficient to simulate in too general a framework. This thesis provides some specific optimizations for the lambda model, but the idea of using optimizations, and shows the general framework of how to optimize for different models. Although the examples given are specific to the lambda model, these are just examples of the underlying framework.

Part II

Background

Chapter 2 Biological Systems and Physical Chemistry

2.1 Summary

This chapter does two things. First, it explains how to break a complicated biological system into a set of (simple) chemical reactions. Although this content is very familiar to most biologists and chemists, it may not be to the computer scientist or mathematician reading this thesis, and hence is included. Second, the chapter explains the basic physical chemistry underlying systems of chemical reactions, including why the usual macroscopic view of chemical kinetics is inappropriate for some systems, and how it should be supplanted with a mesoscopic view, i.e., a stochastic framework. A more detailed version of this stochastic framework may be found in the next chapter.

2.2 How to Represent Biological Systems

The process of creating a predictive model from biochemical details can be split into two parts. This chapter develops the concept of a calculation independent model, i.e., a formal, precise, and quantitative representation of biological processes. Subsequent chapters describe how to start with a calculation independent model, do calculations, and make predictions about the behavior of the system. There are numerous ways to do the calculation, depending on the assumptions one makes; this thesis will consider only the mesoscopic stochastic framework. For a review of other frameworks, see [14].

There are two reasons why having a calculation independent model is useful. First, biologists can develop a model — a precise representation of the processes involved in some biological process — without regard to the computational problems involved. For instance, certain areas of the computational theory have not been worked out fully; precise description of biological processes should not be held hostage to computational problems. (This thesis will develop some of the computational theory that was lacking, but certainly not all.) Rather, a precise model should be made, and when computations are to be done, additional assumptions and constraints can be added, which are understood to be computational assumptions and constraints, not biological. Further, as more powerful theoretical and computational techniques do become available, they can be applied to existing models, rather than necessitating a new (and time-consuming) round of modeling. The second use of calculation independent models is that theorists can develop the tools — both computational and mathematical — to deal with all models that fit into this calculation independent framework, rather than ad-hoc methods that apply only to a particular biological system.

The rest of this chapter introduces the notation of chemical reactions and describes some fundamental ideas underlying physical chemistry: kinetics, equilibrium and the connection to thermodynamics.

2.3 Chemical Reactions

Chemical reactions are the lingua franca of biological modeling. They provide a unifying notation in which to express arbitrarily complex chemical processes, either qualitatively or quantitatively. Specifying chemical reactions is so fundamental that the same set of chemical reactions can lead to different computational models, e.g., a detailed differential equations model or a detailed stochastic model. In this sense, representing processes by chemical equations is more basic than using either differential equations or stochastic processes to run calculations to make predictions.

A generic chemical reaction, such as:

$$n_a A + n_b B \xrightarrow{k} n_c C + n_d D$$

states that some molecules of type A react with some of type B to form molecules of types C and D. The terms on the left of the arrow are called the *reactants*; those on the right are called the *products*. There can be an arbitrary number of reactants and an arbitrary number of products, not just always two, and the number of reactants and products do not have to match.

In the reaction given, n_a molecules of A react with n_b molecules of B to give n_c molecules of C and n_d of D. The n terms are called stoichiometric coefficients and are small integers. For example, the reaction

$$A \xrightarrow{k} A'$$

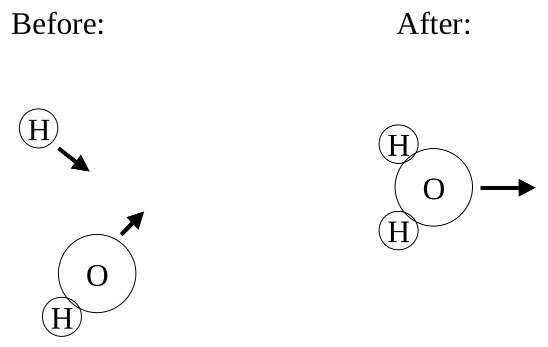


Figure 2-1: Reaction in the previous example.

has one reactant, A, and one product, A'; n_a is 1 and $n_{a'}$ is also 1. In other words, this reaction says that one molecule of type A reacts to give one molecule of type A'. Note that, as in this example, stoichiometric coefficients of 1 are frequently omitted.

Example 1 *A common reaction everyone remembers from high school chemistry is*

$$H^+ + OH^- \xrightarrow{k} H_2O$$

This reaction states that a hydrogen ion from an acid, H^+, reacts with OH^- from an alkali, to form water, H_2O. This reaction occurs, for example, when one mixes vinegar and baking soda.

It is very important to point out that a single chemical reaction can represent either an *elementary* step, i.e., a physically occurring simplest reaction, or the conglomeration of many such elementary steps. In the latter case, the stoichiometric coefficients are the *net* change caused by the reaction. For most of what follows, we shall be interested in elementary reactions, which have the additional property that the total number of molecules on the left side (i.e., the sum of the stoichiometric coefficients of the reactants) is usually 1 or 2. Most reactions with more reactants are not elementary, and can be split into elementary reactions.

Example 2 *The process illustrated in Figure 2-2 is a simple example of transcription factors binding to DNA (as discussed in [1] and in [37]). In this particular example, two different proteins, P and Q, can bind to DNA. There are six types of molecules (technical term:* molecular species*) to keep track of: P, Q, DNA_{free}, $P \bullet DNA$, $Q \bullet DNA$, and*

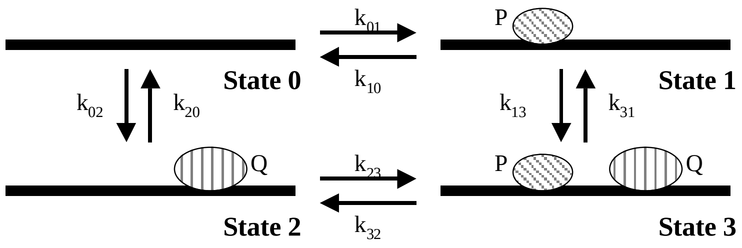


Figure 2-2: Simple model of proteins P and Q binding to and unbinding from DNA. The DNA, a single molecule, is represented as a line.

$P{\bullet}Q{\bullet}DNA$. The first three are self-explanatory. The notation $P{\bullet}DNA$ means "P bound to DNA;" the second half of the molecular species are new molecule types created when P, Q, and DNA_{free} react. In other words, $P \bullet DNA$ is one molecule, of the type shown in State 1 of Figure 2-2. Once a reaction occurs, we no longer care what the molecules were before the reaction, only what they are after it.

$$P + DNA_{free} \xrightarrow{k_{01}} P \bullet DNA$$
$$Q + DNA_{free} \xrightarrow{k_{02}} Q \bullet DNA$$
$$P + Q \bullet DNA_{free} \xrightarrow{k_{23}} P \bullet Q \bullet DNA$$
$$Q + P \bullet DNA_{free} \xrightarrow{k_{13}} P \bullet Q \bullet DNA$$
$$P \bullet DNA \xrightarrow{k_{10}} P + DNA_{free}$$
$$Q \bullet DNA \xrightarrow{k_{20}} Q + DNA_{free}$$
$$P \bullet Q \bullet DNA \xrightarrow{k_{32}} P + Q \bullet DNA_{free}$$
$$P \bullet Q \bullet DNA \xrightarrow{k_{31}} Q + P \bullet DNA_{free}$$

In these examples, the value k on the reaction arrow is a rate constant. Chemical reactions do not occur instantaneously, but rather, take some time to occur. The value k is a way of specifying the amount of time a reaction takes. Suppose a molecule of A collides with a molecule of B; the rate constant measures the probability that this collision will occur with sufficient energy for the molecules to react and give the products. The rate constant depends on temperature — at higher temperatures, collisions will tend to occur more frequently and with greater force, and hence will be more likely to cause reactions, at lower temperatures, the reactants will tend not to collide, or to bounce harmlessly off each other.

Remark 1 *There is a conceptual difference between the rate constant in deterministic (macroscopic) kinetics and the stochastic (mesoscopic) reaction constant. The two constants are related in a straightforward way, but are not necessarily identical. This thesis will concern itself primarily with the latter, but the former are almost always reported, so Section 3.3.1 will discuss converting between the two.*

2.4 The Importance of Physical Chemistry

Chemical equations provide a convenient notation for writing a model of a complicated biological system in terms of elementary interactions or reactions. In and of itself, this is not terribly useful; one really needs *physical chemistry* [4], which tells how these elementary reactions occur. Once one knows that, a model makes very specific predictions about the behavior of systems. We shall discuss three views of physical chemistry: microscopic, mesoscopic, and macroscopic.

2.4.1 The Views

The *Microscopic* View of Physical Chemistry

In principle, one could write out the full molecular dynamics of a system. In other words, one can represent the system as a set of particles, each with a position and a momentum. These particles move around subject to Newton's laws of motion. When particles collide, they may bounce off each other, or, if they collide with sufficient energy, they may react, according to one of the chemical reactions. (Here the rate constants k are related to how much collision energy is required for a collision to result in a reaction.) Newton's laws can be expressed as deterministic differential equations whose variables are positions and momenta. The number of variables is 6 times the number of molecules, 3 each for position and momentum of each individual molecule.

This view is the lowest level of the three and hence the most general. It can incorporate spatial information easily, e.g., local non-uniformities, and can deal with arbitrary reaction geometries. Its main problem is that it contains (and requires!) too much information. For typical systems, there are a nearly infinite number of possible initial conditions, each of which give rise to similar outcomes. For large systems, this becomes intractable; although one might be able to simulate the folding of a short protein on a picosecond time scale, given enough supercomputer time, a system of hundreds of proteins on a timescale of tens of minutes is not possible. Even if it were, many possible realizations of the process differ by insignificant parts — the position of one molecule in a system of hundred, for example — and this level of detail makes it hard to separate important differences from unimportant ones.

The *Mesoscopic* View of Physical Chemistry

In the mesoscopic view of physical chemistry, one counts particles, but does not keep track of their individual positions or momenta. Rather, one assumes the solution is well mixed, at least locally, so that the probability of finding a given molecule anywhere in the volume is equal [19]. Intuitively, this means that one ignores fast, non-reactive collisions, and focuses on the less frequent reactive collisions. The system state is represented by the total number of molecules of species A, the total number of B, etc. When a reaction occurs, the state changes instantaneously and discretely. The reaction constants specify the probability per unit time of a fixed set of reactants reacting — all this will be clarified in the next chapter. The key points are: the state is discrete, changes in the state are discrete and instantaneous, and changes in the state occur probabilistically (in continuous time), according to the laws in the next chapter.

The mesoscopic view cannot take into account spatial information, such as local-non-homogeneities. On the other hand, it is much more tractable, as it uses a single variable per molecule type. In other words, a system of 10 molecules of type A requires a single variable (whose value is 10) rather than 60 variables.

The *Macroscopic* View of Physical Chemistry

The macroscopic view of physical chemistry, typically taught as *the* physical chemistry, deals with systems of many molecules. One approximates the number of molecules as a continuously varying quantity, rather than as discrete numbers. There is another slight difference: one usually talks about concentrations, i.e., number of molecules per unit volume, rather than pure molecule count, but this difference is not the significant one. Again, one assumes the solution is well-mixed, so concentration is a well-defined quantity. Further, this view assumes concentration varies 1) continuously and 2) deterministically — i.e., that the number of molecules is sufficiently large that 1) the discrete changes that occur when individual molecules react cause a small enough change in concentration that said change can be approximated as continuous, and 2) statistical fluctuations in the number of molecules can be neglected [21]. One writes differential equations *whose variables are concentrations* that describe how concentrations change over time.

This view, like the mesoscopic, cannot take into account arbitrary geometries or local non-homogeneities. Additionally, it cannot even account for the stochastic fluctuations caused by the discrete nature of molecules. For systems with many molecules, this approximation is sufficient, and can be used. There is a great computational advantage to this view, as there is only a single variable per molecule type, and those variables obey deterministic, not stochastic, laws.

2.4.2 How the Microscopic, Mesoscopic, and Macroscopic View Relate to Each Other

The *microscopic* view is the most general and exact. It can handle low-level details. However, it is computationally intractable, and contains too much irrelevant information, so one must make additional assumptions. The assumptions made should be noted. Defining precisely and justifying rigorously the conditions — number of molecules, time scale, etc. — under which each of the modeling frameworks is appropriate, remains an open problem.

To use a stochastic model, the *mesoscopic* chemical kinetics, rather than microscopic, one assumes the solution is well-mixed, at least locally, i.e., that a given molecule is equally likely to be anywhere in the solution, or equivalently that the rate of molecular collisions is much greater than the rate of reactions.

Example 3 *The diffusion rate of green fluorescent protein in the bacterium Escherischia coli has been measured to be on the order of 1-10 $\mu m^2/s$ [9]. E. coli has dimensions of order 1 μm, so for gene regulation on the order of seconds or tens of seconds, this assumption is fine.*

Two additional assumptions are required to use the *macroscopic*, differential equations framework: one, that the number of molecules is sufficiently high that discrete changes of a single molecule can be approximated as continuous changes in concentration, and two, that the fluctuations about the mean are small compared to the mean itself.

It is straightforward to check whether the first condition is met. The second often appeals for its justification to the Central Limit Theorem of probability theory [10], which states that the sum of independent, identically distributed (i.i.d.) random variables with finite mean and variance tends to a Gaussian distribution, whose variance grows as the square root of the number of molecules. Typically, one assumes this $1/\sqrt{N}$ noise is sufficiently small for $N \geq 100 - 1000$. (Beware that sums of i.i.d. random variables with heavy tails — infinite mean or variance, e.g., the Cauchy distribution — do not follow this theorem, nor do distributions that are statistically dependent.)

Example 4 *Consider a large number of molecules, each of which can degrade or not degrade independently. At a fixed time, the number remaining is simply the sum of the random variable θ, defined for a single molecule to be 1 if the molecule is present and 0 if it has degraded. This sum meets the criteria of the Central Limit Theorem, so the differential equations approach is probably valid.*

Example 5 *Consider the process of transcriptional elongation, i.e., when DNA is transcribed into mRNA nucleotide by nucleotide. One may model each nucleotide step as taking an exponentially-distributed amount of time, so the total time to move several steps down the DNA is a random variable that meets the criteria of the Central Limit Theorem.*

For more complex processes, it is not so simple to apply the Central Limit Theorem; doing so may not even be legitimate. Under what precise conditions one can and cannot remains an open problem.

Two standard approaches are used to justify using differential equations rather than stochastic models:

- Assert “it is appropriate to use differential equations in this case.”
- Run a subset of the analysis using a stochastic simulator and show that it does not differ qualitatively from the differential equations approach.

Ideally, one would have some better way of justifying when differential equations are and are not valid, or at least a way to point out models where one should be particularly careful so as not to generate incorrect predictions.

2.5 Physical Chemistry - the Concept of State

The state of a system is a snapshot of the system at a given time that contains enough information to predict the behavior of the system for all future times. Intuitively, the state of the system is the set of variables that must be kept track of in a model. Different models of biological systems have different representations of the state:

- In microscopic models of gene regulation, the state is a list of the positions and momenta of each molecule.
- In mesoscopic models, the state is a list of the actual number of molecules of each type.
- In macroscopic models, the state is a list of the concentrations of each chemical type.

Physical chemistry deals with two problems: changes of state (kinetics) and which states are steady states (equilibria). Amazingly, the latter question can be answered by thermodynamics, just by knowing the energy differences of the different possible states of the system, without knowing the initial state or anything about the kinetics. We shall consider in turn kinetics, equilibria, and thermodynamics, as they apply to biological models. Note that these concepts apply to all three views; however, this thesis is primarily interested in the mesoscopic, and will tend to give examples in that framework.

2.6 Kinetics - Changes of State

The chemical equation

$$A + B \xrightarrow{k} C$$

deals with three chemical species, A, B, and C. According to the equation, A and B are transformed to C at a rate of k. In other words, the chemical equation specifies how the state of the system changes, and how fast that change occurs. (See Example 2-1.)

In the microscopic approach, this equation specifies that when a molecule of A and a molecule of B collide with sufficient force (loosely specified by k), they react to form a new kind of molecule, C.

In the mesoscopic approach, this equation specifies that the state changes in a different way: one single molecule of A and one of B are converted into a molecule of C; the total number of molecules of A decreases by one, as does the total number of molecules of B, and the total number of molecules of C increases by one. The probability that this reaction occurs in a small time dt is given by $k \times (\#A) \times (\#B) \times dt$, where $(\#A)$ is the number of molecules of A present, etc.

In the macroscopic approach, this equation specifies that state changes in the following way: the concentration of A decreases, the concentration of B decreases by the same amount, and the concentration of C increases by the same amount. For small times dt, the amount of the change is given by $k \times [A] \times [B] \times dt$, where $[A]$ is the concentration of A, etc.

2.7 Equilibrium

A system is said to be at equilibrium when its state ceases to change. Consider the example:

$$A \xrightarrow{k} A'$$

As long as there is available A, the state will change at a rate determined by k. Thus, when this system reaches equilibrium, all of the A will be gone, and only A' will remain. At the equilibrium point, once all the A has been used up, there can be no more changes of state.

It is not in general true that equilibrium only occurs when one of the components has been used up. Consider, for example, a certain DNA binding protein X, whose binding to and unbinding from DNA follow simple and complementary chemical kinetics:

$$X + DNA \xrightarrow{k_1} X \bullet DNA \tag{2.1}$$

$$X \bullet DNA \xrightarrow{k_{-1}} X + DNA \tag{2.2}$$

This set of reactions involves chemicals of three types: X, DNA and $X \bullet DNA$. Equilibrium will occur when there is no more *net* change.

Example 6 *One way that could happen, in the macroscopic view, is if the rate at which X and DNA are converted to $X \bullet DNA$ (according to Eq 2.1) exactly equals the rate at which*

$X \bullet DNA$ is converted to X and DNA (according to Eq 2.2). Note that in this equilibrium, there are still changes: both reactions still occur constantly. However, there are no net changes. And the state of the system (i.e., the list of concentrations of all chemical types involved) does not change.

In the mesoscopic and microscopic views, equilibrium is not so simple; it is a statistical property.

Example 7 *(Equilibrium in the Stochastic Framework) One computational method (described in the next chapter) for the mesoscopic framework is to consider the* probability *that the system is in a particular state. From the equations above, a single DNA molecule can be in two states, bound or unbound, i.e., $X \bullet DNA$ or DNA. As will be shown in the next chapter, one can write a differential equation (called a master equation) for the probability of the molecule being in each state, given the initial state s_0.*

Assuming that there are x molecules of X, that equation is:

$$\frac{d}{dt}\begin{bmatrix} P(DNA, t|s_0, t_0) \\ P(X \bullet DNA, t|s_0, t_0) \end{bmatrix} = \begin{bmatrix} -x \times k_1 & k_{-1} \\ x \times k_1 & -k_{-1} \end{bmatrix} \times \begin{bmatrix} P(DNA, t|s_0, t_0) \\ P(X \bullet DNA, t|s_0, t_0) \end{bmatrix} \tag{2.3}$$

The off-diagonal terms correspond to the two reactions above, the diagonal terms correspond to no reaction occurring, and are negative to ensure conservation of probability. (How to generate this equation will be explained in the next chapter, in Section 3.4.)

Using Eq 2.3 and the initial state, one could solve for both probabilities as a function of time. Doing so would be the kinetics *approach to the problem. Instead, assume the two competing processes have reached equilibrium, i.e., there are no further net changes. Thus,*

$$\frac{d}{dt}\begin{bmatrix} P(DNA, t|s_0, t_0) \\ P(X \bullet DNA, t|s_0, t_0) \end{bmatrix} = 0 \tag{2.4}$$

This leads to the following equation for the equilibrium probability distribution:

$$
\begin{aligned}
\begin{bmatrix} P(DNA, t|s_0, t_0) \\ P(X \bullet DNA, t|s_0, t_0) \end{bmatrix}_{eq} &= \frac{1}{k_{-1} + xk_1} \begin{bmatrix} k_{-1} \\ xk_1 \end{bmatrix} \\
&= \frac{1/k_{-1}}{k_{-1}/k_{-1} + xk_1/k_{-1}} \begin{bmatrix} k_{-1} \\ xk_1 \end{bmatrix} \\
&= \frac{1}{1 + xK_{eq}} \begin{bmatrix} 1 \\ xK_{eq} \end{bmatrix}
\end{aligned}
\tag{2.5}
$$

Here K_{eq} is called the equilibrium constant of these reactions. Notice that applying the equilibrium condition in (2.4) reduced a differential equation to an algebraic equation, and that the equilibrium probabilities depend only on K_{eq}, i.e., on the ratio of reaction constants, not on the constants themselves. Applying the equilibrium condition in an arbitrary framework removes the time-dependence of the kinetic equations, and the resulting equations are algebraic. In addition to the rate constants of the previous subsection, chemical reactions may have an equilibrium constant, which is a property only of the system, not the computational framework of the system.

Remark 2 *In the microscopic framework, Newton's laws can be represented as differential equations whose variables are positions and momenta. In the mesoscopic, for systems with a small number of possible states, one can write a master equation, another differential equation, whose variables are probabilities. In the macroscopic, one writes differential equations whose variables are concentrations. One should not get confused by these different kinds of differential equations. The frameworks and the variables are very different, even if some of the math overlaps.*

Real physical systems tend toward equilibrium unless energy is continually added. So another interpretation of equilibrium is the state of the system as time$\rightarrow \infty$, in the absence of energy inputs. To simplify models (and experiments), reactions that are fast (compared to the main reactions of interest) are often assumed to be at equilibrium. Given this assumption, for example, the pair of equations 2.1 and 2.2 can be abbreviated as

$$X + DNA \overset{K_{eq}}{\rightleftharpoons} X \bullet DNA \tag{2.6}$$

The previous example considered a single DNA molecule. More generally, consider the reaction above with some arbitrary number of DNA molecules. Then the state can be specified as $S = (\#X,\ \#DNA,\ \#X \bullet DNA)$. The reaction above specifies that this can change state to $S = (\#X - 1,\ \#DNA - 1,\ \#X \bullet DNA + 1)$. From the material to be developed in the next chapter, the probability of the forward reaction occurring in a little bit of time dt is $k_1 \times (\#X) \times (\#DNA) \times dt$ and the reverse (ending in the same state) is $k_{-1} \times (\#X \bullet DNA + 1) \times dt$. The property of *detailed balance* says that at equilibrium, these two rates should be equal. (Detailed balance is a special condition that comes from the physical chemistry of the situation, not just from the math.) That means that at equilibrium:

$$\begin{aligned} & P_{(\#X,\#DNA,\#X\bullet DNA)} \times k_1 \times (\#X) \times (\#DNA) \\ = \ & P_{(\#X-1,\#DNA-1,\#X\bullet DNA+1)} \times k_{-1} \times (\#X \bullet DNA + 1) \end{aligned} \tag{2.7}$$

or

$$\frac{P_{(\#X-1,\#DNA-1,\#X\bullet DNA+1)} \times (\#X \bullet DNA + 1)}{P_{(\#X,\#DNA,\#X\bullet DNA)} \times (\#X) \times (\#DNA)} = \frac{k_1}{k_{-1}} = K_{eq} \tag{2.8}$$

The above example was the special case $(\#X,\ \#DNA,\ \#X \bullet DNA) = (x, 1, 0)$.

Definition 1 *(Detailed Balance) The special condition that, at equilibrium, in addition to the net change in probability balancing to 0, each reaction individually balances to 0.*

It is possible to go directly from the chemical equation 2.6 to the algebraic equation 2.8 by multiplying all the concentrations of the products (in this case $\#X \bullet DNA + 1$) by the equilibrium probability of that state, and dividing by the product of the concentrations of the reactants (in this case $(\#X) \times (\#DNA)$) times the equilibrium probability of that state.

Now consider Eq 2.6 and the additional equilibrium equation:

$$X \bullet DNA \overset{K_{eq2}}{\rightleftharpoons} X' \bullet DNA$$

For example, this could mean that X undergoes a conformational change to X' while bound to DNA. For simplicity, assume (as in the previous example) a single DNA molecule, which may be bound or unbound. Then the state can be specified ($\#X$, $\#DNA$, $\#X \bullet DNA$, $\#X' \bullet DNA$), and the equilibrium equations become:

$$\frac{P_{(\#X-1,0,1,0)} \times (1)}{P_{(\#X,1,0)} \times (\#X) \times (1)} = K_{eq}$$

and

$$\frac{P_{(\#X-1,0,0,1)} \times (1)}{P_{(\#X-1,0,1,0)} \times (1)} = K_{eq2}$$

This leads to

$$\frac{P_{(\#X-1,0,0,1)} \times (1)}{P_{(\#X,1,0)} \times (\#X) \times (1)} = \frac{P_{(\#X-1,0,1,0)} \times (1)}{P_{(\#X,1,0)} \times (\#X) \times (1)} \times \frac{P_{(\#X-1,0,0,1)} \times (1)}{P_{(\#X-1,0,1,0)} \times (1)} = K_{eq} \times K_{eq2} \tag{2.9}$$

This is an important property of equilibria: if A and B are in equilibrium, and B and C are in equilibrium, then A and C are in equilibrium, and the resulting A-C equilibrium constant is simply the product of the A-B and B-C equilibrium constants.

The key points to remember about equilibrium:

- Equilibrium ignores the dynamics of state change, and only considers a static state.
- Equilibrium depends only on equilibrium constants, i.e., a ratio of reaction constants, not the reaction constants themselves.
- Multiple equilibria can be treated by multiplying equilibrium constants, as in Eq 2.9.

2.8 Thermodynamics

Two ways of determining equilibrium constants have been presented thus far: calculating equilibrium constants from the forward and reverse rate constants, and calculating them from the equilibrium concentrations of the chemicals. There is a third way — equilibrium constants can be calculated from thermodynamics [11], using just the energy difference between the products and reactants. From this energy difference alone, one can predict the final state of the system, but not the time-course of the state from initial to final.

For chemical reactions, the Gibbs free energy, G, is defined by

$$\begin{aligned} G &= \text{(Total internal energy)} - \text{(absolute temperature)} \times \text{(entropy)} + \text{(pressure)} \times \text{(volume)} \\ &= \sum \text{(chemical potential)} \times \text{(particle number)} \end{aligned}$$

Typically, values of ΔG are reported instead of the values of K_{eq}. From thermodynamics in dilute solutions [4,11] it follows that, for reactants and products with free energy difference ΔG,

$$K_{eq} = e^{-\Delta G/RT}$$

where T is the absolute temperature, and R is the ideal gas constant — namely Boltzmann's constant time Avogadro's number.

To deal with multiple states in equilibrium with each other, one uses partition functions. In general, the fraction of the system in a certain configuration c (e.g., the fraction of the DNA bound to protein P) is given by:

$$frac_c = \frac{\exp(-\Delta G_c/RT) \times [Species_1]^{power_1} \times \cdots \times [Species_n]^{power_n}}{\sum_i \exp(-\Delta G_i/RT) \times [Species_1]^{power_1} \times \cdots \times [Species_n]^{power_n}}$$

The power of chemical species x in configuration c is simply the number of molecules of type x present in configuration c. The denominator of this equation is called the *partition function.*

Example 8 *(Partition Functions): Consider the example in Figure 2-2, but this time, assume that equilibrium has been reached. Then*

$$\begin{aligned} P_{eq}(0) &= \frac{1}{Z} \\ P_{eq}(1) &= \frac{K_1[P]}{Z} \\ P_{eq}(2) &= \frac{K_2[Q]}{Z} \\ P_{eq}(3) &= \frac{K_3[P][Q]}{Z} \end{aligned}$$

where each of the K's is defined to be the equilibrium constant between a given state and

State 0, and

$$Z = 1 + K_1[P] + K_2[Q] + K_3[P][Q]$$

Here Z is the partition function.

Proof. (of the previous example) Let P_0 be the concentration of DNA in State 0, and let [P] and [Q] be the concentration of free (i.e., unbound) protein P and Q, respectively. Then, there are equilibrium constants K_1, K_2, and K_3, such that at equilibrium

$$\begin{aligned} \frac{P_1}{[P] \times P_0} &= K_1 \\ \frac{P_2}{[Q] \times P_0} &= K_2 \\ \frac{P_3}{[P] \times [Q] \times P_0} &= K_3 \end{aligned}$$

The total probability must sum to 1, so

$$\begin{aligned} 1 &= P_{total} \\ &= P_0 + P_1 + P_2 + P_3 \\ &= P_0 + (K_1 \times [P] \times P_0) + (K_2 \times [Q] \times P_0) + (K_3 \times [P] \times [Q] \times P_0) \\ &= P_0 \times Z \end{aligned}$$

The result of the previous example follows directly. ■

2.9 Key Concepts

- A detailed biological system can be described by many simple chemical equations.
- Chemical equations specify what chemical reactions occur, and how frequently they occur.
- There are three interpretations of chemical equations:
  - microscopic interpretation: positions and momenta. Chemical equations specify discrete changes in the particles present.

  - mesoscopic interpretation: count the number of molecules of each time. Discrete changes occur probabilistically in continuous time.
  - macroscopic interpretation: approximate the number of molecules as continuous. The number changes continuously and deterministically.

- This thesis will focus on mesoscopic, as that view allows us to deal with systems of tens or hundreds of molecules.

- In the mesoscopic view, the state is specified by the number of molecules of each type.

- Kinetics: the mesoscopic state changes discretely at random times. These times are chosen from continuous distributions. Simple laws for those distributions are covered in the next chapter.

- Equilibrium = steady state. State of the system as $t \to \infty$. In the mesoscopic view, one specifies an equilibrium probability distribution.

- The equilibrium condition simplifies a set of differential equations to a set of algebraic equations and is a useful approximation for reactions that are fast compared to the ones of interest.

- The equilibrium constant is a ratio of rate constants and helps specify the equilibrium probability distribution.

- Multiple equilibria can be treated separately. This property is called detailed balance.

- Equilibrium constants tie in to thermodynamics and can be determined from the energy differences between states.

- Partition functions provide a useful tool for dealing with multiple equilibria.

Chapter 3 The Stochastic Framework

3.1 Summary

This chapter provides a more detailed explanation of the mesoscopic view of chemistry. It shows how to make specific predictions about a system described by a set of chemical reactions. There are two main ways to make exact predictions: 1) a state-space approach, which leads to a *master equation*, a system of differential equations whose variables are the probabilities of all the possible states, and 2) numerical methods for generating realizations of the system according to the correct probability distributions.

3.2 Probability Theory Background

As the mesoscopic framework involves stochastic processes (random processes that depend on time), a few quick words about probability and stochastic process theory are in order. Several introductory books discuss probability theory, for example [26] and [10], and several more advanced books discuss stochastic processes, for example [18] and [38]. A couple stray facts are useful in dealing with stochastic processes that do not commonly appear in elementary books or courses. Also, there are different notations, so here are the ones used in this thesis:

- Random variables will be denoted by capital letters, e.g., X is a random variable.
- The expected value of a random variable X will be denoted $E[X]$.
- Samples from random variables will be denoted by lowercase letters, e.g., x is a sample from the random variable X. (One can imagine a bucket with X painted on the side. When one asks for the value of X, one takes a sample x out of the bucket.) Samples are simply numbers.
- A random variable X is defined by its probability density function, P_X. For continuous random variables, $P_X(x)dx =$ the probability that a sample of X will fall in

$(x, x + dx)$. Note that $P_X(x)$ is not a probability, it has units of 1/length, while probability is unitless.

- For stochastic processes, it is frequently useful to state the initial conditions explicitly. So for example, if the initial state of the system at time t_0 is denoted x_0, it is preferable to deal with the conditioned random variable $P(x, t|x_0, t_0)$ rather than the unconditioned $P(x, t)$. Taking care to explicitly state initial conditions avoids certain paradoxes, for example [20].

- A common operation in probability theory is constructing a random variable Y from a random variable X by some function $Y = f(X)$. (In the bucket analogy, this amounts to having a bucket X and a bucket Y. One takes samples x from bucket X, transforms them according to $y = f(x)$, and drops the resulting number into Y.) A common question is: given P_X and f, what is P_Y? For single variable distributions, it can be shown [17, 18] that

$$P_Y(y) = \int_{-\infty}^{\infty} P_X(x)\delta\left[y - f(x)\right] dx$$

This result is called the Random Variable Transformation Theorem, and will be frequently referred to as the RVT theorem.

3.3 The Basics of the Stochastic Framework

Consider the following system of 4 chemical species, in which 3 reactions can occur:

$$\begin{aligned} A + B &\xrightarrow{k_1} C \\ 2C &\xrightarrow{k_2} A + D \\ D &\xrightarrow{k_3} B \end{aligned}$$

In the mesoscopic framework, the state of this system can be described by the list $(\#A, \#B, \#C, \#D)$: the number of molecules of A, the number of B, etc. Discrete changes of state occur whenever a reaction is executed.

Example 9 *Suppose the state is* $(10, 8, 15, 3)$. *Then Reaction 1 would cause the state to change to* $(9, 7, 16, 3)$*, Reaction 2 would cause the (original) state to change to* $(11, 8, 13, 4)$*,* *and Reaction 3 would cause the state to change to* $(10, 9, 15, 2)$. *If no reaction occurs, the state does not change.*

In the mesoscopic view, the probability that a given molecule of A will react with a given molecule of B in the next infinitesimal time dt is given by $k_1 dt$. (For a physical derivation of this, see [19].) In other words, that *probability* that a given pair of molecules react in the next infinitesimal time dt is well defined and is proportional to dt. The proportionality constant — in this case k_1 — is independent of dt. Typically, it is a real-valued constant, although it may instead be a function of time (but not of dt!), or of some other physical quantity.

Remark 3 *The macroscopic view assumes that a reaction rate per unit time is well defined, but for small dt, one cannot have fractional reactions, so the mesoscopic view is more rigorous.*

The probability that *a particular pair* of molecules of A and B will react in the next infinitesimal time dt is given by $k_1 dt$. The probability that *some pair* of molecules of A and B will react in the next infinitesimal time dt is given by $k_1 \times \#A \times \#B \times dt$, since there are $\#A \times \#B$ possible pairs that can react. (Here $\#A$ is the number of molecules of A, etc.) The probability of more than one reaction occurring in the same infinitesimal dt is proportional to $(dt)^2$ and may be ignored.

It is occasionally useful to refer to some of these quantities by name. The combination $\#A \times \#B$ will be called the *redundancy function* of Reaction 1, abbreviated h_1. The combination $k_1 \times \#A \times \#B$ will be called the *propensity function* of Reaction 1, abbreviated a_1.

Remark 4 *The key assumptions that allow the transition from the microscopic view to the mesoscopic view are that the solution is well-mixed and in thermal (although not necessarily chemical) equilibrium. Without those assumptions, it would still be possible to write expressions for the probability of given pairs reacting, but those expressions would vary rapidly with time and would almost certainly not be the same for different particular pairs of reactants. The assumption provides an enormous simplification to the framework.*

Reactions with more than one copy of the same reactant are a little tricky. In addition to the state changing by ± 2 instead of ± 1, the redundancy function h is a little different.

Recall that the redundancy function counts the number of distinct reactant pairs that can undergo a reaction. In Reaction 1 above, that number is simply $\#A \times \#B$. In Reaction 2, two copies of C react with each other. The total number of pairs of C is $\binom{\#C}{2} = \#C \times (\#C - 1)/2$, not $(\#C)^2$. One should be careful with this kind of reaction.

3.3.1 The Relationship Between Mesoscopic Reaction Constants and Macroscopic Rate Constants

Typically, macroscopic rate constants are reported, not mesoscopic reaction constants, so it is useful to know how to convert between the two. Dimensional analysis can be used to calculate the dimensions of mesoscopic reaction constants.

$$\text{Probability} = a \times dt = k \times h \times dt$$

Probability is dimensionless, dt has dimensions of time, and h is just a dimensionless integer. Therefore, k has dimensions of $(\text{time})^{-1}$.

First order reactions (reactions with a single reactant) have *macroscopic* rate constants with dimensions $(\text{time})^{-1}$. Second order *macroscopic* rate constants have dimensions of $(\text{moles/liter})^{-1}(\text{time})^{-1}$. Thus, up to a dimensionless factor, first order reactions have the same macroscopic and mesoscopic constants, whereas second order constants follow $k_{macroscopic} = k_{mesoscopic} \times A \times V \times C$, where A is Avogadro's number, V is the volume, and C is a dimensionless constant. In particular, $C = 1$ for reactions involving two different reactants, and $C = 1/2$ for reactions involving two copies of the same molecule type. (For a more detailed discussion see [16] or [29].) The following table illustrates the redundancy function, h, the propensity function, a, and the macroscopic rate constants that correspond to the mesoscopic rate constants k_1, k_2, and k_3:

Reaction	h	a	Macroscopic rate constant
1	$\#A \times \#B$	$k_1 \times \#A \times \#B$	$k_1 \times A \times V$
2	$\binom{\#C}{2} = \frac{\#C \times (\#C-1)}{2}$	$\frac{k_2 \times \#C \times (\#C-1)}{2}$	$k_2/2 \times A \times V$
3	$\#D$	$k_3 \times \#D$	k_3

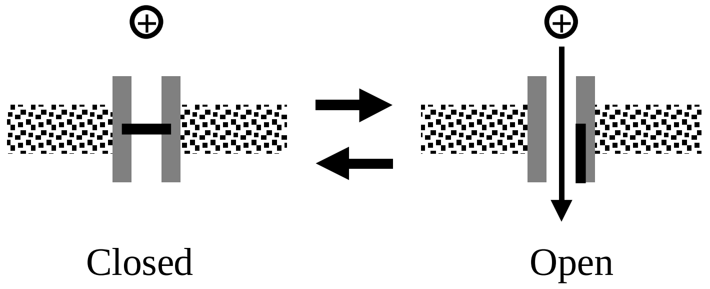


Figure 3-1: A simple, two-state ion channel. The channel is a protein that sits in the cell membrane. When it is closed, ions may not pass through. When it is open, ions may pass through. This figure shows a positive ion — or cation — but certain channels allow negative ions — anions.

3.4 The Chemical Master Equation

Calculations in the mesoscopic framework use three techniques: the Chemical Master Equation, exact stochastic simulation, or approximation. This thesis will only consider exact methods, i.e., the first two. Exact stochastic simulation algorithms, presented in the next section, will turn out to be the method of choice, but we first present the Chemical Master Equation approach [18, 30, 38], as it is useful in justifying certain theoretical results.

3.4.1 The Basic Idea Behind the Chemical Master Equation

The basic approach can be summarized as follows:

- For each possible state S of the system, let $P(S, t|S_0, t_0)$ be the probability that the state is S at time t, given that the state is S_0 at time t_0.
- Use the rules of the previous section to specify how $P(S, t|S_0, t_0)$ varies as a function of t.
- This will lead to a system of linear differential equations with constant coefficients, called a Chemical Master Equation. Solve this system.

The easiest way to illustrate this approach is through a simple example.

Example 10 *(Defining States) Consider a very simple model of an ion channel as in Figure 3-1. An ion channel is a protein that can conduct electricity in the form of ions. At the simplest level, the channel may be open (conducting) or closed (non-conducting). Suppose* *this particularly simple ion channel has only those two states, and call the states O and C (real ion channels may have multiple open and multiple closed states). Suppose the chemical equations governing transitions between the states are:*

$$C \xrightarrow{\alpha} O$$
$$O \xrightarrow{\beta} C$$

(The non-biologically inclined may take these equations as given, and ignore the motivating example.) Consider a single channel, i.e., a single molecule. Thus, the system state is equal to the state of that one molecule, which is either O or C.

Example 11 *(Specifying transitions) From the previous section, the probability of a transition from state C to state O in the infinitesimal time step dt is given by*

$$P(O, t + dt|C, t) = \alpha \times dt$$

and, analogously,

$$P(C, t + dt|O, t) = \beta \times dt$$

Because probabilities must sum to 1, we have

$$\begin{aligned} P(C, t + dt|C, t) &= 1 - \alpha \times dt \\ P(O, t + dt|O, t) &= 1 - \beta \times dt \end{aligned}$$

The functions of interest are $P(C, t|S_0, t_0)$ and $P(O, t|S_0, t_0)$, i.e., the probabilities conditioned on the initial state (which may be either O or C) and on the initial time, not the current time. By the laws of probability:

$$\begin{aligned} P(C, t + dt|S_0, t_0) &= P(C, t + dt|C, t)P(C, t|S_0, t_0) + P(C, t + dt|O, t)P(O, t|S_0, t_0) \\ P(O, t + dt|S_0, t_0) &= P(O, t + dt|C, t)P(C, t|S_0, t_0) + P(O, t + dt|O, t)P(O, t|S_0, t_0) \end{aligned}$$

Writing this in matrix form:

$$\begin{aligned}
\overrightarrow{P}(t+dt|S_0,t_0) &= \begin{bmatrix} P(C,t+dt|S_0,t_0) \\ P(O,t+dt|S_0,t_0) \end{bmatrix} \\
&= \begin{bmatrix} 1-\alpha\times dt & \beta\times dt \\ \alpha\times dt & 1-\beta\times dt \end{bmatrix} \begin{bmatrix} P(C,t|S_0,t_0) \\ P(O,t|S_0,t_0) \end{bmatrix} \\
&= \begin{bmatrix} 1-\alpha\times dt & \beta\times dt \\ \alpha\times dt & 1-\beta\times dt \end{bmatrix} \overrightarrow{P}(t|S_0,t_0) \\
&= \left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + dt \begin{bmatrix} -\alpha & \beta \\ \alpha & -\beta \end{bmatrix} \right) \overrightarrow{P}(t|S_0,t_0) \\
&= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \overrightarrow{P}(t|S_0,t_0) + dt \begin{bmatrix} -\alpha & \beta \\ \alpha & -\beta \end{bmatrix} \overrightarrow{P}(t|S_0,t_0) \\
&= \overrightarrow{P}(t|S_0,t_0) + dt \begin{bmatrix} -\alpha & \beta \\ \alpha & -\beta \end{bmatrix} \overrightarrow{P}(t|S_0,t_0)
\end{aligned}$$

Rearranging,

$$\frac{\overrightarrow{P}(t+dt|S_0,t_0) - \overrightarrow{P}(t|S_0,t_0)}{dt} = \begin{bmatrix} -\alpha & \beta \\ \alpha & -\beta \end{bmatrix} \overrightarrow{P}(t|S_0,t_0)$$

In the limit as $dt \to 0$, the left-hand side becomes a derivative, which leads to a Chemical Master Equation:

$$\frac{d\overrightarrow{P}(t|S_0,t_0)}{dt} = \begin{bmatrix} -\alpha & \beta \\ \alpha & -\beta \end{bmatrix} \overrightarrow{P}(t|S_0,t_0)$$

This is sometimes written as

$$\frac{d\overrightarrow{P}(t|S_0,t_0)}{dt} = \mathcal{W}\overrightarrow{P}(t|S_0,t_0) \tag{3.1}$$

where $\mathcal{W}$ is a special matrix, sometimes called a $\mathcal{W}$ -matrix [38] or a consolidated characterizing function matrix [18].

In some sense, writing Chemical Master Equation is enough, as solving this sort of differential equation subject to its initial conditions is a well-studied problem that can be found in elementary differential equations or linear algebra texts. However, for good measure, here is how to solve it.

Example 12 *Look at the eigenvalues and eigenvectors of $\mathcal{W}$. It has an eigenvalue of 0, corresponding to the (un-normalized) eigenvector $[\beta, \alpha]^\top$ and an eigenvalue of $-(\alpha+\beta)$, corresponding to the (un-normalized) eigenvector $[1, -1]^\top$. Thus,*

$$\overrightarrow{P}(t) = c_1 \begin{bmatrix} \beta \\ \alpha \end{bmatrix} e^{0(t-t_0)} + c_2 \begin{bmatrix} 1 \\ -1 \end{bmatrix} e^{-(\alpha+\beta)(t-t_0)}$$

where c_1 and c_2 are constants that depend only on the initial probability distribution. In particular, the formula must hold at $t = t_0$, so

$$\begin{aligned} \overrightarrow{P}_0 &= \overrightarrow{P}(t_0) \\ &= c_1 \begin{bmatrix} \beta \\ \alpha \end{bmatrix} + c_2 \begin{bmatrix} 1 \\ -1 \end{bmatrix} \\ &= \begin{bmatrix} \beta & 1 \\ \alpha & -1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} \end{aligned}$$

Hence

$$\begin{aligned} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} &= \begin{bmatrix} \beta & 1 \\ \alpha & -1 \end{bmatrix}^{-1} \overrightarrow{P}_0 \\ &= \frac{1}{\alpha+\beta} \begin{bmatrix} 1 & 1 \\ \alpha & -\beta \end{bmatrix} \overrightarrow{P}_0 \end{aligned}$$

So, for example, if $P(C, t_0) = 1$ (and hence $P(O, t_0) = 0$), then $c_1 = \frac{1}{\alpha+\beta}$ and $c_2 = \frac{\alpha}{\alpha+\beta}$. Thus

$$\overrightarrow{P}(t|C, t_0) = \frac{1}{\alpha+\beta} \begin{bmatrix} \beta \\ \alpha \end{bmatrix} + \frac{\alpha}{\alpha+\beta} \begin{bmatrix} 1 \\ -1 \end{bmatrix} e^{-(\alpha+\beta)(t-t_0)}$$

Remark 5 *Another way to do the same problem is to recall that the solution to $\frac{dP}{dt} = \mathcal{W}P$*

is given by $P(t) = \exp(\mathcal{W}t)P_0$, which holds for vectors equations as well as one-variable equations. One can break $\mathcal{W}$ into an eigenvector matrix, a diagonal eigenvalue matrix, and the inverse eigenvector matrix as follows:

$$\begin{aligned}
\mathcal{W} &= \begin{bmatrix} -\alpha & \beta \\ \alpha & -\beta \end{bmatrix} \\
&= \begin{bmatrix} \frac{\beta}{\alpha^2+\beta^2} & \frac{1}{\sqrt{2}} \\ \frac{\alpha}{\alpha^2+\beta^2} & \frac{-1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & -(\alpha+\beta) \end{bmatrix} \begin{bmatrix} \frac{\alpha^2+\beta^2}{\alpha+\beta} & \frac{\alpha^2+\beta^2}{\alpha+\beta} \\ \frac{\sqrt{2}\alpha}{\alpha+\beta} & \frac{-\sqrt{2}\beta}{\alpha+\beta} \end{bmatrix} \\
&= E\Lambda E^{-1}
\end{aligned}$$

Simplifying things a bit by using $u = t - t_0$, and using the previous result

$$\begin{aligned}
\exp(\mathcal{W}u) &= I + \mathcal{W}u + \frac{1}{2}(\mathcal{W}u)^2 + \frac{1}{3!}(\mathcal{W}u)^3 + \dots \\
&= I + E\Lambda E^{-1}u + \frac{1}{2}(E\Lambda E^{-1}u)^2 + \frac{1}{3!}(E\Lambda E^{-1}u)^3 + \dots \\
&= E\left(I + \Lambda u + \frac{1}{2}(\Lambda u)^2 + \frac{1}{3!}(\Lambda u)^3 + \dots\right)E^{-1} \\
&= \begin{bmatrix} \frac{\beta}{\alpha^2+\beta^2} & \frac{1}{\sqrt{2}} \\ \frac{\alpha}{\alpha^2+\beta^2} & \frac{-1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} \exp(0u) & 0 \\ 0 & \exp(-(\alpha+\beta)u) \end{bmatrix} \begin{bmatrix} \frac{\alpha^2+\beta^2}{\alpha+\beta} & \frac{\alpha^2+\beta^2}{\alpha+\beta} \\ \frac{\sqrt{2}\alpha}{\alpha+\beta} & \frac{-\sqrt{2}\beta}{\alpha+\beta} \end{bmatrix}
\end{aligned}$$

And hence

$$\overrightarrow{P}(t) = \begin{bmatrix} \frac{\beta}{\alpha^2+\beta^2} & \frac{1}{\sqrt{2}} \\ \frac{\alpha}{\alpha^2+\beta^2} & \frac{-1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & \exp(-(\alpha+\beta)(t-t_0)) \end{bmatrix} \begin{bmatrix} \frac{\alpha^2+\beta^2}{\alpha+\beta} & \frac{\alpha^2+\beta^2}{\alpha+\beta} \\ \frac{\sqrt{2}\alpha}{\alpha+\beta} & \frac{-\sqrt{2}\beta}{\alpha+\beta} \end{bmatrix} \overrightarrow{P}(t_0)$$

3.4.2 The General Case

The general case is completely analogous to the example. The goal is to get a Chemical Master Equation of the form

$$\frac{d\overrightarrow{P}(t|S_0,t_0)}{dt} = \mathcal{W}\overrightarrow{P}(t|S_0,t_0) \tag{3.2}$$

Here $\overrightarrow{P}$ is a vector containing the probabilities of all possible states, conditioned on the initial conditions: $\left[\overrightarrow{P}(t|S_0,t_0)\right]_1 = \overrightarrow{P}(S=1,t|S_0,t_0)$, etc. In the previous example, there were two possible states, so $\overrightarrow{P}$ was a 2-vector.

Once the meaning of $\overrightarrow{P}$ is defined, all that remains is $\mathcal{W}$. In particular,

$$\mathcal{W}_{ij} = \begin{cases} \text{the propensity, } a_{ij}\text{, of going from state } j \text{ to state } i & \text{if } i \neq j \\ -\sum_{k\neq i} \mathcal{W}_{kj} & \text{if } i = j \end{cases}$$

The first line of this definition relates the propensities discussed in the previous chapter to the $\mathcal{W}$-matrix. The second line is necessary to conserve probability; this corresponds to the condition that probability must sum to 1, which implies that $\frac{d}{dt}(\sum \overrightarrow{P})$ must be 0.

Example 13 *In Example 11, $\overrightarrow{P}$ is the two vector $\begin{bmatrix} \overrightarrow{P}(C,t|S_0,t_0) \\ \overrightarrow{P}(O,t|S_0,t_0) \end{bmatrix}$. $\mathcal{W}_{CO}$ = (the propensity of going from state O to state C)= β, and $\mathcal{W}_{OC} = \alpha$. Finally, $\mathcal{W}_{CC} = -\sum_{k\neq C} \mathcal{W}_{kC} = -\mathcal{W}_{OC} = -\alpha$, and analogously, $\mathcal{W}_{OO} = -\beta$.*

More examples of writing chemical master equations follow. Solving such equations is a standard matter in differential equations and in linear algebra, so we shall be content to write the equations. (Also, as we shall see later, the number of states is typically so large that one cannot solve anyway.)

Example 14 *Consider a more complicated example, defined by the chemical equation*

$$X + DNA \xrightarrow{k_1} X \bullet DNA$$

Here the state consists of every possible combination of number of X molecules, number of DNA molecules, and number of $X \bullet DNA$ molecules. Assuming some total number of molecules, the number of states will be finite (although possibly large). Suppose j is the index of the state $(\#X, \#DNA, \#X \bullet DNA) = (a, b, c)$, for some fixed $a, b,$and c. Then let i be the index of the state $(\#X, \#DNA, \#X \bullet DNA) = (a-1, b-1, c+1)$. Then

$$\mathcal{W}_{kj} = \begin{cases} k_1 \times a \times b & \textit{if } k = i \\ -k_1 \times a \times b & \textit{if } k = j \\ 0 & \textit{otherwise} \end{cases}$$

Example 15 *Consider both the forward and reverse reactions:*

$$X + DNA \xrightarrow{k_1} X \bullet DNA$$
$$X \bullet DNA \xrightarrow{k_{-1}} X + DNA$$

The state definition remains the same. Using the same definition of j and i, and letting i' be the state $(\#X, \#DNA, \#X \bullet DNA) = (a+1, b+1, c-1)$, gives

$$\mathcal{W}_{kj} = \begin{cases} k_1 \times a \times b & \textit{if } k = i \\ k_{-1} \times c & \textit{if } k = i' \\ -k_1 \times a \times b - k_{-1} \times c & \textit{if } k = j \\ 0 & \textit{otherwise} \end{cases}$$

In the special case of a single DNA molecule (bound or unbound), there are only two states: $a = x, b = 1, c = 0$ and $a = x-1, b = 0, c = 1$, and

$$\mathcal{W} = \begin{bmatrix} -x \times k_1 & k_{-1} \\ x \times k_1 & -k_{-1} \end{bmatrix}$$

This corresponds to Example 7 of the previous chapter.

Example 16 *Consider a simple degradation reaction:*

$$A \xrightarrow{k} \textit{no } A$$

For simplicity, suppose the initial state contains 3 molecules of A. Then

$$\frac{d\overrightarrow{P}(t|S_0, t_0)}{dt} = \frac{d}{dt}\begin{bmatrix} P(\#A = 3, t|S_0, t_0) \\ P(\#A = 2, t|S_0, t_0) \\ P(\#A = 1, t|S_0, t_0) \\ P(\#A = 0, t|S_0, t_0) \end{bmatrix} = \begin{bmatrix} -3k & 0 & 0 & 0 \\ 3k & -2k & 0 & 0 \\ 0 & 2k & -k & 0 \\ 0 & 0 & k & 0 \end{bmatrix} \overrightarrow{P}(t|S_0, t_0)$$

3.4.3 Pros and Cons of This Approach

The chemical master equation has a nice feel to it: one is able to capture all the information of a system into a compact vector equation. Solving that equation gives the complete probability distribution at any point in time. Even better, it is a linear differential equation with constant coefficients, so unlike most differential equations, one can actually solve it.

There is, of course, a catch: the chemical master equation needs one variable for each possible state of the system. For all but the simplest systems, this number of variables becomes huge, and so one cannot even write out the full master equation, let alone solve it.

Example 17 *Consider the degradation reactions*

$$A \xrightarrow{k_1} \textit{no } A$$
$$B \xrightarrow{k_2} \textit{no } B$$
$$C \xrightarrow{k_3} \textit{no } C$$

Suppose the initial conditions are $(\#A, \#B, \#C) = (99, 99, 99)$*. Each chemical species can vary independently, so all possible combinations of 3 integers between 0 and 99 are possible. There are precisely* $100 \times 100 \times 100 = 10^6$ *such combinations. Thus, this simple example of 3 reactions leads to a vector* $\overrightarrow{P}$ *of size 1 million, and a* $\mathcal{W}$*-matrix of size 1 million by 1 million.*

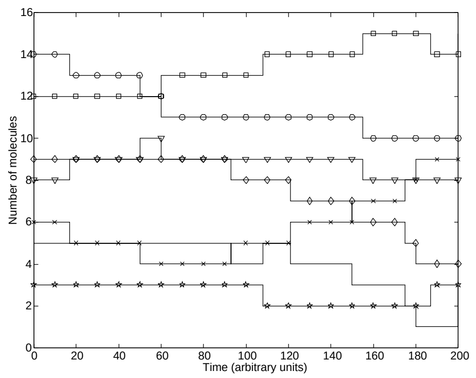
Example 18 *In the Arkin et al. [3] model of lambda phage, reasonable limits on the number of each kind of molecule lead to a number of states on the order of* 10^{70}*. Even if a miscalculation, or a clever reduction of the number of states, say by a factor of* 10^{50}*, leaves* 10^{20} *states, which is still intractable. Exact stochastic simulation provides a more feasible approach.*

For a more detailed example of the stochastic framework, please see [13].

3.5 Exact Stochastic Simulation

Consider, for example, the set of reactions:

(a)



(b)

Time	0	17	50	60	93	108	121	150	155	175
#A	6	5	4	4	5	5	6	7	7	8
#B	14	13	12	11	11	11	11	11	10	10
#C	8	9	10	9	9	9	9	9	8	8
#D	12	12	12	13	13	14	14	14	15	15
#E	9	9	9	9	8	8	7	6	6	5
#F	3	3	3	3	3	2	2	2	2	2
#G	5	5	5	5	4	5	4	3	3	2
Reaction	—	1	1	2	5	4	5	5	2	5

Figure 3-2: Example trajectory. (a) Graphical representation. Legend: A-X, B-circle, C-triangle, D-square, E-diamond, F-star, G-line. (b) State representation. The ‘Reaction’ row merely indicates which reaction occurred; it is not part of the state.

$$\begin{aligned}
A + B &\xrightarrow{k_1} C \\
B + C &\xrightarrow{k_2} D \\
D + E &\xrightarrow{k_3} E + F \\
F &\xrightarrow{k_4} D + G \\
E + G &\xrightarrow{k_5} A \qquad (3.3)
\end{aligned}$$

As a different way of dealing with the stochastic framework, consider the problem of generating a single sample trajectory of a chemical process in the stochastic framework, as in Figure 3-2(a). The (intractable) Master Equation approach tries to write a system of equations and solve simultaneously for the probability of *all* possible trajectories. Generating *a single* trajectory is significantly easier; as in Figure 3-2(b), one needs to generate a sequence of state transitions and the times at which they occur. A naive way to generate legal trajectories is to start with the initial state, repeatedly pick reactions arbitrarily, and execute them, thus generating a legal trajectory. A better way to generate trajectories is to pick reactions and times *according to the correct probability distributions* so that the probability of generating a given trajectory with the simulation algorithm is exactly the probability that would come out of the solution of the Master Equation. Amazingly, it is possible to create an algorithm that has this property, *even if* it is not possible to write out the entire Master Equation explicitly, let alone solve it. (There are also *in*exact stochastic simulation algorithms that generate trajectories according to approximately the correct distribution. Our interest is only in exact methods, however.)

Given the ability to generate a single trajectory with the correct probability, one may estimate any parameter of interest by generating many trajectories, calculating the value of the parameter for each trajectory, and observing the statistics of those calculated values. For example, to find the average number of molecules of B present at time t, one can run many (say hundreds or thousands of) trajectories and plot a histogram of the values of the number of molecules of B at time t.

Gillespie [15, 16] developed two *exact stochastic simulation* algorithms, discussed in the next section. The tricky mathematical part of such an algorithm is specifying how to generate random numbers so that they will have the correct distributions. The tremendous success of these exact stochastic simulation algorithms has led to them being applied to much larger systems than was originally designed for. For example Arkin *et al.* [3] used exact stochastic simulation to simulate a model of a simple virus, lambda phage, containing 75 equations in 57 chemical species. Because the original algorithms do not scale readily to large systems, this thesis presents new versions that *do* scale well with number of reactions. (The tricky computer science part is to develop *efficient* algorithms that do the right thing.)

3.6 Gillespie's Algorithms

Consider a system of r reactions as in Eq 3.3. For now, assume that all rate constants (e.g., $k_1 \ldots k_5$ in Eq 3.3) are true constants; time-varying rate constants will be covered in Chapter 6. Gillespie [16] proposed two *exact stochastic simulation* algorithms. At each time step, the system is in exactly one state. A transition consists of executing a reaction, so there are at most r possible transitions from a given state. The key is to choose random numbers using a computer's random number generator, and use those to pick transitions.

One must be careful to choose from the correct distribution at each point in the algorithm.

Gillespie proposed two methods for accomplishing the simulation. The first method, which he calls the *Direct Method*, calculates explicitly *which* reaction occurs next and *when* it occurs. The second method, which he calls the *First Reaction Method*, generates, for each reaction μ, a putative time τ_μ at which reaction μ occurs, then chooses the reaction μ^* with the smallest τ_μ^* (the *first* reaction) and executes reaction μ^* at time τ_μ^*. This section describes both of these methods.

3.6.1 Gillespie's Direct Method

For a system in a given state, Gillespie's direct algorithm asks two questions:

- Which reaction occurs next?
- When does it occur?

Clearly, both of these questions must be answered probabilistically, by specifying the probability density $P(\mu, \tau)$ that the next reaction is μ and it occurs between times τ and $\tau + d\tau$. Theorem 1 (see also [16]) will show that

$$\begin{aligned} P(\mu,\tau)d\tau &= a_\mu \exp\left(-\tau \sum_j a_j\right) d\tau \\ &= \left[\frac{a_\mu}{\sum_j a_j}\right]\left[\left(\sum_j a_j\right)\exp\left(-\tau\sum_j a_j\right) d\tau\right] \end{aligned} \tag{3.4}$$

Theorem 1 *Eq 3.4 holds.*

Proof. By definition

$$P(\mu,\tau)d\tau = \text{Prob(the next reaction is } \mu \text{ and it occurs between times } \tau \text{ and } \tau + d\tau)$$

This is equal to

$$\begin{aligned}
& \text{Prob(no reaction occurs before } \tau\text{, and reaction } \mu \text{ occurs between times } \tau \text{ and } \tau + d\tau) \\
= \; & \text{Prob(no reaction occurs before } \tau) \\
& \times \text{Prob(reaction } \mu \text{ occurs between times } \tau \text{ and } \tau + d\tau | \text{no reaction occurs before } \tau)
\end{aligned}$$

The second multiplicative term is simply equal to $a_\mu \times d\tau$. Define

$$q(t) = \text{Prob(no reaction occurs before } t)$$

Then, by the laws of probability

$$\begin{aligned}
q(t + dt) &= \text{Prob(no reaction occurs before } t + dt) \\
&= \text{Prob(no reaction occurs before } t) \\
&\quad \times \text{Prob(no reaction occurs between times } t \text{ and } t + dt | \text{no reaction occurs before } t) \\
&= q(t) \times \left[1 - dt \sum_j a_j \right]
\end{aligned}$$

Rearranging,

$$\frac{dq}{dt} = \frac{q(t + dt) - q(t)}{dt} = -\sum_j a_j$$

Thus

$$q(t) = q(t_0) \exp \left[-\int_{t_0}^{t} \left(\sum_j a_j \right) du \right]$$

In particular, $q(t_0) = 1$, and for the present chapter, it is assumed that a_j is time-independent, and $t_0 = 0$, so

$$q(t) = \exp \left[-t \left(\sum_j a_j \right) \right]$$

Putting this all together,

$$\begin{aligned} P(\mu,\tau)d\tau &= q(\tau)\ \times a_\mu \times d\tau \\ &= \exp\left[-\tau\left(\sum_j a_j\right)\right] \times a_\mu \times d\tau \end{aligned}$$

■

This leads directly to the answers of the two questions above. The probability distribution for reactions and the probability distribution for times are independent, and are given by

$$\Pr(\text{ Reaction } = \mu\ ) = a_\mu / \sum_j a_j \tag{3.5}$$

and

$$P(\tau)d\tau = \left(\sum_j a_j\right) \exp\left(-\tau \sum_j a_j\right) d\tau \tag{3.6}$$

These two distributions lead to Gillespie's direct algorithm [16]:

Algorithm 1 *Exact Stochastic Simulation — Direct Method (Gillespie)*

```
1. Initialize(i.e., set initial numbers of molecules, set $t \leftarrow 0$.)
2. Calculate the propensity function, $a_i$, for all $i$.
3. Choose $\mu$ according to the distribution in Eq 3.5.
4. Choose $\tau$ according to an exponential with parameter $\sum_j a_j$ (as in Eq 3.6).
5. Change the number of molecules to reflect execution of reaction $\mu$. Set $t \leftarrow t + \tau$.
6. Go to Step 2.
```

As written, this algorithm uses two random numbers per iteration, takes time proportional to the number of reactions to update the a_i's, and takes time proportional to the number of reactions to calculate $\sum_j a_j$ and to generate a random number according to the distribution in Eq 3.5. Chapter 7 will show how to make the algorithm more efficient, so that the time it takes is proportional to the logarithm of the number of reactions.

3.6.2 Gillespie's First Reaction Method

The algorithm of the previous subsection is direct in the sense that it generates μ and τ directly. Gillespie also developed the First Reaction Method [15], which generates a putative time τ_i for each reaction to occur — a time the reaction would occur if no other reaction occurred first — then lets μ be the reaction whose putative time is first, and lets τ be the putative time τ_μ. Formally:

Algorithm 2 *(Exact Stochastic Simulation — First Reaction Method)*

1. *Initialize(i.e., set initial numbers of molecules, set $t \leftarrow 0$.)*
2. *Calculate the propensity function, a_i, for all i.*
3. *For each i, generate a putative time, τ_i, according to an exponential distribution with parameter a_i.*
4. *Let μ be the reaction whose putative time, τ_μ, is least.*
5. *Let τ be τ_μ.*
6. *Change the number of molecules to reflect execution of reaction μ. Set $t \leftarrow t + \tau$.*
7. *Go to Step 2.*

As written, this algorithm uses r random numbers per iteration (where r is the number of reactions), takes time proportional to r to update the a_i's, and takes time proportional to r to identify the smallest τ_μ.

3.7 Equivalence of Algorithms With Each Other, and With the Master Equation

At first glance, these two algorithms may seem very different, but they are provably equivalent [15], i.e., the probability distributions used to choose μ and τ are the same.

Proof. In Algorithm 2, the μ chosen is the one whose τ_μ is least. Let R be the random variable describing the choice of reaction, and let T_μ be the random variable describing the choice of τ_μ. By assumption, the random numbers used are statistically independent. Then

$$\begin{aligned}
\Pr(R = \mu) &= \Pr(\min_\alpha(T_\alpha) = T_\mu) \\
&= \int_{u=\mathbf{0}}^{\infty} P(u) \prod_{\alpha \neq \mu} \Pr(T_\alpha > u) du \\
&= \int_{u=\mathbf{0}}^{\infty} a_\mu \exp(-a_\mu u) \prod_{\alpha \neq \mu} \exp(-a_\alpha u) du \\
&= a_\mu \int_{u=\mathbf{0}}^{\infty} \exp(-\sum_\alpha (a_\alpha) u) du \\
&= \frac{a_\mu}{\sum_\alpha (a_\alpha)}
\end{aligned}$$

$$\begin{aligned}
\Pr(T &> u) = \Pr(\min_\alpha(T_\alpha) > u) \\
&= \prod_\alpha \Pr(T_\alpha > u) \\
&= \prod_\alpha \exp(-a_\alpha u) \\
&= \exp(\sum_\alpha (-a_\alpha) u)
\end{aligned}$$

$$\begin{aligned}
P(u) &= -\frac{d \Pr(T > u)}{du} \\
&= (\sum_\alpha a_\alpha) \exp(\sum_\alpha (-a_\alpha) u) \qquad (3.7)
\end{aligned}$$

■

Chapter 4 The Lambda Model - a Motivating Example

4.1 Summary

The Arkin et al. [3] model of the temperate bacteriophage lambda provides an example of the issues one encounters in stochastic modeling of large systems. This chapter summarizes the model. Recall that the focus of this thesis is developing general computational methods — not calculations for a specific model. Subsequent chapters will show the general computational framework and how easy it is to incorporate optimizations for this specific model.

4.2 Introduction to Lambda Phage

Lambda phage [34] is a virus that infects the bacterium *E. coli.* It is called a temperate phage, because it has two possible developmental pathways (see Figure 4-1) — it can either:

- replicate and lyse (dissolve) the host cell, thus releasing about 100 progeny or
- integrate its DNA into the bacterial DNA and form a lysogen.

In the latter case, the virus will replicate passively whenever the bacterium replicates. A lysogen has immunity from subsequent lambda infections; this protects the lysogen from being destroyed should another phage infect the host cell.

Under the right conditions (e.g., exposure to UV light), a lysogen can be induced, i.e., the viral DNA excises itself from the bacterial DNA and undergoes normal replication and lysis. (See [34].)

Lambda has been studied extensively because it is one of the simplest developmental switches - systems with multiple possible end states. Its complete genome (50,000 base pairs) was sequenced long before the era of large-scale genome sequencing. The details of which genes are expressed, when, and in what quantity have been studied extensively, as has lambda's host, *E. coli.* As a result, lambda is one of the best-understood organisms in existence. It is also an organism that exhibits probabilistic or stochastic behavior — some

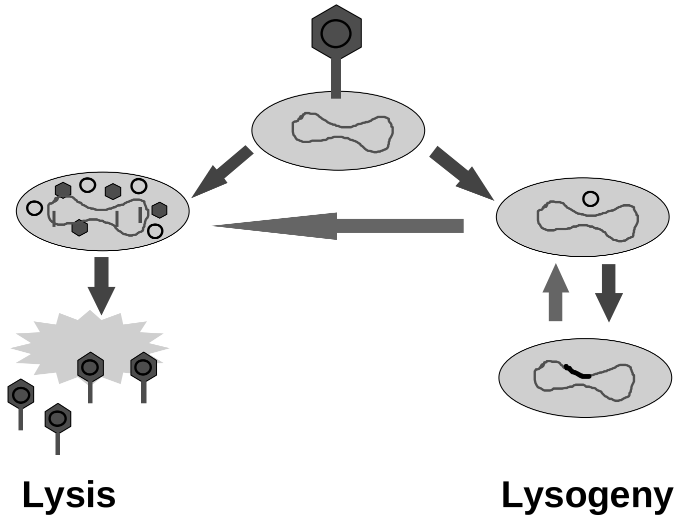


Figure 4-1: The life cycle of lambda phage. A phage injects its DNA into the *E. coli* host, then either (left) replicates and destroys the host cell — lysis — or (right) integrates its DNA into the host DNA — lysogeny.

infected cells end up in lysis, others in lysogeny. Hence it is a perfect system in which to demonstrate the stochastic framework of gene regulation modeling.

4.2.1 Origins of Stochastic Behavior

Where does this probabilistic or stochastic behavior come from?

An *E. coli* cell, which acts as the host for lambda, is a rod shaped bacterium 2 μm long with a diameter of 1 μm [39]. The volume of an *E. coli* cell is

$$V = \pi r^2 l = \frac{\pi}{2} \times 10^{-15} \text{ liters}$$

As will be explained later, significant differences in the amount of binding occur in the range 10^{-9} M to 10^{-7} M. (Remember that M = moles/liter.) The number of molecules that corresponds to, say, 10^{-8} M is

$$(\frac{\pi}{2} \times 10^{-15} \text{ liters})(10^{-8} \text{ moles/liter})(6 \times 10^{23} \text{ molecules/mole}) \approx 10 \text{ molecules}$$

The lambda model deals with 1 molecule of DNA, a few molecules of mRNA, and 10s to 100s of molecules of proteins.

4.3 The Lambda Model

4.3.1 Model Overview

The central dogma of molecular biology is that genetic information travels from DNA to messenger RNA (mRNA) to proteins. This is frequently abbreviated

$$\text{DNA} \rightarrow \text{RNA} \rightarrow \text{protein}$$

DNA and mRNA store information in a linear form. Proteins are complex macromolecules whose three-dimensional shape is very important, and which can have complex functions. Information stored in DNA is called a *gene*, and a gene is said to be *expressed* if its protein is made.

The lambda model deals with *transcription* (the processes by which RNA is created from DNA), *translation* (the process by which a protein is created from mRNA), various protein-protein interactions, and feedback via protein-DNA binding. (For an extended overview of these processes, see [14].)

The basic steps that occur when lambda infects a host are:

- Lambda injects its DNA into the host.
- The machinery in the host treats lambda DNA just like native host DNA, and transcribes lambda mRNA. (The host is unable to distinguish lambda DNA from its native DNA, because DNA is all the same chemically.)
- The machinery in the host treats lambda mRNA just like native host mRNA, and translates lambda proteins. (The host is unable to distinguish lambda mRNA from its native mRNA, because mRNA is all the same chemically.)
- In *lysis*, the proteins that are created: make shells for new viruses, copy the lambda DNA, dissolve (lyse) the host cell, etc.
- In *lysogeny*, the proteins that are created: cut host DNA, insert lambda DNA into the cut, glue it all back together, turn off the expression of lysis genes, etc.
- Certain early genes control whether the lysis or the lysogeny genes are expressed; the protein products of these genes bind to DNA and affect the rate at which mRNA of

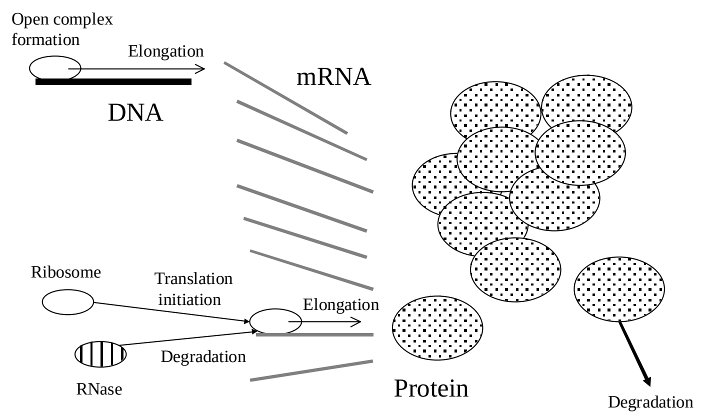


Figure 4-2: Processes included in the lambda phage model.

each type is transcribed.

The Arkin et al. [3] model of this process consists of chemical equations for:

- transcription: mRNA being made from DNA,
- translation: proteins being made from mRNA,
- degradation: proteins and mRNA being recycled, and
- protein-DNA binding, and how it affects the rate of mRNA synthesis.

All these processes are illustrated in Figure 4-2.

4.3.2 Specifics

The model deals with 5 genes, their 5 mRNAs, and their 5 protein products. By convention, genes are in italics, and protein products are in normal text:

- *cro* controls lysis. The cro protein turns off *cI* and activates those genes that are required to duplicate DNA, make virus shells, and lyse the cell.
- *cI* is present in lysogeny. The cI protein shuts everything down, especially *cro*. (The cI protein is typically called repressor, but for simplicity, we shall call it cI.)
- *N* is a temporal regulator. It is expressed very early in the infection cycle, and turns on early genes that are necessary for both lysis and lysogeny. It is turned off by either cI or cro protein.

$$\text{OR, right} \rightarrow \textit{cro} \rightarrow \text{NUT-R} \rightarrow \text{TR}_1 \rightarrow \textit{cII}$$
$$\textit{cIII} \leftarrow \text{TL}_1 \leftarrow \textit{N} \leftarrow \text{NUT-L} \leftarrow \text{PL, left} \leftarrow \textit{cI} \leftarrow \text{OR, left} \leftarrow \text{PRE, left}$$

Table 4.1: The linear order of genes (in italics), promoters (OR, PRE, PL), termination sites (TR_1, TL_1) and anti-termination sites (NUT-R, NUT-L).

- *cII* and *cIII* are "decision" genes. Under the right conditions, these will be stably expressed, which leads to *cI* expression. If conditions are such that cII and cIII proteins are unstable, *cI* is not expressed, and *cro* is instead.

For more information on lambda biology (including the function of these and other genes) see Ptashne [34] or Lambda II [23]. For more details on the model (including justification of various assumptions) see Arkin et al. [3]. Reference [13] contains a subset of the full model, namely the regulation of N by cI and cro.

4.4 Chemical Equations for the Model

4.4.1 Transcription

Transcription occurs when the enzyme RNA Polymerase (RNAP), bound at a promoter, unzips the DNA double helix, thus forming an open complex, and begins transcribing DNA into mRNA (e.g., Reaction 1 in Table 4.2). RNAP moves along the DNA step by step and elongates the mRNA (e.g., Reaction 2 in Table 4.2). The end result of elongation is an increase in the amount of mRNA of the gene in question (e.g., mRNA_{cro} in Reaction 2 of Table 4.2).

Notation 1 *We shall use the notation RNAP•DNA(X) to mean "RNAP bound to DNA at position X," and mRNA$_Y$ to mean "mRNA corresponding to gene Y." The position names are really just labels, but correspond to significant positions on the DNA, such as the promoter, the end of a gene, or anti-termination sites (see below).*

It is instructive to record the relative order of positions, as in Table 4.1.

The remainder of the transcription reactions are Reactions 3-34. Reactions 1, 15, 17, and 20 are transcription initiation reactions. Reactions 2, 3, 8, 9, 13, 14, 16, 18, 19, 21, 26, 27, 28, 29, 33, and 34 are transcription elongation reactions.

The N-utilization sites *NUTR* and *NUTL* are places where the transcription process slows down (Reactions 4 & 22) and the RNAP/DNA complex can bind N with some probability (Reactions 5-7, 23-25). Once the RNAP/DNA complex (with or without N) moves past these sites, it continues as normal (Reactions 8-9, 26-29). When the RNAP/DNA complex reaches the termination sites TR_1 and TL_1, complexes without N bound terminate transcription with some probability: the RNAP falls off the DNA at that point (Reactions 10,11, 30 & 31). Anti-terminated complexes, i.e., complexes with N bound, pass through the termination sites (Reactions 12, 32).

4.4.2 Translation

Mathematically, translation is very similar to transcription; the molecular players are different, but the processes are similar. A *ribosome* binds the mRNA (Reactions 40, 42, 44, 46, and 48) and elongates step by step, translating mRNA into protein (Reactions 41, 43, 45, 47, and 49). There is none of the complication of anti-termination to deal with. However, we shall also consider degradation of mRNA: instead of a ribosome binding mRNA, a ribonuclease (RNase) can bind to mRNA and degrade it (Reactions 35-39). Both ribosome binding and RNase binding are modeled as pseudo-first order processes, i.e., the concentration of ribosome and RNase are assumed to be constant, so that the reaction constant subsumes the concentration of ribosome or RNase.

4.4.3 Dimerization and Simple Protein Degradation

The protein products cro and cI can form dimers, i.e., two copies bind together (Reactions 50-53). It is these dimer forms that bind DNA. The (monomer) forms of cI and cro protein, as well as N protein, follow simple degradation laws (Reactions 54-56). The protein products of cII and cIII follow the more complicated scheme in the next subsection.

4.4.4 Complex Protein Degradation

Degradation of cII and cIII proteins is highly interconnected, and involves two proteins P1 and P2 from the host. (Quite possibly these proteins are hflA and hflB, but that is not important for the purposes of calculation.) This complex degradation obeys the equations in Reactions 57-68.

4.4.5 Cell

The cell volume doubles over the course of 35 minutes (“Reaction” 69). The host proteins RNAP, P1, and P2 must be generated, in order to keep their concentrations roughly the same as the cell grows (Reactions 70-72). The reactions to do this may seem strange — certainly these proteins come from somewhere — but the key point is that the process by which these proteins are made is not part of the model.

4.5 Promoters

All of the rate constants labeled ‘see 4.5’ in the tables above belong to reactions of the form RNAP•DNA$(X)_{closed}$ →RNAP•DNA$(X)_{open}$. The astute reader will also note that there are no reaction whose product is RNAP•DNA$(X)_{closed}$. These species, and the resulting rate constants, come from an equilibrium binding model of promoters [1, 37]. Recall Example 7 of Chapter 2, which gave a very simple equilibrium model, and Section 2.8, which showed how to use partition functions to deal with complicated models. The more complex models use the same techniques, but with more states. Each state has an associated rate constant for open complex formation, whose interpretation is: the rate constant given that the system is in that state. The equilibrium models are in Tables 4.4 through 4.6, and come from [3]. (Note that the O_R operator has two promoters: one left and one right.)

Example 19 *(Equilibrium probability distribution for P_{RE}.) Call the states of P_{RE} (from Table 4.4) 1, 2, 3, and 4. From Chapter 2, P(state s) will be of the form*

$$\frac{\exp(-\Delta G_s/RT) \times [cII]^{power} \times [RNAP]^{power}}{Z}$$

where Z is the partition function, and consists of the sum of all the different numerators for all four possible states. To simplify notation, let $K_s = \exp(-\Delta G_s/RT)$ for $s \in \{1,2,3,4\}$. Note that all ΔG values are given in kcal/mole. The temperature used is 37C, so using 1 kcal = 4184 J, R = 8.314 J/(K×mole) and converting degrees C to K, $\Delta G/RT = (\Delta G \times$

$4184)/(8.314 \times (273+37))$. *Thus*

$$\begin{aligned}
K_1 &= \exp(-\Delta G_1/RT) = \exp(0) = 1 \\
K_2 &= \exp(-\Delta G_2/RT) = \exp(-((-9.9) \times 4184)/(8.314 \times (273+37))) = 9.54 \times 10^6 \\
K_3 &= \exp(-\Delta G_3/RT) = \exp(-((-9.7) \times 4184)/(8.314 \times (273+37))) = 6.90 \times 10^6 \\
K_4 &= \exp(-\Delta G_4/RT) = \exp(-((-21.5) \times 4184)/(8.314 \times (273+37))) = 1.44 \times 10^{15}
\end{aligned}$$

So,

$$\begin{aligned}
P(1) &= \frac{K_1}{Z} \\
P(2) &= \frac{K_2 \times [RNAP]}{Z} \\
P(3) &= \frac{K_3 \times [cII]}{Z} \\
P(4) &= \frac{K_4 \times [RNAP] \times [cII]}{Z}
\end{aligned}$$

where

$$\begin{aligned}
Z &= K_1 + K_2 \times [RNAP] + K_3 \times [cII] + K_4 \times [RNAP] \times [cII] \\
[RNAP] &= \frac{\#RNAP}{N_{Avagadro} \times V} = \frac{\#RNAP}{10^9} \\
[cII] &= \frac{\#cII}{N_{Avagadro} \times V} = \frac{\#cII}{10^9}
\end{aligned}$$

Example 20 *(Average rate of open complex formation.) The values of k_{oc} reported in Table 4.4 are conditioned on the system being in each respective state. We can define the average rate by*

$$\begin{aligned}
\langle k_{oc} \rangle &= \sum_s P(s) \times k_{oc}(s) \\
&= P(1) \times 0 + P(2) \times 0.00004 + P(3) \times 0 + P(4) \times 0.015 \\
&= P(2) \times 0.00004 + P(4) \times 0.015
\end{aligned}$$

Using the formulas from the previous example, let $\#RNAP = 30$, and $\#cII = 0$. So

$[RNAP] = 30 \times 10^{-9}$ *and* $[cII] = 0$. *Then,*

$$\begin{aligned} Z &= K_1 + K_2 \times [RNAP] + K_3 \times [cII] + K_4 \times [RNAP] \times [cII] \\ &= 1 + \left(9.54 \times 10^{6}\right) \times \left(30 \times 10^{-9}\right) + 0 + 0 \\ &= 1.29 \\ P(2) &= \frac{\left(9.54 \times 10^{6}\right) \times \left(30 \times 10^{-9}\right)}{1.29} = 0.223 \\ P(4) &= 0 \\ \langle k_{oc} \rangle &= 0.223 \times 0.00004 = 8.9 \times 10^{-6} \end{aligned}$$

Now, let $\#RNAP = 30$, *and* $\#cII = 20$. *So* $[RNAP] = 30 \times 10^{-9}$ *and* $[cII] = 20 \times 10^{-9}$. *Then,*

$$\begin{aligned} Z &= K_1 + K_2 \times [RNAP] + K_3 \times [cII] + K_4 \times [RNAP] \times [cII] \\ &= 1 + \left(9.54 \times 10^{6}\right) \times \left(30 \times 10^{-9}\right) + \left(6.90 \times 10^{6}\right) \times \left(20 \times 10^{-9}\right) \\ &\quad + \left(1.44 \times 10^{15}\right) \times \left(30 \times 10^{-9}\right) \times \left(20 \times 10^{-9}\right) \\ &= 2.29 \\ P(2) &= \frac{\left(9.54 \times 10^{6}\right) \times \left(30 \times 10^{-9}\right)}{2.29} = 0.125 \\ P(4) &= \frac{\left(1.44 \times 10^{15}\right) \times \left(30 \times 10^{-9}\right) \times \left(20 \times 10^{-9}\right)}{2.29} = 0.377 \\ \langle k_{oc} \rangle &= 0.125 \times 0.00004 + 0.377 \times 0.015 = 0.0057 \end{aligned}$$

Thus, changing the amount of cII by 20 molecules increases the rate of open complex formation by a factor of nearly 1000.

4.6 Discussion

To a first approximation, this model consists of many reactions and an equilibrium binding model. Our first cut at calculation (in Chapter 5) will discuss how to deal with large numbers of reactions. Calculations involving the equilibrium binding model will be discussed in Chapter 8.

There are a couple other things to note. First, the transcription and translation reactions consist of many identical steps. Although one could deal with them by splitting to one step per reaction, there are more efficient ways, detailed in Chapter 6. Finally, second (and higher) order reactions are not simple exponentials when volume is allowed to change. One way to handle that is to assume volume changes slowly and recalculate second order rate constants once in a while. Another way is presented in Chapter 6.

#	Reaction	Constant (s^{-1})	Other
1	RNAP•DNA($O_{R,right}$)$_{closed}$ $\longrightarrow$RNAP•DNA($O_{R,right}$)$_{open}$	see 4.5	
2	RNAP•DNA(O_R)$_{open}$ $\longrightarrow$RNAP•DNA(cro)+mRNA$_{cro}$	30	240 steps
3	RNAP•DNA(cro) $\longrightarrow$RNAP•DNA(NUTR)	30	24 steps
4	RNAP•DNA(NUTR) $\longrightarrow$RNAP•DNA(NUTR+1)	5	
5	RNAP•DNA(NUTR)+N$\longrightarrow$RNAP•N•DNA(NUTR)	0.145	
6	RNAP•N•DNA(NUTR) $\longrightarrow$RNAP•DNA(NUTR)+N	0.1	
7	RNAP•N•DNA(NUTR) $\longrightarrow$RNAP•N•DNA(NUTR+1)	30	
8	RNAP•DNA(NUTR+1) $\longrightarrow$RNAP•DNA(TR$_1$)	30	68 steps
9	RNAP•N•DNA(NUTR+1) $\longrightarrow$RNAP•N•DNA(TR$_1$)	30	68 steps
10	RNAP•DNA(TR$_1$) $\longrightarrow$RNAP$_{free}$	15	
11	RNAP•DNA(TR$_1$) $\longrightarrow$RNAP•DNA(TR$_1 + 1$)	15	
12	RNAP•N•DNA(TR$_1$) $\longrightarrow$RNAP•N•DNA(TR$_1 + 1$)	30	
13	RNAP•DNA(TR$_1 + 1$) $\longrightarrow$RNAP$_{free}$+mRNA$_{cII}$	30	315 steps
14	RNAP•N•DNA(TR$_1 + 1$) $\longrightarrow$RNAP$_{free}$+N+mRNA$_{cII}$	30	315 steps
15	RNAP•DNA(P_{RE})$_{closed}$ $\longrightarrow$RNAP•DNA(P_{RE})$_{open}$	see 4.5	
16	RNAP•DNA(P_{RE})$_{open}$ $\longrightarrow$RNAP•DNA($O_{R,left}$)$_{open}$	30	400 steps
17	RNAP•DNA($O_{R,left}$)$_{closed}$ $\longrightarrow$RNAP•DNA($O_{R,left}$)$_{open}$	see 4.5	
18	RNAP•DNA($O_{R,left}$)$_{open}$ $\longrightarrow$RNAP•DNA(cI)+mRNA$_{cI}$	30	720 steps
19	RNAP•DNA(cI) $\longrightarrow$RNAP•DNA(P_L)$_{open}$	30	1640 steps
20	RNAP•DNA(P_L)$_{closed}$ $\longrightarrow$RNAP•DNA(P_L)$_{open}$	see 4.5	
21	RNAP•DNA(P_L)$_{open}$ $\longrightarrow$RNAP•DNA(NUTL)	30	60 steps
22	RNAP•DNA(NUTL) $\longrightarrow$RNAP•DNA(NUTL+1)	5	
23	RNAP•DNA(NUTL)+N$\longrightarrow$RNAP•N•DNA(NUTL)	0.145	
24	RNAP•N•DNA(NUTL) $\longrightarrow$RNAP•DNA(NUTL)+N	0.1	
25	RNAP•N•DNA(NUTL) $\longrightarrow$RNAP•N•DNA(NUTL+1)	30	
26	RNAP•DNA(NUTL+1) $\longrightarrow$RNAP•DNA(N)+mRNA$_N$	30	490 steps
27	RNAP•N•DNA(NUTL+1) $\longrightarrow$ RNAP•N•DNA(N)+mRNA$_N$	30	490 steps
28	RNAP•DNA(N) $\longrightarrow$RNAP•DNA(TL$_1$)	30	480 steps
29	RNAP•N•DNA(N) $\longrightarrow$RNAP•N•DNA(TL$_1$)	30	480 steps
30	RNAP•DNA(TL$_1$) $\longrightarrow$RNAP$_{free}$	25	
31	RNAP•DNA(TL$_1$) $\longrightarrow$RNAP•DNA(TL$_1 + 1$)	5	
32	RNAP•N•DNA(TL$_1$) $\longrightarrow$RNAP•N•DNA(TL$_1 + 1$)	30	
33	RNAP•DNA(TL$_1 + 1$) $\longrightarrow$RNAP$_{free}$+mRNA$_{cIII}$	30	1260 steps
34	RNAP•N•DNA(TL$_1 + 1$) $\longrightarrow$RNAP$_{free}$+N+mRNA$_{cIII}$	30	1260 steps
35	mRNA$_{cI}$+RNase$\longrightarrow$RNase	0.03	$= k\times$(#RNase)
36	mRNA$_{cII}$+RNase$\longrightarrow$RNase	0.03	$= k\times$(#RNase)
37	mRNA$_{cII}$+RNase$\longrightarrow$RNase	0.03	$= k\times$(#RNase)
38	mRNA$_{cro}$+RNase$\longrightarrow$RNase	0.03	$= k\times$(#RNase)
39	mRNA$_N$+RNase$\longrightarrow$RNase	0.03	$= k\times$(#RNase)

Table 4.2: The complete set of reactions for the lambda model, Part I.

#	Reaction	Constant (s^{-1})	Other
40	$mRNA_{cI}$+Ribosome⟶Ribosome•$mRNA_{cI}$	0.3	$= k\times$(#Ribosome)
41	Ribosome•$mRNA_{cI}$ ⟶Ribosome+cI+$mRNA_{cI}$	100	710 steps
42	$mRNA_{cII}$+Ribosome⟶Ribosome•$mRNA_{cII}$	0.3	$= k\times$(#Ribosome)
43	Ribosome•$mRNA_{cII}$ ⟶Ribosome+cII+$mRNA_{cII}$	100	290 steps
44	$mRNA_{cIII}$+Ribosome⟶Ribosome•$mRNA_{cIII}$	0.3	$= k\times$(#Ribosome)
45	Ribosome•$mRNA_{cIII}$ ⟶Ribosome+cIII+$mRNA_{cIII}$	100	160 steps
46	$mRNA_{cro}$+Ribosome⟶Ribosome•$mRNA_{cro}$	0.3	$= k\times$(#Ribosome)
47	Ribosome•$mRNA_{cro}$ ⟶Ribosome+cro+$mRNA_{cro}$	100	200 steps
48	$mRNA_{N}$+Ribosome⟶Ribosome•$mRNA_{N}$	0.3	$= k\times$(#Ribosome)
49	Ribosome•$mRNA_{N}$ ⟶Ribosome+N+$mRNA_{N}$	100	320 steps
50	cro+cro⟶cro•cro	0.05	
51	cro•cro⟶cro+cro	0.5	
52	cI+cI⟶cI•cI	0.05	
53	cI•cI⟶cI+cI	0.5	
54	cI⟶nothing	0.0007	
55	cro⟶nothing	0.0025	
56	N⟶nothing	0.0023	
57	cII+P1⟶cII•P1	0.01	
58	cII•P1⟶cII+P1	0.01	
59	cII•P1⟶P1	0.015	
60	cIII+P1⟶cIII•P1	0.05	
61	cIII•P1⟶cIII+P1	0.001	
62	cIII•P1⟶P1	0.0001	
63	cII+P2⟶cII•P2	0.0001	
64	cII•P2⟶cII+P2	0.065	
65	cII•P2⟶P2	0.6	
66	cIII+P2⟶cIII•P2	0.01	
67	cIII•P2⟶cIII+P2	0.01	
68	cIII•P2⟶P2	0.001	
69	Volume $=V_0(1+ct)$	$\frac{1}{35\times60}$	
70	nothing⟶RNAP	0.0146	
71	nothing⟶P1	0.0116	
72	nothing⟶P2	0.0465	
	Temperature		37 C

Table 4.3: The complete set of reactions for the lambda model, continued.

Binding sites		ΔG	k_{oc}
—	—	0	0
—	RNAP	-9.9	0.00004
cII	—	-9.7	0
cII	RNAP	-21.5	0.015

Table 4.4: Equilibrium model of P_{RE} binding site. The units of ΔG are kcal/mole and the units of k_{oc} are s^{-1}.

Binding sites		ΔG	k_{oc}
—	—	0	0
cro•cro	—	-10.9	0
—	cro•cro	-12.1	0
cI•cI	—	-11.7	0
—	cI•cI	-10.1	0
—	RNAP	-12.5	0.011
cro•cro	cro•cro	-22.9	0
cro•cro	cI•cI	-20.9	0
cI•cI	cro•cro	-22.8	0
cI•cI	cI•cI	-23.7	0

Table 4.5: Equilibrium model of P_L binding site. The units of ΔG are kcal/mole and the units of k_{oc} are s^{-1}.

Binding Sites			ΔG	$k_{oc,left}$	$k_{oc,right}$
—	—	—	0		
—	—	cI•cI	-11.7		
—	cI•cI	—	-10.1		
cI•cI	—	—	-10.1		
—	—	cro•cro	-10.8		
—	cro•cro	—	-10.8		
cro•cro	—	—	-12.1		
RNAP	—	—	-11.5	0.001	
—	RNAP	—	-12.5		0.014
—	cI•cI	cI•cI	-23.7		
cI•cI	—	cI•cI	-21.8		
cI•cI	cI•cI	—	-22.2		
—	cro•cro	cro•cro	-21.6		
cro•cro	—	cro•cro	-22.9		
cro•cro	cro•cro	—	-24.0		
RNAP	RNAP	—	-22.5	0.001	0.014
—	cro•cro	cI•cI	-20.9		
—	cI•cI	cro•cro	-23.8		
cI•cI	—	cro•cro	-20.9		
cro•cro	—	cI•cI	-23.8		
cI•cI	cro•cro	—	-20.9		
cro•cro	cI•cI	—	-22.2		
cI•cI	RNAP	—	-22.6		0.014
RNAP	cI•cI	—	-21.6	0.011	
RNAP	—	cI•cI	-23.2	0.001	
cro•cro	RNAP	—	-24.6		0.014
RNAP	cro•cro	—	-22.3	0.001	
RNAP	—	cro•cro	-22.3	0.001	
cI•cI	cI•cI	cI•cI	-33.8		
cro•cro	cro•cro	cro•cro	-33.7		
cro•cro	cI•cI	cI•cI	-35.8		
cI•cI	cro•cro	cI•cI	-32.6		
cI•cI	cI•cI	cro•cro	-33.0		
cI•cI	cro•cro	cro•cro	-31.7		
cro•cro	cI•cI	cro•cro	-33.0		
cro•cro	cro•cro	cI•cI	-34.6		
RNAP	cI•cI	cI•cI	-35.2	0.011	
RNAP	cro•cro	cro•cro	-33.1	0.001	
RNAP	cro•cro	cI•cI	-34.0	0.001	
RNAP	cI•cI	cro•cro	-32.4	0.011	

Table 4.6: Equilibrium model of O_R binding site. The units of ΔG are kcal/mole and the units of k_{oc} are s^{-1}.

Part III

Exact, Efficient Stochastic Simulation

Chapter 5 The Next Reaction Method

5.1 Summary

This chapter describes an efficient algorithm, the Next Reaction Method, for exact simulation of chemical reactions in the stochastic framework. The Next Reaction Method takes time proportional to the logarithm of the number of reactions, not to the number of reactions, and uses a single random number per simulation event. This chapter contains the basic description of the algorithm (as applied to time-invariant Markov processes) and a detailed description of the data structures used. The extension to time-varying and non-Markovian processes is done in the next chapter.

5.2 The Next Reaction Method

Gillespie's First Reaction Method has three activities that occur every iteration and take time proportional to the number of reactions, r:

1. updating all r of the a_is,
2. generating a putative time, τ_i, for each i, and
3. identifying the smallest putative time, τ_μ.

The *Next Reaction Method* will do away with each of these in turn. The main ideas used are:

- *Store τ_i, not just a_i.*
- *Be extremely sensitive in recalculating a_i (and τ_i); recalculate a_i only if it changes.* The preceding statement may seem circular: how can one know that a_i has changed or not changed without calculating it and comparing to its old value? In fact, one can analyze the set of reactions beforehand and determine which reactions change which a_is. Section 5.3 will introduce a data structure, called a *dependency graph*, which allows one to update the minimum number of a_is.

- *Reuse τ_is where appropriate.* In general, Monte Carlo simulations assume statistically independent random numbers, so it is *usually not* legitimate to reuse random numbers. In this particular special case, we shall prove that it *is* legitimate. Specifically, Theorem 2 in Section 5.6 plus two simple transformations make it possible to reuse all τ_is except for τ_μ, the time of the reaction that was just executed. Because re-using random numbers is not valid in general, it is critically important to justify such reuse in this special case.

- *Switch from relative time (time between reactions) to absolute time.* This will obviate the need for one of the two transformations above; for reactions whose underlying a_i has not changed, the putative time τ_i will not have to change, either.

- *Use appropriate data structures to store a_is (and τ_is), so that updating those that change will be a very efficient operation.* Section 5.4 shows a data structure, called an *indexed priority queue*, that achieves this goal.

The formal statement of the algorithm is in Section 5.5, following the definitions of the data structures used.

5.3 Dependency Graphs

Consider, once again, the reactions in Eq 3.3.

Definition 2 *Let* Reactants(ρ) *and* Products(ρ) *be the sets of reactants and products, respectively, of reaction ρ. So, for example,* Reactants*(Reaction 1)* $= \{A, B\}$ *and* Products*(Reaction 1)* $= \{C\}$.

Definition 3 *Let* DependsOn*(a_μ) be the set of substances which affect the value a_μ.*

Evidently, *Reactants*(μ) = *DependsOn*(a_μ). It is sometimes useful to add additional dependencies (e.g., in the lambda model of Chapter 4), so we make this distinction.

Definition 4 *Let* Affects(μ) *be the set of substances that change quantity when reaction μ is executed.*

Reaction	a_μ	*DependsOn*(a_μ)	*Affects*(μ)
$A+B \xrightarrow{k_1} C$	$k_1 \times (\#A) \times (\#B)$	A, B	A, B, C
$B+C \xrightarrow{k_2} D$	$k_2 \times (\#B) \times (\#C)$	B, C	B, C, D
$D+E \xrightarrow{k_3} E+F$	$k_3 \times (\#D) \times (\#E)$	D, E	D, F
$F \xrightarrow{k_4} D+G$	$k_4 \times (\#F)$	F	D, F, G
$E+G \xrightarrow{k_5} A$	$k_5 \times (\#E) \times (\#G)$	E, G	A, E, G

Table 5.1: Example reactions used in the text.

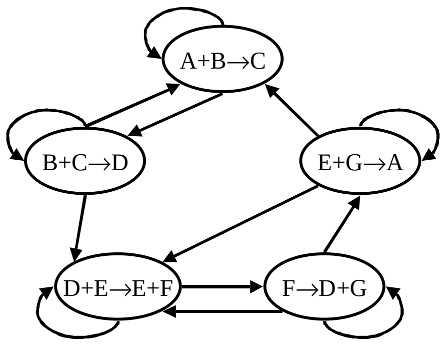


Figure 5-1: Dependency graphs for example equations from Table 5.1.

Typically, *Affects*(μ) = *Reactants*(μ) $\cup$ *Products*(μ), but again, there may be exceptions (e.g. catalytic reactions, such as Reaction 3).

Table 5.1 illustrates each of these concepts.

Definition 5 (Dependency Graph) Let a set of reactions $\mathcal{R}$ be given. Let $\mathcal{G}(V, E)$ be a directed graph with vertex set $V = \mathcal{R}$, and with a directed edge from v_i to v_j if and only if Affects(v_i) $\cap$ DependsOn*(* a_{v_j} *)*$\neq \emptyset$. (If for some strange reason, the self edges from v_i to v_i are not included in this definition, include them as well.) Then $\mathcal{G}$ is called the dependency graph of the set of reactions $\mathcal{R}$.

In other words, a dependency graph is a data structure which tells *precisely* which a_is to change when a given reaction is executed. Using the dependency graph allows one to recalculate only the minimum number of a_is in Step 5 of the Next Reaction Method. The dependency graph of the sample reactions is illustrated in Figure 5-1.

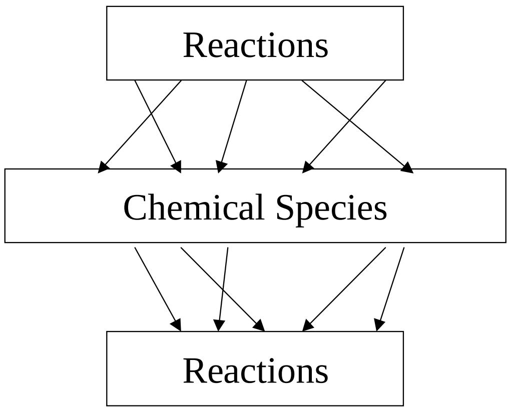


Figure 5-2: Schematic view of the two-step process of generating the dependency graph. First, one records which reactions affect which chemical species, then one records which chemical species affect which reactions and uses this information to infer which reactions affect which other reactions.

5.3.1 Efficient Generation of Dependency Graphs

The definition of dependency graph calls for a vertex for each reaction and an edge from v_i to v_j if and only if Affects(v_i) $\cap$ DependsOn(a_{v_j})$\neq \emptyset$. A simple way to compute the dependency graph is to check, for each possible edge, whether the edge meets the dependency graph criterion. This approach requires (# reactions)^2 operations.

The simple approach may well be sufficient, as a dependency graph need only be generated once per reaction set, regardless of how many simulation events occur per iteration or how many iterations are run. (Further, in the Multiple Next Reaction Method of Chapter 9, only one dependency graph will be used regardless of how many simulation events, iterations, or parameter sets.) For completeness, though, here is a more efficient algorithm for generating dependency graphs.

The basic idea is simple: store a temporary array, **ChemicalAffects**, with one entry for each chemical species. Loop through the reactions once, looking at dependencies: make links in the **ChemicalAffects** array for each (chemical c, reaction r') pair, where chemical c affects the propensity function $a_{r'}$ of reaction r'. Loop through the reactions again, looking at which chemicals each reaction affects: for each (reaction r, chemical c) pair, where reaction r affects chemical c, add to the dependency graph all edges (r, r') where r affects c and c affects r'. In particular, (r, c) is stored in Affects(r) for each reaction, and (c, r') is stored in the c entry of the array **ChemicalAffects**.

The formal statement is:

Algorithm 3

```
1. For each chemical $C$
   $\textbf{ChemicalAffects}(C) = \emptyset$.

2. For each reaction $R$
   For each chemical $C \in$ DependsOn($a_R$)
   $\textbf{ChemicalAffects}(C) = \textbf{ChemicalAffects}(C) \cup \{R\}$

3. For each reaction $R$
   For each chemical $C \in$ Affects(R)
   For each reaction R' $\in \textbf{ChemicalAffects}(C)$
   Add edge (R, R') to dependency graph.
```

This algorithm does not examine every edge, so it is not $\mathcal{O}($ (#reactions)$^2)$. Rather, it depends on the number of chemicals affected by or affecting each reaction (typically, these two quantities sum to ≤ 3), and the number of actual edges in the dependency graph. For systems with many reactions but few dependencies, this algorithm is preferable.

5.4 Indexed Priority Queues

Typically, the dependency graph is *sparse*, i.e., that the number of edges from a given vertex is small. It is important to have data structures that are very efficient at handling *a small number* of updates.

The Next Reaction Method deals with two kinds of variables, τ_is and a_is. The latter are easy to handle: the operations required are READ and UPDATE; they can be stored in a simple array. (A purist might not even store them, but rather recalculate them as needed.) The τ_is require the operations: FIND_MINIMUM (in Step 2) and UPDATE (in Step 5d). The former is one of the operations of a priority queue (which is often implemented as a heap), and with a little thought, the other can be implemented in terms of the standard priority queue algorithms ADD_ELEMENT and DELETE_ELEMENT [8]. However, the standard algorithms, although used in some contexts for this speedup [27], are not really what is called for in this context. A better UPDATE, which takes into account the structure of the data, requires an indexing scheme and a separate UPDATE algorithm.

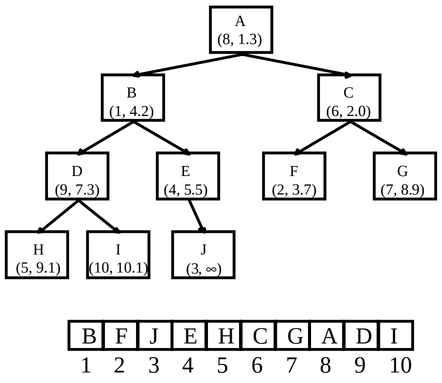


Figure 5-3: Example indexed priority queue. Top: tree structure. The positions in the tree structure are labeled with letters A–J for pedigogical purposes. Bottom: Index structure. Each number has a pointer to the corresponding position in the tree structure; these pointers are illustrated as letters A–J.

Definition 6 *An* indexed priority queue *consists of (a) a tree structure of ordered pairs of the form (i, τ_i), where i is the number of a reaction, and τ_i is the putative time when reaction i occurs, and (b) an index structure whose i-th element is a pointer to the position in the tree that contains (i, τ_i). The tree structure in (a) has the property that each parent has a lower τ_i than either of its children.*

Figure 5-3 shows an example priority queue. Note the following: a) finding the minimum element takes constant time — it is always in the top node, b) the ordering is *only* vertical, not horizontal, c) the number of nodes is precisely the number of reactions r, not twice the number of reactions as in the efficient version of the Direct Method in Chapter 7, d) because of the indexing scheme, it is possible to find any arbitrary reaction in constant time, and e) $\tau_3 = \infty$, which corresponds to reaction 3 never occurring, i.e., $a_3 = 0$. In fact, ∞ is a perfectly legitimate floating point number, so it is possible to implement this feature (in the C programming language, for example) without any major headaches.

There are several algorithms that need to be defined in order to use the priority queue. Most of them are analogous to algorithms for priority queues. In particular, one needs:

- SWAP(i, j), which swaps the tree nodes i and j *and updates the index structure appropriately,*
- BUILD, which takes a tree and an index structure and moves entries until the tree has the property that each parent is less than its children,

- UPDATE(r), which updates a given reaction number.

SWAP is easy to implement. BUILD is completely analogous to the standard heap/priority queue BUILD operation, but uses SWAP so as to keep the index structure correct. UPDATE is non-standard and deserves comment.

Algorithm 4 *UPDATE(node **n**, value **new_value**)*

```
Change value of n to new_value
UPDATE_AUX( n )
```

Algorithm 5 *UPDATE_AUX(node **n**)*

```
    If value(n) < value( parent( n ) )
        SWAP n and parent( n )
        Update_aux( parent( n ) )
    Else If value( n ) > minimum value( children( n ) )
        SWAP n and minimum child( n )
        Update_aux( minimum child( n ) )
    Else
        Return
```

An example is called for.

Example 21 *Suppose the value of τ_1 changes from 4.2 to 16. Looking in the index array, τ_1 is stored in node B. In the tree structure, one changes node B's τ value to 16. Calling UPDATE_AUX on node B, one executes the 'Else If' statement and swaps the ordered pairs in nodes B and E and the corresponding indices (1 and 4) in the array. Calling UPDATE_AUX recursively on node E, one notes that the new value of 16 is in the correct position ($5.5 < 16 < \infty$), so the final 'Else' clause is executed, and the algorithm stops with: ordered pair (4, 5.5) in node B, (1, 16) in node E, index 'E' in position 1 of the array, and index 'B' at position 4. The rest of the structure remains unchanged.*

The converse case, where the new value is less than the old value, is completely analogous.

One way to implement UPDATE is simply to delete the offending node, and insert a new node with the same reaction number but a different time value. This takes something like $2 \log r$ operations. Our approach, on the other hand, changes the node in place, then bubbles it up or down the tree structure until the priority property is re-established. This evidently takes $\log r$, but has the advantage that if there are a small number of reactions that have fast rate constants compared to the others, say there are r' such reactions, most of the updates will involve those, and take $\log r'$ time, because once the algorithm reaches a node that is already in the right spot, it does not continue further. So, for example, if some of the reactions are "disabled" or "not possible" in the given state, and have $a = 0$ and $\tau = \infty$, they will not slow down the computation. This effect can be significant, for example, the chemotaxis system of [32] contains a large number of reactions which will not be "active" at any given time. Because of these inactive reactions, [32] avoided the standard Gillespie algorithm (i.e., the Direct Method), which grows with the number of reactions, and instead developed one that is not exact, but scales with number of molecules, since the number of molecules is much less than the number of possible reactions. Note that our algorithm is not only exact, but also scales with the *logarithm* of the number of "*active*" reactions. Chapter 8 gives some performance numbers.

5.5 Statement of Algorithm and Timing Analysis

Algorithm 6 *(Exact Stochastic Simulation — Next Reaction Method)*

1. *Initialize:*
   - (a) *Set initial numbers of molecules, set $t \leftarrow 0$, generate a dependency graph $\mathcal{G}$.*
   - (b) *Calculate the propensity function, a_i, for all i.*
   - (c) *For each i, generate a putative time, τ_i, according to an exponential distribution with parameter a_i.*
   - (d) *Store the τ_i values in an indexed priority queue $\mathcal{P}$.*
2. *Let μ be the reaction whose putative time, τ_μ, stored in $\mathcal{P}$, is least.*
3. *Let τ be τ_μ.*
4. *Change the number of molecules to reflect execution of reaction μ. Set $t \leftarrow \tau$.*
5. *For each edge (μ, α) in the dependency graph $\mathcal{G}$,*

(a) Update a_α.

(b) If $\alpha \neq \mu$, set

$$\tau_\alpha \leftarrow (a_{\alpha,old}/a_{\alpha,new})(\tau_\alpha - t) + t \tag{5.1}$$

(c) If $\alpha = \mu$, generate a random number, ρ, according to an exponential distribution with parameter a_μ, and set $\tau_\alpha \leftarrow \rho + t$ (see note[1])

(d) Replace the old τ_α value in $\mathcal{P}$ with the new value.

6. Go to Step 2.

Consider the time used by the algorithm. Step 1 of the Next Reaction Method is only executed once; Steps 2-6 are executed once for each simulation event. Steps 3, 4, and 6 do not depend on the number of reactions, r. Step 2 does not either, because of the properties of indexed priority queues. Step 5 is executed once for every edge (μ, α) in $\mathcal{G}$; suppose there are k such edges, where k is typically much less than r. Step 5a, executed k times, depends on the number of reactants for each (elementary) reaction, so it should take no more than 3 multiplications (as was explained in the introduction). Step 5b, executed $k-1$ times, requires an addition, a subtraction, a multiplication and a division. Step 5c, executed 1 time, requires a call to the random number generator, which can be very slow compared to the other operations discussed (a simple test on our system indicates that a single call to the random number generator takes 10 times as long as a division). Step 5d, executed k times, requires *at most* $2\log(r)$ operations, although it may effectively take far fewer (see the discussion in Section 5.4). (Throughout this paper, log means logarithm base 2, as per the typical computer science usage.)

The total number of operations per iteration is at most $c_{2,3,4,5a,6} + c_{5b}(k-1) + c_{5c} + c_{5d}(k)(2\log(r))$, where each c is a machine specific constant. From a computer science perspective, this is $\mathcal{O}(\log(r))$, i.e., for very large r, only the last term will matter. From a more practical perspective, for r of 50 or 100, the other terms, particularly c_{5c}, may not be negligible. Let us be very clear on this point: the Next Reaction Method words even if k is

[1] *It may happen that $a_\alpha = 0$ for some $\alpha \neq \mu$, in which case Step 5b is incomplete. As long as $a_\alpha = 0$, τ_α should be set to ∞. Let t_1 be the time at which a_α first becomes 0, let t_2 be the time at which a_α ceases to be 0, let $a_{\alpha,old}$ be the last pre-0 propensity, and let $a_{\alpha,new}$ be the first post-0 propensity. Then the correct transformation for Step 5b is $\tau_\alpha \leftarrow (a_{\alpha,old}/a_{\alpha,new})(\tau_\alpha - t_1) + t_2$.*

large, but will achieve more of a speedup if k is small, relative to the number of reactions. (An equivalent way of saying the same thing is 'if the dependency graph is *sparse*.')

Lukkien *et al.* [27] discuss ways to improve the Direct Method and the First Reaction Method (a more detailed treatment can be found in Segers [36]). Their improved First Reaction Method, which we shall call the Absolute Time First Reaction Method, consists of switching from relative to absolute times and using a standard priority queue. They conclude that for time-invariant processes, the Direct Method is preferable to the Absolute Time First Reaction Method for two reasons, which do not apply to the Next Reaction Method. First, in their domain, in which position is important, it is difficult to do the indexing necessary to implement the efficient update algorithm (Algorithm 4 of Section 5.4); specifically, the time-consuming part of their problem is not the priority queue, but rather maintaining the data structures that store position-dependent information, which is irrelevant in the present position-independent context. Second, the Absolute Time First Reaction Method generates too many random numbers. Because typical computer pseudo-random number generators cycle with some regularity, using too many random numbers will quickly exhaust the abilities of the generator, and should be avoided with extreme prejudice. (Also, from a purely practical standpoint, generating random numbers is relatively slow.)

Amazingly, the Next Reaction Method uses just a single random number per iteration. Clearly, the optimum would be *exactly* one random number per iteration. Our algorithm is slightly sub-optimal, in that the initialization step will generate r extra random numbers, and at the end of the algorithm, r random numbers will be left over. As the number of iterations increases, this initialization effect becomes negligible in comparison. The only new random number generated, τ_μ, corresponds to the reaction that was just executed and is generated in Step 5c. It is clear that reaction μ requires a new random number, since the value of the old random number has been used explicitly, thus reducing it to a sure variable. Section 5.6 will show that it is correct to do the other manipulations in 5, so as not to regenerate any other random numbers.

For this reason we assert that the Next Reaction Method is superior to the Direct Method.

5.6 Reusing τ_is

This section will demonstrate that the Next Reaction Method, with its switch from relative to absolute times and all of the strangeness in Step 5, is equivalent to the First Reaction Method. This demonstration, necessary to show that the algorithm works and why it works, is somewhat more mathematical than the rest of the paper. The reader whose primary interest is implementing the Next Reaction Method may skip ahead with impunity. As mentioned before, it is usually not legitimate to reuse random numbers; this section will prove that it is permissible in this special case.

In what follows, T_i will denote the *random variable* corresponding to the i-th reaction, and τ_i, a number, will denote a *sample* from that random variable.

One of the differences between the First Reaction Method and the Next Reaction Method is that the former uses relative times, while the latter uses absolute times. This should not be a stumbling block or a source of confusion. Suppose that during the n-th iteration of the First Reaction Method, the random variables are denoted R_α, for $1 \leq \alpha \leq$ (number of reactions). Then $R_\alpha =$Exp(a_α), and the density of R_α is given by $P_{R_\alpha}(\tau) = \theta(\tau) a_\alpha \exp(-a_\alpha \tau)$. ($\theta(\tau)$, the Heaviside function, is 0 for $\tau' < 0$ and 1 for $\tau' \geq 0$.) The corresponding absolute time is given by the random variable $T_\alpha = R_\alpha + t_n$, the sum of the relative time and the variable t during the n-th iteration. (The n-th iteration ends, and the $n+1$-st begins, when t is updated in Step 5.) What is the density of T_α? By the random variable transformation (RVT) theorem [18],

$$P_{T_\alpha}(\tau) = \int_{-\infty}^{\infty} P_{R_\alpha}(\tau')\delta(\tau - [\tau' + t_n])d\tau' = P_{R_\alpha}(\tau - t_n) = \theta(\tau - t_n) a_\alpha \exp(-a_\alpha(\tau - t_n)),$$

or, equivalently,

$$\Pr(T_{\alpha,n} > u) = \begin{cases} \exp(-a_{\alpha,n}(u - t_n)) & \text{if } u > t_n \\ 1 & \text{otherwise} \end{cases} \tag{5.2}$$

Clearly, an absolute-time version of the First Reaction Method with no other changes would be entirely equivalent to the original relative-time version.

Now we turn our attention to the Next Reaction Method. After Step 1, the random variables follow the distribution in Eq 5.2; t_0 was set to 0 in Step 1a. The real core of the Next Reaction Method is that each subsequent iteration maintains Eq 5.2.

At the risk of being overly mathematical, we state the key property that allows the Next Reaction Method as a theorem:

Theorem 2 *Assume that Eq 5.2 holds at the beginning of Step 2. Then, before Step 5 of the n-th iteration, for all $i \neq \mu$, τ_i is distributed according to*

$$\Pr(T_i > u) = \begin{cases} \exp(-a_{i,n}(u - t_{n+1})) & \textit{if } u > t_{n+1} \\ 1 & \textit{otherwise} \end{cases} \tag{5.3}$$

Proof. By assumption, before Step 2 of the n-th iteration, τ_i is distributed according to Eq 5.2. Steps 2 and 3 identify the least τ, namely τ_μ. The act of identification reduces uncertainty. In particular, T_μ becomes the sure variable τ_μ, and all of the other τ_is must be larger than τ_μ. Hence each of the other T_is is distributed according to $\Pr(T_i > u | T_i > \tau_\mu)$. By definition, this is $\Pr((T_i > u) \text{ AND } (T_i > \tau_\mu)) / \Pr(T_i > \tau_\mu)$. There are two cases. Case 1) for $u > \tau_\mu$, the numerator simplifies to $\Pr(T_i > u)$, and the resulting division is $\exp(-a_{i,n}(u - t_n)) / \exp(-a_{i,n}(\tau_\mu - t_n)) = \exp(-a_{i,n}(u - \tau_\mu))$. Case 2) $u \leq \tau_\mu$, and the numerator simplifies to $\Pr(T_i > \tau_\mu)$. In this case, the numerator cancels the denominator, leaving 1. In Step 4, t_{n+1} is set to τ (which was set to τ_μ in Step 3), so the theorem holds. ■

Showing that Eq 5.2 is maintained is just a matter of collecting the details:

- For those $i \neq \mu$ whose a_i remains constant from the n-th to $n+1$-st iteration, $a_{i,n+1} = a_{i,n}$, so Eq 5.3 is equivalent to Eq 5.2. There is no need to change these τ_is in Step 5. In fact, reactions whose a_i does not change are not in the dependency graph, so the τ_is are not changed.

- For those $i \neq \mu$ whose a_i *does* change, τ_i is now distributed according to Eq 5.3. Simply plugging in to the RVT theorem shows that the random variable T_i', constructed by $\tau_i' = (a_{i,n}/a_{i,n+1})(\tau_i - t_{n+1}) + t_{n+1}$, is distributed according to Eq 5.2[2].

[2]Similarly, plugging into the RVT theorem shows directly that the formula for the case $a_i = 0$ produces

What is the intuition behind this transformation? By the theorem, τ_i is distributed according to Eq 5.3. Going back to relative times, $\tau_i - t_{n+1}$ is distributed according to Exp($a_{i,n}$). It can be shown (by the RVT, for example) that $(a_{i,n}/a_{i,n+1})$Exp($a_{i,n}$) = Exp($a_{i,n+1}$). Returning to absolute times gives the transformation.

This transformation is applied to all the appropriate is in Step 5b.

- Finally, for $i = \mu$, it is necessary to generate a new random number. Note that the theorem only holds for $i \neq \mu$. For $i = \mu$ the variable T_μ was reduced to a sure variable in Step 3, so a new random variable is needed. That new value is supplied in Step 5c.

One key point has been overlooked thus far: the First Reaction Method requires *statistically independent* random numbers. To complete the correctness argument amounts to showing that the manipulations done by the Next Reaction Method do not introduce any statistical dependencies.

At each step in the algorithm $T_i = f_i(R_i)$; each random variable in the Next Reaction Method is a transformed version of the corresponding random variable in the First Reaction Method, and there are no cross dependencies. By the RVT theorem,

$$\begin{aligned} P_{T_1 \cdots T_N}(\tau_1, \ldots \tau_N) &= \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} \left\{ \prod_{i=1}^{N} \theta(r_i) \exp(-a_i r_i) \right\} \prod_{j=1}^{N} \delta\left(\tau_j - f_j(r_j)\right) dr_1 \cdots dr_N \\ &= \prod_{i=1}^{N} \left\{ \int_{-\infty}^{\infty} \theta(r_i) \exp(-a_i r_i) \delta\left(\tau_i - f_i(r_i)\right) dr_i \right\} \end{aligned}$$

The “product form” of this joint distribution function tells us that since the *original* variables R_i were statistically independent, then the *transformed* variables T_i are as well.

In summary, Step 1 sets up the T_i according to Eq 5.2. Each subsequent iteration maintains that distribution, without introducing any statistical dependencies between the random variables. Thus the Next Reaction Method is equivalent to the First Reaction Method, and in turn to the Direct Method and the Master Equation approach.

the correct new distribution for T_i'. We shall outline a more intuitive argument, but not make it entirely rigorous: Intuitively, one can multiply the original transformation by $a_{i,n+1}$, giving $a_{i,n+1}\tau_i' = a_{i,n}\tau_i - a_{i,n}t_{n+1} + a_{i,n+1}t_{n+1}$. One can view this as a transformation $\varsigma' = \varsigma - a_{i,n}t_{n+1} + a_{i,n+1}t_{n+1}$ of a new random variable ς, which is well-defined even if $a = 0$. Applying the $\varsigma \rightarrow \varsigma'$ transformation for each time that $a_{i,n+1} = 0$ and the first non-zero $a_{i,n+1}$, and applying $\varsigma = a_{i,n}\tau_i$ to the last pre-zero a_i and the first post-zero a_i, gives the desired transformation.

Chapter 6 Extensions of the NRM

6.1 Summary

The Next Reaction Method presented in the previous chapter assumed that the probability of a reaction μ occurring in a little bit of time dt (a) is given by $a_\mu \times dt$, where a_μ is a constant, and (b) depends only on the current state, not on the previous state or states of the system. It can be shown [19] that many reasonable chemical systems have these properties.

This chapter will show how to deal with systems in which (a) and (b) do not hold. In particular, it first relaxes assumption (a) by letting a_μ be a function time (as is necessary to model systems whose rate "constants" change, due to changing temperature, volume, etc.), and second it relaxes assumption (b), showing how to deal with non-Markov processes. Even though *elementary* reactions in the stochastic framework are Markov (i.e., have property (b)), it is sometimes useful to group consecutive steps to form a composite process. The full model of that process is, of course, still Markov, but if one is only interested in a subset of the variables, the resulting mathematical process is not guaranteed to be Markov.

Computationally, it is amazingly simple to do the extension: one simply replaces the exponential random number generator with some other random number generator. Much mathematical machinery is required to show that it is legitimate to extend the NRM in this way.

6.2 Time-Dependent Markov Processes

Consider a system in which the probability of a reaction μ occurring in a little bit of time dt is given by $a_\mu \times dt$, *but* a_μ *is a function of time.* For now, assume that the transition probabilities do not depend on history, but only on the current state. For example, in Table 5.1, the first reaction has $a_1 = k_1 \times (\#A) \times (\#B)$. The rate constant k_1 is a function of temperature and of volume. In an engineering system, one typically affects the rate constants by heating or cooling the reaction. In a biological system, cell growth changes the volume. Either of these mechanisms, or others, might change rate constants as a function of time, which requires a modification to the algorithms presented.

In place of the simple exponential distribution, the putative times are distributed [18] according to

$$P_\mu(\tau|S,t_n) = a_\mu(S,\tau)\exp\left(-\int_{t_n}^{\tau} a_\mu(S,t)dt\right) \tag{6.1}$$

Notice two things: first, it is not easy in general to find a closed form solution of Eq 6.1 for arbitrary functions of time a_μ, and second, for non-constant a_μ, the resulting answer will not be a simple exponential distribution, and hence will not have the temporal homogeneity property that $\Pr(T_i > u|T_i > \tau_\mu) = \Pr(T_i > u - \tau_\mu)$. As a consequence of the latter, it will not, in general, be possible to let $t_0 = 0$, so absolute times should be used.

6.2.1 How to do it: Next Reaction Method, Markov Processes

To extend the Next Reaction Method to arbitrary Markov processes, one simply changes Step 3 to generate τ_i according to the new process[1]. This has two advantages over the time variant version of the Direct Method in the appendix:

- Since one considers each reaction separately, the computation is easier, and may be analytically solvable for some processes (e.g., in the example of the next section). The Direct Method, which considers all reactions at once, involves a sum within the integral in Eq 6.1; the exact form is given in the appendix.

- Conditioning. Since the Next Reaction Method stores τ_is and not just a_is, it does not have to recondition on each iteration. Specifically, after executing reaction μ it does not need to regenerate τ_i according to $\Pr(T_i > u|T_i > \tau_\mu)$: the fact that the algorithm chose to execute reaction μ implies $T_i > \tau_\mu$. Therefore, T_i is already distributed according to the correct distribution, for all $i \neq \mu$ whose a_i has not changed.

[1]For time varying processes, there may be a non-zero probability that a given reaction does not occur at all (in other words, integrating Eq. (9) from t_n to ∞ may result in some $p < 1$). In this case, one must choose random numbers in such a way that $P_\mu(\infty|S,t_n) = 1 - p$. This is easy to do in the Next Reaction Method: one generates a uniform random number r; if $r > p$, the reaction never occurs — so $\tau = \infty$, otherwise one transforms r (or uses some other method) to find $\tau < \infty$.

We now show an example of how to generalize the Next Reaction Method for time varying Markov processes, then consider the problem of reusing random numbers with the generalization.

Example 22 *(Changing Volume)*

Reaction 1 in Table 3.3 has propensity $k \times (\#A) \times (\#B)$. For second order reactions, such as this one, the k term depends on the volume, and should be replaced with $k'/V(t)$, where $V(t)$ is the volume and k' is independent of volume. This leads to the propensity $a'/V(t)$, where $a' = k' \times (\#A) \times (\#B)$ is a constant independent of volume (and hence time). For simple $V(t)$, Eq 6.1 can be solved analytically, for example, in Arkin et al., the volume is modeled as increasing linearly. Thus $V(t) = V_0 + ct$, which leads (by a simple integration) to the distribution

$$P(t|t_0) = \frac{a' \left(V(t_0) + ct\right)^{-a'/c-1}}{V(t_0)^{-a'/c}} \tag{6.2}$$

Note also that in the limit as c goes to zero, this distribution reduces to an exponential with parameter a'/V_0, as expected.

It is a straightforward operation to generate random numbers according to this distribution, using the inversion generating method [18, 26]: one calculates the cumulative distribution function F (a simple integral of P), takes a sample U from a uniform random number generator, and the variable $F^{-1}(U)$ has the correct distribution. For the preceding example, the variable $R = V(t_0)[U^{-c/a'} - 1]/c$ is distributed according to Eq 6.2.

6.2.2 Generating Fewer Random Numbers

It was remarkably simple in the time-independent, exponential case to reuse the same random numbers. The extension is somewhat more difficult. The method presented here works for reusing random variables generated by the inversion generating method. Of course, not every random variable is generated that way, and in practice it may be hard to reuse other random numbers.

Theorem 3 *Let τ be a random number generated according to an arbitrary distribution with parameter a_n and distribution $F_{a,n}$. Suppose the current simulation time is t_n, and* *the new parameter (after the update in Step 2) is a_{n+1}. Then the transformation*

$$\tau' = F^{-1}_{a,n+1}([F_{a,n}(\tau) - F_{a,n}(t_n)]/[1 - F_{a,n}(t_n)]) \tag{6.3}$$

generates a new random variable from the correct (new) distribution.

Before proving this theorem, here are some examples:

Example 23 *For exponentials,*

$$F_{a,n}(u) = \Pr(T_n \le u) = \begin{cases} 1 - \exp(-a_n(u - t_{n-1})) & \textit{if } u > t_{n-1} \\ 0 & \textit{otherwise} \end{cases}$$

and

$$F^{-1}_{a,n}(U) = \begin{cases} -\ln(1-U)/a_n + t_{n-1} & \textit{if } 0 \le U \le 1 \\ \textit{undefined} & \textit{otherwise} \end{cases}$$

So, by the theorem, $\tau' =$

$$\begin{aligned} & F^{-1}_{a,n+1}([F_{a,n}(\tau) - F_{a,n}(t_n)]/[1 - F_{a,n}(t_n)]) \\ = \ & \frac{-1}{a_{n+1}} \ln(1 - \frac{[1 - \exp(-a_n(\tau - t_{n-1}))] - [1 - \exp(-a_n(t_n - t_{n-1}))]}{1 - [1 - \exp(-a_n(t_n - t_{n-1}))]}) + t_n \\ = \ & (a_n/a_{n+1})(\tau - t_n) + t_n \end{aligned}$$

This is the transformation used by the Next Reaction Method.

Example 24 *The previous section considered a process with $V(t) = V(t_{n-1}) + c(t - t_{n-1})$,*

$$F_{a,n}(u) = 1 - \left(\frac{V(t)}{V(t_{n-1})}\right)^{-a_n/c}$$

and

$$F^{-1}_{a,n}(U) = V(t_{n-1})[(1-U)^{-c/a_n} - 1]/c + t_{n-1}$$

By the theorem, one can reuse random numbers by the transformation

$$\begin{aligned}
\tau' &= F^{-1}_{a,n+1}([F_{a,n}(\tau) - F_{a,n}(t_n)]/[1 - F_{a,n}(t_n)]) \\
&= \tfrac{V(t_n)}{c} \left[\left(1 - \frac{\left(1 - \left(\frac{V(\tau)}{V(t_{n-1})}\right)^{-a_n/c}\right) - \left(1 - \left(\frac{V(t_n)}{V(t_{n-1})}\right)^{-a_n/c}\right)}{\left(\frac{V(t_n)}{V(t_{n-1})}\right)^{-a_n/c}} \right)^{-c/a_{N+1}} - 1 \right] + t_n \\
&= \frac{V(t_n)}{c} \left[\left(\frac{V(\tau)}{V(t_n)} \right)^{a_n/a_{N+1}} - 1 \right] + t_n
\end{aligned}$$

In some cases, as in the examples above, it is possible to calculate a closed form solution of the equations that is relatively simple, so this method is practical. In general, it may not be at all practical, and it may be easier to generate fresh random numbers.

Proof. (Theorem 3) The random number τ is originally distributed according to distribution $F_{a,n}$, with density $P_{a,n}$. After Step 6, it is distributed according to $P'_{a,n} = \Pr_{a,n}(T = u|T > t_n)$, which is equal to

$$P'_{a,n}(u) = \begin{cases} P_{a,n}(u)/[1 - F_{a,n}(t_n)] & \text{if } u > t_n \\ 0 & \text{otherwise} \end{cases}$$

By the Random Variable Transform Theorem [18], the random variable $Y = [F_{a,n}(\tau) - F_{a,n}(t_n)]/[1 - F_{a,n}(t_n)]$ has the density

$$\begin{aligned}
Q(y) &= \int_{-\infty}^{\infty} P'_{a,n}(u) \delta \left(y - \frac{F_{a,n}(u) - F_{a,n}(t_n)}{1 - F_{a,n}(t_n)} \right) du \\
&= \frac{1}{1 - F_{a,n}(t_n)} \int_{t_n}^{\infty} P_{a,n}(u) \delta \left(y - \frac{F_{a,n}(u) - F_{a,n}(t_n)}{1 - F_{a,n}(t_n)} \right) du \\
&= \frac{1}{1 - F_{a,n}(t_n)} \int_{F_{a,n}(t_n)}^{F_{a,n}(\infty)=1} \delta \left(y - \frac{v - F_{a,n}(t_n)}{1 - F_{a,n}(t_n)} \right) dv \\
&= \int_0^1 \delta (y - w)\, dw \\
&= \begin{cases} 1 & 0 < y \le 1 \\ 0 & \text{otherwise} \end{cases}
\end{aligned}$$

The second line is just the definition of $P'_{a,n}(u)$. The third line comes from the transformation $v = F_{a,n}(u)$, $dv = \frac{dF_{a,n}(u)}{du} du = P_{a,n}(u) du$. The fourth line comes from the

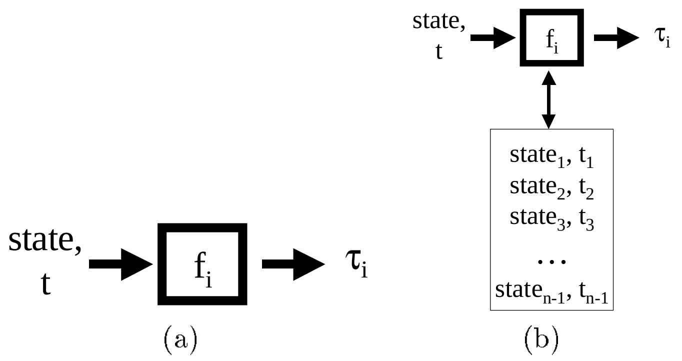


Figure 6-1: Procedures, f, for generating random numbers. (a) A Markov process, in which no history is stored. (b) A non-Markov process, which requires storing history.

transformation $w = \frac{v - F_{a,n}(t_n)}{1 - F_{a,n}(t_n)}$, $dw = \frac{1}{1 - F_{a,n}(t_n)} dv$. The final line comes from the definition of the delta function.

Hence, the random variable Y is distributed uniformly on $(0, 1]$. Finally, the inverse generation method works by transforming a uniform random number U and to $F^{-1}(U)$. Here Y is such a uniform random number, which proves the theorem. ■

6.3 Non-Markov Processes

Even though elementary reactions are Markov (i.e., do not depend on history) in the stochastic framework, it is sometimes useful to deal with non-Markov processes. For example, one may model a system using a certain set of variables for which the system is Markov, or one may use a smaller set for which the system is not. Provided the simulation algorithm still works for non-Markov processes, a smaller number of variables may be significantly faster to simulate.

For Markov processes, one generates τ_i directly from the state and the time, as in Figure 6-1a. For example, in the time independent case, the value a_i is calculated from the state, and τ_i is the sum of t and a random variable with exponential distribution and parameter a_i. Notice that 1) f_i is a *random* function, i.e., calling it multiple times with the same parameters will give multiple answers, and 2) f_i is a *function* in the mathematical usage (or in the computer science sense of functional programming), i.e., it does not contain any internal state. For non-Markov processes, as in Figure 6-1b, the distribution of τ_i depends on the history of (possibly all) states of the system from the initial time to the present. Hence one must use a *procedure* (in the computer science sense of procedural programming), which can store previous values of the system state and time.

In general, non-Markov processes are very difficult to deal with [18]. The distribution of next states, or of transition times to the next state, may depend on the entire history of the system. Fortunately, the sort of non-Markov processes that occur in chemical reaction simulations have some nice properties that make them easier to deal with. First, the complete history of the system is uniquely determined by the series of *discrete* transitions and transition times. Given the transitions and transition times, the state at any time t is the same as the state at the last transition before t. This is an enormous simplification: since continuous transitions are not possible, one can hope to store the entire state history. Second, one may not even need the entire history. For any given reaction μ, one only needs to store that fraction of the history that affects (in the dependency graph sense) the reaction μ. For systems with many reactions, that leads to another significant reduction in the amount of storage.

For *arbitrary* non-Markov chemical reaction models, even this reduction in storage may not be enough. It may be very difficult to generate τ_i given the appropriate subset of history. In that case, it may be preferable to include the full gamut of variables so as to make the system Markov. In those special cases where it *is* possible to generate τ_i, one may achieve a substantial performance improvement. An example follows.

6.3.1 Example: Gamma Distribution

Consider the set of equations:

$$\begin{aligned}
A + B &\xrightarrow{k_1} S_0 \\
S_0 &\xrightarrow{k} S_1 \\
S_1 &\xrightarrow{k} S_2 \\
&\dots \\
S_{n-1} &\xrightarrow{k} S_n \\
S_n &\xrightarrow{k_2} C + D
\end{aligned}$$

Systems of equations very much like this comes up in the Arkin *et al.* [3] model of lambda phage, both for transcription and for translation. Physically, this means that at some time a molecule of type S_0 is produced, it undergoes an n-step process, and then the resulting molecule, S_n, affects the rest of the system.

The first and last equations are different, but all the n intervening equations are identical. Assuming time-independence (as is the case in the model; first order reactions are not affected by change in volume) one may solve these n equations analytically. Rather than n exponentials, the combined waiting time is a gamma distribution. Specifically, consider a single molecule of S_0, produced at t_0, with no other molecules of S_0 produced. Then:

$$\begin{aligned} & \Pr\left(\text{one molecule of } S_n \text{ is produced between } t \text{ and } t+dt \mid \text{one molecule of } S_0,\, t_0\right) \\ = \ & \frac{k[k(t-t_0)]^{n-1}}{(n-1)!}\exp[-k(t-t_0)] \times dt \end{aligned} \tag{6.4}$$

This is simply a gamma distribution, and there are efficient ways to generate random numbers according to this distribution.

Now consider several molecules undergoing this process. This composite system can be described by ordered pairs of the form (molecule identity, state). There are two ways to simplify this: grouping by state and grouping by molecule identity. Thus far, the grouping has always been by state, i.e., the number of molecules in state S_0, the number in state molecules of S_1, etc.

For the context in which this problem occurs, with many more states than molecules, one achieves a smaller system by grouping by molecule identity; by Eq 6.4, one can simplify the n-step exponential process into a 1-step gamma process, thus the number of processes to consider is equal to the number of distinct molecules, much less than the number of states. (Note that this is possible because the reactions involved are first order.) These two possibilities are schematized in Figure 6-2.

The procedure f_i is as follows: rather than store state and time directly, f_i will keep a list L of processed values τ'. Every time a new molecule of S_0 is produced, f_i generates a τ' value for it according to Eq 6.4 and adds that value to L. The value of τ_i' that f_i returns is simply the minimum of the τ values in L. (The astute reader will note that the operations required on L are insert, delete and minimum, so one could implement L as a priority queue. This implementation of L as a priority queue should not be confused with

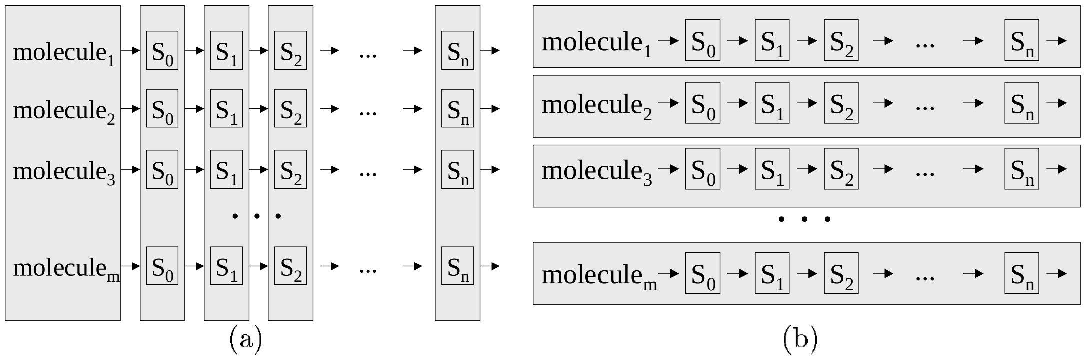


Figure 6-2: Ways of splitting reactions. (a) By state, (b) by molecule.

the indexed priority queue in Section 5.4 of Chapter 5.)

In the current context, the transcription or translation lengths, and hence the n values, may be in the hundreds or even thousands, so this enhancement achieves quite a speedup (see Chapter 8).

Chapter 7 Improving the Direct Method

7.1 Summary

The underlying ideas of the Next Reaction Method — use a dependency graph to update the minimal number of variables, and use an efficient data structure — are applied to the Direct Method to create an efficient version of it. However, the NRM is preferable for two reasons: 1) it is more difficult to extend the Direct Method to time varying and non-Markov processes and 2) the Direct Method requires two random numbers per simulation event.

7.2 Efficient Direct Method

Algorithm 7 *(Efficient Exact Stochastic Simulation — Direct Method)*

Replace Steps 1 and 2 of the Direct Method with:

1. *Initialize (i.e., set initial numbers of molecules, set $t = 0$, generate a dependency graph $\mathcal{G}$).*

2. *Calculate the propensity function, a_i, for the following i:*

   If this is the initial iteration, calculate a_i for all i.

   Otherwise, let μ be the reaction that was just executed. For each edge (μ, α) in the dependency graph $\mathcal{G}$, update a_α.

It is clear from the definition of the dependency graph that this will update only the a_is that need to be updated. The only thing remaining is to use the right data structure to speed up the updates.

To complete the speed up of the Direct Method, one must do Steps 2, 3 and 4 efficiently. One might consider using a simple array to store each of the a_is. In this scheme, updates would be very fast. However, Step 3 of the Direct Method would then take time proportional to the number of reactions. A better data structure, which takes time proportional to the logarithm of the number of reactions, follows.

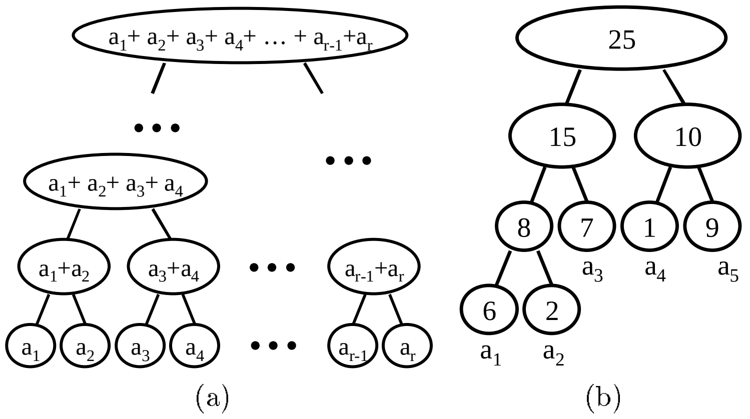


Figure 7-1: Data structure used for a_i's for an efficient version of the direct method. Each leaf contains an a_i value. Each other node contains the sum of its left and right child. (a) generic construction, (b) numerical example used in text.

Store the a_is as the leaves of a complete tree, and store in each non-leaf node the sum of its left child and right child (see Figure 7-1a). Thus, the root will have value $\sum_i a_i$. When a_is change, update 1) those a_is that have changed and 2) their ancestors. Notice that the tree contains r leaves and $r/2 + r/4 + r/8 + ... + 1 \cong r$ non-leaves. The height of the tree is simply $\log 2r = 1 + \log r$. Each update affects one node at each level, hence is $\mathcal{O}(\log r)$.

Generating the random numbers τ will be easy, since the root of the tree contains the appropriate parameter. For the μ value, generate a random number x between 0 and $\sum_i a_i$ (which can be found at the root) and then use the following algorithm, starting at the root:

Algorithm 8 (Efficient Uniform Random Number Generation)

1. If the current node is a leaf, let μ be its index.

2. Otherwise, if $0 \leq x \leq$ (left child value), then call this algorithm recursively on the left child with parameter x.

3. Otherwise, (left child value) $\leq x$, so call this algorithm recursively on the right child with parameter $x -$ (left child value).

The discussion thus far has been somewhat abstract and calls for an example.

Example 25 *Consider the numerical values in Figure 7-1b, the tree structure for* $a_1 = 6$, $a_2 = 2$, $a_3 = 7$, $a_4 = 1$ *and* $a_5 = 9$. *To calculate* τ, *generate an exponential random variable with parameter 25. For the* μ *value, generate a random number* x *between 0 and 25. Suppose*

$x = 15.5$. *Because $x > 15$ (the left child), go to Step 3 of the algorithm, and descend the right subtree (i.e., the one whose root is "10") with parameter $x' = x - 15 = 0.5$. Now, $0 \leq x' \leq 1$, so use Step 2 of the algorithm, and go to the left node, labeled "1." Finally, this is the leaf corresponding to a_4, so stop and let μ be "4."*

One standard method of generating a random number of this distribution is to generate a random number R between 0 and 1, then find the index μ such that $\sum_{i=1}^{\mu-1} a_i \leq R(\sum_i a_i) < \sum_{i=1}^{\mu} a_i$. In fact, Algorithm 8 does precisely that, in an efficient way, and takes time proportional to the height of the tree, not the total number of nodes in the tree.

With this algorithm, an update takes $1+\log r$ operations. By the sparseness assumption, each simulation event (time through the loop) takes at most $k(1 + \log r)$ operations, where k is a constant independent of r. For E simulation events, the algorithm takes $\mathcal{O}(E \log r)$ operations, not counting the initialization in Step 1.

As a side note, there are other efficient ways to generate random variates of a discrete distribution [28], which are somewhat esoteric but have better expected times. One could do a thorough analysis of the trade-off between programming complexity, run time, numerical stability, and number of uniform random numbers required. However, since the Next Reaction Method is more easily enhanced to use fewer uniform random numbers and handle time-varying processes, both Markov and non-Markov, we shall favor it.

7.3 Time Varying Direct Method, Markov Processes

It is well known how to generalize the Direct Method to arbitrary functions of time $a_i(t)$ [18, 24]. One writes an equation that is analogous to Eq 3.4. If S is the state at time t_0, then the equation is:

$$P(\mu, \tau | S, t_0) = a_\mu(S, \tau) \exp\left(- \int_{t_0}^{\tau} \sum_j a_j(S, t) dt \right)$$

At each subsequent step of the algorithm, one must recondition, i.e., change the density $P(\mu, \tau | S, t_i)$ to the density $P(\mu, \tau | S, t_{i+1})$. Doing so works out to changing the lower limit of integration.

It may be hard to generate random numbers according to this distribution for arbitrary functions of time a_i. (Note, in particular, that the lower limit of integration changes each iteration, so methods which involve numerical storage of partial values of the integral will have to do significant recalculation each iteration.) If all the a_is change *in the same way*, then one can use the enhancements of the previous section; if not, it is not immediately clear how to run this algorithm efficiently for many reaction channels. Once again, the Next Reaction Method is preferable.

Chapter 8 Application to Lambda

8.1 Summary

This chapter details how the Next Reaction Method of Chapter 5 can be used to simulate the lambda model of Chapter 4. Section 8.2 describes the basic principles used to run the lambda model through the Next Reaction Method. Sections 8.3 and 8.4 explain technical details of how to implement promoter binding. Section 8.5 presents simulation results.

8.2 The Basics

The reactions in Table 4.2 can be divided into four groups:

1. *Zeroth and first order reactions.* These reactions obey simple exponential probability distributions as detailed in Chapter 5. The reactions that fall into this category are: 4, 6, 7, 10, 11, 12, 22, 24, 25, 30, 31, 32, 51, 53, 54, 55, 56, 58, 59, 61, 62, 64, 65, 67, 68, 70, 71, and 72. The pseudo-first order reactions 35, 36, 37, 38, 39, 40, 42, 44, 46, and 48 also fall into this category.

2. *Second order reactions.* These reactions follow the more complicated time-varying distribution in 6. The reactions that fall into this category are: 5, 23, 50, 52, 57, 60, 63, and 66.

3. *Gamma reactions.* All the transcription and translation reactions with multiple steps can be simplified using the gamma distribution of Chapter 6. (The original Arkin et al. paper does not do this in all cases, so as to model collisions between RNA polymerase molecules traveling in opposite directions across the same DNA. We shall ignore that effect.) The reactions that fall into this category are: 2, 3, 8, 9, 13, 14, 16, 18, 19, 21, 26, 27, 28, 29, 33, 34, 41, 43, 45, 47, and 49.

4. *Promoter-related reactions.* These follow the equilibrium binding model of Section 8.3. The reactions that fall into this category are: 1, 15, 17, and 20.

Chapter 5 explains how to implement category 1 in NRM, Chapter 6 explains how to implement categories 2 and 3. Category 4 is discussed in the next subsection.

8.3 Promoter-related Equations

The promoters are assumed to be in equilibrium, i.e., the binding and unbinding reactions are much faster than the other reactions considered in the simulation. There are several possible methods for dealing with this equilibrium assumption:

1. *Ignore it; write out the kinetics of each reaction, apply the NRM.* Pros: (1) This approach is exact and (2) relatively efficient. Cons: (1) Requires knowledge of reaction rate constants (kinetics data), not just free energies (equilibrium data), and (2) if the equilibrium assumption is correct, these reactions are very fast compared to others, so NRM may spend a disproportionate amount of time on these reactions.

2. *Calculate the average rates, add dependencies based on rate constant dependencies.* For example, k_1 depends on the number of molecules of cI_2, so add an edge in the dependency graph from any reaction that affects cI_2 to Reaction 1. Pros: (1) This simplifies the simulation by using the equilibrium assumption, thus getting rid of the cons of Method 1. Cons: (1) Very computationally intense. Requires an iterative calculation at each promoter whenever one of the concentrations changes. (2) Even then, this approach is not exact, because some terms in the partition functions depend on volume, which is changing. (This would mainly be a problem if the number of molecules of each type that affects a promoter were to remain constant for a long time.)

3. *Use a "Monte Carlo" approach: randomly pick a promoter state every so often at fixed time intervals.* Pros: (1) Much faster than Methods 1 and 2, (2) if the time step is sufficiently small, gets around the volume problem, and (3) does not do any averaging — thus one sees the true stochastic nature of binding and unbinding. Cons: (1) Not exact, because it depends on the value of the time interval Δt. (To integrate this into NRM, one can make a pseudo-reaction that is executed at regular intervals, and whose only effect is to update the state of the promoters and then adjust the promoter-related τs appropriately.)

4. *Various modifications of Method 3: e.g., use a variable time interval or update promoters every second reaction, etc.* Pros: (2) and (3) above. Possibly (1), depending on how cleverly one does the updating. Cons: (1) above: still not exact, no matter how cleverly one uses discrete time jumps.

Remark 6 *There is no rigorous theoretical method to use the equilibrium assumption in stochastic kinetics.*

These different approaches have various interpretations of the equilibrium assumption. In Method 1, one simply ignores it. In Method 2, one assumes that the rate at any given time is equal to the equilibrium rate calculated from statistical mechanics. In Methods 3 and 4, one assumes that the probability of finding the system in a given state is given by the equilibrium distribution. While that is true, it still does not specify how frequently one should sample the state to get statistically valid results.

We have opted for Method 3. Because the NRM is efficient at handling large sets of chemical reactions, calculating the promoter probabilities becomes the most time-consuming part of the simulation using Method 2. (Also, kinetics data is not available, so Method 1 is right out.) We originally set time interval Δt to 0.1s (recall that simulations run for 35 minutes of simulation time), but the simulation was still very slow. A time interval of 1.0s is significantly faster (the time to update promoters is on order of the time to run NRM) and yielded indistinguishable results.

8.4 Issues

There are two big issues with the equilibrium assumption used in modeling promoter binding: correctness and computational efficiency. As mentioned above, there is no theoretical justification for treating certain reactions as having reached equilibrium, yet modeling the kinetics of other reactions. The only rigorously justified approach is Method 1, which is computationally very costly and requires kinetic rate constants that may not be available.

As for computational efficiency, there is one trick we can pull out that helps somewhat. Suppose the system in question has four states:

The partition function of this system is given by

$$Z = 1 + K_2[P] + K_3[P] + K_4[P]^2$$

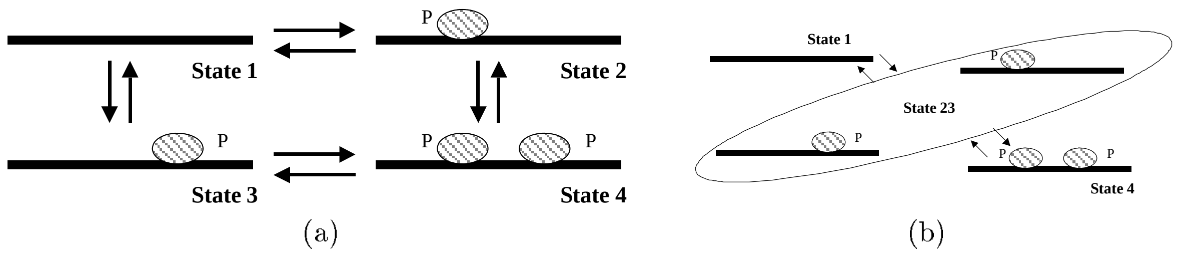


Figure 8-1: (a) Four state system used in example. (b) Simplified three state system.

where $[P]$ is the concentration of P. The probabilities of the four states are given by

$$\begin{aligned} P_1 &= \frac{1}{Z} \\ P_2 &= \frac{K_2[P]}{Z} \\ P_3 &= \frac{K_3[P]}{Z} \\ P_4 &= \frac{K_4[P]^2}{Z} \end{aligned}$$

(See Chapter 2, Section 2.8, for more discussion of partition functions.) The average rate is given by

$$\begin{aligned} \langle rate \rangle &= k_1P_1 + k_2P_2 + k_3P_3 + k_4P_4 \\ &= \frac{k_1 \times 1 + k_2 \times K_2[P] + k_3 \times K_3[P] + k_4 \times K_4[P]^2}{Z} \end{aligned}$$

One can create an equivalent system with three states, by combining State 2 and State 3. Call this new state '23' and let

$$\begin{aligned} K_{23} &= K_2 + K_3 \\ k_{23} &= \frac{k_2K_2 + k_3K_3}{K_{23}} \end{aligned}$$

Theorem 4 *This three state system has the same equilibrium properties as the four state system.*

Proof.

$$\begin{aligned} Z_3 &= 1 + K_{23}[P] + K_4[P]^2 \\ &= 1 + K_2[P] + K_3[P] + K_4[P]^2 \\ &= Z_4 \end{aligned}$$

Further,

$$\begin{aligned} P_1 &= \frac{1}{Z} \\ P_{23} &= \frac{K_{23}[P]}{Z} \\ P_4 &= \frac{K_4[P]^2}{Z} \end{aligned}$$

so

$$\begin{aligned} \langle rate \rangle_3 &= k_1 P_1 + k_{23} P_{23} + k_4 P_4 \\ &= \frac{k_1 \times 1 + k_{23} \times K_{23}[P] + k_4 \times K_4[P]^2}{Z_3} \\ &= \frac{k_1 \times 1 + (k_2 K_2 + k_3 K_3)[P] + k_4 \times K_4[P]^2}{Z_3} \\ &= \frac{k_1 \times 1 + k_2 \times K_2[P] + k_3 \times K_3[P] + k_4 \times K_4[P]^2}{Z_4} \\ &= \langle rate \rangle_4 \end{aligned}$$

■

More generally, this trick can be used to combine any set of states with the same stoichiometry into a new state, and letting

$$K_{new} = \sum K_j$$

and

$$k_{new} = \frac{\sum k_j K_j}{K_{new}}$$

(Here the sum is over all states j that are being combined into the new state.) Note that this trick only works for states with the same stoichiometry.

Theorem 5 *The simplified system has the same equilibrium properties as the original system.*

Proof.

$$\begin{aligned}
Z_{orig} &= \sum_i K_i \times [X_1]^{power(i,1)} \times [X_2]^{power(i,2)} \times \cdots \\
&= \sum_s \sum_j K_j \times [X_1]^{power(j,1)} \times [X_2]^{power(j,2)} \times \cdots \\
&= \sum_s \left([X_1]^{power(s,1)} \times [X_2]^{power(s,2)} \times \cdots \right) \times \left(\sum_j K_j \right) \\
&= \sum_s \left([X_1]^{power(s,1)} \times [X_2]^{power(s,2)} \times \cdots \right) \times K_{new} \\
&= Z_{new}
\end{aligned}$$

The sums are over: all states i (first line); all stoichiometries s and states i with stoichiometry s (subsequent lines). We have used the fact that the powers to which one raises the concentrations depend on the stoichiometry of the state only, not on the state itself. Analogously,

$$\begin{aligned}
\langle rate \rangle_{orig} &= \sum_i k_i P_i \\
&= \frac{\sum_i k_i K_i \times [X_1]^{power(i,1)} \times [X_2]^{power(i,2)} \times \cdots}{Z_{orig}} \\
&= \frac{\sum_s \left([X_1]^{power(s,1)} \times [X_2]^{power(s,2)} \times \cdots \right) \times \left(\sum_j k_j K_j \right)}{Z_{orig}} \\
&= \frac{\sum_s \left([X_1]^{power(s,1)} \times [X_2]^{power(s,2)} \times \cdots \right) \times K_{s,new} \left(\sum_j \frac{k_j K_j}{K_{s,new}} \right)}{Z_{orig}} \\
&= \frac{\sum_s \left([X_1]^{power(s,1)} \times [X_2]^{power(s,2)} \times \cdots \right) \times K_{s,new} \times k_{s,new}}{Z_{new}} \\
&= \sum_s P_{s,new} \times k_{s,new} \\
&= \langle rate \rangle_{new}
\end{aligned}$$

Again, the sums are over: all states i (first two lines); all stoichiometries s and states i with stoichiometry s (subsequent lines) ∎

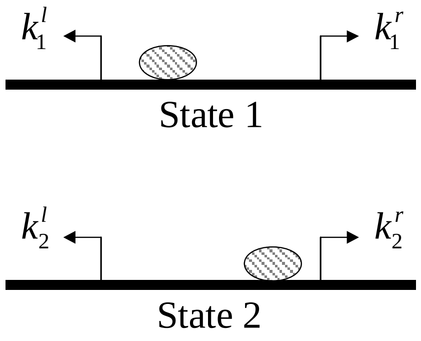


Figure 8-2: Operator with left and right promoters. The simplification presented thus far ignores statistical dependencies between reaction rates for different promoters at the same operator.

Using this technique, the promoter $\mathrm{P_L}$ (in Table 4.5) can be simplified from 10 states to 7, and $\mathrm{O_R}$ (in Table 4.6) can be simplified from 40 states to 17.

This trick, in its current form, applies to Method 2. The simplification may give different results for Method 3 for operators with multiple promoters:

Method 3 keeps a statistical dependence between the two rates, whereas Method 2 averages out that statistical dependence. Depending on how one interprets the equilibrium assumption, it may be legitimate to average out. (Here the interpretation would be "the reactions in question are so much faster than the other reactions in the simulation that an observer cannot disambiguate between individual states, and the statistical dependencies of individual states average out.") If one is worried about this problem, one can combine only those states with identical stoichiometries and having all rate constants the same. In this mode, $\mathrm{P_L}$ can still be simplified from 10 states to 7, and $\mathrm{O_R}$ can be simplified from 40 states to 22, not the 17 above.

Remark 7 *This technique only works for reactions at equilibrium; the kinetics of state change are very different if there is one state than if there are multiple states. Also, the original K_is and k_is depend on temperature, so this simplification will not work if temperature is changing. (In other words, the combined K_{new}s will not change in the same way with temperature that the original K_is and k_is do.) Changing volume is okay, though, since volume is subsumed in the stoichiometric part, not the equilibrium or rate constants.*

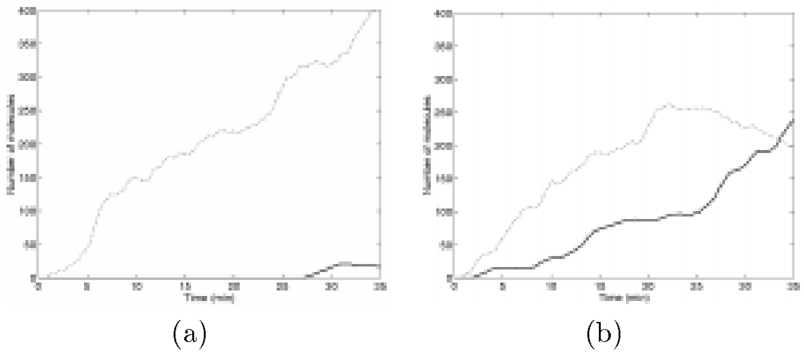


Figure 8-3: Total amount of cI (dotted line) and cro (solid line) present in two runs at MOI 3. (a) Run ending in lysis, (b) run ending in lysogeny.

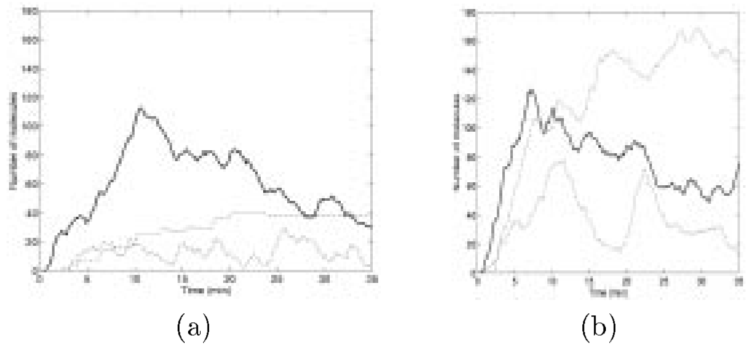


Figure 8-4: Total amount of N (solid line), cII (dotted line), and cIII (dotted-dashed line) present in the same two runs at MOI 3. (a) Run ending in lysis, (b) run ending in lysogeny.

8.5 Results, Times, etc.

8.5.1 Basic Results

Figure 8-3 shows the concentration of protein dimers cI_2 and cro_2 for two typical trajectories. In Figure 8-3a, the concentration of cro_2 increases and the concentration of cI_2 remains very low — this possibility leads to lysis. The converse case, shown in Figure 8-3b, leads to lysogeny. Figure 8-4 shows the concentrations of other proteins — N, cII, and cIII — for these two cases.

Note that these trajectories are generated from the same rate constants and the same initial conditions. The only difference between trajectories is the specific set of random numbers chosen.

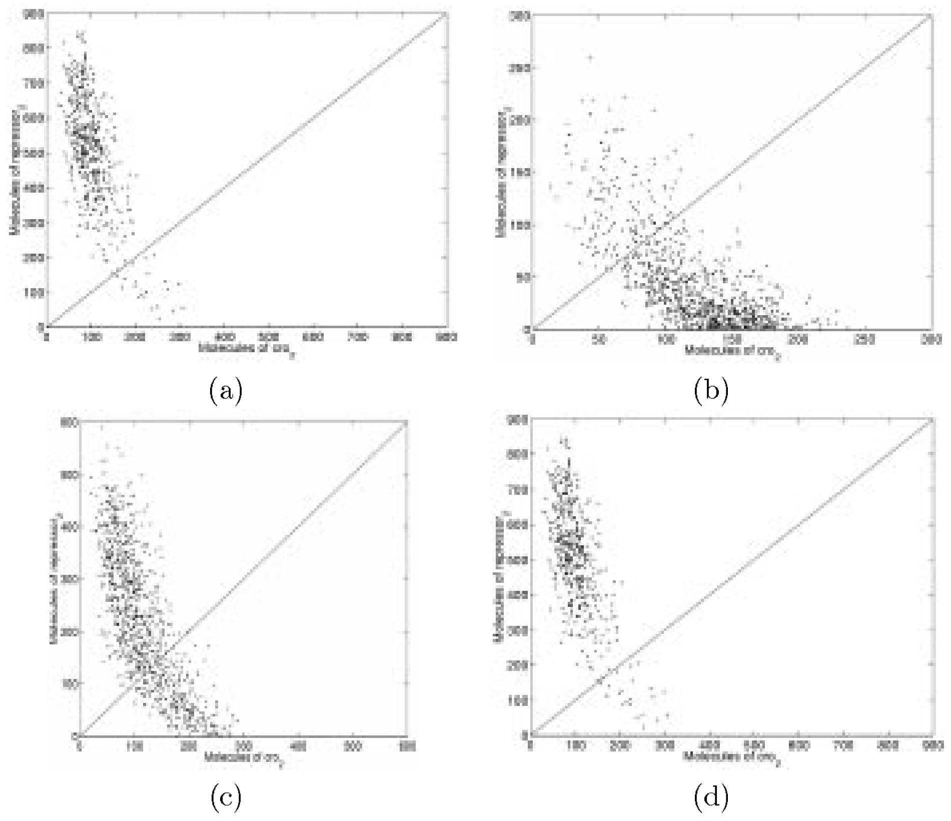


Figure 8-5: Final amount of cI_2 and cro_2 for MOI (a) 1, (b) 3, (c) 6, and (d) 10.

8.5.2 Lambda Fates

An *E. coli* host cell infected with more lambda phages has a higher probability of lysogeny than a host cell with fewer phages. Recalling that lysis or lysogeny is determined largely by the final concentrations of cI_2 and cro_2, we can plot these concentrations as a function of Multiplicity of Infection (MOI). Figure 8-5a shows MOI=1, 8-5b shows MOI=3, 8-5c shows MOI=6, and 8-5d shows MOI=10. (For reference, the line shows equal amounts of cI_2 and cro_2.) Note the statistical nature of the simulations: if the system were deterministic, there would be a single point per graph.

Note that as MOI increases, the probability of lysis decreases and the probability of lysogeny increases.

Figure 8-6 shows the probability of lysogeny (calculated as the fraction of trajectories where, at time = 35 minutes, the amount of cI_2 exceeded cro_2) as a function of MOI.

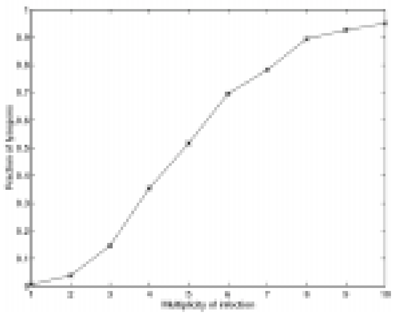


Figure 8-6: Fraction of lysogens as a function of MOI.

8.5.3 Time

The simulations done all ended at time = 35 minutes; thus the number of simulation events depends on how quickly events are occurring. For the parameter values used in the original Arkin et al. paper, the NRM took approximately n seconds per trajectory at MOI $= n$ on a Pentium 400. Thus, running 500 trajectories each for MOIs 1 to 10 takes

$$500 \times 1 + 500 \times 2 + \ldots + 500 \times 10 = 500 \times 55 = 27,500 \text{ sec} = 7.6 \text{ hours}$$

We were running on a cluster of ten such machines, so the actual time was about 45 minutes. For other parameter sets (not the values used in Arkin et al.), as in Chapter 10, many more simulation events occurred, and hence the simulation was much slower.

The original lambda work of Arkin et al. was done on a 200-node supercomputer. (Note that to be statistically correct, one must run different numbers of trajectories to estimate each probability to a fixed percent error, so the calculation above is just an estimate.) With the algorithmic improvements here, it is possible to run the model on a desktop computer (at the time a high-end desktop computer) overnight, or on a cluster of a few computers on the order of an hour.

Part IV

Sensitivity Analysis

Chapter 9 The Multiple Next Reaction Method

9.1 Summary

This chapter presents the Multiple Next Reaction Method (MNRM), an algorithm for simultaneously creating trajectories corresponding to different parameter sets (e.g., rate constants). In particular, MNRM will outperform NRM when run on parameter sets which are very similar. MNRM can be used by itself to gauge the sensitivity of a model to its parameters, or as a building block for the more sophisticated sensitivity analysis of the next chapter.

9.2 Introduction

Consider a set of reactions, such as

$$\begin{aligned} A+B &\xrightarrow{k_1} C \\ B+C &\xrightarrow{k_2} D \\ D+E &\xrightarrow{k_3} E+F \\ F &\xrightarrow{k_4} D+G \\ E+G &\xrightarrow{k_5} A \end{aligned}$$

or the reactions for lambda in Chapter 4. These reactions consist of stoichiometric information (one A and one B react to form one C) and also rate/propensity information (k_1). Stoichiometry is relatively straightforward — it consists of small integers. Stoichiometric measurements done *in vitro* (in a test tube) are likely to result in the same reactions as the corresponding measurements done *in vivo* (in a cell). The rate/propensity information is much more subtle. The (real-number valued) constant k depends on many factors: volume, temperature, salt concentration, etc. For that reason, rate/propensity measurements done *in vitro* will likely differ from ones done *in vivo.*

How does this affect modeling and simulation?

A modeler must be aware that rate constants, typically measured *in vitro* or sometimes guessed, will not be exact. For that reason, it is critical to understand the behavior of a model whose rate constants are uncertain. Recent theoretical [5] and experimental [2] work has suggested that some biological systems are robust to uncertainty in components, i.e., that changes in k values or steady state concentrations of certain proteins will not affect the qualitative behavior of the system. (In other words, the system is *robust* to such uncertainty.) Rather than the system working *because* of the exact value of its parameters, the system works *despite* the exact value of its parameters; system structure, not fine tuning of parameters, leads to the observed behavior. It seems likely that this design principle, robustness, will play a role in many biological systems. (In the Barkai & Leibler system, Yi et al. [40] have suggested that the particular structural consideration or "pathway motif" is integral feedback, which occurs in Barkai and Leibler's model, and is theoretically necessary and sufficient for deterministic systems to have perfect adaptation at steady state, as is observed in the system. This theoretical result does not hold as-is for the stochastic framework, as will be discussed in Chapter 13.)

As a first step toward considering robustness to uncertainty in rate constants, it is useful to see how trajectories and measurable variables of interest change as rate constants change. This chapter presents a first cut algorithm that can efficiently run several simultaneous simulations, each with a different parameter set. This algorithm, MNRM, is most efficient if the parameter sets are similar, differing only in a small number of parameters. So MNRM by itself is ideal for simulating some "nominal" or "base" parameter set, $\overrightarrow{k}$, and parameter sets $\overrightarrow{k^p}$ most of whose components are identical to the corresponding $\overrightarrow{k}$ components. The next chapter will build on this, using MNRM as a tool for more sophisticated analysis, and will use $\overrightarrow{k^p}$s that differ from $\overrightarrow{k}$ in precisely one parameter.

9.3 Fundamentals

In what follows, it will be important to specify not only which reaction is meant, but also which parameter set.

Notation 2 *Let subscripts correspond to reactions and superscripts to parameter sets. For example, x_r^p would be the quantity x corresponding to reaction r and parameter set p.*

Parameters will be specified k_r^p, the reaction constant k corresponding to reaction r in parameter set p. A parameter set $\overrightarrow{k^p}$ is a vector of parameter values: $\overrightarrow{k^p} = (k_1^p, k_2^p, \ldots, k_{\max r}^p)$. We acknowledge a special parameter set, the *main* parameter set, $\overrightarrow{k^{main}}$, which is the nominal or base parameter set, and all other parameter sets differ little from it.

Example 26 *Consider a set of 10 reactions. Then $\overrightarrow{k^{main}}$ is a 10-vector, and k_1^{main} is the k parameter of Reaction 1 for the main parameter set, etc. Suppose we define a new parameter set $\overrightarrow{k^1}$ by*

$$k_r^1 = \begin{cases} 10 \times k_r^{main} & \textit{if } r = 1 \\ k_r^{main} & \textit{otherwise} \end{cases}$$

Another parameter set, $\overrightarrow{k^2}$, could be defined analogously by replacing the "$r = 1$" condition with "$r = 2$" and so forth.

The problem that MNRM addresses is

Problem 1 *Simulate a system with each of the parameter sets $\overrightarrow{k^{main}}$, $\overrightarrow{k^1}$, $\overrightarrow{k^2}$, ..., $\overrightarrow{k^{pmax}}$.*

A simple way to solve Problem 1 is:

Algorithm 9 *For $p \in \{main, 1, 2, \ldots, pmax\}$,*
Simulate the system corresponding to $\overrightarrow{k^p}$ using NRM.

The complexity of this algorithm is

$$\begin{aligned} & E_{main} \log r + E_1 \log r + \ldots + E_{pmax} \log r \\ = & \left(\sum_{p \in \{main, 1, 2, \ldots, pmax\}} E_p \right) \log r \\ = & E_{total} \log r \end{aligned}$$

where E_p is the number of simulation events in the simulation corresponding to parameter set p, and r is the total number of reactions.

One way to accomplish this simulation is to store the following data for each p:

- t^p, a real number representing the time in the pth simulation

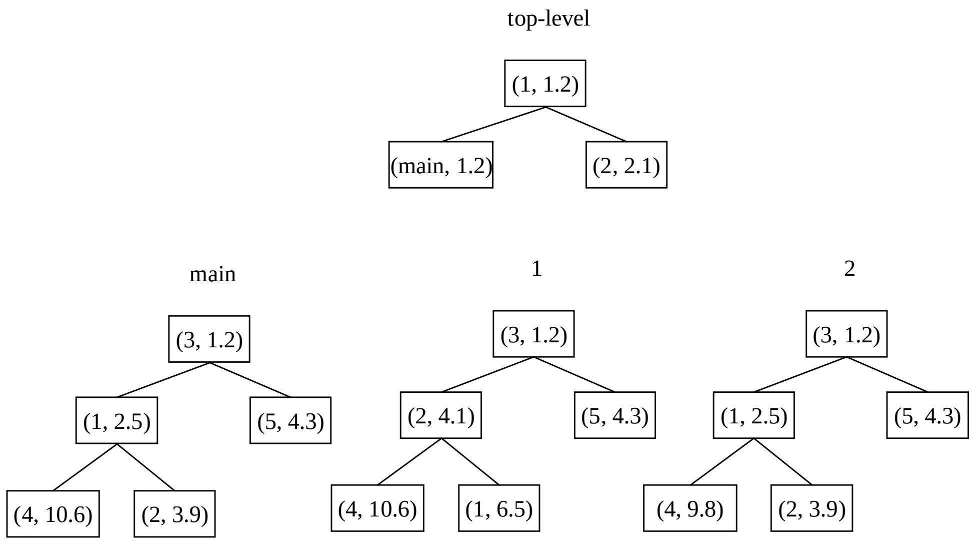


Figure 9-1: Priority queues used by Algorithm 10: one top-level and one for each parameter set.

- $(\#X)^p$, an array with one element for each chemical species, representing the state in the pth simulation
- $\mathcal{P}^p$, a priority queue consisting of the τ_r^p values for the pth simulation
- μ^p, τ^p, α^p, working variables for the simulation.

Also, one needs $\mathcal{G}$, the dependency graph of the system. Only a single copy of $\mathcal{G}$ is needed, as it is the same for all parameter sets.

Note that Algorithm 9 does not really require p copies of all these things, because it runs simulations *in serial.* Subsequent algorithms will run simulations *in parallel,* and hence will require separate copies.

As a baby step toward MNRM, consider the following way to solve Problem 1: Store a single t, and a single copy of the working variables, but multiple priority queues $\mathcal{P}^p$ and multiple chemical numbers $(\#X)^p$.

Algorithm 10 *(Top level)*

1. *Create a priority queue $\mathcal{P}^{top-level}$, which stores (p, τ) pairs, where $p \in \{main, 1, 2, \ldots, pmax\}$ and τ is a time. Like all other priority queues used in NRM and MNRM, $\mathcal{P}^{top-level}$ is ordered by τ with smaller values having higher priority.*
2. *Run simulations for $\overrightarrow{k^{main}}$, $\overrightarrow{k^1}$, $\overrightarrow{k^2}$, ..., $\overrightarrow{k^{pmax}}$ in parallel in the following way:*

(a) *For each p, do the initialization step of NRM.*

(b) *Store $(p, \tau^p_{\min})$ in $\mathcal{P}^{top-level}$, where $\tau^p_{\min}$ is the minimum time in $\mathcal{P}^p$.*

(c) *Pick the minimum element $(p*, \tau^{p*}_{\min})$ of $\mathcal{P}^{top-level}$.*

(d) *Do one NRM step in simulation $p*$, including updating τ^{p*}_r as necessary.*

(e) *Set $t \longleftarrow \tau^{p*}_{\min}$.*

(f) *Update the $(p*, \tau^{p*}_{\min})$ entry of $\mathcal{P}^{top-level}$ with the updated $\tau^{p*}_{\min}$,* which may correspond to a different reaction *in simulation $p*$.*

(g) *Go to Step c and repeat, as long as $t < t_{\max}$.*

This is similar to Algorithm 9. The difference is that we now ask the questions "What is the next reaction that occurs in any parameter set? When does it occur? In which parameter set?" repeatedly.

It is not hard to show that Algorithm 10 is equivalent to running Algorithm 9 and sorting the generated events by time, rather than by parameter set.

It turns out that there is no advantage to doing this. In fact there is a disadvantage. Specifically, it still takes $E_{total} \log r$ operations to accomplish the simulations, but it now takes an additional $E_{total} \log pmax$ operations to put the E_{total} events into the correct temporal order, so the total number of operations is $E_{total}(\log r + \log pmax)$.

Another possible algorithm is:

Algorithm 11 *Store (p, r, τ^p_r) triples in a single priority queue, $\mathcal{P}$.*

In analogy with NRM:

1. *Choose the minimum triple, $(p*, r*, \tau^{p*}_{r*})$.*

2. *Execute reaction $r*$ in parameter set $p*$ (i.e., update $(\#X)^{p*}$ according to execution of $r*$).*

3. *Update certain $(p*, \alpha, \tau^{p*}_{\alpha})$ based on the dependency graph $\mathcal{G}$.*

The priority queue $\mathcal{P}$ in this algorithm is size $r \times pmax$, so the resulting algorithm requires $E_{total} \log(r \times pmax) = E_{total}(\log r + \log pmax)$ operations, the same as the previous algorithm. This algorithm also suffers the problem of ordering all the times across reactions.

It is not immediately clear whether it would be simpler to implement Algorithm 10 or Algorithm 11. In any case, Algorithm 9 is better, so it is time to introduce a new idea.

Use the same random numbers for the simulation $p = main$, the simulation $p = 1$, etc. Then, in Algorithm 10 or Algorithm 11, it will frequently be the case that $(r, \tau_r^{p1}) = (r, \tau_r^{p2})$, i.e., that the τs chosen for the same reaction in different parameter sets will be the same. The main idea of MNRM is not to do too much calculation in this case.

In particular, we decompose E_{total} into E_{unique} and E_{copy}, where E_{unique} counts the first time a particular (r, τ_r^{p1}) pair is executed, and E_{copy} counts executions of the same pair in different parameter sets. Of course $E_{total} = E_{unique} + E_{copy}$. Then:

- The simple algorithm, Algorithm 9, requires $E_{total} \log r$ operations.
- Algorithms 10 and 11 require $E_{total}(\log r + \log pmax)$ operations.
- MNRM requires $E_{unique}(\log r + \log pmax) + E_{copy}$ operations.

Note that there is no log term in the final summand of the MNRM line.

Example 27 *(Good case) Suppose we have r reactions and p parameter sets, and nearly all the events are the same across parameter sets. (For example, if the parameter sets differ in parameters that do not affect the system much.) Then $E_{total} \approx pE_{main}$, $E_{unique} \approx E_{main}$, and $E_{copy} \approx (p-1)E_{main}$. So the simple algorithm takes $pE_{main} \log r$ reactions, while MNRM takes $E_{main}(\log r + \log p) + (p-1)E_{main}$. The savings incurred by using MNRM is thus $E_{main}\left[(p-1)(\log r - 1) - \log p\right] \approx E_{main} p \log r$ operations.*

Example 28 *(Bad case) Suppose the numbers of events E_s are about the same for each s, but the events themselves are nearly always different. (For example, if the parameter set is very different, or if the trajectories from similar parameter sets diverge quickly.) Then $E_{total} \approx pE_{main}$, $E_{unique} \approx pE_{main}$, and $E_{copy} \approx 0$. So the simple algorithm takes $pE_{main} \log r$ reactions, while MNRM takes $pE_{main}(\log r + \log p)$, which is higher than the number of operations in the simple algorithm by $pE_{main}(\log p)$.*

The basic savings in MNRM comes from simplifying the processing for identical reactions across different parameter sets. The problematic cost is due to running the p simulations in lock-step, i.e., ordering all the events from different parameter sets relative to each other.

Section 9.4 details MNRM. Section 9.5 proves its correctness and explains where the algorithm comes from. Section 9.6 extends it to time-varying and non-Markov processes. Section 9.7 analyzes the time needed to run MNRM, and presents two techniques to lower or eliminate the $\log p$ cost for very different or divergent parameter sets.

9.4 Details of MNRM

The key idea of MNRM is to use the same random numbers for each simulation, as far as that is practical. (In those cases where the cost of using the same random numbers outweighs the benefit gained, MNRM will use distinct random numbers.) Imagine applying this basic idea to Algorithm 10. The priority queues in Algorithm 10 store certain data, in particular, ignoring for now $\mathcal{P}^{top-level}$, the stored data can be described as:

- For each parameter set p, for each reaction r, store the pair (r, τ_r^p) in $\mathcal{P}^p$, where τ_r^p is a random number valid as a time for reaction r in parameter set p.

By using the same random numbers across parameter sets, it will now be the case that many τ_rs, for the same reaction but different parameter sets, are equal. (See Figure 9-1.) In particular, it will frequently be the case that $\tau_r^{p1} = \tau_r^{p2} = \tau_r^{main}$. (Recall the assumption that *main* is somehow the base or nominal parameter set, about which all others are centered.) Assuming the parameter sets are similar and the system is not too sensitive to the differences in parameters, a lot of redundant information is stored. MNRM does away with (some of) that redundancy by using a different concept of what to store:

- For each r, store (r, τ_r^{main}, C) in $\mathcal{P}^{main}$, where τ_r^{main} is a random number valid as a time for reaction r in parameter set *main*, and C is a collection $\{p_1, p_2, \dots\}$ of parameter sets for which $\tau_r^{main} = \tau_r^{p1} = \tau_r^{p2}$, etc.

Example 29 *The entry* $(12, 18.3, \{main, 1, 2, 5\})$ *in* $\mathcal{P}^{main}$ *stores data for reaction 12, namely* $\tau_{12}^{main} = \tau_{12}^1 = \tau_{12}^2 = \tau_{12}^5 = 18.3$. *Equivalently,* $\tau_{12}^{\{main,1,2,5\}} = 18.3$.

- For each parameter set $p \neq main$, and certain (but not all) reactions r, store the pair (r, τ_r^p) in $\mathcal{P}^p$, where τ_r^p is a random number valid as a time for reaction r in parameter set p, and the (r, τ_r^{main}, C) entry in $\mathcal{P}^{main}$ is such that $p \notin C$. The "certain" reactions r are those where a) $\tau_r^p \neq \tau_r^{main}$ or b) it is convenient, as described in Section 9.5.

In other words, each τ_r^p will be stored in exactly one place, either in the (r, τ_r^{main}, C) entry of $\mathcal{P}^{main}$ (in which case $p \in C$) or in the (r, τ_r^p) entry of $\mathcal{P}^p$. We have been deliberately vague about why a certain entry would be stored in $\mathcal{P}^{main}$ versus in $\mathcal{P}^p$; the exact specification is somewhat tedious:

1. For all r, the value τ_r^{main} will always be stored in $\mathcal{P}^{main}$.
2. For all r, and for all $p \neq main$, if $a_r^p \neq a_r^{main}$, then $\tau_r^p \neq \tau_r^{main}$, so τ_r^p will be stored in $\mathcal{P}^p$. Specifically,
   (a) If $k_r^p \neq k_r^{main}$, then $a_r^p \neq a_r^{main}$, so Condition (2) will hold.
   (b) Suppose $a_r = k_r \times (\#X_1) \times (\#X_2)$. If $(\#X_1)_r^p \neq (\#X_1)_r^{main}$ (for example), then $a_r^p \neq a_r^{main}$, so Condition (2) will hold.
3. For all r, and for all $p \neq main$, if at some point in the past a_r^p was not equal to a_r^{main}, so τ_r^p was stored in $\mathcal{P}^p$. Even if changes in system state cause a_r^p to regain equality with a_r^{main}, the value τ_r^p will still be stored in $\mathcal{P}^p$. (The rationale behind this is that it would take more time and effort to check this condition than would be gained by moving τ_r^p back to $\mathcal{P}^{main}$.)

We are now in a position to outline MNRM. Along the lines of Algorithm 10, it creates a top-level priority queue $\mathcal{P}^{top-level}$ with (p, τ) pairs, and additional priority queues $\mathcal{P}^1, \mathcal{P}^2, \ldots, \mathcal{P}^{\max}$ with (r, τ) pairs, and $\mathcal{P}^{main}$ with (r, τ, C) triples. What data is where was described above, and is illustrated in Figure 9-2. MNRM also keeps separate $(\#X)^p$ arrays. Each iteration picks the minimum element of $\mathcal{P}^{top-level}$, executes the corresponding reaction, and updates the τs and priority queues appropriately.

Algorithm 12 *(MNRM)*

1. *Create a priority queue $\mathcal{P}^{top-level}$, which stores (p, τ) pairs, where $p \in \{main, 1, 2, \ldots, pmax\}$ and τ is a time. Like all other priority queues used in this chapter, $\mathcal{P}^{top-level}$ is ordered by τ with smaller values having higher priority.*
2. *Run simulations for $\overrightarrow{k^{main}}$, $\overrightarrow{k^1}$, $\overrightarrow{k^2}$, ..., $\overrightarrow{k^{pmax}}$ in parallel in the following way:*
   (a) MNRM-Initialize

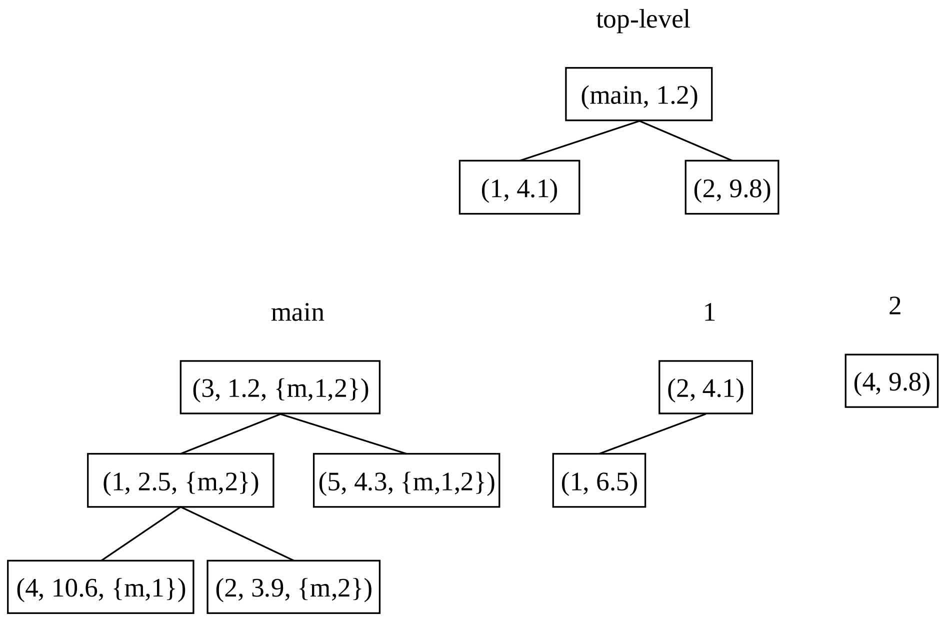


Figure 9-2: Priority queues used by MNRM: one top-level and one for each parameter set.

(b) *Store* $(p, \tau^p_{\min})$ *in* $\mathcal{P}^{top-level}$*, where* $\tau^p_{\min}$ *is the minimum time in* $\mathcal{P}^p$*.*

(c) *Pick the minimum element* $(p*, \tau^{p*}_{\min})$ *of* $\mathcal{P}^{top-level}$*.*

(d) *MNRM-Execute*

(e) *Set* $t \longleftarrow \tau^{p*}_{\min}$*.*

(f) *MNRM-Update*

(g) *Go to Step c and repeat, as long as* $t < t_{\max}$*.*

Figure 9-2 shows samples of the priority queues used by MNRM. Compare them with the corresponding priority queues for Algorithm 10, in Figure 9-1. Algorithm 10 stores a lot of redundant information, and all priority queues are the same size, namely r. In MNRM, the redundancy is eliminated by the collections C added to the *main* priority queue. As a result, $\mathcal{P}^1$, $\mathcal{P}^2$, etc., may be much smaller than the corresponding priority queues in Algorithm 10. ($\mathcal{P}^{main}$ is, of course, still size r.)

The data stored by Algorithm 10 and MNRM is different, so the operations that manipulate the data structures must also be different. The MNRM-Initialize, MNRM-Execute and MNRM-Update steps of MNRM differ from the corresponding steps of Algorithm 10. We discuss each of these three in turn.

9.4.1 Initialization

Algorithm 13 (MNRM-Initialize)

```
1. $\mathbf{t} = 0$
2. Set initial number of molecules (for each parameter set).
3. For each reaction $r$:
    (a) Calculate initial $a_r^{main}$, i.e., propensity for reaction $r$ in parameter set main.
    (b) Generate initial $\tau_r^{main}$, i.e., putative time for reaction $r$ in parameter set main.
    (c) Put the triple $(r, \tau_r^{main}, \{main, 1, 2, \ldots, n\})$ into $\mathcal{P}^{main}$.
4. For each parameter set $\overrightarrow{k^p}$ $(p \neq main)$:
   For each reaction $r$:
   If $k_r^{main} \neq k_r^p$
    (a) Calculate initial $a_r^p$, i.e., propensity for reaction $r$ in parameter set $p$.
    (b) Generate initial $\tau_r^p$, i.e., putative time for reaction $r$ in parameter set $p$.
    (c) Put the pair $(r, \tau_r^p)$ into $\mathcal{P}^p$.
    (d) Remove $p$ from the collection $C$ in the $(r, \tau_r^{main}, C)$ entry of $\mathcal{P}^{main}$.
5. For each parameter set $\overrightarrow{k^p}$ (including $p = main$), put $(p, \tau_{min}^p)$ into $\mathcal{P}^{top-level}$.
6. Generate dependency graph $\mathcal{G}$.
```

The formal proof of correctness is in the next section, but the present section will outline why MNRM-Initialize works: for all r, it puts $(r, \tau_r^{main}, \{main, 1, 2, \ldots, n\})$ into $\mathcal{P}^{main}$, then moves those r, p pairs where $a_r^p \neq a_r^{main}$ to $\mathcal{P}^p$. (Specifically, it never moves $p = main$, in accordance with Specification (1), and it will move pairs where $k_r^p \neq k_r^{main}$, according to Specification (2a). Now, at the initial time, $(\#X)^p = (\#X)^{main}$, so Specification (2b) cannot occur, and because $t = 0$, Specification (3) cannot happen either.

	Parameter Set	#A	#B	#C	#D	#E	#F	#G
(a)	*main*	15	12	7	13	4	9	3
	1	15	12	7	13	4	9	3
	2	15	12	7	13	4	9	3
	Parameter Set	#A	#B	#C	#D	#E	#F	#G
(b)	*main*	15	12	7	13	4	9	3
	1	15	11	6	14	4	9	3
	2	15	12	7	13	4	9	3
	Parameter Set	#A	#B	#C	#D	#E	#F	#G
(c)	*main*	15	12	7	12	4	10	3
	1	15	12	7	12	4	10	3
	2	15	12	7	12	4	10	3

Table 9.1: Values of $(\#X)^p$ for $p = main$, 1, and 2. (a) Initial values, before any reactions are executed. (b) Values, starting from initial, after Reaction 2 ($B + C \longrightarrow D$) is executed in parameter set 1. (c) Values, starting from initial, after Reaction 3 ($D + E \longrightarrow E + F$) is executed in all parameter sets.

9.4.2 Execution

The MNRM-Execute algorithm is also straightforward. Looking at Figure 9-2, suppose we execute $\mathcal{P}^1$ entry $(2, 4.1)$. (We would not normally do this, as the top entry of $\mathcal{P}^{main}$ occurs at an earlier time. This example is for illustration purposes only.) Executing this entry means "execute reaction 2 in parameter set 1 at time 4.1," so we change $(\#X)^1$ according to reaction 2. See Table 9.1.

Executing an entry in $\mathcal{P}^{main}$ is more complicated. Suppose, for example, we execute the entry $(3, 1.2, \{main, 1, 2\})$. In this case, the τ value, 1.2, is valid for parameter sets *main*, 1, and 2. We change $(\#X)^{main}$ according to reaction 3, change $(\#X)^1$ according to reaction 3, and change $(\#X)^2$ according to reaction 3. See Table 9.1.

In this example, the collection C contained all the parameter sets. In general, it will not, and the reaction will be executed in some of the parameter sets only, as specified by C.

Algorithm 14 (MNRM-Execute)

1. *If $p \neq main$, execute reaction r at time τ in parameter set p.*

2. *If $p = main$*

```
    (a) Let $C$ be the collection of parameter sets in the $(r, \tau_r^{main}, C)$ entry in $\mathcal{P}^{main}$.
    (b) For each parameter set $\pi \in C$, execute reaction $r$ at time $\tau$ in parameter set $\pi$.
```

9.4.3 Updating

The most complicated part of MNRM is updating τs. First we give the formal statement, then walk through it.

Algorithm 15 (MNRM-Update)

```
1. If $p \neq main$
   For each edge $(r, \alpha)$ in $\mathcal{G}$
    (a) If $(\alpha, \tau_\alpha)$ is in $\mathcal{P}^p$, update $\alpha$ in $\mathcal{P}^p$ in the same manner as NRM.
    (b) Else /* $(\alpha, \tau_\alpha)$ is in $\mathcal{P}^{main}$ */
         i. Remove $p$ from the collection $C$ in the $(\alpha, \tau_\alpha^{main}, C)$ entry of $\mathcal{P}^{main}$.
        ii. Let $\tau_\alpha^p =$ the reused form of $\tau_\alpha^{main}$. Put the pair $(\alpha, \tau_\alpha^p)$ into $\mathcal{P}^p$.
2. If $p = main$
   For each edge $(r, \alpha)$ in $\mathcal{G}$
    (a) For each parameter set $\overrightarrow{k^\pi}$ $(\pi \neq main)$:
         i. Consider the $(r, \tau_r, C_r)$ and $(\alpha, \tau_\alpha, C_\alpha)$ entries of $\mathcal{P}^{main}$.
        ii. If $\pi \in C_r$ AND $\pi \in C_\alpha$, do nothing.
       iii. If $\pi \in C_r$ AND $\pi \notin C_\alpha$, update $\alpha$ in $\mathcal{P}^\pi$ in the same manner as NRM.
        iv. If $\pi \notin C_r$ AND $\pi \in C_\alpha$, remove $\pi$ from the collection $C_\alpha$. Let $\tau_\alpha^\pi = \tau_\alpha^{main}$.
            Put the pair $(\alpha, \tau_\alpha^\pi)$ into $\mathcal{P}^\pi$.
         v. If $\pi \notin C_r$ AND $\pi \notin C_\alpha$, do nothing.
    (b) Let $(\tau_\alpha^{main})' =$ the reused form of $\tau_\alpha^{main}$ (of course, if $\alpha = r$, then $\tau_\alpha^{main}$ must be
        regenerated). Update the triple $(\alpha, (\tau_\alpha^{main})', C_\alpha)$ in $\mathcal{P}^{main}$.
```

Recall the key points about updating in NRM: 1) we need to generate a new random number for the reaction r that was just executed, and 2) for reactions α that depend on r,

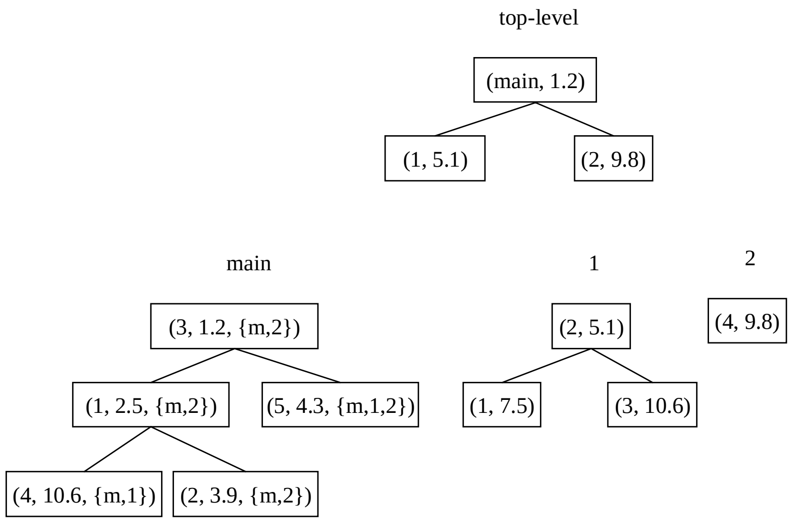


Figure 9-3: Priority queues used by MNRM after the update operation in Example 30.

a_α changes, and so τ_α needs to be transformed (according to Eq 5.1 or Eq 6.3) to regain the correct distribution. MNRM adds a new wrinkle: we need to make sure τ_r and all the τ_αs wind up in the right priority queue(s).

As in the MNRM-Execute algorithm, there are two cases: events that are in $\mathcal{P}^{main}$ and events that are not. The latter are easier, so we consider them first.

Example 30 *Starting with the priority queues in Figure 9-2, suppose we execute the top element of* $\mathcal{P}^1$*, namely* $(2, 4.1)$*. (Note: as mentioned above, we would not typically do this, and this example is only an illustration.) In the MNRM-Execute algorithm, we changed* $(\#X)^p$ *for* $p = 1$ *only, as in Table 9.1. From the dependency graph, Reaction 2 affects reactions 1, 2, and 3. To update* $p = 1, r = 1$*, one runs MNRM-Update Step 1a and transforms* $(1, 6.5)$ *in* $\mathcal{P}^1$ *according to Eq 5.1. Suppose the new value is 7.5. To update* $p = 1, r = 2$*, one also runs MNRM-Update Step 1a. This time, we need to generate a new random number, and update it in* $\mathcal{P}^1$*. Suppose the new random number is 5.3. To update* $p = 1, r = 3$*, one runs MNRM-Update Step 1b. Here the entry is in* $\mathcal{P}^{main}$*, so one not only transform the value 1.2 (say the new value is 10.6) but also removes it from* $\mathcal{P}^{main}$ *and puts it in* $\mathcal{P}^1$*. These manipulations are illustrated in Figure 9-3.*

Example 31 *Starting with the priority queues in Figure 9-2, suppose we execute the top element of* $\mathcal{P}^{main}$*, namely* $(3, 1.2, \{main, 1, 2\})$*. From the dependency graph, we need to update Reactions 3 and 4 in all three parameter sets. Which case in MNRM-Update we*

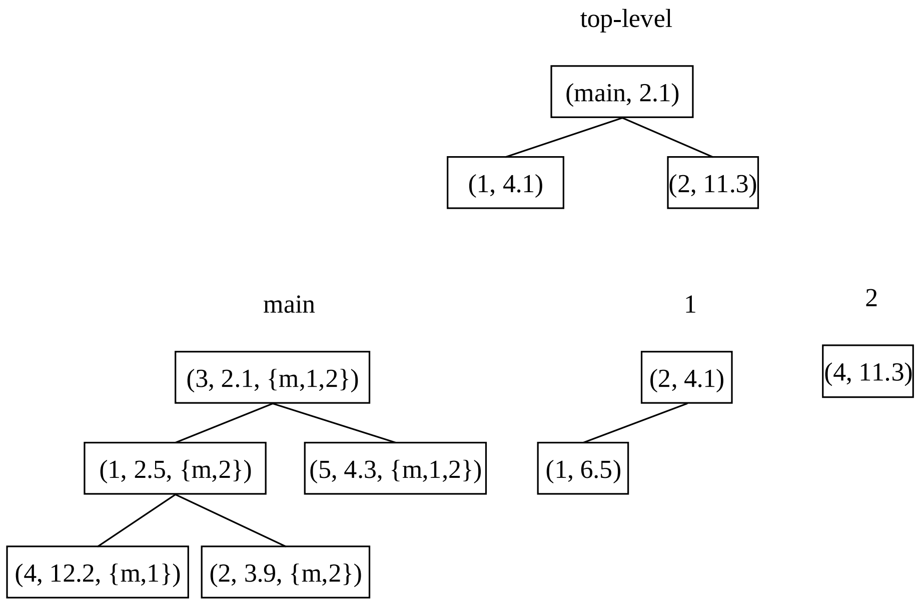


Figure 9-4: Priority queues used by MNRM after the update operation in Example 31.

execute depends on where the target reactions are. For Reaction 3, all three parameter sets use the same τ, stored in $\mathcal{P}^{main}$. Because the τs for those three parameter sets were equal before execution of Reaction 3, they must be equal after its execution (the conditions that assure this will be discussed in the correctness proof of the next section). We executed Reaction 3, so we will have to generate a new τ_3, say 2.1 and update it in $\mathcal{P}^{main}$, according to Steps 2a,ii and 2b. Updating Reaction 4 in parameter sets main and 1 *uses the same step, but only requires a transformation (Eq 5.1), not a new random number. Suppose the transformed value is 12.2. Updating Reaction 4 in parameter set* 2 *involves transforming the* $(4, 9.8)$ *entry of* $\mathcal{P}^2$*, to 11.3. These manipulations are illustrated in Figure 9-4.*

There is one other interesting case, Step 2a, iv. Starting with our original priority queues, suppose we had to update Reaction 3 in parameter sets *main* and 1, but not 2. (Again, this is a contrived example.) Then τ_3^{main} and τ_3^1 would change, and the priority queue would be updated appropriately. However, τ_3^2 would not change, and after the update $\tau_3^2(new) = \tau_3^2(old) = \tau_3^{main}(old) \neq \tau_3^{main}(new)$, so we need to move τ_3^2 to $\mathcal{P}^2$.

From the discussion thus far, Steps 2a and 2b could occur in either order. In fact, for Markov processes, the order does not matter. For non-Markov processes, as in Section 9.6, the order will matter, and they should be executed as written.

9.5 Correctness of MNRM

We shall prove that MNRM is essentially equivalent to Algorithm 10, except that MNRM allows statistical dependencies between random numbers *in the simulations corresponding to different parameter sets.* It is critical that NRM and MNRM not have statistical dependencies between random numbers *in the same parameter set*; however, there is no such restriction *across parameter sets.*

Definition 7 The inputs of the propensity function are the variables that determine the propensity. For the reaction $A+B \xrightarrow{k} C$, the propensity a is given by $a = k \times (\#A) \times (\#B)$, so the inputs are k, $\#A$, and $\#B$. (The "constant" k is variable in the sense that it may vary from one parameter set to another.) The function valin(a) will be the values of the inputs, and $valin(a) = valin(b)$ if and only all inputs to a equal the corresponding input to b.

We shall prove the following:

Lemma 1 *In the MNRM, at time t, the contents of the priority queues are:*

1. *The τ value corresponding to reaction r, parameter set p will be in the main branch (i.e., $\mathcal{P}^{main}$ will contain an entry (r, τ_r^{main}, C) and $p \in C$) if and only if $\forall t' \in [0,t]$, $\mathrm{valin}(a_r^p(t')) = \mathrm{valin}(a_r^{main}(t'))$.*
2. *Otherwise, the τ value corresponding to reaction r, parameter set p will be in the (r, τ_r^p) entry of $\mathcal{P}^p$.*
3. *The entry for reaction r in $\mathcal{P}^p (p \in \{1, 2, \ldots, n, main\})$ is associated with a well-defined a_r^p and $\tau_r^p \sim Exp(a_r^p) + t$.*

Lemma 2 *MNRM is equivalent to running n instances of NRM, subject to: the values τ_{r1}^{p1} and τ_{r2}^{p2} are statistically dependent if and only if $r1 = r2$ and ($\forall t' \in [0, t^*]$, $\mathrm{valin}(a_{r1}^{p1}(t')) = \mathrm{valin}(a_{r2}^{p2}(t')) = \mathrm{valin}(a_{r1}^{main}(t'))$), where $t^* \leq t$ and neither reaction $r1$ in $p1$ or reaction $r2$ in $p2$ has occurred in $[t^*, t]$.*

Theorem 6 *MNRM is correct; i.e., $\forall p$, the trajectories generated for the pth parameter set follow the same statistics one would get running NRM on parameter set p. (And* *thus, MNRM is equivalent to applying the chemical master equation approach for multiple parameter sets.)*

Proof. (Lemma 1, Part 1) The proof proceeds by induction on t. Let C_r denote the collection of parameter sets in the (r, τ_r^{main}, C_r) entry of $\mathcal{P}^{main}$. Let $H'(p,r,t)$ be the statement "$p \in C_r$ if and only if $\forall t' \in [0,t]$, $valin(a_r^p(t')) = valin(a_r^{main}(t'))$." The induction hypothesis, $H(t)$, is "$H(p,r,t)$ holds for all reactions r and all parameter sets p."

Base case: $t = 0$.

Let p and r be given. The routine MNRM-Initialize sets up the induction hypothesis. Specifically, it puts the triple $(r, \tau_r, \{main, 1, 2, \ldots, pmax\})$ into $\mathcal{P}^{main}$ in Step 3. Step 4 removes those p from C_r for which $k_r^{main} \neq k_r^p$. Note that this inequality implies inequality of the corresponding *valin* functions. In fact, the implication goes both ways, because the initial number of molecules is the same for each parameter set. So, after MNRM-Initialize, $p \in C_r \Leftrightarrow k_r^{main} = k_r^p \Leftrightarrow valin(a_r^{main}(t=0)) = valin(a_r^p(t=0))$. This establishes $H'(p,r,0)$. As p and r were arbitrary, this establishes $H(0)$.

Induction step: Assume $H(t_{\text{old}})$, and suppose the next reaction occurs at $t_{\text{new}} \geq t_{\text{old}}$ (in other words, no reaction occurs between t_{old} and t_{new}). We shall show that after running the functions MNRM-Execute and MNRM-Update, $H(t_{\text{new}})$ holds. Let p and r be given, and consider the (r, τ, p) triple picked by MNRM-Execute.

Case 1 $p \neq main$

This implies $p \notin C_r(t_{old})$. In this case, we execute MNRM-Execute Step 3, which affects $(\#X)^p$, and MNRM-Update Step 1. The change in $(\#X)^p$ only affects the values stored for the pth parameter set, some of which are stored (by the induction hypothesis) in $\mathcal{P}^p$ and some of which are stored in $\mathcal{P}^{main}$. Thus for all reactions r and all parameter sets $pp \notin \{p, main\}$, $(\#X)^{pp}$ does not change, and so τ_r^{pp} does not change, and so $H(t_{old})$ immediately implies $H'(pp, r, t_{new})$.

Let reaction rr be arbitrary. To establish $H'(p, rr, t_{new})$, consider three cases:

- $(r, rr) \notin \mathcal{G}$

  Then $\mathrm{valin}(a_{rr}^p(t_{old})) = \mathrm{valin}(a_{rr}^p(t_{new}))$. *Because equation r is not executed in the main parameter set, only in the pth parameter set,* $\mathrm{valin}(a_{rr}^{main}(t_{old})) = \mathrm{valin}(a_{rr}^{main}(t_{new}))$. *Thus* $\mathrm{valin}(a_{rr}^p(t_{new})) = \mathrm{valin}(a_{rr}^{main}(t_{new}))$ *if and only if* $\mathrm{valin}(a_{rr}^p(t_{old})) = \mathrm{valin}(a_{rr}^{main}(t_{old}))$.

- $(r, rr) \in \mathcal{G}$, *and* $p \notin C_{rr}(t_{old})$

  Then Step 1a will be executed, and C_{rr} *will not change. So* $p \notin C_{rr}(t_{new})$, *and by the induction hypothesis,* $\neg(\forall t' \in [0, t_{old}], \operatorname{valin}(a_r^p(t')) = \operatorname{valin}(a_r^{main}(t'))$, *which implies* $\neg(\forall t' \in [0, t_{new}], \operatorname{valin}(a_r^p(t')) = \operatorname{valin}(a_r^{main}(t'))$.

- $(r, rr) \in \mathcal{G}$, *and* $p \in C_{rr}(t_{old})$

  Then Step 1b, c, and d will be executed. $(r, rr) \in \mathcal{G}$, *so* $\operatorname{valin}(a_{rr}^p(t_{old})) \neq \operatorname{valin}(a_{rr}^p(t_{new}))$. *Using the fact that* $p \in C_{rr}(t_{old})$ *and the induction hypothesis, we have* $\operatorname{valin}(a_{rr}^p(t_{new})) \neq \operatorname{valin}(a_{rr}^p(t_{old})) = \operatorname{valin}(a_{rr}^{main}(t_{old})) = \operatorname{valin}(a_{rr}^{main}(t_{new}))$. *Step 1c ensures that* $p \notin C_{rr}(t_{new})$.

In the first case, $H'(p, rr, t_{old}) \Rightarrow H'(p, rr, t_{new})$. *In the second and third cases,* $p \notin C_{rr}(t_{new})$ *and* $\neg(\forall t' \in [0, t_{new}], \operatorname{valin}(a_r^p(t')) = \operatorname{valin}(a_r^{main}(t'))$, *so* $H'(p, rr, t_{new})$ *holds in those cases as well. Because all rr fit into one of these cases, this establishes* $H'(p, rr, t_{new})$ *for all rr, subject to* $p \notin C_r(t_{old})$.

Case 2 $p = main$

In this case, we execute MNRM-Execute Step 4, which affects the number of molecules in simulation$_{pp}$ *for all* $pp \in C_r(t_{old})$, *and MNRM-Update Step 2. Let reaction rr and parameter set* $pp \in C_r(t_{old})$ *be given, and consider cases:*

- $pp = main$

Case 3

- *If* $(r, rr) \in \mathcal{G}$, *we execute Step 2b. Whether we do or not, pp remains in* C_{rr}. *By definition,* $H'(main, r, t_{new})$ *holds.*

- $pp \neq main$

  The key here is to see whether a_{rr}^{pp} *should change. If* a_{rr}^{pp} *is in the main branch, this can be tricky — some elements of* C_{rr} *may need to change their* a_{rr}*s, while others may not, in which case one will need to move things around between priority queues. Breaking this up into cases:*

- $(r, rr) \notin \mathcal{G}$

  This is completely analogous to the subcase above in the $p \neq main$ *case.*

- *$(r, rr) \in \mathcal{G}$, and $pp \in C_r(t_{old})$ and $pp \in C_{rr}(t_{old})$*

  The fact that $pp \in C_r(t_{old})$ means we will execute reaction r in parameter set pp (MNRM-Execute Step 4). Because $(r, rr) \in \mathcal{G}$, a_{rr}^{pp} should change. Here is the clever part: by the induction hypothesis, $\forall t' \in [0, t_{old}]$, $\mathrm{valin}(a_{rr}^{main}(t')) = \mathrm{valin}(a_{rr}^{pp}(t'))$; in particular, it is true at $t' = t_{old}$. When we execute r, we change certain inputs. They were the same at t_{old}, and they are changed in the same way (i.e., by the effect of r), so they must still be the same[1]. Hence $\mathrm{valin}(a_{rr}^{main}(t_{new})) = \mathrm{valin}(a_{rr}^{pp}(t_{new}))$, and $H'(pp, rr, t_{new})$ holds.

- *$(r, rr) \in \mathcal{G}$, and $pp \in C_r(t_{old})$ and $pp \notin C_{rr}(t_{old})$*

  Again, the fact that $pp \in C_r(t_{old})$ means we will execute reaction r in parameter set pp (MNRM-Execute Step 4). Because $(r, rr) \in \mathcal{G}$, a_{rr}^{pp} should change. The value τ_{rr}^{pp} corresponding to a_{rr}^{pp} is being stored in $\mathcal{P}^{pp}$ (because $pp \notin C_{rr}(t_{old})$), so it is updated there. Formally, by the induction hypothesis, $\neg(\forall t' \in [0, t_{old}], \mathrm{valin}(a_{rr}^{main}(t')) = \mathrm{valin}(a_{rr}^{pp}(t')))$ implies $\neg(\forall t' \in [0, t_{new}], \mathrm{valin}(a_{rr}^{main}(t')) = \mathrm{valin}(a_{rr}^{pp}(t')))$, and the update in 2aiii keeps $pp \notin C_{rr}(t_{new})$. Thus $H'(pp, rr, t_{new})$ holds.

- *$(r, rr) \in \mathcal{G}$, and $pp \notin C_r(t_{old})$ and $pp \in C_{rr}(t_{old})$*

  This time, we do not execute reaction r in parameter set pp. a_{rr}^{pp} should not change. However, a_{rr}^{main} is going to change at Step 2b. Thus $\mathrm{valin}(a_{rr}^{pp}(t_{new})) = \mathrm{valin}(a_{rr}^{pp}(t_{old})) = \mathrm{valin}(a_{rr}^{main}(t_{old})) \neq \mathrm{valin}(a_{rr}^{main}(t_{new}))$, so rr no longer qualifies to be in $\mathcal{P}^{main}$. This is taken care of in Step 2aiv, after which $pp \notin C_{rr}(t_{new})$.

- *$(r, rr) \in \mathcal{G}$, and $pp \notin C_r(t_{old})$ and $pp \notin C_{rr}(t_{old})$*

  In this case, we do not change: $\mathrm{valin}(a_{rr}^{main}(t_{old}))$, $\mathrm{valin}(a_{rr}^{pp}(t_{new}))$, $C_r(t_{old})$, or $C_{rr}(t_{old})$. Thus $H'(pp, rr, t_{old}) \Rightarrow H'(pp, rr, t_{new})$.

Together, these five cases cover all possibilities. Thus $H'(pp, rr, t_{new})$ holds for all pp and rr, so $H(t_{new})$ holds.

This completes the induction, and hence the proof. ■

[1] *This is why we require equality of input values, not just equality of a_rs. For example, suppose $a_r = k \times (\#A) \times (\#B)$. If $a_r^1 = k \times (1) \times (2) = k \times (2) \times (1) = a_r^2$. the change will not have the same effect, and the a_rs will not be equal after the change.*

Proof. (Lemma 1, Part 2) Every (reaction, parameter set) pair is always contained in one of the priority queues (all the places where such a pair is removed from one priority queue, it is immediately added to another). The previous lemma showed under which conditions (reaction, parameter set) is in $\mathcal{P}^{main}$. Whenever a pair is removed from $\mathcal{P}^{main}$, either in MNRM-Initialize Step 4 or MNRM-Update Step 1c or Step 2aiv, the pair is immediately inserted into $\mathcal{P}^{p}$. This proves Part 2. ■

Proof. (Lemma 1, Part 3) Again, the proof is by induction. The basic idea here is that

$$\tau_r^p \sim Exp(a_r^p) + t \tag{9.1}$$

after the initialization step, then each time we change or do not change a_r^p, in all the cases above, we change or do not change τ_r^p accordingly. More formally,

Base case: Clearly MNRM-Initialize, Steps 1–3 set Eq 9.1 up for all r and τ_r^{main} at $t = 0$. Step 4 sets Eq 9.1 up for all r and p where $\tau_r^p \neq \tau_r^{main}$ at $t = 0$.

Induction step: Suppose each τ_r^p follows Eq 9.1 at the beginning of an iteration. MNRM picks a certain reaction μ in one or more parameter sets and executes it. By direct analogy of the proof for NRM,

- Any reaction not affected by μ may keep the same τ_r^p. In particular, the reactions in MNRM-Update Step 2aiv are not affected by μ, so they fall under this case.

- Any reaction updated in the same priority queue (MNRM-Update Steps 1a, 2aiii, and 2b) uses the standard NRM update, and hence is correct.

- The only troublesome case is MNRM-Update Step 1b, moving priority queues. By the induction hypothesis, $\tau_r^p = \tau_r^{main} \sim Exp(a_r^{main}) + t$ before we choose μ. Also, by Part 1 of the lemma, $a_r^{main} = a_r^p$ before we execute μ. Thus, after μ, the reused form of τ_r^p equals the reused form of τ_r^{main}.

Thus Part 3 of the lemma holds. ■

Another approach would be to separate simulations as soon as the first event occurred that causes trajectories from different parameter sets to diverge. MNRM is more efficient, because conglomeration is done at the reaction level, not the trajectory level: two trajectories that differ by a single reaction will continue to incur savings.

9.6 Time-varying and non-Markov Processes

MNRM can be extended to time-varying and non-Markov processes. Time-varying Markov processes are particularly simple: one replaces the exponential function with some other function, as was done in Chapter 6 for NRM.

Non-Markov processes are more challenging. From the discussion of NRM in Chapter 6, recall that a non-Markov process must store state information, and so the next-time code is a procedure, not a function. Each non-Markov reaction in MNRM must store state *for each parameter set.* However, just as the times for different parameter sets can be combined, so can the stored states. (In fact, the value τ is a special case of state — it is the amount of state stored by the NRM for Markov processes.) When a reaction in a certain parameter set diverges from the main branch, it must copy the stored state from the main branch, and from then on maintain state separate from the main branch.

For the gamma reaction in NRM, we had to store a separate list L with times of events for different molecules undergoing the gamma process. For example, suppose reaction r is a gamma process, which conglomerates the elementary reactions $S_0 \to S_1 \to \cdots \to S_n$, all with a common rate constant k. Whenever a molecule of S_0 is generated, NRM generates a τ' value according to a gamma distribution (Chapter 6, Eq 6.4) and adds that value to L. The value τ_r is simply $\min L$. Here L stores the state of the gamma reaction r. (One may implement L as a priority queue, but whether one does or not is irrelevant for the current discussion.)

To extend non-Markov reactions to MNRM, suppose $(r, \tau_r^{main}, C_r) \in \mathcal{P}^{main}$ and $p \in C_r$. Then one needs to store state L_r^{main} in order to generate τ_r^{main}. Similarly, one needs to store state L_r^p in order to generate τ_r^p. However, as long as $p \in C_r$, one can use $L_r^p = L_r^{main}$ and $\tau_r^p = \tau_r^{main}$. The tricky part is when (and if) p needs to be removed from C_r. To accomplish this removal, one duplicates L_r^{main}; the copy becomes L_r^p. Subsequent manipulations of L affect either the original, which is now L_r^{main}, or the copy, which is now L_r^p, but not both.

Example 32 *Suppose reaction r is a gamma-type reaction whose τ values coincide for parameter sets main, 1, and 2. As in Table 9.2a, suppose $L_r^{main} = L_r^1 = L_r^2 = \{14.6, 18.2, 19.3\}$. Now, suppose that executing some other reaction $\mu \neq r$ in parameter set 1 requires that we update L_r by adding the value 20.8. Because we only update in parameter set 1, it is necessary to copy the L_r^{main} entry, and change the C_r set in the $\mathcal{P}^{main}$ entry (Table 9.2b). Upon*

	Parameter Set	Variable	Value
(a)	main	$\mathcal{P}^{main}$ entry	$(r, 14.6, \{main, 1, 2\})$
	main	L_r^{main}	$\{14.6, 18.2, 19.3\}$
	Parameter Set	**Variable**	**Value**
(b)	main	$\mathcal{P}^{main}$ entry	$(r, 14.6, \{main, 2\})$
	main	L_r^{main}	$\{14.6, 18.2, 19.3\}$
	1	$\mathcal{P}^1$ entry	$(r, 14.6)$
	1	L_r^1	$\{14.6, 18.2, 19.3\}$
	Parameter Set	**Variable**	**Value**
(c)	main	$\mathcal{P}^{main}$ entry	$(r, 14.6, \{main, 2\})$
	main	L_r^{main}	$\{14.6, 18.2, 19.3\}$
	1	$\mathcal{P}^1$ entry	$(r, 14.6)$
	1	L_r^1	$\{14.6, 18.2, 19.3, 20.8\}$

Table 9.2: Copying of state for gamma reaction. (a) Before any manipulations, when parameter sets $\{main, 1, 2\}$ agree. (b) Parameter set 1 has split off and been copied. (c) Subsequent manipulation affecting parameter set 1 only.

executing μ in parameter set 1, the new τ' value is added to L_r^1 only (Table 9.2c).

More generally, non-Markov processes will require some sort of shared state S_r^{main}. One may use a single copy corresponding to the main branch, and as other parameter sets diverge (in steps MNRM-Update 1b and 2aiv, for example), it is necessary to copy S_r^{main} to create S_r^p.

In principle, one can apply the MNRM-ideas to the stored state S to minimize the amount of duplication. In practice, the amount of state may be small, and the code necessary may outweigh the benefits.

9.7 Timing Analysis and Improvements

This section analyzes the time it takes to run MNRM and suggests two ways to improve its worst-case performance.

The operations in MNRM-Initialize are executed only once, whereas the operations in MNRM-Execute and MNRM-Update are executed once for each simulation event. As there will typically be very many simulation events, we shall ignore the initialization cost.

At each iteration, MNRM chooses an event to execute, which consists of a parameter set p and a time τ, which together constitute the top entry of $\mathcal{P}^{top-level}$. The event executed will then be the top entry of $\mathcal{P}^p$. There are 2 cases:

Case 4 $p \neq main$

In this case, MNRM executes a single event in a single parameter set. MNRM-Execute runs Step 1, which takes constant time, MNRM-Update executes Step 1, which takes for each α *with* $(r, \alpha) \in \mathcal{G}$*, either* $\log |\mathcal{P}^p|$ *(Step 1a) or* $\log |\mathcal{P}^p| + const$ *(Step 1b). Assuming, as before, that* $|\{(r, \alpha) \in \mathcal{G}\}|$ *is a constant independent of* r *(in other words, that the dependency graph is sparse), then the total amount of time is* $\mathcal{O}(\log |\mathcal{P}^p|)$*. (Here we have used the fact that although* $\mathcal{P}^p$ *may grow, its total growth is bounded by* $|\{(r, \alpha) \in \mathcal{G}\}|$*, a constant.) Note that* $|\mathcal{P}^p|$ *will always be* $\leq R$*, and typically will be* $\ll R$*. At the end of the update process, one updates the* $(p, \tau^p_{\min})$ *entry of* $\mathcal{P}^{top-level}$*, which takes worst case* $\mathcal{O}(\log pmax)$*.*

In short, one event takes $\mathcal{O}(\log R + \log pmax)$ *operations in the worst case.*

Case 5 $p = main$

In this case, MNRM executes events for each $p \in C_r$ *for the* (r, τ^{main}_r, C_r) *entry of* $\mathcal{P}^{main}$*. There are* $|C_r|$ *such events.*

The function MNRM-Execute requires $pmax + |C_r| \times const$ *operations; the first term is to loop through all possible parameter sets p (assumed to be stored as bits in an array) and const to execute each reaction.*

For each $(r, \alpha) \in \mathcal{G}$*, Step 2ai takes p operations, and exactly one of 2aii-2av is executed: 2aii and 2av require a constant amount of time, 2aiii requires* $\log r^\pi$ *and 2aiv requires* $const + \log r^\pi$*. Thus the total time for Step 2 is* $|\{(r, \alpha) \in \mathcal{G}\}| \times (pmax + |C_r \otimes C_\alpha| \log r^\pi)$*, where* $\otimes$ *denotes the symmetric difference, i.e., elements in* C_r *or* C_α *but not in both.*

Step 2b requires $|\{(r, \alpha) \in \mathcal{G}\}| \times \log R$ *operations.*

Updating $\mathcal{P}^{top-level}$ *takes worst case* $\mathcal{O}(pmax)$ *operations, in the case that all its entries change and it is necessary to rebuild it from scratch, rather than updating each entry individually.*

In short, $|C_r|$ *events take* $\mathcal{O}(pmax + |C_r| + |C_r \otimes C_\alpha| \log r^\pi + \log R)$ *operations in the worst case.*

In the latter case, amortizing across events gives us that each event takes

$$\mathcal{O}(\frac{pmax}{|C_r|} + \frac{|C_r|}{|C_r|} + \frac{|C_r \otimes C_\alpha| \log r^\pi}{|C_r|} + \frac{\log R}{|C_r|})$$

operations in the worst case. If $|C_r|$ is small, this is $\mathcal{O}(pmax)$ operations *per simulation* *event*, which is atrocious. Therefore, to get reasonable performance, $|C_r|$ needs to be close to $pmax$, which leads to our first improvement:

If $|C_r| \ll pmax$, move all $p \in C_r$ into $\mathcal{P}^p$, and mark the r entry of $\mathcal{P}^{main}$ as "main only." When executing a "main only" event, do not bother looping over all parameter sets, only execute in the *main* parameter set.

Given this improvement, either $|C_r| \approx pmax$ or $|C_r| = 1$. The order per event is

$$\begin{cases} \mathcal{O}(\frac{|C_r \otimes C_\alpha| \log r^\pi}{|C_r|} + \frac{\log R}{|C_r|}) \approx \mathcal{O}(\log r^\pi) & \text{if } |C_r| \approx pmax \\ \mathcal{O}(|C_r \otimes C_\alpha| \log r^\pi + \log R) & \text{if } |C_r| = 1 \end{cases}$$

In the former case, the order per event is very good, in the latter, no (as $|C_r \otimes C_\alpha|$ can be close to $pmax$). To rectify this, we introduce improvement 2:

If $|r^\pi|$ gets to be too big, move all the π elements of $\mathcal{P}^{main}$ into $\mathcal{P}^p$ and continue the simulation of parameter set π independently of the others.

This reduces the complexity to $\log R$, for individual simulations, as they are removed from $\mathcal{P}^{top-level}$. Further, since no non-*main* branch can be "too big," not only is $\log r^\pi$ small, but $|C_r \otimes C_\alpha|$ is small on average.

(At long last we are in a position to say why we prefer Algorithm 10 to Algorithm 11, namely it is easier to remove parameter sets from Algorithm 10 to accomplish improvement 2.)

9.8 Conclusions

This chapter has presented the Multiple Next Reaction Method (MNRM), an algorithm to run multiple simulations in parallel for different parameter sets. By keeping track of which as and τs are the same across parameter sets, and using the improvements in the previous section, MNRM is never worse (in order) than running separate instances of NRM, and is frequently much better.

Chapter 10 Sensitivity Analysis

10.1 Summary

A very important problem in analyzing models is: to what extend does the model depend on the exact parameter values (rate constants, etc.) chosen? [25]

To deal with this question, this chapter provides a numerical sensitivity analysis algorithm for calculating the gradient of observable system properties with respect to the system parameters. (Sections 10.3 and 10.4 define more clearly the types of properties and what their gradients mean.) Section 10.5 explains the Mesoscopic Efficient Sensitivity Analysis (MESA) algorithm. Section 10.6 shows the results of applying MESA to the lambda model of Chapter 4. Section 10.7 shows how to bound the difference in a model's predictions given uncertainties in its parameters.

10.2 Introduction

Work done on deterministic biological systems suggests that certain model properties are robust [2, 5]; changing parameters should not affect those properties. Specifically, perfect adaptation is robust in *E. coli*: an *E. coli* cell tumbles with the same frequency regardless of the absolute amount of chemoattractant. Barkai and Leibler's work [5] suggests that the structure of the system, not fine-tuning of parameters, maintains the properties. Perfect adaptation is robust not because of the exact values of the system's parameters (rate constants, etc.), but despite the exact values. Of course, not every property is robust — for example, changing parameters may lead to the same qualitative behavior but different timing.

When one creates a model, certain parameters are unknown, so one is forced to estimate or even guess their values. Sensitivity analysis provides a way to see whether the model depends critically on those parameters (in which case one might want to get better estimates of them) or whether the parameters do not matter at all. In other words, one can study the model at a nominal parameter set (the operating point) and make statements about what the model will do in an area near that operating point, without having to calculate the model's behavior everywhere in the region, rather only at key points. (See Figure 10-2 below.)

This chapter presents a way to deal with robustness and its converse, sensitivity, in the stochastic framework. Section 10.3 gives a simple example of a system with one parameter and a small number of states. This system is used to illustrate the concept of an observable of a system and sensitivity of observables with respect to a parameter. Section 10.3.3 presents an important use of sensitivity — to approximate the value of an observable at different points in parameter space. Section 10.4 formalizes these concepts, and generalizes them to multi-variable systems. Section 10.5 presents a numerical method to approximate sensitivity gradients using the MNRM algorithm of the previous chapter. Section 10.6 applies sensitivity analysis to the running example, the lambda model. Section 10.7 presents another use of sensitivity analysis, namely singular value decomposition.

10.3 Example of Sensitivity Analysis

Consider a simple chemical system

$$A \xrightarrow{k} \text{no } A$$

with initial condition "3 molecules of A." Using the approach of Chapter 3, one can write a chemical master equation for this system, namely

$$\frac{d\overrightarrow{P}}{dt} = \frac{d}{dt}\begin{bmatrix} P_3 \\ P_2 \\ P_1 \\ P_0 \end{bmatrix} = \begin{bmatrix} -3k & 0 & 0 & 0 \\ 3k & -2k & 0 & 0 \\ 0 & 2k & -k & 0 \\ 0 & 0 & k & 0 \end{bmatrix} \overrightarrow{P} \tag{10.1}$$

Solving this gives

$$\overrightarrow{P}(t) = \begin{bmatrix} e^{-3kt} \\ 3e^{-2kt} - 3e^{-3kt} \\ 3e^{-kt} - 6e^{-2kt} + 3e^{-3kt} \\ 1 - 3e^{-kt} + 3e^{-2kt} - e^{-3kt} \end{bmatrix}$$

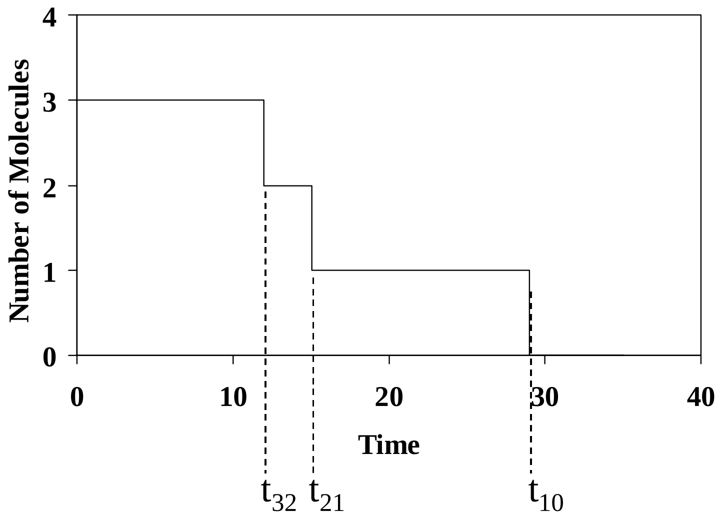


Figure 10-1: Sample trajectory of the system in Example 10.3.

A plot of the number of molecules of A as a function of time for a single trajectory would look something like Figure 10-1. Many plots, corresponding to many trajectories, would give the correct statistical distributions.

Various statistics relate to this process. For example:

- t_{mean}, the mean amount of time from the initial time till there are no molecules of A left, which can be shown to be $\frac{11}{6k}$,
- $P_0(t)$, the probability that there are 0 molecules of A at time t, given by $P_0(t) = 1 - 3e^{-kt} + 3e^{-2kt} - e^{-3kt}$.
- $E[\#A_t]$, the expected number of molecules at time t, given by

$$\begin{aligned} E[\#A_t] &= 3\left(e^{-3kt}\right) + 2\left(3e^{-2kt} - 3e^{-3kt}\right) + 1\left(3e^{-kt} - 6e^{-2kt} + 3e^{-3kt}\right) \\ &\quad +0\left(1 - 3e^{-kt} + 3e^{-2kt} - e^{-3kt}\right) \\ &= 3e^{-kt} \end{aligned}$$

- $Var[\#A_t]$, the variance of the number of molecules at time t.

The simple example system chosen has precisely one parameter, k[1]. This chapter asks the question: *How sensitive is the system to the parameter k?*

[1]One could consider the initial number of molecules to be a parameter, but that will not be done here.

10.3.1 Observables

Suppose k in the example above were replaced with k'. The previous analysis goes through, but with k' instead of k. In particular, the system starts with 3 molecules and ends with none. The plot in Figure 10-1 is identical, the only difference being the probability distribution on transition times t_{32}, t_{21}, and t_{10}. (In other words, while Figure 10-1 is the same, the probability that a given trajectory will look like Figure 10-1 changes.) The statistics change: t_{mean} becomes $\frac{11}{6k'}$, $P_0(t)$ changes, $E[\#A_t]$ becomes $3e^{-k't}$, etc.

So how sensitive is the system to k?

- The states of the system do not change at all.
- The state transitions do not change either.
- The trajectories (Figure 10-1) do not change either, but the probability of getting any particular trajectory changes.
- $P_0(t)$ follows the same type of distribution, but with different parameters.
- t_{mean} changes.
- $E[\#A_t]$ changes.

In other words, whether the system is sensitive to k or not, and how sensitive, depends on the quantity of interest.

Definition 8 *An* observable *is a measurable quantity of a system. Each of the bullets above corresponds to an observable.*

The key to sensitivity analysis is to define an observable and determine the sensitivity *of that observable* with respect to a given parameter. In particular, this chapter will be concerned with observables of the form

$$y(t) = \sum_{\text{state } s} C_s P_s(t) \tag{10.2}$$

The rate constant k is real-valued, initial numbers of molecules are integer-valued, and it will not make sense to take derivatives (in Section 10.3.2) with respect to integer-valued variables. This is not the severe restriction it appears to be: in general, one could have certain probability distributions on the initial number of molecules of each type, and take derivatives with respect to the parameters of those distributions.

where C_s is a function of state only.

Example 33 *$y(t) = P_0(t)$ is such an observable, with $C_0 = 1$, $C_i = 0$ otherwise.*

Example 34 *$y(t) = E[\#A_t]$ is also such an observable, with $C_i = i$.*

Example 35 *$y(t) = E[(\#A_t)^2]$ is also such an observable, with $C_i = i^2$.*

Example 36 *t_{mean} is not of the form Eq 10.2, nor is the shape of Figure 10-1, nor the type of any of the distributions.*

Remark 8 *One may object that Eq 10.2 only allows* linear *functions. This is not entirely correct. Eq 10.2 only allows functions that are linear in probability, but they may be of arbitrary order in the state. For example, one could construct $\mathcal{C}$ so as to observe $\overrightarrow{y}(t) =$ the k-th moment of the number of molecules of type A. (This quantity, $\sum_s (\#A_s)^k P(s)$, is linear in P, but not in $\#A$.)*

10.3.2 Sensitivity of an Observable

This section shifts focus slightly and views $y(t)$ as a function of k, not of t. To reinforce this notion, we shall write $y_t(k)$. In other words, one treats t as fixed. Given that $y_t(k)$ is a function of k, it is meaningful to discuss its derivative with respect to k, which provides a measure of the sensitivity of y to changes in k.

Definition 9 *The derivative $\left(\frac{dy_t}{dk}\right)_{k^*}$ is called "the sensitivity of y_t with respect to parameter k, evaluated at k^*."*

Example 37 *Let $t = 3$ and $k^* = 1$. For the observable $y_t(k) = P_0(t)$, we have*

$$\begin{aligned}
\left(\frac{dy_t}{dk}\right)_{k^*} &= \frac{d}{dk}\left(1 - 3e^{-kt} + 3e^{-2kt} - e^{-3kt}\right)_{k^*} \\
&= \left(+3te^{-kt} - 6te^{-2kt} + 3te^{-3kt}\right)_{k^*} \\
&= 0.4046
\end{aligned}$$

Example 38 *Using the same t and k^*, let the observable $y_t(k) = E[\#A_t]$. Then*

$$\begin{aligned}\left(\frac{dy_t}{dk}\right)_{k^*} &= \frac{d}{dk}\left(3e^{-kt}\right)_{k^*} \\ &= \left(-3te^{-kt}\right)_{k^*} \\ &= -0.4481\end{aligned}$$

10.3.3 Using Sensitivity to Approximate Observables

In general, $y_t(k)$ is a messy, non-linear function. Using a Taylor expansion about a point k^* (in parameter space) gives

$$y_t(k) = y_t(k^*) + (k - k^*)\left(\frac{dy_t}{dk}\right)_{k^*} + \text{ Higher order terms}$$

One important use of the sensitivity is to approximate the value of y_t for parameter values close to the operating point k^*. To a first approximation, one may drop the higher order terms approximate about k^* using the linear terms only. Other uses of sensitivity will be shown in Section 10.7.

Example 39 *In Example 37, approximate y_t at $k = 1.2$. At the operating point ($k^* = 1$),*

$$y_t(k^*) = \left(1 - 3e^{-kt} + 3e^{-2kt} - e^{-3kt}\right)_{k^*} = 0.8580$$

and the approximated $y_t(k)$ is

$$\begin{aligned}y_t(k) &= y_t(k^*) + (k - k^*)\left(\frac{dy_t}{dk}\right)_{k^*} \\ &= 0.8580 + (1.2 - 1)(0.4046) \\ &= 0.9389\end{aligned}$$

For comparison, the actual calculated value is 0.9202.

Example 40 *Using the same k, use the observable from Example 38. Then,*

$$y_t(k^*) = \left(3e^{-kt}\right)_{k^*} = 0.1494$$

and so the approximated $y_t(k)$ is

$$\begin{aligned} y_t(k) &= y_t(k^*) + (k - k^*) \left(\frac{dy_t}{dk}\right)_{k^*} \\ &= 0.1494 + (1.2 - 1)(-0.4481) \\ &= 0.0598 \end{aligned}$$

For comparison, the actual calculated value is 0.0820.

10.4 Generalization - Multi-dimensional Sensitivity

Chapter 3 described the chemical master equation approach to stochastic kinetics. Although that approach is impractical for large systems, it provides a simple theoretical framework for sensitivity analysis. More practical techniques will be discussed in Section 10.5.

Recall that in the chemical master equation framework, one considers a probability vector $\overrightarrow{P}(t)$ that changes over time and contains a huge number of states. Specifically, the i-th component of $\overrightarrow{P}(t)$ is the probability of being in state i at time t, conditioned on the initial state. $\overrightarrow{P}(t)$ evolves with time according to the equation:

$$\frac{d\overrightarrow{P}(t)}{dt} = \mathcal{W}\overrightarrow{P}(t) \tag{10.3}$$

where $\mathcal{W}$ is a coefficient matrix that is independent of $\overrightarrow{P}(t)$. The matrix $\mathcal{W}$ is determined from the chemistry using propensity functions a_r corresponding to each reaction (see Chapter 3), which are in turn functions of kinetic rate constants, k_r. Note that *in principle*, one could solve this equation to get $\overrightarrow{P}(t)$, but *in practice*, $\overrightarrow{P}(t)$ is too big (e.g., in the lambda model it has something like 10^{60} elements). Even writing out $\mathcal{W}$ is impractical, although any given $\mathcal{W}_{ij}$ can be calculated easily.

Definition 10 *A* mesoscopic chemical system *is one that obeys Eq 10.3.*

Now consider an observable. The observables in the previous section were all one-dimensional, but one can equally well have a multi-dimensional observable.

Definition 11 *An observable of a mesoscopic chemical system is a function $y(t)$ defined* *by*

$$\overrightarrow{y}(t) = \mathcal{C}\overrightarrow{P}(t) \tag{10.4}$$

Where $\mathcal{C}$ is a matrix.

In particular, if $\mathcal{C}$ is the identity matrix, one observes the probabilities of all states directly. If $\mathcal{C}$ is invertible, one may infer the probabilities of all states from the observation. In Eq 10.2 of the previous subsection, $y(t)$ was a scalar (i.e., a real number) and $\mathcal{C}$ was an n-vector, or equivalently a $1 \times n$ matrix. Typically, the chemical master equation will be very high dimensional, and the observable will be very low dimensional, so one will not be able to find all the original probabilities.

Example 41 *In the lambda model, one is interested in the statistics of the final fate of the system, namely P(lysis), P(lysogeny), and P(undecided) at t_{final} = 35 minutes. In the chemical master equation framework, the system has very many states (call this number n) and obeys Eq 10.3, and the observable is*

$$\overrightarrow{y}(t) = \begin{bmatrix} P(lysis) \\ P(lysogeny) \\ P(undecided) \end{bmatrix} = \mathcal{C}\overrightarrow{P}(t)$$

where $\mathcal{C}$ is a 3×n matrix with

$$\begin{aligned} \mathcal{C}_{1,i} &= \begin{cases} 1 & \textit{if state } i \textit{ is lysis} \\ 0 & \textit{otherwise} \end{cases} \\ \mathcal{C}_{2,i} &= \begin{cases} 1 & \textit{if state } i \textit{ is lysogeny} \\ 0 & \textit{otherwise} \end{cases} \\ \mathcal{C}_{3,i} &= \begin{cases} 1 & \textit{if state } i \textit{ is undecided} \\ 0 & \textit{otherwise} \end{cases} \end{aligned}$$

The final cell fate probabilities are given by $\overrightarrow{y}$(35 minutes).

Example 42 *Consider the system $A \xrightarrow{k_1} \text{no } A$, $B \xrightarrow{k_2} \text{no } B$, subject to the initial conditions*

"1 molecule of A, 1 molecule of B." This system obeys the master equation

$$\frac{d\overrightarrow{P}(t)}{dt} = \frac{d}{dt}\begin{bmatrix} P(1A,1B) \\ P(1A,0B) \\ P(0A,1B) \\ P(0A,0B) \end{bmatrix} = \begin{bmatrix} -k_1-k_2 & 0 & 0 & 0 \\ k_2 & -k_1 & 0 & 0 \\ k_1 & 0 & -k_2 & 0 \\ 0 & k_1 & k_2 & 0 \end{bmatrix} \overrightarrow{P}(t)$$

Let $y(t)$ be $\begin{bmatrix} E[\#A] \\ E[\#B] \end{bmatrix}$. *Then* $\mathcal{C} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix}$.

10.4.1 Sensitivity Gradients

The previous section considered only a single rate constant, k, but now consider multiple rate constants, k_1, k_2, etc.

Definition 12 *For a mesoscopic chemical system obeying Eq 10.3 and an observable $\overrightarrow{y}_t$ (possibly multi-dimensional) obeying Eq 10.4, the derivative of y with respect to the rth parameter, namely $\frac{\partial \overrightarrow{y_t}}{\partial k_r}$, is called the sensitivity of $\overrightarrow{y}_t$ with respect to k_r.*

Definition 13 *The* sensitivity gradient $\nabla\overrightarrow{y}_t$ *of an observable of a mesoscopic chemical system is a matrix* $\left[\frac{\partial \overrightarrow{y_t}}{\partial k_1}, \frac{\partial \overrightarrow{y_t}}{\partial k_2}, \ldots\right]$.

Example 43 *Consider the system of Example 42. Solving the master equation,*

$$\overrightarrow{P}(t) = \begin{bmatrix} e^{-(k_1+k_2)t} \\ e^{-k_1 t}(1-e^{-k_2 t}) \\ (1-e^{-k_1 t})e^{-k_2 t} \\ (1-e^{-k_1 t})(1-e^{-k_2 t}) \end{bmatrix}$$

So,

$$\begin{aligned}
\overrightarrow{y}(t) &= \mathcal{C}\overrightarrow{P}(t) \\
\begin{bmatrix} E[\#A] \\ E[\#B] \end{bmatrix} &= \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} e^{-(k_1+k_2)t} \\ e^{-k_1 t}(1-e^{-k_2 t}) \\ (1-e^{-k_1 t})e^{-k_2 t} \\ (1-e^{-k_1 t})(1-e^{-k_2 t}) \end{bmatrix} \\
&= \begin{bmatrix} e^{-k_1 t} \\ e^{-k_2 t} \end{bmatrix}
\end{aligned}$$

And

$$\nabla \overrightarrow{y_t} = \begin{bmatrix} -te^{-k_1 t} & 0 \\ 0 & -te^{-k_2 t} \end{bmatrix}$$

10.4.2 Using Multi-dimensional Sensitivity to Approximate Observables

Let $\overrightarrow{y_t}(\overrightarrow{k}^*)$ be the value of y at some operating point $\overrightarrow{k}^* = (k_1^*, k_2^*, \dots)$ in parameter space. At a nearby point $\overrightarrow{k}$, the value of $\overrightarrow{y_t}(\overrightarrow{k})$ is given by

$$\begin{aligned}
\overrightarrow{y_t}(\overrightarrow{k}) &= \overrightarrow{y_t}(\overrightarrow{k}^*) + \left(\frac{\partial \overrightarrow{y_t}}{\partial k_1}\right)_{\overrightarrow{k}^*}(k_1 - k_1^*) + \left(\frac{\partial \overrightarrow{y_t}}{\partial k_2}\right)_{\overrightarrow{k}^*}(k_2 - k_2^*) + \dots \\
&\quad + \left(\frac{\partial \overrightarrow{y_t}}{\partial k_n}\right)_{\overrightarrow{k}^*}(k_n - k_n^*) + \text{ higher order terms} \qquad (10.5)
\end{aligned}$$

In other words, one can approximate $\overrightarrow{y_t}(\overrightarrow{k})$ anywhere in a multi-dimensional neighborhood of an operating point $\overrightarrow{k}^*$ using just 1) the value at the operating point, and 2) the sensitivities, $\frac{\partial \overrightarrow{y_t}}{\partial k_r}$. So by calculating n one-dimensional functions, one can approximate y anywhere in an n-dimensional neighborhood centered at $\overrightarrow{k}^*$, as in Figure 10-2. This first-order approximation should be relatively good near the operating point, but may get progressively worse away from the operating point. In fact, it would be wrong to assume that system properties far from the operating point are similar to those close to the operating point.

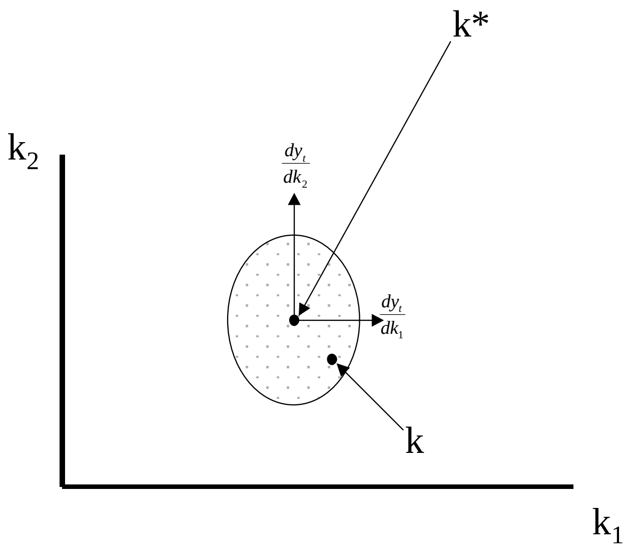


Figure 10-2: The sensitivity gradients $\frac{\partial y_t}{\partial k_1}$ allow one to approximate y at any point k in a neighborhood (shaded) about k^*.

10.5 Calculating Sensitivity

In principle, one can calculate $\frac{\partial \overrightarrow{y}(t)}{\partial k_j}$:

$$\frac{\partial \overrightarrow{y}(t)}{\partial k_j} = \frac{\partial}{\partial k_j}\overrightarrow{y}(t) = \frac{\partial}{\partial k_j}\left(\mathcal{C}\overrightarrow{P}(t)\right) = \mathcal{C}\frac{\partial \overrightarrow{P}(t)}{\partial k_j}$$

and

$$\begin{aligned}
\frac{d}{dt}\left(\frac{\partial \overrightarrow{P}(t)}{\partial k_j}\right) &= \frac{\partial}{\partial k_j}\left(\frac{d\overrightarrow{P}(t)}{dt}\right) \\
&= \frac{\partial}{\partial k_j}\left(\mathcal{W}\overrightarrow{P}(t)\right) \\
&= \mathcal{W}\left(\frac{\partial \overrightarrow{P}(t)}{\partial k_j}\right) + \left(\frac{\partial \mathcal{W}}{\partial k_j}\right)\overrightarrow{P}(t)
\end{aligned}$$

Letting $u = \frac{\partial \overrightarrow{P}(t)}{\partial k_j}$ and $\mathcal{W}' = \frac{\partial \mathcal{W}}{\partial k_j}$, these equations simplify to:

$$\begin{aligned}
\frac{\partial \overrightarrow{y}(t)}{\partial k_j} &= \mathcal{C}u \\
\frac{du}{dt} &= \mathcal{W}u + \mathcal{W}'\overrightarrow{P}(t)
\end{aligned}$$

So, in principle, one can solve for $\overrightarrow{P}(t)$, use that to solve for u, and use that to solve for $\frac{\partial \overrightarrow{y}(t)}{\partial k_j}$.

Example 44 *For the example of Section 10.3, with y as in Example 38 and $k_j = k$,*

$$\frac{\partial \mathcal{W}}{\partial k_j} = \begin{bmatrix} -3 & 0 & 0 & 0 \\ 3 & -2 & 0 & 0 \\ 0 & 2 & -1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

so

$$\frac{d}{dt}\left(\frac{\partial \overrightarrow{P}(t)}{\partial k_j}\right) = \begin{bmatrix} -3k & 0 & 0 & 0 \\ 3k & -2k & 0 & 0 \\ 0 & 2k & -k & 0 \\ 0 & 0 & k & 0 \end{bmatrix} \frac{\partial \overrightarrow{P}(t)}{\partial k_j} + \begin{bmatrix} -3 & 0 & 0 & 0 \\ 3 & -2 & 0 & 0 \\ 0 & 2 & -1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \overrightarrow{P}(t)$$

where $\overrightarrow{P}(t)$ satisfies Eq 10.1.

Of course, this technique is not practical for larger systems. Rather, we shall use the fact that $\frac{\partial \overrightarrow{y}(t)}{\partial k_j}$ is a well-defined, meaningful quantity, and develop numerical techniques to approximate it.

10.5.1 Calculating Derivatives

It is straightforward to calculate y at any point $\overrightarrow{k}$ in parameter space: one uses the parameters $\overrightarrow{k}$ and runs the Next Reaction Method to calculate y. In particular, NRM provides samples from $\overrightarrow{P}(t)$, and $\overrightarrow{y}(t) = \mathcal{C}\overrightarrow{P}(t)$, so in the limit of large numbers, the estimated y approaches the correct y.

Example 45 *In the running lambda example, y is a 3-vector, $\mathcal{C}$ is a $3{\times}n$ matrix, where n is very large (10^{60}) and P is an n-vector. Each trajectory run by the simulator picks a state from P according to the correct probability distribution, so to estimate the probabilities in P to some accuracy ε would take a number of trajectories proportional to some function* *of n and ε^2. Clearly, since n is so large, this is not practical. However, to estimate y, one needs only a number of trajectories proportional to ε and 3, so it is tractable.*

The following algorithm approximates the derivatives $\frac{\partial \overrightarrow{y_t}(\overrightarrow{k})}{\partial k_j}$:

Algorithm 16 (Mesoscopic Efficient Sensitivity Analysis (MESA))

1. *Calculate $\overrightarrow{y_t}(\overrightarrow{k^*})$.*
2. *$\forall j$, calculate $\overrightarrow{y_t}(\overrightarrow{k^*} + \Delta k_j)$ and $\overrightarrow{y_t}(\overrightarrow{k^*} - \Delta k_j)$.*
3. *Use $\overrightarrow{y_t}(\overrightarrow{k^*} + \Delta k_j)$ and $\overrightarrow{y_t}(\overrightarrow{k^*} - \Delta k_j)$ to approximate $\frac{\partial \overrightarrow{y_t}(\overrightarrow{k})}{\partial k_j}$, using the formula*

$$\frac{\partial \overrightarrow{y_t}(\overrightarrow{k})}{\partial k_j} \approx \frac{\overrightarrow{y_t}(\overrightarrow{k^*} + \Delta k_j) - \overrightarrow{y_t}(\overrightarrow{k^*} - \Delta k_j)}{2\Delta k_j} \tag{10.6}$$

The formula in Eq 10.6 is a standard three-point approximation of a derivative [7] (the three points are $\overrightarrow{k^*} - \Delta k_j$, $\overrightarrow{k^*}$, and $\overrightarrow{k^*} + \Delta k_j$, even though the middle term is not used) and is accurate to $O((\Delta k_j)^2)$. There are also higher order approximations, using four or more points, which may be used instead without changing the basic idea. The remainder of this chapter will assume the three-point method.

To calculate $\overrightarrow{y_t}(\overrightarrow{k^*})$ and $\overrightarrow{y_t}(\overrightarrow{k^*} \pm \Delta k_j)$ for all j, one can use the NRM $2n + 1$ times. This approach is entirely doable: e.g., the NRM, applied to the lambda model for MOI 1 to 10, takes a couple of hours on a desktop, so one can run the entirety of MESA on a cluster of desktop machines in a couple of days. However, the $2n + 1$ calculations done are highly redundant and can be reduced significantly. Consider, as an extreme example, a parameter k_i that the system is not sensitive to. Then $\overrightarrow{y_t}(\overrightarrow{k^*}) = \overrightarrow{y_t}(\overrightarrow{k^*} + \Delta k_i) = \overrightarrow{y_t}(\overrightarrow{k^*} - \Delta k_i)$, so there should be no reason to do the same calculation three times. For other parameters — even sensitive ones — some trajectories may not depend on the parameter at all, i.e., the reaction in question may never occur, and other trajectories may be sensitive only from a certain point on. In either case, there is computational savings to be had.

The problem to be solved is to run NRM on many similar parameter sets, precisely the problem that MNRM handles.

[2] *If one is interested in estimating each probability to a fixed percentage error, the number of trajectories would depend on the values in P, not just its size.*

Parameters	P(lysis)	P(lysogeny)
$\overrightarrow{k}^*$	0.77	0.23
$\overrightarrow{k}^* + \Delta k_4$	0.81	0.19
$\overrightarrow{k}^* - \Delta k_4$	0.73	0.27
$\overrightarrow{k}^* + \Delta k_5$	0.83	0.17
$\overrightarrow{k}^* - \Delta k_5$	0.80	0.20
$\overrightarrow{k}^* + \Delta k_6$	0.79	0.21
$\overrightarrow{k}^* - \Delta k_6$	0.83	0.17
$\overrightarrow{k}^* + \Delta k_7$	0.75	0.25
$\overrightarrow{k}^* - \Delta k_7$	0.78	0.22

Table 10.1: Sensitivity data for four equations of the lambda model.

Algorithm 17 *Let $\overrightarrow{k^{main}} = \overrightarrow{k}^*$, let $\overrightarrow{k^{2i+1}} = \overrightarrow{k}^* + \Delta k_i$, and let $\overrightarrow{k^{2i+2}} = \overrightarrow{k^*} - \Delta k_i$, and run MNRM to get the y values for MESA.*

10.6 Sensitivity of the Lambda Model

In the lambda model, let

$$\overrightarrow{y_t} = \begin{bmatrix} P(\text{lysis}) \\ P(\text{lysogeny}) \end{bmatrix}$$

At MOI=3 (chosen because neither P(lysis) nor P(lysogeny) is close to 0) $\overrightarrow{y_t}$ is approximately $\begin{bmatrix} 0.77 & 0.23 \end{bmatrix}^\top$. This section will consider the first four elementary reactions, Reactions 4–7 in Table 4.2. (Elementary reactions are non-gamma, non-promoter reactions.) Data for all 46 elementary reactions can be found in Table 10.2, for now the first four will be used to illustrate some of the principles involved.

MNRM was used to generate $\overrightarrow{y_t}$ estimates at $\overrightarrow{k}^*$ = the values in Table 4.2. Each Δk_i was half the value of k_i, so, for example, k_4 is 5 sec^{-1}, so Δk_i is 2.5 sec^{-1}. Under these conditions, the following $\overrightarrow{y_t}$ values were generated:

Note that some of these appear to be well-approximated by a linear fit — Δk_4 adds 0.04 and $-\Delta k_4$ subtracts 0.04, for example — but others (e.g., k_5) do not. Such is the danger of assuming linearity, particularly with the statistical uncertainty inherent in mesoscopic chemical systems. To fix this, one would need to decrease Δk_5 and make statements about $\overrightarrow{y_t}$ on a smaller part of parameter space. To get the full parameter space, one would need to do multiple runs of MESA about different operating points, and approximate the more complex non-linear behavior by a series of piecewise linear segments. We shall ignore this concern for now, and assume that it is legitimate (within statistical error) to approximate as linear on the space of interest.

The formula in Eq 10.6 of the MESA algorithm in Section 10.5 can be used to estimate $\nabla \overrightarrow{y_t}$. To do this, one needs to use the data from Table 10.1 and the nominal values of the rate constants (as the Δks were derived from them). (It is unclear whether this normalization of the "error as a fraction of rate constant" normalization one would get by not dividing by nominal rate constants is more useful. We shall assume the former.) This gives

$$\nabla \overrightarrow{y_t} = \begin{bmatrix} 0.02 & 0.21 & -0.40 & -0.001 \\ -0.02 & -0.21 & 0.40 & 0.001 \end{bmatrix}$$

As in previous sections, one can use this sensitivity matrix to approximate $\overrightarrow{y_t}$ anywhere in a four-dimensional neighborhood of the nominal parameter values. We shall leave that as an exercise for the interested reader.

Using NRM, one could do the calculation in this section. The key to MNRM is that it is much faster, so one can do sensitivity analysis for all the elementary reaction (see Section 10.7) rather than just a small subset.

10.7 Other Uses of Sensitivity

So far, only one use of sensitivity has been presented, namely, using sensitivity derivatives to approximate an observable anywhere in a neighborhood around the nominal parameters. This is an important feat, because it allows one to do a small number of calculations and approximate a function over a much larger space. This section shows another use of sensitivity derivatives, namely to find the singular values of the sensitivity and to figure out what *combination* of parameter uncertainties has the biggest effect on the model's predictions.

10.7.1 Singular Value Decomposition

One can linearize Eq 10.5, and rewrite it in matrix form (using the sensitivity gradient in Definition 13):

$$\overrightarrow{y_t}(\overrightarrow{k}) \approx \overrightarrow{y_t}(\overrightarrow{k}^*) + (\nabla \overrightarrow{y_t})_{\overrightarrow{k}^*}(\Delta k)$$

The vector $\overrightarrow{\Delta k} = \overrightarrow{k} - \overrightarrow{k}^*$. The matrix $\nabla \overrightarrow{y_t}$ consists of n columns, where the ith column is $\frac{\partial \overrightarrow{y_t}}{\partial k_i}$.

One analysis tool one can apply is Singular Value Decomposition (SVD) [41]. Basically, any matrix M can be decomposed into $Q_1 \Sigma Q_2$ where Σ is a diagonal matrix and Q_1 and Q_2 are unitary matrices.

Example 46 *For the four lambda reactions above, $\nabla \overrightarrow{y_t}$ can be rewritten as*

$$\begin{aligned}
\nabla \overrightarrow{y_t} &= Q_1 \Sigma Q_2 \\
&= \begin{bmatrix} 0.707 & -0.707 \\ -0.707 & -0.707 \end{bmatrix} \begin{bmatrix} 0.637 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0.036 & 0.459 & -0.888 & -0.002 \\ -0.151 & 0.880 & 0.449 & 0.010 \\ -0.988 & -0.118 & -0.101 & -0.002 \\ 0 & -0.008 & -0.007 & 1 \end{bmatrix}
\end{aligned}$$

In particular, the maximum singular value of Σ is the maximum change in $\|\overrightarrow{y_t}\|$ for any normalized vector $\overrightarrow{\Delta k}$, and that maximum change occurs when the vector is in the direction corresponding to the row of Q_2 that corresponds to the maximum singular value.

Example 47 *In the lambda example 46, the manipulation that will change $\|\overrightarrow{y_t}\|$ the most is $\overrightarrow{\Delta k} = \begin{bmatrix} 0.036 & 0.459 & -0.888 & -0.002 \end{bmatrix}$, the first row of Q_2. Recalling that*

$$\overrightarrow{\Delta k} = \begin{bmatrix} \Delta k_4 & \Delta k_5 & \Delta k_6 & \Delta k_7 \end{bmatrix}$$

note that k_3 is the most important of the three parameters; changing k_3 by a unit will change $\|\overrightarrow{y_t}\|$ by $(-0.40)^2 + (0.40)^2 = 0.32$. However, changing k_2 by 0.459 and k_3 by $-.0888$ (a total of a unit as $(0.459)^2 + (-0.888)^2 = 1$) will result in a change to $\|\overrightarrow{y_t}\|$ of $(0.637)^2 = 0.41$, greater than a unit change in any individual parameter.

Sensitivity analysis, in particular the $\nabla \overrightarrow{y_t}$ matrix, shows how much the model will change per unit change of individual parameters. Singular value decomposition considers combinations of parameters and makes definite statements about the maximum change in $\|\overrightarrow{y_t}\|$ that can be achieved by *any* normalized combination of changes in parameters.

10.7.2 SVD of Lambda

MESA was run on the lambda model for all elementary reactions (i.e., not gamma reactions or promoter reactions, but all others) at MOI=3. (Chosen because $0 \ll P(\text{lysis at MOI=3}) \ll 1$.) There are at least two different ways to do this: if one uses straight unit-changes, then Reaction 63 is the most sensitive. This is to be expected, as the rate constant for that reaction, at 0.0001, is the lowest in the simulation; the second most sensitive is 62, other reaction with constant 0.0001. So a unit change, 0.0001 to 1.0001, is very big relative to the rate constant (in fact, so big that the linear analysis almost certainly does not hold), while a unit step of a rate constant 100 is small compared to the rate constant. A much better comparison is to normalize, so a "unit step" is a constant fraction of the rate constant. Under these conditions, no one reaction sticks out. The sensitivities range from 0 to (absolute value) 0.19. Seven reactions (see Table 10.2) have sensitivities greater (in absolute value) than 0.1. Table 10.2 shows the reactions used, their sensitivities, and the normalized combination of them that gives the largest change in $\|\overrightarrow{y_t}\|$, namely $(0.673)^2 = 0.45$.

This is the sort of analysis one can do with no *a priori* knowledge of the system. Really, though, the right way to do this is to define error bounds on each k_r based on the amount of data one has available to get the value k_r, then normalize *with respect to these error bounds.* (Note that creating such bounds is a modeling issue, not an algorithmic issue, and will not be addressed here. It will, however, be alluded to in the chapter on parameter estimation, Chapter 11.) One can then run MESA and find out how sensitive the model is *within the parameter space of interest*, defined by the error bounds. For example, if k_1 is known very precisely, but k_2 is not, a modeler should use MESA on a parameter space that is small in the k_1 dimension and large in the k_2 dimension.

Reaction #	Sensitivity	Component	Reaction #	Sensitivity	Component
4	0.080	0.168	48	0.000	-0.011
5	0.030	0.063	50	-0.090	-0.189
6	-0.040	-0.084	51	-0.110	-0.231
7	-0.030	-0.063	52	0.000	-0.000
10	0.030	0.063	53	-0.040	-0.084
11	-0.040	-0.084	54	0.070	0.147
12	-0.020	-0.042	55	-0.050	-0.105
22	-0.190	-0.399	56	0.020	0.042
23	-0.070	-0.158	57	-0.110	-0.231
24	0.070	0.147	58	-0.050	-0.105
25	-0.040	-0.084	59	-0.010	-0.021
30	-0.120	-0.252	60	-0.130	-0.273
31	0.030	0.084	61	0.010	0.010
32	0.070	0.147	62	0.060	0.126
35	-0.010	-0.021	63	-0.100	-0.210
36	0.040	0.084	64	-0.080	-0.168
37	-0.050	-0.105	65	-0.030	-0.063
38	-0.030	-0.063	66	-0.050	-0.105
39	0.040	0.084	67	0.150	0.315
40	0.030	0.063	68	0.010	0.021
42	-0.060	-0.137	70	0.080	0.168
44	-0.040	-0.073	71	0.020	0.042
46	0.130	0.273	72	-0.030	-0.063

Table 10.2: Data for singular value decomposition of lambda phage.

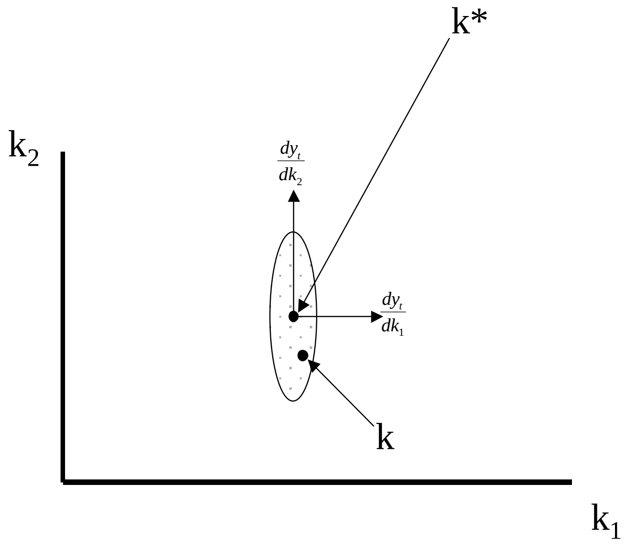


Figure 10-3: The sensitivity gradients $\frac{\partial y_t}{\partial k_1}$ allow one to approximate y at any point k in a neighborhood (shaded) about k^*. Here the neighborhood is smaller in the k_1 dimension, because uncertainty in the value of k_1 is less than uncertainty in the value of k_2.

Part V

Parameter Estimation

Chapter 11 Estimation of Stochastic Parameters

11.1 Summary

Previous chapters have considered the problems of simulation and sensitivity analysis, both of which ask what predictions (in the form of trajectories, etc.) come from a given model. This chapter considers the inverse problem: *given trajectories, how much information (rate constants, etc.) can one retrieve about the underlying model?*

The fundamental result presented is that a trajectory where reaction r occurs n times contains the same amount of information as n statistically independent random draws from a distribution with parameter k_r. (This chapter will concern itself with time-invariant distributions, so the distribution will be an exponential.) These draws allow one to estimate $1/k_r$ efficiently (in the statistical sense of efficient estimators) with variance $1/(nk_r^2)$. A standard result in statistics shows that this bound is the best one can hope to achieve, and any other unbiased estimator must have higher variance.

Moreover, the following surprising result comes out of the analysis: in a system with multiple reactions, one can estimate the constant of each individual reaction independently and correct for the effects of other coupled reactions, even if those reactions are unknown. For example, to estimate the parameter k of the reaction $A + B \xrightarrow{k} C$, one needs only measure $\#A$, $\#B$, and $\#C$, even if other (possibly unknown) reactions affect $\#A$, $\#B$, and $\#C$; the results of this chapter show how to correct for other reactions.

Section 11.3 introduces NRM^{-1}, an exact inverse to NRM, which recovers the random numbers used by NRM up to a factor depending on the rate constants k_r. Section 11.4 presents the Estimation of Stochastic Parameters (ESP) algorithm, which is exactly equivalent to NRM^{-1}, but shows more explicitly how to do the estimation for individual reactions. Section 11.6 explains the statistical results and discusses the issue of separability.

11.2 NRM Simplified

Chapter 5, which presented the Next Reaction Method (NRM), was very interested in efficiency. The present chapter is interested in generating an inverse, NRM^{-1}, and making that inverse efficient. Toward that end, we provide NRM', a simplified version of NRM, which

- does not contain the efficient data structures of NRM,
- has a different take on random numbers: the first time it uses a random number, the random number may depend on the rate constants k. All subsequent manipulations do not depend on the rate constants, and
- uses the random numbers ς explained in the footnotes of NRM, rather than only the τs, to avoid problems when the propensity a becomes 0.

Recall from Chapter 3 that the propensity function a of an equation (say $A + B \xrightarrow{k} C$) can be written as $k \times h$, where h is a dimensionless number called the *redundancy function* that tells how many identical (redundant) copies of a reaction are possible. In this example, $h = (\#A) \times (\#B)$.

Algorithm 18 *(NRM′)*

```
1. Initialize
    (a) $t \leftarrow 0$.
    (b) For each chemical $c$, set initial $\#X_c$.
    (c) For each reaction $r$, calculate $h_r$.
    (d) For each reaction $r$, generate a random number $u$ according to a uniform distribution, let $\rho_r \leftarrow \frac{1}{k_r} \ln\left(\frac{1}{u}\right)$, and let $\varsigma_r \leftarrow \rho_r$.
    (e) For each reaction $r$, if $h_r = 0$, then let $\tau_r \leftarrow \infty$. Otherwise, let $\tau_r \leftarrow \varsigma_r / h_r$.
2. Pick and execute
    (a) $\tau \leftarrow \min \tau_r$.
    (b) $\mu \leftarrow$ the corresponding $r$.
```

   (c) *Update $\#X$ according to executing reaction μ.*

   (d) $t \leftarrow \tau$.

3. *Update*

   (a) *For each reaction r, $\varsigma_r \leftarrow \varsigma_r - h_r \times t$.*

   (b) *For each reaction r, recalculate h_r.*

   (c) *Generate a single random number u according to a uniform distribution, let $\rho_r \leftarrow \frac{1}{k_r} \ln\left(\frac{1}{u}\right)$, and let $\varsigma_r \leftarrow \rho_r$.*

   (d) *For each reaction r, $\varsigma_r \leftarrow \varsigma_r + h_r \times t$.*

   (e) *For each reaction r, if $h_r = 0$, then let $\tau_r \leftarrow \infty$. Otherwise, let $\tau_r \leftarrow \varsigma_r / h_r$.*

4. *Loop (go to Step 2).*

We now provide an extended example before proving that NRM$'$ is equivalent to NRM.

Example 48 *Consider again the reactions in Table 11.1, subject to the initial conditions $[\#A,\ \#B,\ \ldots,\ \#G]_0 = [10,\ 8,\ 7,\ 3,\ 6,\ 8,\ 12]$ and using rate constants $[12.3,\ 18.2,\ 6.4,\ 15.0,\ 0.9]$. Table 11.2 shows an execution of NRM$'$.*

Stepping through this: At $t = 0$, the initial $\#X$s are as above, and the hs can be calculated from those numbers and Table 11.1. The initial random numbers from Step 1d, ρ_r, are shown in the first column of the 'ρ_rs' rows. (Subsequent columns of these rows show the values at the end of Step 3c.) In Step 1d, these initial ρ_rs become the initial ς_rs. After Step 1e, the τ_rs are set. Based on the initial τ_rs, $\tau = 0.0001$ in Step 2a and $\mu = 1$ in Step 2b. The second *column of Table 11.2 shows the new $\#X$s and t after executing Steps 2b and c. The second column also shows the recalculated h_rs, the single new random number generated in Step 3c, the fully transformed ς_rs after Step 3d, and the new τ_rs after Step 3e. NRM$'$ continues by picking the minimum of the new τ_rs, etc., as shown in the rest of Table 11.2.*

11.2.1 Correctness

It is not that hard to show that NRM$'$ is entirely equivalent to NRM. Note first of all that the standard way to generate an exponential random variate ρ with parameter a is to

Reaction	h_μ
$A + B \xrightarrow{k_1} C$	$(\#A) \times (\#B)$
$B + C \xrightarrow{k_2} D$	$(\#B) \times (\#C)$
$D + E \xrightarrow{k_3} E + F$	$(\#D) \times (\#E)$
$F \xrightarrow{k_4} D + G$	$(\#F)$
$E + G \xrightarrow{k_5} A$	$(\#E) \times (\#G)$

Table 11.1: Example reactions used in the text and their redundancy functions h_r.

t	0	.0001	.004	.0015	.00177	.00178	.0038	.0040	.0057	.00575	.00578
$\#A$	10	9	8	8	9	8	7	7	6	7	7
$\#B$	8	7	6	5	5	4	3	3	2	2	2
$\#C$	7	8	9	8	8	9	10	10	11	11	11
$\#D$	3	3	3	4	4	4	4	3	3	3	4
$\#E$	6	6	6	6	5	5	5	5	5	4	4
$\#F$	8	8	8	8	8	8	8	9	9	9	8
$\#G$	12	12	12	12	11	11	11	11	11	10	11
μ	—	1	1	2	5	1	1	3	1	5	4
h_1	80	63	48	40	45	32	21	21	12	14	14
h_2	56	56	54	40	40	36	30	30	22	22	22
h_3	18	18	18	24	20	20	20	15	15	12	16
h_4	8	8	8	8	8	8	8	9	9	9	8
h_5	72	72	72	72	55	55	55	55	55	40	44
ρ_1	.004	.022	.064	—	—	.066	.040	—	.007	—	—
ρ_2	.081	—	—	.219	—	—	—	—	—	—	—
ρ_3	.078	—	—	—	—	—	—	.036	—	—	—
ρ_4	.048	—	—	—	—	—	—	—	—	—	.116
ρ_5	.128	—	—	—	.219	—	—	—	—	.337	—
ς_1	.004	.025	.083	.071	.080	.123	.120	.120	.075	.087	.087
ς_2	.081	.081	.080	.278	.278	.271	.248	.248	.202	.202	.202
ς_3	.078	.078	.078	.087	.080	.080	.080	.096	.096	.079	.102
ς_4	.048	.048	.048	.048	.048	.048	.048	.052	.052	.052	.162
ς_5	.128	.128	.128	.128	.316	.316	.316	.316	.316	.567	.590
τ_1	.0001	.0004	.0017	.0018	.0018	.0038	.0057	.0057	.0063	.0062	.0062
τ_2	.0014	.0014	.0015	.0070	.0070	.0075	.0083	.0083	.0092	.0092	.0092
τ_3	.0043	.0043	.0043	.0036	.0040	.0040	.0040	.0064	.0064	.0066	.0064
τ_4	.0060	.0060	.0060	.0060	.0060	.0060	.0060	.0058	.0058	.0058	.0203
τ_5	.0018	.0018	.0018	.0018	.0058	.0058	.0058	.0058	.0058	.0142	.0134

Table 11.2: Example execution of NRM′.

generate a uniform [0,1] random variate u and apply the inverse-generating method [18,26]: $\rho = \frac{1}{a}\ln\left(\frac{1}{u}\right)$. This fact was not explicitly mentioned in the discussion of NRM.

We shall only consider cases where $a_r \neq 0$ (and hence $h_r \neq 0$). The zero cases follow easily.

Lemma 3 *The τ_rs generated in Step 1 of NRM′ follow the correct distributions.*

Proof. Consider a certain reaction r. In NRM, τ_r is generated according to exponential distribution with parameter a. By the remark above, this is equivalent to generating a uniform [0,1] random variate u and letting $\tau_r = \frac{1}{a}\ln\left(\frac{1}{u}\right)$. By arithmetic, this is the same as generating u and letting $\tau_r = \frac{1}{h}\left[\frac{1}{k}\ln\left(\frac{1}{u}\right)\right]$. The part inside the brackets is accomplished in Step 1d, the multiplication by $1/h$ in Step 1e. As r was arbitrary, the lemma holds. ■

Lemma 4 *The τ_rs generated in Step 3 of NRM′ follow the correct distributions.*

Proof. By induction, we shall show that the τ_rs follow the correct distributions and that $\tau_r \times h_r = \varsigma_r$. (Remember we are only considering the case where $h_r \neq 0$.)

The previous lemma established the base case. For the inductive case, assume both parts of the induction hypothesis hold before Step 3.

Case 6 *$r \neq \mu$, and h_r remains constant.*

Then a_r remains constant. Further, Steps 3a and 3d cancel each other out, so ς_r remains constant. Thus τ_r at the end of Step 3 is equal to τ_r at the beginning of Step 3, as it should be.

Case 7 *$r \neq \mu$, and h_r changes.*

Denoting the new values with primes, $\tau'_r = \varsigma'_r/h'_r$ after Step 3e (this is half the induction). Further, $\varsigma'_r = \varsigma_r - h_r \times t + h'_r \times t$ by Steps 3a and 3d. So,

$$\begin{aligned}
\tau'_r &= \varsigma'_r/h'_r \\
&= \frac{1}{h'_r}\left(\tau_r \times h_r - h_r \times t + h'_r \times t\right) \\
&= \frac{h_r}{h'_r}(\tau_r - t) + t \\
&= \frac{a_r}{a'_r}(\tau_r - t) + t
\end{aligned}$$

The second equality follows from the induction hypothesis. The third follows from algebra. The last is simply a restatement of $a = k \times h$. Note that this final result is precisely the transformation used by NRM.

Case 8 $r = \mu$

Denoting the new values with primes, $\tau'_\mu = \varsigma'_\mu / h'_\mu$ after Step 3e (again, this is half the induction). Then

$$\begin{aligned}
\tau'_\mu &= \varsigma'_\mu / h'_\mu \\
&= \frac{1}{h'_\mu}\left(\frac{1}{k_\mu}\ln\left[\frac{1}{u}\right] + h'_\mu \times t\right) \\
&= \frac{1}{a'_\mu}\ln\left[\frac{1}{u}\right] + t
\end{aligned}$$

The second equality follows from Steps 3c, d, and e. The final equality follows from algebra and the identity $a = k \times h$. By our previous comment on generating exponential random numbers, τ'_μ follows an exponential distribution with parameter a'_μ, plus t, as in NRM.

These three cases cover all possibilities, so the lemma holds. ■

From the two lemmas, it is immediately clear that NRM$'$ is equivalent to NRM. The former is simply a different way of specifying precisely the same steps as the latter, minus some of the optimizations for efficiency. (Such optimizations should not affect correctness.)

Why introduce NRM$'$ at all? Basically, NRM$'$ isolates the parts of the code that have knowledge of the rate constants k_r. The next section will introduce NRM^{-1}, an inverse to NRM$'$. In particular, NRM^{-1} will not have knowledge of the rate constants k_r, so it will not be able to retrieve the random numbers u, rather only the exponential random numbers labeled ρ_r in Table 11.2.

11.3 NRM^{-1}

NRM and NRM$'$ are equivalent simulation algorithms that transform random numbers into trajectories. The current section introduces a parameter estimation algorithm, NRM^{-1}, that transforms trajectories into random numbers and uses those random numbers in an optimal way to estimate the parameters (i.e., rate constants) of the system.

In what follows, it will be assumed that NRM^{-1} knows what reactions are present, but does not know the rate constants k_r. (Section 11.6 will come back to the case where one does not know the reactions either.) Because NRM^{-1} does not know the rate constants, it cannot hope to recover the random numbers u in Steps 1d and 3c of NRM$'$. (We use NRM$'$ rather than NRM as a starting point, because NRM$'$ isolates the parts of the code that require knowledge of rate constants.) Rather, NRM^{-1} takes as input t and $\#X_c$ for each chemical type c, and outputs the random numbers ρ_r. For now, assume the inputs are a vector of t values and a vector of $\#X_c$ values for each chemical type c, which includes each discrete event. For example, the first two parts of Table 11.2 could be input to NRM^{-1}. These restrictions will be relaxed later.

Algorithm 19 *(NRM^{-1})*

1. *Input $\overrightarrow{t}$ and $\overrightarrow{\#X_c}$ for each chemical type c.*
2. *Calculate $\overrightarrow{\mu}$ as well as possible.*
3. *Calculate $\overrightarrow{h}$.*
4. *Calculate $\overrightarrow{\varsigma_r}$ for each reaction r as well as possible.*
5. *Calculate $\overrightarrow{\rho_r}$ for each reaction r.*
6. *Calculate estimates $\widehat{k_r}$, which approximate k_r, for each reaction r.*

Step 1 is self-explanatory.

Step 2: By assumption, the inputs contain *every* discrete event. A given discrete event will look something like $[\#A, \#B, \ldots, \#G] \to [\#A, \#B, \ldots, \#G] + \Delta$, where Δ is a signature of a given reaction. (For example, the first event in Table 11.2 — the change from Column 1 to Column 2 — has $\Delta = [-1, -1, +1, 0, 0, 0, 0]$, corresponding to the reaction $A + B \to C$, which implies $\mu = 1$.)

It is possible there could be some ambiguity here. For example, suppose one of the possible reactions is $A + B \xrightarrow{k_1} C$ and another is $A + B \xrightarrow{k_2} C$. Because both reactions have the same signature, NRM^{-1} is unable to disambiguate them. In such a case, we would suggest grouping the reactions into $A + B \xrightarrow{k} C$, where $k = k_1 + k_2$, and estimating k. To disambiguate k_1 and k_2, one would have to do some experimental manipulation that changes k_1 and k_2 independently, then estimate k again, and use the combination of the k estimates to separate k_1 and k_2.

This kind of reaction ambiguity could also come up, for example, in the set of reactions $A + E_1 \rightarrow B + E_1$ and $A + E_2 \rightarrow B + E_2$, i.e., the same reaction catalyzed by two different enzymes. However, one could disambiguate by breaking these reactions into elementary steps such as $A+E_1 \rightarrow A \bullet E_1 \rightarrow B+E_1$, etc. Again, one would have to do an experimental manipulation — one should measure the amount of chemical species $A \bullet E_1$, etc.

Up to the kind of ambiguity described, one can identify which reaction μ occurred. In particular, the way to get around the ambiguities is to redesign the experiment, which is not an estimation issue.

Step 3: The $\overrightarrow{h_r}$s follow directly from the $\overrightarrow{\#X_c}$s, just as they did in NRM$'$.

Step 4: This is the key step of NRM^{-1}. Assume that $\overrightarrow{t}$ and $\overrightarrow{\#X_c}$ (and hence $\overrightarrow{h_r}$) are indexed from 1 to M, and μ is indexed from 2 to M. The key is to calculate values $\widehat{\varsigma_r}(1)$ to $\widehat{\varsigma_r}(M)$, which will be shown to equal $\varsigma_r(1)$ to $\varsigma_r(M)$ up to some end effects.

- **Algorithm 20**

```
1. For each reaction $r$, let $\widehat{\varsigma_r}(M) \leftarrow \emptyset$.
2. For $i = M - 1$ downto $1$
    (a) if $r = \mu(i+1)$
            $\widehat{\varsigma_r}(i) \leftarrow t(i+1) \times h_r(i)$
    (b) else if $\widehat{\varsigma_r}(i+1) = \emptyset$
            $\widehat{\varsigma_r}(i) \leftarrow \emptyset$
    (c) else
            $\widehat{\varsigma_r}(i) \leftarrow \widehat{\varsigma_r}(i+1) + t(i+1) \times h_r(i) - t(i+1) \times h_r(i+1)$
```

Table 11.3 shows the result of running Algorithm 20 on the example of the previous section.

How does this work?

Basically, NRM$'$ does the following:

- iterates forward through time,
- outputs $t = \tau_\mu$ at Step 2d,

$\widehat{\varsigma}_1$	0.004	0.025	0.083	0.071	0.080	0.123	0.120	0.120	$\emptyset$	$\emptyset$	$\emptyset$
$\widehat{\varsigma}_2$	0.081	0.081	0.080	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$\widehat{\varsigma}_3$	0.078	0.078	0.078	0.087	0.080	0.080	0.080	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$\widehat{\varsigma}_4$	0.048	0.048	0.048	0.048	0.048	0.048	0.048	0.052	0.052	0.052	$\emptyset$
$\widehat{\varsigma}_5$	0.128	0.128	0.128	0.128	0.316	0.316	0.316	0.316	0.316	$\emptyset$	$\emptyset$

Table 11.3: Inverse values calculated by NRM^{-1}.

- generates a single new random number ς_μ, at Step 3c,
- transforms the existing ς_r values for all $r \neq \mu$ in Steps 3a and 3d.

NRM^{-1} inverts these steps. It

- iterates backward through time,
- inputs $t = \tau_\mu$,
- uses $t = \tau_\mu$ to get a single random number $\widehat{\varsigma}_r \leftarrow t \times h_r$ in Step 2a,
- undoes the NRM$'$ transformation $\widehat{\varsigma}_r(new) \leftarrow \widehat{\varsigma}_r(old) - t(new) \times h_r(old) + t(new) \times h_r(new)$ (Steps 3a and 3d of NRM$'$), giving $\widehat{\varsigma}_r(old) \leftarrow \widehat{\varsigma}_r(new) + t(new) \times h_r(old) - t(new) \times h_r(new)$ (Step 2c of NRM^{-1}).

NRM^{-1} does not have knowledge of the last random number used for each reaction, because random numbers only become available to trajectories as they are converted into t values, i.e., when events occur. Said another way, the last random number used for each reaction is part of the internal state of NRM$'$, not part of the output trajectory. To signify this, NRM^{-1} uses the value $\emptyset$ for $\widehat{\varsigma}_r$s whose value is unknown, and carries the value $\emptyset$ from right to left until the first (i.e., last in time) occurrence of the reaction r.

Note that $\widehat{\varsigma}_r$ agree with ς_r, i.e., all non-$\emptyset$ values of $\widehat{\varsigma}_r$ equal the corresponding ς_r values. Thus the notation $\widehat{\varsigma}_r$ is slightly misleading — it might imply to some readers that $\widehat{\varsigma}_r$ is simply an estimate, not an actual value. In fact, $\widehat{\varsigma}_r$ is a calculated value, which is an internal variable of the estimation algorithm, and agrees with the corresponding value of the simulation algorithm.

Step 5: This step inverts Steps 3c and 3d to get ρ_r values.

Algorithm 21

L_1	$\{0.004, 0.022, 0.064, 0.066, 0.040\}$
L_2	$\{0.081\}$
L_3	$\{0.078\}$
L_4	$\{0.048\}$
L_5	$\{0.128, 0.219\}$

Table 11.4: L_r values generated by NRM^{-1}.

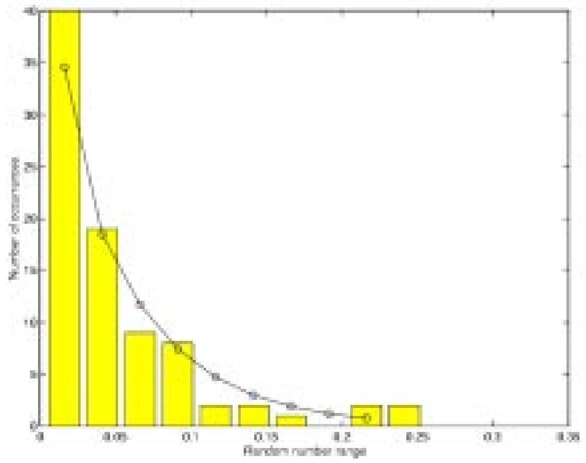


Figure 11-1: Histogram of L_2 values, with predicted line overlaid.

```
1. For each reaction $r$, let $L_r \leftarrow \{\widehat{\varsigma_r}(1)\}$
2. For $i$ = 2 to $M$
    (a) $\mu \leftarrow \overrightarrow{\mu}(i)$
    (b) $L_\mu \leftarrow L_\mu \cup \{\widehat{\varsigma_\mu}(i) - t(i) \times h_\mu(i)\}$
```

For each reaction r, the list L_r will contain ρ_r values. Step 1 records the initial ρ_r values (from Step 1d of NRM$'$). Each subsequent i records the single new random number ρ_μ generated during the ith iteration. Step 2b is the inverse of NRM$'$ Step 3d. (Note that all $\widehat{\varsigma_r}$ values for $r \neq \mu$ are transformed values of previous $\widehat{\varsigma_r}$ values and are not statistically independent random numbers, so they are ignored.)

At the end of Step 5, we are left with a list L_r for each reaction r of the ρ_r values used (minus the last ρ_r value for each r). Table 11.4 shows this for the running example. Recall that the original ρ_rs were statistically independent random numbers distributed according to an exponential distribution of parameter k_r.

Figure 11-1 expounds on this idea. Here we have run NRM$'$ on the same system with

Value	Reaction 1	Reaction 2	Reaction 3	Reaction 4	Reaction 5
Events	61	85	43	2	9
ξ_r	0.0917	0.0500	0.1493	0.0368	1.4846
$1/k_r$	0.0813	0.0549	0.1563	0.0667	1.1111
$1/(\sqrt{n}k_r)$	0.0104	0.0060	0.0238	0.0471	0.3704

Table 11.5: Sanity check for NRM^{-1}.

different initial conditions for 200 simulation events[1]. Of the 200 simulation events, 85 correspond to Reaction 2. Figure 11-1 shows a histogram of those 85 values, with the predicted curve for the actual k_r value superimposed.

Step 6: Informally, we could draw for each reaction r a histogram of the L_r values, then find a parameter $\widehat{k_r}$ that provides a good fit like the superimposed curve in Figure 11-1.

More formally, the L_r values constitute independent samples from an exponential distribution with parameter k_r. Estimating k_r, namely calculating $\widehat{k_r}$, from such samples is a standard problem in statistics [22, 35]. It turns out that

$$\xi_r = \frac{1}{n}\sum_{i=1}^{n} L_r(i)$$

is an efficient estimator for $1/k_r$, i.e., $E[\xi_r] = 1/k_r$ and $Var[\xi_r] = 1/(nk_r^2)$. It can be shown that any other unbiased estimator has at least this much variance. It can also be shown that for no other function of k_r does an efficient estimator exist (i.e., any estimator of another function of k_r will have variance strictly greater than the bound, not equal to it.)

Table 11.5 shows the results of a simple sanity check: 200 simulation events were run (as mentioned above). First, the breakdown of the 200 events by reaction. Second, the values of ξ_r calculated by NRM^{-1}. Third, the actual values of $1/k_r$, using the real k_r values (which are not known to NRM^{-1}). Finally, the predicted standard deviations of the estimates, namely the square roots of the variances, again using the real k_r values.

[1]The simple example system has the unfortunate property that the total number of molecules decreases with time, so different (larger) initial conditions were needed to generate 200 simulation events without running out of molecules. The original initial conditions times 20 were used.

11.3.1 Correctness

A proof of correctness proceeds by induction backward in time, showing that the steps of NRM^{-1} invert the steps of NRM$'$. The individual steps of that induction were explained above.

11.4 Estimation of Stochastic Parameters, the ESP Algorithm

This section presents the Estimation of Stochastic Parameters (ESP) algorithm. ESP is equivalent to NRM^{-1}, in fact, it consists of precisely the same mathematical steps, simply grouped in a different way.

Algorithm 22 *(ESP) For each reaction r,*

1. *Input $\overrightarrow{t}$ and the appropriate $\overrightarrow{\#X_c}$s.*
2. *Find those indices i such that $\overrightarrow{\mu}(i) = r$.*
3. *Calculate $\overrightarrow{h_r}$.*
4. *Calculate $\overrightarrow{\rho_r}$.*
5. *Calculate an estimate $\widehat{1/k_r}$.*

Notice the first key to the ESP algorithm — it separates reactions completely. In point of fact, NRM^{-1}contains a lot of parallel structures of the form "calculate something for all reactions r." The reader can verify that none of these parallel constructions in NRM^{-1}depend on each other — the variables corresponding to reaction r_1 only depend on other r_1 variables, not r_2 variables. So ESP amounts to filling in Tables 11.2 and 11.3 by rows, rather than by columns. Further, though, the details of Steps 1 to 5 differ from NRM^{-1}.

Step 1: Inputting $\overrightarrow{t}$ is self-explanatory. For any given reaction r, one only needs to input $\overrightarrow{\#X_c}$ for those chemicals c that are reactants of products of reaction r. For example, for the reaction $A + B \to C$, one need only input $\#A$, $\#B$, and $\#C$, not $\#D$, $\#E$, etc.

Step 2: One need not calculate $\overrightarrow{\mu}(i)$ for every $i \in 2 \dots M$. (Recall that $\overrightarrow{\mu}(1)$ is undefined, since $\overrightarrow{\mu}(i)$ is the reaction going from time $i - 1$ to time i.) Rather, one only needs to know whether $\overrightarrow{\mu}(i) = r$. This can be determined by comparing $\overrightarrow{\#X_c}(i) - \overrightarrow{\#X_c}(i-1)$ for those chemicals c in the reactants and products of reaction r, and comparing with the signature of the reaction.[2]

Step 3: The $\overrightarrow{h_r}$ array follows directly from the $\overrightarrow{\#X_c}$s of the reactants.

Step 4: The following algorithm is algebraically equivalent to Steps 4 and 5 of NRM^{-1}, as will be proved below:

Algorithm 23

```
1. Let $i_1 \leftarrow 1$, let $i_2, i_3, \ldots, i_I$ be the set of indices such that $\overrightarrow{\mu}(i_k) = r$.
2. $L_r \leftarrow \emptyset$.
3. For $k = 1$ to $I - 1$
    (a) $\widehat{\rho} = \sum_{j=i_k}^{i_{k+1}-1} \overrightarrow{h_r}(j) \times \left[\overrightarrow{t}(j+1) - \overrightarrow{t}(j)\right]$
    (b) $L_r \leftarrow L_r \cup \{\widehat{\rho}\}$
```

Theorem 7 *Algorithm 23 is equivalent to Steps 4 and 5 of NRM^{-1}, in the sense that L_r at the end of NRM^{-1} equals L_r at the end of Algorithm 23.*

The proof proceeds by a chain of equivalent algorithms, starting with Algorithms 20 and 21, leading to Algorithm 23. Recall that each of the following algorithms will be executed for all reactions r, by virtue of being within the main loop of the ESP algorithm.

Algorithm 24

```
1. Let $\widehat{\varsigma}_r(M) \leftarrow \emptyset$.
2. For $i = M - 1$ downto 1
    (a) if $r = \mu(i+1)$
            if $\widehat{\varsigma}_r(i+1) \neq \emptyset$ then $L_r \leftarrow L_r \cup \{\widehat{\varsigma}_r(i+1) - t(i+1) \times h_r(i+1)\}$
            $\widehat{\varsigma}_r(i) \leftarrow t(i+1) \times h_r(i)$
```

[2]In cases where this could lead to ambiguity, one should read in additional $\overrightarrow{\#X_c}$s to disambiguate. See also the discussion in Section 11.3.

```
    (b) else if $\widehat{\varsigma}_r(i+1) = \emptyset$
            $\widehat{\varsigma}_r(i) \leftarrow \emptyset$
    (c) else
            $\widehat{\varsigma}_r(i) \leftarrow \widehat{\varsigma}_r(i+1) + t(i+1) \times h_r(i) - t(i+1) \times h_r(i+1)$
3. if $\widehat{\varsigma}_r(1) \neq \emptyset$ then $L_r \leftarrow L_r \cup \{\widehat{\varsigma}_r(1)\}$
```

Lemma 5 *Steps 4 and 5 of NRM^{-1} are equivalent to Algorithm 24.*

Proof. Algorithm 24 is essentially Algorithm 20, with extra lines to add elements to L_r. In particular, the $\widehat{\varsigma}_r(i+1) - t(i+1) \times h_r(i+1)$ values added are the same as the values added by Algorithm 21, with a slight difference in indices. In Algorithm 21, $r = \mu(i)$, so the correct index to use is i. Here, $r = \mu(i+1)$, so the correct index to use is $i+1$. The final element added, in Step 3, is the first element added in Step 1 of Algorithm 21. Note that Algorithm 24 adds elements to L_r in the reverse order of Algorithm 21. ■

Algorithm 25 *For $k = I$ downto 2*

```
1. $\widehat{\varsigma}_r(i_k - 1) \leftarrow t(i_k) \times h_r(i_k - 1)$
2. For $j = i_k - 2$ downto $i_{k-1}$
       $\widehat{\varsigma}_r(j) \leftarrow \widehat{\varsigma}_r(j+1) + t(j+1) \times h_r(j) - t(j+1) \times h_r(j+1)$
3. $L_r \leftarrow L_r \cup \{\widehat{\varsigma}_r(i_{k-1}) - t(i_{k-1}) \times h_r(i_{k-1})\}$
```

Lemma 6 *Algorithm 25 is equivalent to Algorithm 24.*

Proof. The equivalence comes from the definitions of the i_ks: $i_1 = 1$, and for all $k > 1$, $\overrightarrow{\mu}(i_k) = r$. First, it is no longer necessary to use the dummy value $\emptyset$; that value is for indices $j > i_I$ only. For $k > 2$, and index $j = i_k - 1$, $\overrightarrow{\mu}(j+1) = \overrightarrow{\mu}(i_k) = r$, so Algorithm 24 executes Step 2a. Ignoring (for now) the part about L_r, this is equivalent to Step 1 of Algorithm 25. For indices $j = i_k - 2$ downto i_{k-1}, Algorithm 24 executes Step 2c, which is equivalent to the body of the loop in Step 2 of the current Algorithm. Finally, for index $j = i_{k-1} - 1$, $\overrightarrow{\mu}(j+1) = \overrightarrow{\mu}(i_{k-1}) = r$, so Algorithm 24 executes Step 2a, which puts the term $\{\widehat{\varsigma}_r(i_{k-1} - 1 + 1) - t(i_{k-1} - 1 + 1) \times h_r(i_{k-1} - 1 + 1)\}$ into L_r. This is the part of Step 2a we ignored before. Finally, Step 3 of Algorithm 24 is just the special case where $k = 2$, as $i_{k-1} = i_{2-1} = i_1 = 1$, by definition of i_1. ■

Lemma 7 *Algorithm 25 is equivalent to Step 4 of ESP (i.e., to Algorithm 23).*

Proof. Let x be the quantity that is added into L_r in Step 3 of Algorithm 25. Then

$$\begin{aligned}
x &= t(i_k) \times h_r(i_k - 1) + \sum_{j=i_{k-1}}^{i_k-2} [t(j+1) \times h_r(j) - t(j+1) \times h_r(j+1)] - t(i_{k-1}) \times h_r(i_{k-1}) \\
&= t(i_k) \times h_r(i_k - 1) + \sum_{j=i_{k-1}}^{i_k-2} [t(j+1) \times h_r(j)] \\
&\quad - \sum_{j=i_{k-1}}^{i_k-2} [t(j+1) \times h_r(j+1)] - t(i_{k-1}) \times h_r(i_{k-1}) \\
&= \sum_{j=i_{k-1}}^{i_k-1} [t(j+1) \times h_r(j)] - \sum_{u=i_{k-1}+1}^{i_k-1} [t(u) \times h_r(u)] - t(i_{k-1}) \times h_r(i_{k-1}) \\
&= \sum_{j=i_{k-1}}^{i_k-1} [t(j+1) \times h_r(j)] - \sum_{u=i_{k-1}}^{i_k-1} [t(u) \times h_r(u)] \\
&= \sum_{j=i_{k-1}}^{i_k-1} [t(j+1) \times h_r(j) - t(j) \times h_r(j)] \\
&= \sum_{j=i_{k-1}}^{i_k-1} h_r(j) \times [t(j+1) - t(j)]
\end{aligned}$$

The first equality follows directly from Algorithm 25. The second follows from splitting the sum. The third follows from combining the first term with the first sum and using $u = j + 1$. The fourth follows from combining the second sum and the final term. The fifth follows from combining the two sums, using the index j. The final equality follows by rearrangement, and is precisely the equation used in ESP. ■

Proof. (Theorem 7) By a chain of equivalences, using the lemmas above: NRM^{-1} Steps 4 and 5 are equivalent to Algorithm 24, which is equivalent to Algorithm 25, which is equivalent to ESP Step 4. ■

Step 5: Use precisely the same step as NRM^{-1}.

11.5 Applying ESP to Lambda

Figure 11-2 shows ESP applied to test data generated from the lambda model. The input data for ESP is data for a single reaction only, namely the reaction cro•cro $\longrightarrow$ cro + cro. The values $\overrightarrow{t}$, $\overrightarrow{\#cro \bullet cro}$, and $\overrightarrow{\#cro}$ are inputs. ESP was run on a single trajectory, of

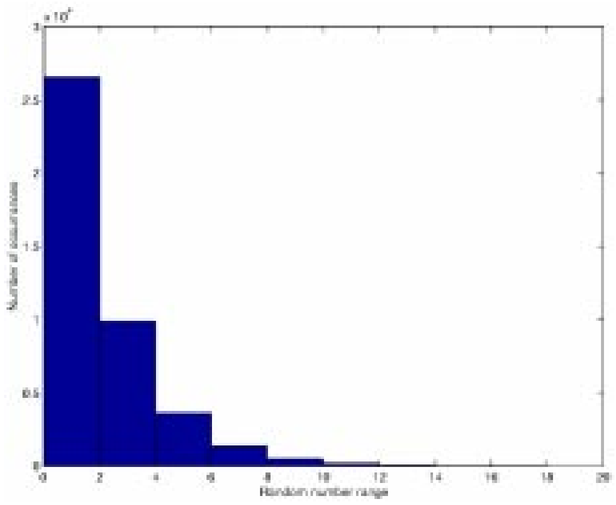


Figure 11-2: Histogram of inverse values for lambda model.

which 42,000 of the events correspond to this particular reaction (the reaction was chosen in part because it occurs so frequently). The L_r values are histogrammed in Figure 11-2. The estimated value, $\widehat{1/k_r}$, is 1.9993. For contrast, the actual value of $1/k_r$ is 2.0, and the predicted standard deviation is 0.0097.

Note that ESP only used two chemical species as input, out of the 57 possible chemical species in the model. This key separability issue will be discussed in Section 11.6.

11.6 Results

This section summarizes the key non-algorithmic issues involved.

11.6.1 Putting it All Together

So far we have introduced NRM, NRM$'$, NRM^{-1}, and ESP. They have the following properties:

1. The Next Reaction Method, NRM, is an exact stochastic simulation algorithm equivalent to the chemical master equation approach.

2. NRM$'$ is a simplified version of NRM that is entirely equivalent.

3. NRM^{-1} is an inverse of NRM$'$.

4. ESP is equivalent to NRM^{-1}.

Putting this all together, one can apply the ESP algorithm to a trajectory of a chemical stochastic process (i.e., a process that obeys a chemical master equation). This results in lists of statistically independent random numbers, such that the elements of list L_r are distributed according to an exponential distribution with parameter k_r. ESP uses standard methods from statistics to estimate k_r from L_r.

11.6.2 Statistics

Given n samples $x_1, \ldots x_n$ of an exponential distribution with parameter k_r, the formula

$$\widehat{1/k_r} = \frac{1}{n} \sum_{j=1}^{n} x_j \tag{11.1}$$

is an unbiased, efficient estimator for $1/k_i$, in the sense of the Cramer-Rao bound [22, 35]. In other words, $E[\widehat{1/k_r}] = 1/k_r$ for all n, and the variance of $\widehat{1/k_r}$ achieves the lower bound established by Cramer-Rao, in this case $Var[\text{any unbiased estimator}] \geq 1/(nk_r^2)$. It can be shown that no unbiased estimator can do better, and that for exponentials, one cannot achieve the lower bound in estimating k_r, but only in estimating $1/k_r$.

So, in other words, given a trajectory with n occurrences of reaction r, ESP will give (in probability) $1/k_r(1 \pm 1/\sqrt{n})$. One should be careful in reading the previous sentence, though, as it is easy to bias one's choice of trajectories in such a way as to negate this result.

11.6.3 Interpretation of Results

- If there is only a single reaction, the summation in ESP simplifies to $h_r(j) \times [t(i_k) - t(i_{k-1})]$. In other words, in the absence of other reactions, just take the time difference between subsequent occurrences of the reaction, multiply by the redundancy function, and that will estimate $1/k_r$.

  The usefulness of the full summation in ESP is that it tells you exactly how to deal with a reaction that is part of a pathway and can be affected by outside reactions. (Note that one need not have full knowledge of the other reactions, however, as will be discussed below.)

- The variance bound in the previous section tells you how much data you need to get an estimate of a fixed uncertainty. For example, if you want to estimate a parameter to 10%, you will need 100 data points.

11.6.4 Separability

It is not immediately clear that one can separate arbitrary systems into their constituent reactions for the purpose of estimation. For systems where one can observe stochastic events, we have proven that it *is* possible, using ESP.

For deterministic systems phrased as differential equations whose variables are concentrations, it is certainly not true that one can separate things so easily. In a coupled deterministic system of n variables, one has three choices:

- measure all the $[X]$s and do an n-dimensional estimation,
- control certain $[X]$s,
- pull out a given reaction and run it *in vitro* (in a test tube).

Each of these approaches is troublesome:

- In a large system, the number of variables may be very large — the simple lambda system had 75 equations in 57 chemical species. It is very difficult to run an experiment where one has to measure 57 variables simultaneously. Further, if there are unmodeled dynamics, i.e., unknown reactions, that affect the 57 chemical species, those reactions may disturb or mess up the estimation procedure.
- Controlling concentrations is also hard experimentally, and also suffers from the problem of unmodeled dynamics.
- In a complex system, the ks may depend on electrolyte concentration, activity (particularly at high local concentrations, or if spatial effects are important locally), etc. One loses all of these variables *in vitro*, so the constants estimated must be taken with a grain of salt.

Thus, for deterministic systems, one is left to choose between several alternatives, none of which is ideal.

The key of ESP is that if one can measure stochastic events, everything changes. One can measure rate constants separately (for example, the rate constant from lambda, estimated above, involved knowledge of two chemical species only). One can do a series of relatively simple experiments, measuring 3 or 4 species at a time; each of these simple experiments will give one rate constant. (Presumably it is easier to do 75 3-variable experiments than one 75-variable experiment; this process might even be automatable.) There is no reason one could not do such an experiment *in vivo*: for deterministic systems, it was necessary to remove the effects of unmodeled dynamics, so experiments were run *in vitro*, for stochastic systems, ESP already takes care of unmodeled dynamics. If one wants to measure the rate constant k in $A + B \xrightarrow{k} C$, one need only measure $\#A$, $\#B$, and $\#C$; the algorithm shows how to correct for unmodeled reactions *without complete knowledge of those reactions*, just how they affect $\#A$, $\#B$, and $\#C$. There is no *a priori* knowledge required — one can simply observe how the observed variables change and correct for such changes. One could, in principle, run ESP without even knowing what reactions are present. One would simply observe discrete events, and those would define the reactions. It will typically be preferable, however, to know what one is looking for, because then one need only observe a subset of the chemical species.

It is certainly desirable to measure stochastic events. The question remains: is it possible? At this writing, no one has done such an experiment for chemical reactions. However, the ion channel community has been measuring stochastic events involving single ion-channel molecules for decades [33]. More recently, biophysicists have measured biological motors, which behave according to stochastic dynamics [31]. In light of how useful it could potentially be, measuring stochastic chemical reaction events is an area that should be seriously considered by experimentalists.

Part VI

Conclusions

Chapter 12 Summary

The preceding chapters have presented three main ideas:

- NRM, an exact, efficient simulation algorithm,
- MNRM and MESA, efficient algorithms for sensitivity analysis, and
- ESP, a parameter estimation algorithm for mesoscopic chemistry.

These three can be used together to create and refine models of chemical processes. Basically, the algorithm for doing so goes like this:

1. Write down a system of chemical equations that have been determined experimentally.
2. Augment with suspected reactions.
3. Pick some set of parameter values, either:
   (a) Use values reported in the literature
   (b) Run *in vitro* experiments
   (c) Use the ESP algorithm to measure and estimate constants *in vivo*, or
   (d) Guess or estimate the parameters.
4. Use NRM to simulate the system.
5. If the simulation is way off, check the set of reactions and go to Step 2.
6. Use MNRM and MESA to figure out how sensitive the model is to its parameters.
7. For those parameters that the model is sensitive to, use ESP to generate better estimates.
8. Go to Step 4.

One can iterate this until the model matches the results of experiment. Further, one can then predict and run experiments that were not used in forming the model. For example, one can create a model entirely from biochemical rate data, and check its predictions on system-level data, such as lysis versus lysogeny in lambda.

So, in short, by iterating between the forward algorithms (NRM, MNRM, and MESA) and the inverse algorithms (ESP and experiments) one can create increasingly good biological models that are based on actual structure, not parameter fitting.

Chapter 13 Research Directions

There are numerous areas for further study.

- Using the tools of mesoscopic chemistry to model more biological systems, and more complex biological systems.
- Developing formal reasoning techniques for simplifying models — for figuring out which parts of the model are most important, and which are not. (MNRM and MESA are numerical techniques, but provide little in the way of reasoning tools.)
- Adapting techniques such as singular value perturbation methods, separation of time scale, bifurcation analysis, and so on, that have been developed within the differential equations framework, to work with stochastic mesoscopic models.
- Formalizing the tie-in between stochastic, mesoscopic chemistry and deterministic, macroscopic chemistry. For example, how many molecules does one need to have before it is legitimate to ignore stochastic effects?
- Developing intermediate formalisms between stochastic and deterministic. Some interesting work has been done using Langevin and Fokker-Planck equations, but little has been done to determine when it is legitimate to use which formalism. Also, there may be room between the full mesoscopic chemistry and Langevin/Fokker-Planck formalisms.
- Developing techniques for multi-mode or hybrid systems, systems where part behaves stochastically and part deterministically, or more generally, where different parts are considered in different frameworks.
- Formalizing the equilibrium assumptions in earlier chapters. When is it legitimate to separate out certain parts of a system and declare them to be in equilibrium?
- How can one use position dependent information (diffusion, proteins bound to the membrane) in mesoscopic chemistry?

The first question is obviously biological in nature. The remainder are theoretical and mathematical. The glue that ties them together will be large-scale computational frameworks such as the techniques employed in this thesis. In particular, the typical biological user of these computational techniques should be presented with a user interface that abstracts away the details; he or she need not care about the underlying theoretical details, in the same way that the user of a numerical package such as Matlab does not have to know the details of numerical techniques for finding eigenvalues.

Bibliography

[1] Ackers, G. K., Johnson, A. D., and Shea, M. A. *Quantitative model for gene regulation by λ phage*, Proceedings of the National Academy of Sciences, 79, 1129–1133 (1982).

[2] Alon, U., Surette, M. G., Barkai, N., and Leibler, S. *Robustness in bacterial chemotaxis*, Nature, 397, 168–171 (1999).

[3] Arkin, A. P., Ross, J., and McAdams, H. *Stochastic kinetic analysis of developmental pathway bifurcation in phage lambda-infected Escherichia coli cells*, Genetics, 149, 1633–1648 (1998).

[4] Atkins, P. W. *Physical Chemistry*, 6th edition. Freeman: New York (1998).

[5] Barkai, N. and Leibler, S. *Robustness in simple biochemical networks*, Nature, 387, 913–917 (1997).

[6] Bray, D. *Reductionism for biochemists: how to survive the protein jungle*, Trends in Biochemical Sciences, 22, 325–326 (1997).

[7] Burden, R. L. and Faires, J. D. *Numerical Analysis*, Fifth edition. PWS Publishing Company: Boston, Massachusetts (1993).

[8] Cormen, T. H., Leiserson, C. E., and Rivest, R. L., *Introduction to Algorithms*, The MIT Press and McGraw-Hill Book Company: Cambridge, Massachusetts and New York, New York (1990).

[9] Elowitz, M. B., Surette, M. G., Wolf, P. E., Stock, J. B. and Leibler, S., *Protein mobility in the cytoplasm of Escherichia coli*, Journal of Bacteriology, 181, 197–203 (1999).

[10] Feller, W. *An Introduction to Probability Theory and Its Applications: Volume I*, 3rd Edition. John Wiley and Sons, Inc.: New York (1966).

[11] Fermi, E. *Thermodynamics*, Dover Publications: New York (1956).

[12] Gibson, M. A. and Bruck, J. *Efficient exact stochastic simulation of chemical systems with many species and many channels*, Journal of Physical Chemistry A, 104, 1876–1889 (2000).

[13] Gibson, M. A. and Bruck, J. *A probabilistic model of a prokaryotic gene and its regulation*, in *Computational Modeling of Genetic and Biochemical Networks*, Bower, J. and Bolouri, H., eds. The MIT Press: Cambridge, Massachusetts (in press).

[14] Gibson, M. A. and Mjolsness. E. *Modeling the Activity of Single Genes*, in *Computational Modeling of Genetic and Biochemical Networks*, Bower, J. and Bolouri, H., eds. The MIT Press: Cambridge, Massachusetts (in press).

[15] Gillespie, D. T. *A general method for numerically simulating the stochastic time evolution of coupled chemical reactions*, Journal of Computational Physics, 22, 403–434 (1976).

[16] Gillespie, D. T. *Exact stochastic simulation of coupled chemical reactions*, Journal of Physical Chemistry, 81, 2340–2361 (1977).

[17] Gillespie, D. T. *A theorem for physicists in the theory of random variables*, American Journal of Physics, 51, 520–533 (1983).

[18] Gillespie, D. T. *Markov Processes: An Introduction for Physical Scientists*, Academic Press: San Diego, California (1992).

[19] Gillespie, D. T. *A rigorous derivation of the chemical master equation*, Physica A, 188, 404–425 (1992).

[20] Gillespie, D. T. *Describing the state of a stochastic process*, American Journal of Physics, 66, 533–536 (1998).

[21] Gillespie, D. T. and Mangel, M. *Conditioned averages in chemical kinetics*, Journal of Chemical Physics, 75, 704–709 (1981).

[22] Gottschalk, T. D. *A short overview of statistical inference*, Course notes, Caltech.

[23] Hendrix, R. W. *et al.*, eds. *Lambda II*, Cold Spring Harbor Laboratory: Cold Spring Harbor, N.Y. (1983).

[24] Jansen, A. P. J. *Monte carlo simulations of chemical reactions on a surface with time-dependent reaction-rate constants*, Computer Physics Communications, 86, 1–12 (1995).

[25] Khalil, H. K. *Nonlinear Systems*, Second edition. Prentice Hall, Inc.: Upper Saddle River, New Jersey (1996).

[26] Leon-Garcia, A. *Probability and Random Processes for Electrical Engineering*, Addison-Wesley Publishing Company: Reading, Massachusetts (1994).

[27] Lukkien, J. J., Segers, J. P. L., Hilbers, P. A. J., Gelten, R. J., and Jansen, A. P. J. *Efficient monte carlo methods for the simulation of catalytic surface reactions*, Physical Review E, 58, 2598–2610 (1998).

[28] Matias, Y., Vitter, J. S. and Ni, W. C. *Dynamic generation of discrete random variates*, Technical Report, Bell Labs (1997).

[29] McAdams, H. H. and Arkin, A. P. *Simulation of prokaryotic genetic circuits*, Annual Review of Biophysics and Biomolecular Structure, 27, 199–224 (1998).

[30] McQuarrie, D. A. *Stochastic approach to chemical kinetics*, Journal of Applied Probability, 4, 413–478 (1967).

[31] Mehta, A. D., Rief, M., Spudich, J. A., Smith, D. A. and Simmons, R. M. *Single-molecule biomechanics with optical methods*, Science, 283, 1689–1695 (1999).

[32] Morton-Firth, C. J. *Stochastic Simulation of Cell Signalling Pathways*, Ph.D. Thesis, University of Cambridge (1998).

[33] Neher, E. and Sakmann, B. *Single-channel currents recorded from membrane of denervated frog muscle fibres*, Nature, 260, 799–802 (1976).

[34] Ptashne, M. *A Genetic Switch: Phage λ and Higher Organisms*, Second edition. Cell Press and Blackwell Scientific Publications: Cambridge, Massachusetts (1992).

[35] Rice, J. A. *Mathematical Statistics and Data Analysis*, Second edition. Duxbury Press, Wadsworth Publishing Company: Belmont, California (1995).

[36] Segers, J. *Algorithms for the Simulation of Surface Processes*, Ph.D. thesis, Eindhoven University of Technology (1999).

[37] Shea, M. A. and Ackers, G. K. *The o_R control system of bacteriophage lambda, a physical-chemical model for gene regulation*, Journal of Molecular Biology, 181, 211–230 (1985).

[38] van Kampen, N. G. *Stochastic Processes in Physics and Chemistry*, Revised and enlarged edition. Elsevier: Amsterdam, The Netherlands (1992).

[39] Watson, J. D., Hopkins, N. H., Roberts, J. W., Steitz, J. A., and Weiner, A. M. *Molecular Biology of the Gene*, Fourth edition. The Benjamin/Cummings Publishing Company, Inc.: Menlo Park, California (1987).

[40] Yi, T.-M., Huang, Y., Simon, M. I., and Doyle, J. *Robust perfect adaptation in bacterial chemotaxis through integral feedback control*, Proceedings of the National Academy of Sciences USA (in press).

[41] Zhou, K., Doyle, J. C. and Glover, K. *Robust and Optimal Control*, Prentice Hall: New Jersey (1996).