

# Systems Biology: Introduction to Pathway Modeling

---

First Python Edition

*Herbert M. Sauro  
University of Washington  
Seattle, WA*

This copy belongs to Eric Freeman



Ambrosius Publishing

Copyright © 2014-2018 Herbert M. Sauro. All rights reserved.

First Edition, version 1.15

Published by Ambrosius Publishing and Future Skill Software

[www.analogmachine.org](http://www.analogmachine.org)

Typeset using L<sup>A</sup>T<sub>E</sub>X 2<sub>&</sub>, TikZ, PGFPlots, WinEdt, InkScape, and  
11pt Math Time Professional 2 Fonts

pgf version is: 3.0.1a

Limit of Liability/Disclaimer of Warranty: While the author has used his best efforts in preparing this book, he makes no representations or warranties with respect to the accuracy or completeness of the contents of this book and specifically disclaim any implied warranties of merchantability or fitness for a particular purpose. The advice and strategies contained herein may not be suitable for your situation. Neither the author nor publisher shall be liable for any loss of profit or any other commercial damages, including but not limited to special, incidental, consequential, or other damages. No part of this book may be reproduced by any means without written permission of the author.

ISBN 13: 978-0-9824773-7-3 (paperback)

ISBN-10: 0982477376 (paperback)

Printed in the United States of America.

Mosaic image modified from Daniel Steger's Tikz image (<http://www.texample.net/tikz/examples/mosaic-from-pompeii/>)

Front-Cover: Cross-section through a single cell of Mycoplasma mycoides. Illustration by David S. Goodsell, the Scripps Research Institute, with permission.

University Disclaimer: Any views, opinions, data, documentation and other information presented in this book are solely those of the author and do not represent those of the University of Washington.

# ***Contents***

Preface . . . . .	x
Cover Image . . . . .	xii
<b>A Note about Software</b>	<b>xiii</b>
<b>Prologue</b>	<b>xv</b>
<b>1 Cellular Networks</b>	<b>1</b>
1.1 Overall Organization . . . . .	1
1.2 Network Representation . . . . .	2
1.3 Metabolic Networks . . . . .	2
1.4 Protein Networks . . . . .	4
1.5 Gene Regulatory Networks . . . . .	12
1.6 Genome Sizes . . . . .	15
1.7 <i>E. coli</i> . . . . .	18
1.8 Network Motifs . . . . .	22
Further Reading . . . . .	27
Exercises . . . . .	29
<b>2 Kinetics in a Nutshell</b>	<b>33</b>
2.1 Introduction . . . . .	33
2.2 Definitions . . . . .	33
2.3 Elementary Mass-Action Kinetics . . . . .	35
2.4 Chemical Equilibrium . . . . .	35
2.5 Mass-action and Disequilibrium Ratio . . . . .	37
2.6 Modified Mass-Action Rate Laws . . . . .	37
Further Reading . . . . .	38

<b>3 Stoichiometric Networks</b>	<b>39</b>
3.1 Stoichiometric Networks . . . . .	39
3.2 Standard Visualization Notation . . . . .	46
3.3 Mass-Balance Equations . . . . .	46
3.4 Stoichiometry Matrix . . . . .	54
3.5 Reversibility . . . . .	55
3.6 Signaling Networks . . . . .	57
3.7 Gene Regulatory Networks . . . . .	58
3.8 Moiety Conserved Cycles . . . . .	60
3.9 The System Equation . . . . .	63
3.10 Tellurium . . . . .	64
Further Reading . . . . .	65
Exercises . . . . .	66
<b>4 Introduction to Modeling</b>	<b>69</b>
4.1 Introduction . . . . .	69
4.2 Open, Closed, and Isolated Systems . . . . .	70
4.3 What is a Model? . . . . .	71
4.4 Building a Model . . . . .	74
4.5 Variables, Parameters and Absolute Constants . . . . .	77
4.6 Mathematical Descriptions of Models . . . . .	80
4.7 Example . . . . .	82
4.8 Dimensions and Units . . . . .	85
4.9 Classification of Models . . . . .	86
4.10 Linear and Nonlinear Models . . . . .	87
4.11 Linearization . . . . .	89
4.12 Approximations . . . . .	93
4.13 Example Model . . . . .	95
4.14 Where to get Data for Building Models . . . . .	96
4.15 Of Exactitude in Science . . . . .	99
Further Reading . . . . .	99
Exercises . . . . .	100

<b>5 Differential Equation Models</b>	<b>103</b>
5.1 Introduction . . . . .	103
5.2 Differential Equation Models . . . . .	103
5.3 Matlab Solvers . . . . .	115
5.4 Python Solvers . . . . .	116
5.5 Other Software . . . . .	117
5.6 Moiety Conserved Cycles . . . . .	119
5.7 Exploiting Fast Processes . . . . .	122
Further Reading . . . . .	129
Exercises . . . . .	130
<b>6 Stochastic Models</b>	<b>133</b>
6.1 Stochastic Kinetic Models . . . . .	133
6.2 Stochastic Kinetics . . . . .	134
6.3 Time to Reaction . . . . .	137
6.4 Running Stochastic Simulations . . . . .	140
6.5 Events at Regular Intervals . . . . .	142
6.6 Stochastic Trajectories . . . . .	143
Further Reading . . . . .	147
Exercises . . . . .	148
<b>7 How Systems Behave</b>	<b>149</b>
7.1 System Behavior . . . . .	149
7.2 Equilibrium . . . . .	149
7.3 Steady State . . . . .	152
7.4 Transients . . . . .	155
7.5 Setting up a Model in Software . . . . .	155
7.6 Robustness and Homeostasis . . . . .	156
Further Reading . . . . .	156
Exercises . . . . .	156
Tellurium Scripts . . . . .	158
<b>8 Multicompartmental Systems</b>	<b>159</b>
8.1 Multicompartment Systems . . . . .	159

8.2	Simple Diffusion . . . . .	159
8.3	Membrane Transporter Protein . . . . .	163
8.4	Three Compartment Model . . . . .	165
	Further Reading . . . . .	167
	Exercises . . . . .	168
<b>9</b>	<b>Fitting Models</b>	<b>169</b>
9.1	Introduction . . . . .	169
9.2	Optimization Algorithms . . . . .	174
9.3	Model Fitting Software . . . . .	197
9.4	Using Python to Fit Data . . . . .	198
	Further Reading . . . . .	201
	Exercises . . . . .	202
	Tellurium Scripts . . . . .	203
<b>10</b>	<b>Parameter Estimation</b>	<b>209</b>
10.1	Introduction . . . . .	209
10.2	Analysis of Residuals . . . . .	209
10.3	$\chi^2$ -Goodness of Fit Test . . . . .	213
10.4	Estimating Confidence Intervals . . . . .	218
10.5	Cross-validation . . . . .	222
10.6	Case studies . . . . .	224
10.7	Final Comments . . . . .	230
	Further Reading . . . . .	230
	Exercises . . . . .	231
	Tellurium Scripts . . . . .	231
<b>11</b>	<b>The Steady State</b>	<b>233</b>
11.1	Steady State . . . . .	233
11.2	Effect of Different Kinds of Perturbations . . . . .	237
11.3	Computing the Steady State . . . . .	239
11.4	Introduction to Stability . . . . .	250
11.5	Sensitivity Analysis . . . . .	252
	Further Reading . . . . .	253

Exercises . . . . .	253
Tellurium Scripts . . . . .	255
<b>12 Stability</b>	<b>261</b>
12.1 Stability . . . . .	261
12.2 Jacobian for Biochemical Systems . . . . .	266
12.3 External Stability . . . . .	267
12.4 Phase Portraits . . . . .	268
12.5 Bifurcation Plots . . . . .	273
Further Reading . . . . .	287
Exercises . . . . .	288
12.6 Appendix . . . . .	290
<b>13 Modeling FeedForward Networks</b>	<b>295</b>
13.1 Coherent Type I Motif . . . . .	296
13.2 Incoherent Type I Motif . . . . .	300
Further Reading . . . . .	301
Exercises . . . . .	303
Tellurium Scripts . . . . .	303
<b>14 Behavior of Stochastic Models</b>	<b>307</b>
14.1 Introduction . . . . .	307
14.2 Stochastic Bursting . . . . .	308
14.3 Stochastic Focusing . . . . .	311
14.4 Chatter . . . . .	314
Further Reading . . . . .	316
Tellurium Scripts . . . . .	319
<b>Appendix A List of Symbols and Abbreviations</b>	<b>325</b>
<b>Appendix B Useful Numbers</b>	<b>329</b>
B.1 Useful Numbers . . . . .	329
<b>Appendix C Answers to Questions</b>	<b>333</b>

<b>Appendix D Enzyme Kinetics in a Nutshell</b>	<b>335</b>
D.1 Michaelis-Menten Kinetics . . . . .	336
D.2 Reversibility and Product Inhibition . . . . .	336
D.3 Reversible Rate laws . . . . .	337
D.4 Haldane Relationship . . . . .	337
D.5 Competitive Inhibition . . . . .	338
D.6 Cooperativity . . . . .	339
D.7 Allostery . . . . .	341
D.8 Elasticities . . . . .	343
Further Reading . . . . .	344
<b>Appendix E Math Fundamentals</b>	<b>345</b>
E.1 Notation . . . . .	345
E.2 Short Table of Derivatives . . . . .	346
E.3 Logarithms . . . . .	347
E.4 Partial Derivatives . . . . .	347
E.5 Differential Equations . . . . .	348
E.6 Taylor Series . . . . .	349
E.7 Total Derivative . . . . .	352
E.8 Eigenvalues and Eigenvectors . . . . .	353
Further Reading . . . . .	354
<b>Appendix F Statistics Reminder</b>	<b>355</b>
F.1 Mean . . . . .	355
F.2 Deviation . . . . .	355
F.3 Standard Error . . . . .	356
F.4 Covariance . . . . .	356
F.5 Normal Distribution . . . . .	356
F.6 <i>z</i> -Scores or Standard Scores . . . . .	357
F.7 Null Hypothesis . . . . .	357
F.8 $\chi^2$ Distribution . . . . .	358
F.9 F-test . . . . .	359
F.10 Confidence Intervals . . . . .	359

F.11 Bootstrapping . . . . .	360
F.12 Maximum Likelihood . . . . .	361
Further Reading . . . . .	364
<b>Appendix G Modeling Standards and Databases</b>	<b>367</b>
G.1 Introduction . . . . .	367
G.2 Graphical Layout . . . . .	368
G.3 MIRIAM . . . . .	368
G.4 SBO – Systems Biology Ontology . . . . .	369
G.5 Other Ontologies and Formats . . . . .	369
G.6 Human Readable Formats . . . . .	370
G.7 Databases . . . . .	370
<b>Appendix H Modeling with Python</b>	<b>373</b>
H.1 Introduction to Python . . . . .	374
H.2 Describing Reaction Networks using Antimony . . . . .	378
H.2.1 Initialization of Model Values . . . . .	381
H.3 Using libRoadRunner in Python . . . . .	381
H.3.1 Time Course Simulation . . . . .	382
H.3.2 Plotting Simulation Results . . . . .	383
H.3.3 Applying Perturbations to a Simulation . . . . .	384
H.3.4 Steady State and Metabolic Control . . . . .	385
H.3.5 Other Model Properties of Interest . . . . .	387
H.4 Generating SBML and Matlab Files . . . . .	387
H.5 Exercise . . . . .	388
<b>References</b>	<b>391</b>
<b>History</b>	<b>403</b>
<b>Index</b>	<b>1</b>

## Preface

---

This book is an introduction to modeling biochemical pathways and is written mainly for people new to the field. The mathematics is generally light and the emphasis is on concepts and exercises carried out at the computer. I would recommend having taken at least one year of college cell biology and chemistry. The material presented in this book is suitable for late sophomore or early junior years, or late first year and second year for the UK university system.

I wrote this book because I felt there weren't any text books specifically on modeling biochemical networks. There are many books on mathematical modeling, but these tend to be broad in scope and cover biochemical networks as one example among many.

Given how broad the field of modeling biochemical systems is, I've had to be strict on what to include and the text reflects this personal view. I concentrate on two popular and successful approaches to building biochemical models, one using differential equations and the other using stochastic kinetics. I leave it to others to write about flux balance models, Boolean networks, Petri nets and the many other modeling approaches that exist. The other topic I do not review is the selection, limitations and assumptions of the various rate laws that one can use in building a biochemical model. Many of these details can be found in my companion book, 'Enzyme Kinetics for Systems Biology'. This book is strictly about modeling techniques and does not cover in great detail the kinds of dynamics one finds in biochemical systems. This means there is little mention of oscillators, bistable switches, negative feedback, and the myriad other interesting behaviors and systems we find in biological networks. I have used one of the last chapters to describe the modeling of feedforward network but other than that I only briefly mention other kinds of systems. Furthermore, I do not discuss the important topic of metabolic control analysis. These topics are reserved for another time. Lastly, one major omission in the book is spatial modeling, an area that is becoming increasingly important as researchers turn to modeling larger spatially extended systems such as tissues and organs. This topic alone would require a separate publication in its own right.

As with my book on Enzyme Kinetics for Systems Biology, I have decided to publish this book myself. I have found that traditional publishers have yet to catch up with modern publishing trends, in particular the loss of copyright on the text as well as any figures and even more problematic, the inability to rapidly update text to correct errors or when new material needs to be added. Publishers still handle corrections via errata pages rather than updating the book itself. With today's print on demand technology, the restrictions imposed by publishers seem unnecessary.

This edition is also a modified version of the original publication: Essentials of Biochemical Modeling. The differences include a change to the book title and redoing all the modeling examples in Python. The change in title is interesting because it appears that the term 'biochemical modeling' is not common place so I decided to change the title that uses terms more familiar with researchers.

There are many people and organizations who I should thank, but foremost must be my infinitely patient wife, Holly, and my two boys Theodore and Tyler who have put up with the many hours I have spent working alone. I would like to thank Holly in particular for helping me edit the text. I am also most grateful to the National Science Foundation and the National Institutes of Health who paid my summer salary so that I could allocate some time to write, edit and research. I would also like to thank the many undergraduates, graduates and colleagues who have contributed to this work. In particular I want to thank my two teachers, David Fell and the late Henrik Kacser who I had the privilege to work with as a graduate student and postdoctoral fellow. I had many hours of fruitful conversations with Luis Azerenza, Frank Bruggeman, Jim Burns, Vijay Chickarmane (who carried out some of the optimization experiments in Chapter 9), Athel Cornish-Bowden, Jannie Hofmeyr and Pedro Mendes. More recently I should thank my graduate students, in particular Frank Bergmann (author of SBW) and Deepak Chandran (author of TinkerCell) who developed a very deep understanding of how networks operate. I thank them for their dedication and steadfast enthusiasm while they worked in my lab. I would also like to thank Kyung Kim, a gifted applied mathematician, who helped me understand some of the subtleties of stochastic systems. I would also like to sincerely thank Kaylene Stocking who translated all the Jarnac scripts from the earlier edition into the Python scripts you see in this version. I would like to thank Joseph Hellerstein<sup>1</sup> at UW who provided many useful suggestions for improving the text throughout the book, and identifying errors in some of the examples. I would also like to sincerely thank my colleague James Glazier at Indiana University Bloomington who made a thorough examination of the book in the more recent versions (1.13/1.14) and highlighted numerous improvements and corrections. Finally I would like to thank my graduate student Veronica Porubsky who helped develop the parameter fitting example in Chapter 9 which was introduced in revision 1.15.

Naturally, I am responsible for the remaining errors. Or as a contributor (Marc Claesen) to stackoverflow once humorously remarked, ‘Making the manuscript error-free is left as an exercise for the reader.’.

Many thanks to the authors of the  $\text{\TeX}$  system, MikTeX (2.9), TikZ (2.1), PGFPlots (1.9), WinEdt (6.0), Inkscape (0.48.4), and Createspace for making available such amazing tools for technical authors. It is these tools that make it possible for individuals like myself to publish. Finally, I should thank Michael Corral (<http://www.mecmath.net/>) and Mike Hucka ([www.sbml.org](http://www.sbml.org)) whose  $\text{\LaTeX}$  work inspired some of the visual styles I used in the text.

August 2014  
Seattle, WA

HERBERT M. SAURO

---

<sup>1</sup><https://sites.google.com/site/josephlhellerstein/>

## Cover Image

---

The cover illustrates a crosssection through a single *Mycoplasma mycoides* cell that is approximately 250 nm in diameter. *Mycoplasma mycoides* is the agent for the contagious disease bovine pleuropneumonia that infects cattle and goats. The organism has been sequenced and found to contain 985 genes and is the smallest known free-living life form (Westberg et al., 2004). Since the genome is greatly reduced in size, the organism relies on the host to provide much of its nutrition.

The Goodsell website<sup>2</sup> provides a key to identify the various parts in the illustration. For example, the larger purple objects are the ribosomes, the yellow strand running through the cell is DNA, the smaller blue objects are enzymes, and the green objects embedded in the membrane are transporters and ATP synthetase.

The image was painted by the artist and scientist David S. Goodsell. The author is most grateful to David Goodsell for his permission to use the image. For further remarkable images painted by Goodsell, the second edition book, The Machinery of Life is highly recommended.

Westberg J, Persson A, Holmberg A, Goesmann A, Lundeberg J, Johansson KE, Pettersson B, Uhlén M. The genome sequence of *Mycoplasma mycoides* subsp. *mycoides* SC type strain PG1T, the causative agent of contagious bovine pleuropneumonia (CBPP). *Genome Res.* 2004 Feb;14(2):221-7.

Goodsell DS (2009) The Machinery of Life, Second Edition, Springer Science, Copernicus, ISBN-13: 978-0387849249.

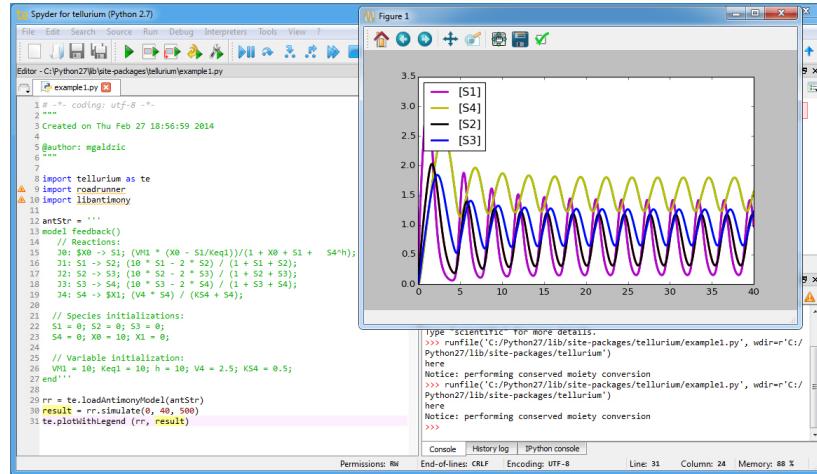
---

<sup>2</sup><http://mgl.scripps.edu/people/goodsell/illustration/mycoplasma>

# A Note about Software

## A Note about Software

Some of the chapters in this book include model listings in an appendix. These listings are expressed in Antimony [165] and libRoadRunner [157, 166] using Python in the integrated modeling environment called Tellurium ([tellurium.analogmachine.org](http://tellurium.analogmachine.org)). Antimony and Python offer a simple way to express models in a concise and readable form. For those who wish to use Matlab or another simulation tool, it is possible to load the scripts into Python and convert them either to Matlab or SBML [76]. Once in SBML format, the models can be loaded into a great variety of other software tools. Section H.4 in Appendix H describes in more detail how to generate SBML or Matlab from an Antimony/Python script. All scripts will work on Tellurium 2.0.19 and above.



Above: Screen-shot of the Python environment we will use in the book.

## Conventions

---

Traditionally concentrations are indicated using square brackets. That is:

$$[A]$$

is interpreted to mean the concentration of  $A$ .  $A$  on its own, as in:

$$A$$

is then interpreted to mean the amount of  $A$ . Given the many times in the text we refer to concentrations, the use of square brackets each and every time will lead to some clutter in the equations. To simplify the notation, the text will instead use the simple  $A$  to represent concentrations and will state in the text if, on those rare occasions, amounts must be specified. A roman letter  $A$  will be used to refer to the chemical species name as in, for example, the reaction:  $A \rightarrow B$ .

## *Prologue*

This book is about a grand vision, that of being able to predict in precise detail how living systems grow, respond and change as they live their lives. Within that vision also lies a hope that living systems can be engineered to the benefit of society and diseases such as cancer finally conquered.

These ideas have a rich history in fiction. For many years science fiction writers have written stories about engineering life. Three in particular stand out in my own mind. Frank Herbert's novel *The Eyes of Heisenberg* (1966), describes a future where micromanipulation of cells at the molecular level is common place. Another author, Harry Harrison in his *Eden* trilogy (1984), describes an earth populated by intelligent dinosaurs who have complete mastery over genetic engineering and can manipulate living organisms at will. Finally, Greg Bear writes a story called *Blood Music* (1985) that describes a renegade bioengineer who reengineers his own lymphocytes with intelligence, and then begin to alter and "improve" his own genetic constitution with dire effects. These and many other stories going all the way back to Shelley's *Frankenstein* have predicted that one day, the ability to control and change living systems at an unprecedented level would become a reality. Obviously such control could be used for good or bad, and science fiction writers have recognized both aspects in their writings. There are considerable ethical issues at stake and many fear that such power could do great harm.

But what of the good? The ability to regenerate a severed spinal column, the replacement of disfiguring skin burns, the regrowth of severed limbs, or a final cure for one of the most feared diseases, cancer, could all eventually become reality if we had a better understanding of living systems.

A key indicator of understanding in science and our goal to engineer is the ability to predict what will happen to a system when perturbed. In biology we would ask the question: how might a cell respond to a certain intervention? Such perturbations might include the action of therapeutic drugs or engineering new abilities via synthetic biology. If we were to administer a drug or cocktail of drugs, or change the expression of one or more genes, can we predict the outcome? At the moment, not very well, and in fact the easiest way to find out what will happen is to actually do the experiment. One reason for this is that a living cell is a complicated beast with many thousands of interacting components. It is also very challenging to characterize molecular based components that operate in a liquid/gel like environment. Nevertheless, great progress has been made in the last sixty years in

cataloging and characterizing the various parts of a cell. This leads to the question whether we can predict the outcome of interventions before they are even attempted given sufficient information. In recent decades there has been a growing interest and some progress in the possibility of building computer models of living organisms to make such predictions. For example, we might build a computer simulation of a signaling pathway and use it to investigate possible targets for disrupting or controlling it. Perhaps a synthetic biologist might want to design a genetic network to control when a cell will replicate or begin to produce a useful commodity. It will be up to the new generation of young scientists and engineers to determine how these tools might be applied and extended to solve problems that affect us today.

This book is about introducing students to some of the concepts and techniques in modeling life processes at the biochemical level. We will start by introducing biological networks before considering their mathematical representation in the form of stoichiometric networks.

# 1

## *Cellular Networks*

The study of cellular networks is one of the defining characteristics of systems and synthetic biology. Such networks involve the coordinated interaction of thousands of molecules that include nucleic acids, proteins, metabolites and other small molecules. Descriptions of these elaborate networks can be found in text books, on wall charts, and more recently in databases such as EcoCyc, RegulonDB, KEGG or STRING (Table 1.1).

### **1.1 Overall Organization**

---

Biological networks can be organized into three broad categories (Fig. 1.5): gene regulatory, protein and metabolic networks. In the metabolic category, small molecules are chemically transformed by enzymes. These molecules – or metabolites – serve either as energy sources or as building blocks for more complex molecules, particularly polymers such as polysaccharides, nucleic acids and proteins.

The protein networks constitute a major part of the decision making and nano-machine apparatus of a cell. We can divide the decision making protein networks into two subgroups. One subgroup involves transcription factor proteins that regulate gene expression, forming what are called gene regulatory networks (GRNs). The second subgroup constitutes the protein signaling pathways that integrate information about the internal and external environments and modulate both the metabolic and gene regulatory networks.

The metabolic, protein, and gene regulatory networks each have a characteristic mode of operation and differ by the molecular mechanisms employed and their respective operating

**Table 1.1** Online *E. coli* resources

Online Resource	URL
EcoCyc	<a href="http://ecocyc.org/">http://ecocyc.org/</a>
RegulonDB	<a href="http://regulondb.ccg.unam.mx/">http://regulondb.ccg.unam.mx/</a>
KEGG	<a href="http://www.genome.jp/kegg/">http://www.genome.jp/kegg/</a>
STRING	<a href="http://string.embl.de/">http://string.embl.de/</a>

time scale. In general, metabolic networks operate on the smallest time scale, followed by protein signaling networks, and gene regulatory networks.

This picture is of course a simplified view. For example, it omits the extensive RNA network that may be present, particularly in eukaryotic cells. Protein signaling networks are also involved in a variety of other related functions such as cytoskeleton control and cell cycle regulation. In addition, there is considerable overlap between the different systems with gene, metabolic, and protein control networks interlinked [14].

## 1.2 Network Representation

---

There are different ways (Figure 1.1) to represent cellular networks depending on how the information will be used and what kinds of questions are asked. Traditionally, cellular networks have been described using a **stoichiometric** formalism. Such networks are mechanistic in nature, consistent with the laws of mass conservation and will often include kinetic laws describing transformations of species from one form to another through binding/unbinding or molecular reorganization. In recent years an alternative representation, which might be termed **non-stoichiometric**, has gained significant popularity with the advent of high-throughput data collection. Non-stoichiometric networks, of which there are a great variety, include interaction networks which describe the relationship, usually via some physical interaction but sometimes also functional, between molecular species or functional entities such as genes or proteins. Non-stoichiometric networks tend to be more coarse grained compared to stoichiometric networks, but their study has proven to be very popular due in large part to the availability of vast new data sources. That, coupled with the unprecedented interest in networks in general, has made the study of non-stoichiometric networks an intellectually interesting area of study [6].

## 1.3 Metabolic Networks

---

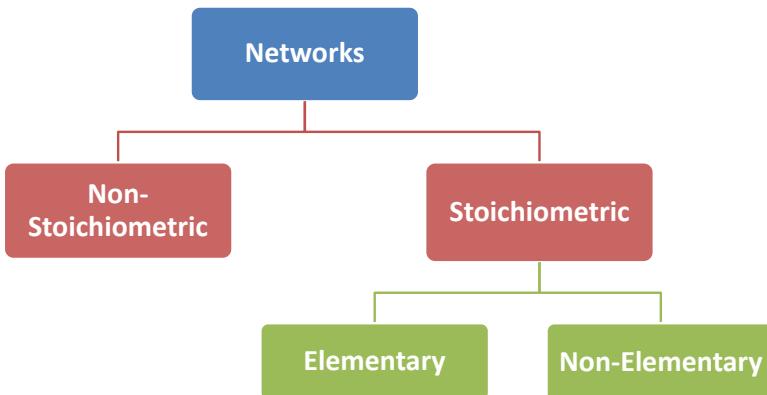
The first cellular networks to be discovered were the metabolic pathways such as Glycolysis in the 1930s and the Calvin cycle in the 1940s. The first metabolic pathways were

elucidated by a combination of enzymatic inhibitors and the use of radioisotopes such as carbon-14. The Calvin cycle for example was discovered by following the fate of carbon when algae was exposed to  $^{14}\text{C}$ -labeled  $\text{CO}_2$ . With the development of microbial genetics, significant progress was made in uncovering other pathways by studying mutants and complementing different mutants of a given pathway to determine the order of steps. The reaction steps in a metabolic pathway are catalysed by enzymes, and we now know there are thousands of enzymes in a given organism catalyzing a great variety of pathways. The collective sum of all reaction pathways in a cell is referred to as metabolism, and the small molecules that are interconverted are called metabolites.

Traditionally, metabolism is classified into two groups, anabolic (synthesis) and catabolic (breakdown) metabolism. Coupling between the two metabolic groups is achieved through cofactors of which a great variety exist. Two widely distributed cofactors include the pyridine nucleotides in the form of  $\text{NAD}^+$  and  $\text{NADP}^+$ , and the adenine nucleotides in the form of ATP, ADP and AMP. These cofactors couple redox and phosphate, respectively, by forming reactive intermediates that enables catabolism to drive anabolism. Cellular respiration is a catabolic process where molecules such as glucose and fatty acids are oxidized in a stepwise fashion. The energy released is captured in the form of ATP and the oxidized products of water and carbon dioxide are released as waste. ATP can be used in turn to drive anabolic processes such as amino acid or nucleotide biosynthesis. In general, metabolic pathways tend to be regulated via allosteric regulation. This is where a metabolite can regulate the reaction rate of an enzyme by binding to a site on the enzyme other than the catalytic site. Such interactions form a network of feedback and feedforward regulation. Figure 1.2 shows a metabolic pathway of glycolysis from *Lactococcus lactis*. On the left, glucose enters the cell which is converted in a series of reactions to ethanol and a variety of other small molecules.

Metabolic networks are the fastest (excluding ion transfer mechanisms) in terms of their response to perturbations and can operate on a time scale from microseconds to seconds. This reflects the need to rapidly adjust the supply of molecular building blocks and energy as supply and demand fluctuate. Physically, the rapid response of metabolic networks is achieved by allosteric control where the fast diffusion of small molecules can bind and rapidly alter the activity of selected enzymes.

Figure 1.4 shows a section of the glycolytic pathway which converts glucose to pyruvate with the production of ATP and NADH. The diagram also shows the many negative and positive feedback and feedforward regulatory loops in glycolysis. Not all of these are present in all organisms, however many are. Note the six regulatory signals that converge on 6-Phosphofructose-1-kinase (also known as phosphofructokinase) and Fructose Bisphosphatase (Labeled 2 and 3). Figure 1.4 uses a standard notation to indicate inhibition and activation. Inhibition is often represented as a blunt ended arrow and in this book activation by a rounded arrow (Figure 1.3).



**Figure 1.1** Cellular networks are often represented using two common approaches, non-stoichiometric and stoichiometric. Non-stoichiometric networks are characterized by a lack of stoichiometric information and mass conservation. Stoichiometric networks are classified according whether they are elementary or not. Elementary networks are those where the reactions cannot be broken into simpler forms. Non-elementary networks may have one or more reaction steps which represent an aggregate of two or more elementary reactions, the aggregation being dependent on assumptions such as quasi-steady state or equilibrium.

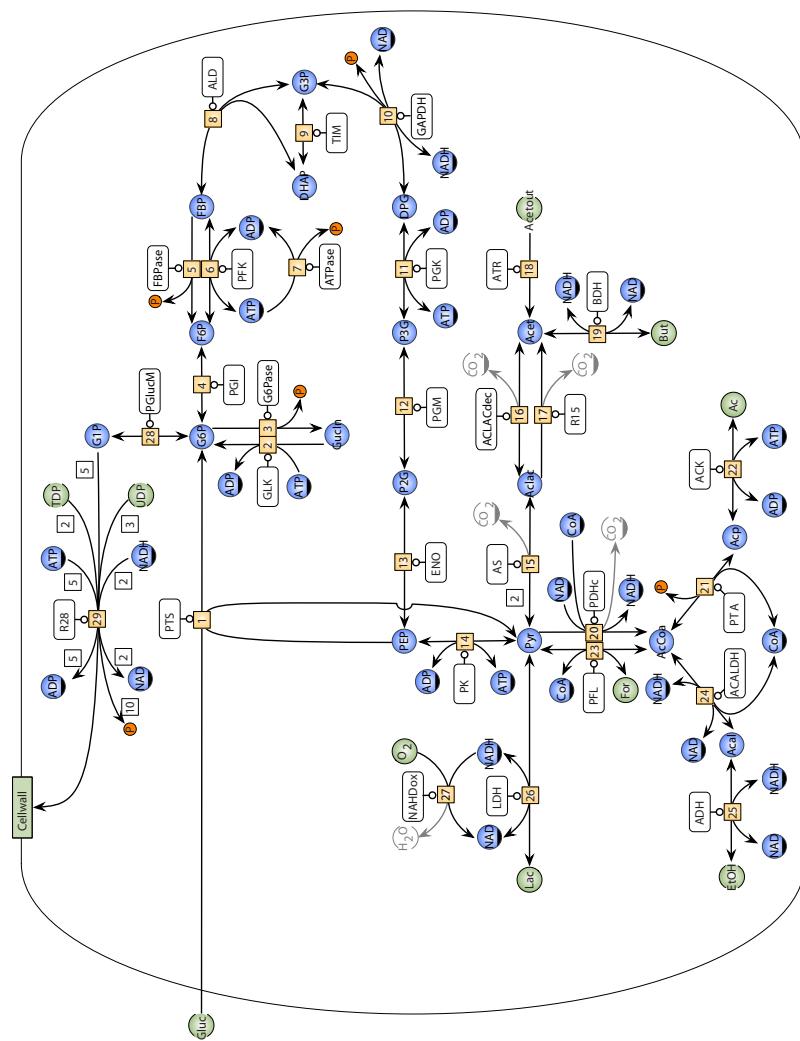
## 1.4 Protein Networks

---

Protein networks are by far the most varied networks found in biological cells. They range from proteins involved in controlling gene expression, the cell cycle, coordinating and processing signals from the internal and external environments, to highly sophisticated nano-machines such as parts of the ribosome or the bacterial flagella motor.

Protein networks can be studied on different levels, broadly classified as either stoichiometric or non-stoichiometric networks. The non-stoichiometric networks can be as simple as considering the physical associations between different proteins (often through the formation of protein complexes). Such networks, also termed interaction networks, have been elucidated largely with the help of high-throughput methods. An interaction is formed if two proteins, *A* and *B*, are known to associate.

Another descriptive level involves functional and stoichiometric networks formed from a consideration of specific stoichiometric binding events, covalent modification (most notably phosphorylation), and degradation. Here two proteins, *A* and *B* might form a complex with a stoichiometric relationship and given association constant.



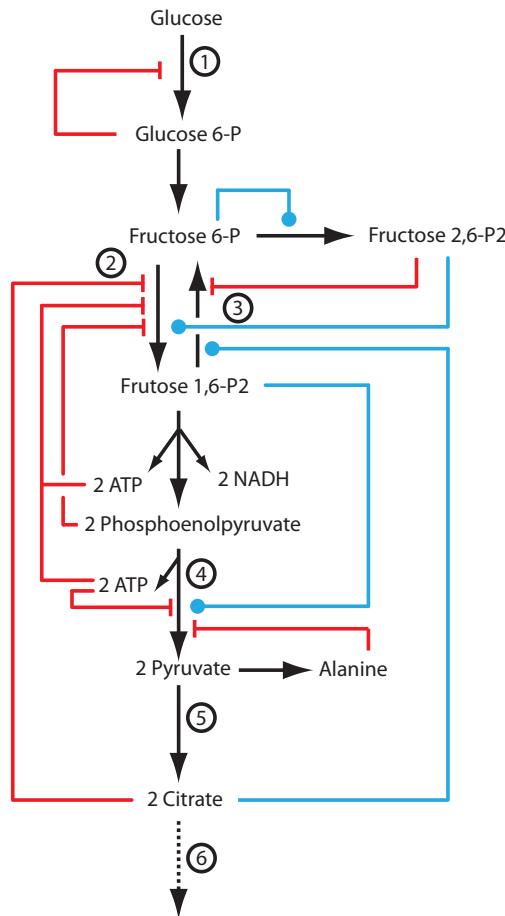
**Figure 1.2** Metabolic Pathway: Metabolic pathway image from JWS online (Jacky Snoep) with permission. The pathway depicts the glycolytic pathway from *Lactococcus lactis* using the Systems Biology Graphical Notation (SBGN) [94, 72].

## Protein-Protein Networks

Work on uncovering protein networks has been ongoing since the 1950s and considerable detail has accumulated on many different pathways across different organisms. Traditional methods, though laborious [33, 60], have been used extensively to gain detailed knowledge

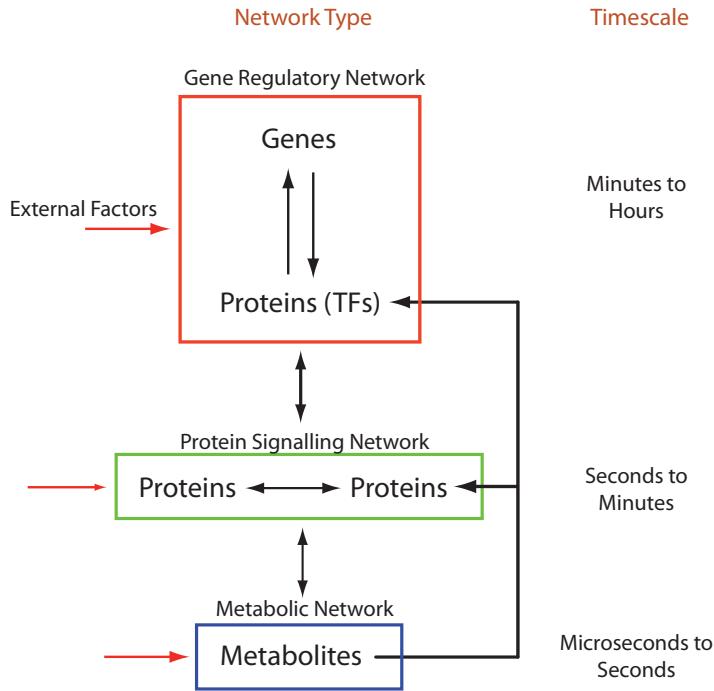


**Figure 1.3** Summary of regulation and reaction symbols.



**Figure 1.4** A section of glycolysis with negative and positive regulation shown. 1. Hexokinase; 2. 6-Phosphofructose-1-kinase; 3. Fructose bisphosphatase; 4. Pyruvate kinase; 5. Entry to Citric acid cycle; 6. To oxidative respiration.

on phosphorylation sites, protein structure, the nature of membrane receptors, and the constitution and function of protein complexes. More recent high-throughput methods, though more course grained, have uncovered large swaths of protein-protein interaction networks. For example, in yeast, large scale studies have identified approximately 500 different pro-



**Figure 1.5** Network Overview. The figure illustrates the three main network layers, metabolic, protein and gene. TF – Transcription Factors. The arrows here represent interactions between the different processes.

tein complexes [51, 91] and their relationships to each other.

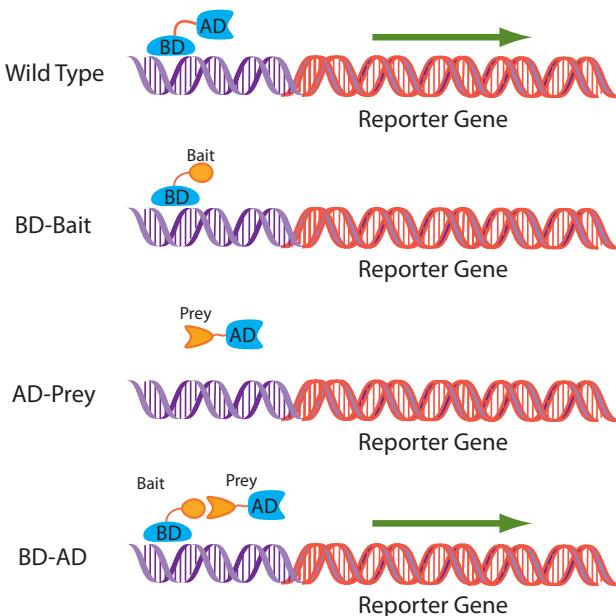
A popular high-throughput technique that has been used to uncover protein-protein interaction networks is the yeast two-hybrid method [44, 137]. Other methods such as phage display [164, 59], affinity purification, and mass spectrometry have also been successfully employed [51, 91]. The yeast two-hybrid method (Figure 1.6) is based on the idea that eukaryotic transcriptional activators consist of two domains, a DNA binding domain (DB) and an activation domain (AD). The activation domain is responsible for recruiting the RNA polymerase to begin transcription. What is remarkable is that the two domains do not have to be covalently linked in order to function correctly; they simply need to be in close proximity. The yeast two-hybrid method is based specifically on this property.

Assume we want to know whether protein X and protein Y interact with each other. In the two-hybrid method, protein X is fused with the DB domain (known as the bait protein) and the second protein, Y, is fused with the AD domain (known as the prey protein). These two fused proteins are now expressed in yeast. If the two proteins X and Y interact in some way, they will bring the DB and AD domains close to each other resulting in an active transcriptional activator. If the gene downstream of the DNA binding sequence is a reporter

gene, the interaction of X and Y can be detected once the reporter gene is expressed.

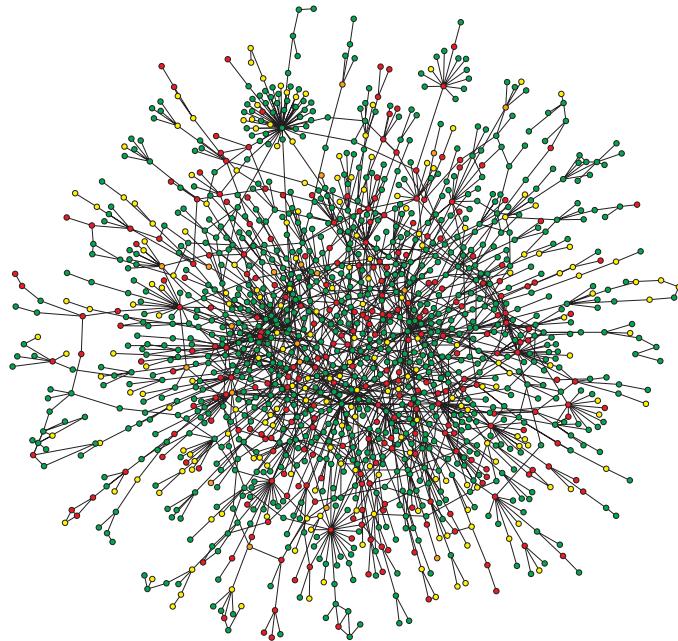
A common reporter gene is the lacZ gene which codes for  $\beta$ -galactosidase and which produces a blue coloring in yeast colonies through the metabolism of exogenously supplied X-gal (5-bromo-4-chloro-3-indolyl- $\beta$ -D-galactoside).

There are some caveats with the yeast two-hybrid method, however. Although two proteins may be observed to interact, the protein in their natural setting may not be expressed at the same time or may be expressed but in different compartments. In addition, using the method to identify interactions between non-yeast proteins may be invalid because of the alien environment of yeast cells. As with many high-throughput methods, caution is advised when interpreting the data.



**Figure 1.6** Yeast two-hybrid. The wild-type transcription factor is composed of two domains, BD and AD. Both are essential for transcription. Two fusion proteins are made, BD-Bait and AD-Prey. Bait and Prey are the two proteins under investigation. If Bait and Prey interact, BD and AD are brought together resulting in a viable transcription factor that can express a reporter gene.

The yeast two-hybrid system helped generate one of the first large scale interaction graphs to be published, the protein interaction graph of *Saccharomyces cerevisiae* [178, 79]. Subsequent analysis of this map was conducted by Jeong et al. [81] and included 1870 proteins nodes and 2240 interaction edges. Such graphs give a birds-eye view of protein interactions in an entire cell (Fig. 1.7).



**Figure 1.7** The poster child of interaction networks, one of the earliest yeast protein interaction networks generated from yeast two-hybrid measurements. Each node represents a protein and each edge an interaction. Although difficult to see in the figure, the graph nodes have been annotated such that red (dark) indicate lethal phenotypic effect if removed, green non-lethal, orange slow growth, and yellow unknown. Adapted from Barabási and Oltvai [5] but originally published in arXiv and Nature [81].

## Signaling and Control Networks

Many protein-protein networks operate as signal processing networks and are responsible for sensing external signals such as nutritional (for example, changes in glucose levels), or cell to cell signals such as insulin or Epidermal growth factor (EGF). Other signaling networks include control networks that are concerned with monitoring and coordinating internal changes, the most well known of these includes the cell cycle control network. Many external signals act by binding to cell-surface receptor proteins such as the large family of receptor tyrosine kinases and G-protein coupled receptors [90]. Once a signal is internalized through the cell-surface receptors, other proteins including protein kinases and phosphatases continue to process the signal often in coordination with other signaling networks. Eventually the signaling pathway terminates on target proteins that leads to a change in cell behavior. Such targets can include a wide variety of processes such as metabolic pathways, ion channels, cytoskeleton, motor proteins, and gene regulatory proteins.

The molecular mechanisms employed by signaling and control pathways include covalent

modification, degradation and complex formation. Covalent modification in particular is a common mechanism used in signaling networks and includes a variety of different modifications such as phosphorylation, acetylation, methylation, ubiquitylation, and possibly others [16]. As a result, the structure and computational abilities [154] of such networks are likely to be elaborate. It has been estimated from experimental studies that in *E. coli*, 79 proteins can be phosphorylated [103] on serine, threonine, and tyrosine side groups whereas in yeast, 4000 phosphorylation events involving 1,325 different proteins have been recorded [140].

The cell cycle control network is a good example of a sophisticated protein control network that coordinates the replication of a biological cell. The cell cycle includes a number of common molecular mechanisms that are found in many other protein networks. These can be grouped into three broad types: phosphorylation, degradation and complex formation. Phosphorylation is a common mechanism for changing the state of a protein and involves phosphorylation on a number of sites on the protein surface including serine/threonine and tyrosine. In prokaryotes, histidine, arginine, or lysine can also be phosphorylated. Phosphorylation is mediated by kinases. The human genome may have over 500 kinase encoding genes [107]. The effect of phosphorylation is varied but most often causes the altered protein to change catalytic activity, to change the protein's 'visibility' to other proteins, or to mark the protein for degradation. For example, Src is a tyrosine kinase protein involved in cell growth. It has two states, active and inactive. When active, it has the capacity to phosphorylate other proteins. Deactivation of src is achieved by phosphorylation of a tyrosine group on the C-terminal end of the protein. Dephosphorylation of the tyrosine group by tyrosine phosphatases results in activation of the protein.

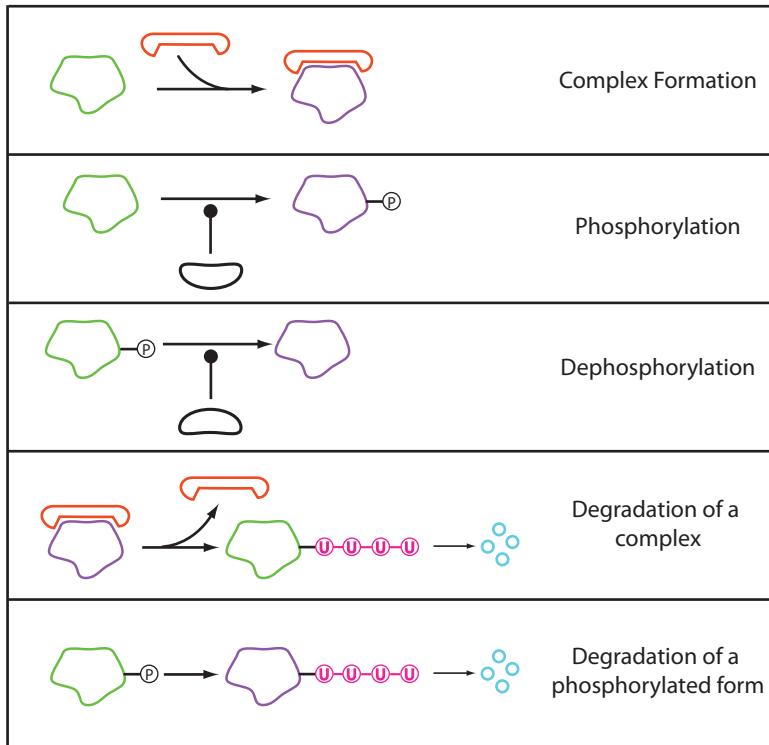
Phosphorylation can also be used to inactivate enzymes such as glycogen synthase by the glycogen synthase kinase 3 protein. In the yeast cell cycle, the protein Wee1 is phosphorylated and inactivated by the complex Cdc2-Cdc13. Active Wee1 in turn (i.e. the unphosphorylated form) can inactivate Cdc2-Cdc13 by phosphorylating the Cdc2 subunit.

In addition to changing the activity of proteins, phosphorylation can also be used to mark proteins for degradation. For example, the protein Rum1 that is part of the yeast cell cycle control network can be phosphorylated by Cdc2-Cdc13. Once phosphorylated, Rum1 is degraded. Degradation itself is an important mechanism used in protein signalling networks and allows proteins to be rapidly removed from a network according to the cell state. Degradation is usually mediated by ubiquitylation. For example, Cdc2-Cdc13, via Ste9 and APC is marked for degradation by ubiquitylation (Rum1 is similarly processed once phosphorylated). Once marked this way, such proteins can bind to the proteasome where they are degraded. Finally, binding of one protein to another can change the target protein's activity or visibility. An example of this is the inactivation of Cdc2-Cdc13 by Rum1. When unphosphorylated, Rum1 binds to Cdc2-Cdc13, rendering the resulting complex inactive.

Different combinations of these basic mechanisms are also employed. For example, phosphorylation of complexes can lead to the dissociation of the complex, or the full activity of a protein may require multiple phosphorylation events. Although signaling networks

can appear highly complex and varied, most of them can be reduced to the three fundamental mechanisms of covalent modification, selective degradation and complex formation (Fig 1.8).

These examples highlight the basic mechanisms by which protein signalling control networks can be assembled into sophisticated decision making systems.



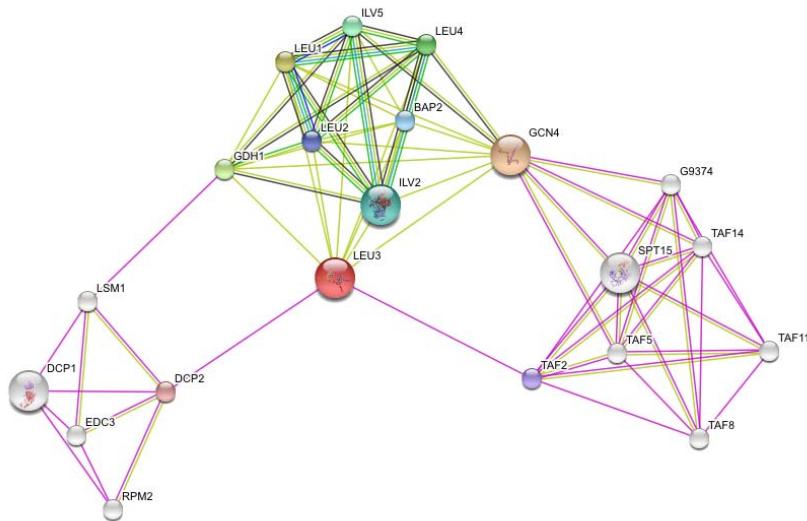
**Figure 1.8** Fundamental Protein Mechanisms.

In higher eukaryotic cells, particularly human, around 2% of the protein-coding part of the genome is devoted to encoding protein kinases, with perhaps 10% of the coding region dedicated to proteins involved in signaling networks. It has also been suggested that as much as 30% of all cellular proteins in yeast and human can be phosphorylated [30].

The actual size of networks is even larger than these numbers suggest because of the significant number of covalent variants and binding permutations. For example, the tumor suppressor protein, p53, has between 17 and 20 phosphorylation sites alone [176]. If every combination were phenotypically significant, as unlikely as that might be, this amounts to at least 131,072 different states. On a side note, there is some discussion [41, 28] on how to deal with networks when there is a proliferation in states due to covalent modification and

protein complex formation. Two issues present themselves, the first is how do we describe such systems to a computer? This has been solved by using a rule-based approach. BioNetgen is an example of a software tool that allows a modeler to describe such models using rules [63]. The second issue is how important is the combinatorial expansion to modeling, are the multiple states actually functional—[tefaeder2005rule](#), [sekar2017introduction](#)?

Ptacek and Snyder [139] have published a review on elucidating phosphorylation networks where more detailed information is provided.



**Figure 1.9** A Small Protein-Protein Interaction Map. This image was taken from the STRING web site (Search Tool for the Retrieval of Interacting Genes/Proteins, <http://string.embl.de/>). The image displays a small segment of the protein interaction map centered around LEU3, the transcription factor that regulates genes involved in leucine and other branched chain amino acid biosynthesis.

## 1.5 Gene Regulatory Networks

The control of gene expression in prokaryotes is relatively well understood. Transcription factors control gene expression by binding to special upstream DNA sequences called operator sites. Such binding results in the activation or inhibition of gene transcription. Multiple transcription factors can also interact to control the expression of a single gene. Such interactions can emulate simple logic functions (such as AND, OR, etc.) or more elaborate computations. Gene regulatory networks can range from a single controlled gene to hundreds of genes interlinked with transcription factors forming a complex decision making circuit. There are different classes of transcription factors. For example, the bind-

ing of some transcription factors to operator sites is modulated by small molecules, the classic example being the binding of allolactose (a disaccharide very similar to lactose) to the lac repressor, or cAMP to the catabolite activator protein (CAP). Alternatively, a transcription factor may be expressed by one gene and either directly modulate a second gene (which could be itself), or via other transcription factors integrating multiple signals onto another gene. Additionally, some transcription factors only become active when phosphorylated or unphosphorylated by protein kinases and phosphatases. Like protein signaling and metabolic networks, gene regulatory networks can be elaborate.

Significant advances have been made in developing high-throughput methods used to determine protein-gene networks. Of particular interest are ChIP-chip [145, 4] and the more recently developed ChIP-seq [108] screening method – Chromatin immunoprecipitation microarray/Sequencing. ChIP works by treating cells with formaldehyde which crosslinks DNA to the transcription binding protein if it is bound to the DNA. The cells are then lysed and the DNA fragmented into small 1 kB or less fragments. A specific antibody is then used to bind to the DNA-binding protein of interest and precipitate the protein and associated DNA fragment. The precipitated DNA pieces are released by reversing the crosslinking. In ChIP-chip, the released DNA pieces are hybridized to a microarray that enables the bound protein to be located on the genome. A more recent version is ChIP-seq. In this procedure the microarray stage is abandoned and instead, the released DNA pieces are sequenced. Once sequenced, the location on the genome can be uncovered. These methods have been successfully used to determine the gene-protein network of a number of organisms, with yeast being the first [96]. Alternatively, other approaches have focused on determining gene-protein networks from literature mining and careful curation or even prediction of putative binding sites.

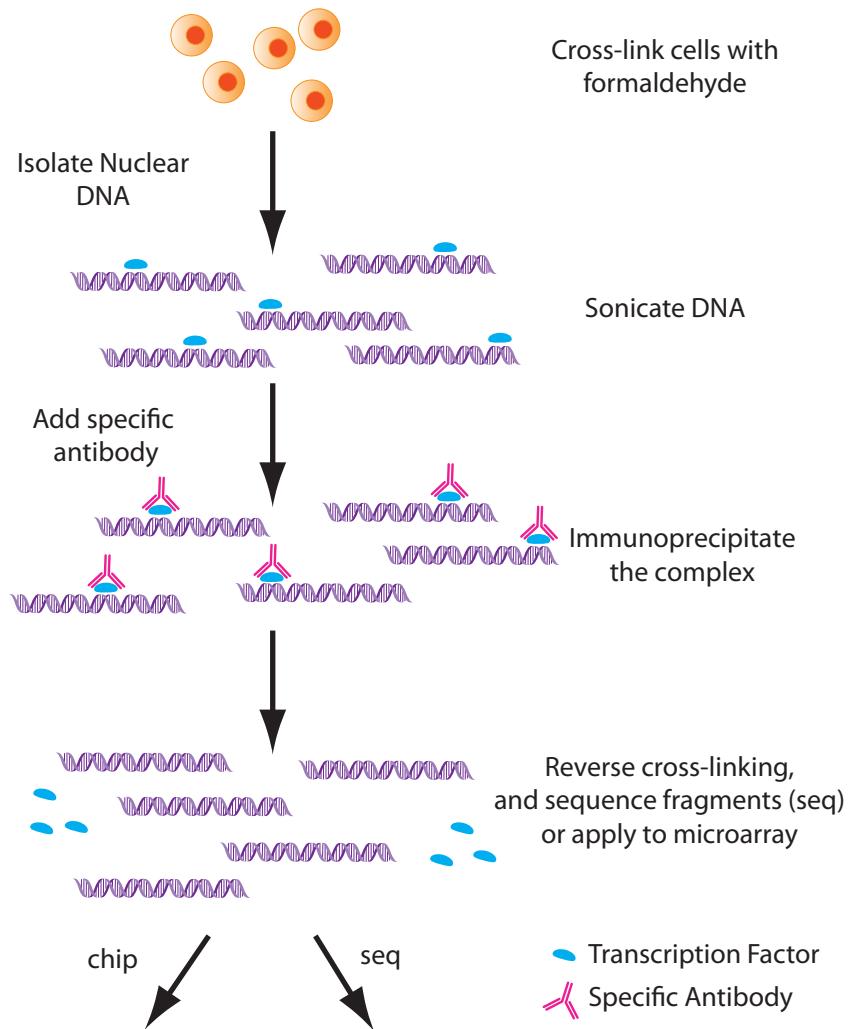
In general, gene regulatory networks are the slowest responding networks in a cell and work from minutes to hours depending on the organism. Bacterial gene regulatory networks tend to operate more quickly compared to eukaryotic gen networks. .

The most extensive gene regulatory network database is RegulonDB [77, 46] which represents the gene regulatory network of *E. coli*. In-depth reviews covering the structure of regulatory networks can be found in the works of Alon [161] and Seshasayee [160].

To visually represent gene regulatory networks we will use a notation very similar to that used by the software package Biotapestry [100, 99] because it offers a clear and concise visual representation. To represent the expression of a gene controlled by a transcription factor,  $P$ , we will use the diagrammatic notation shown in Figure 1.11. This omits details such as translation and transcription, providing a concise representation of a gene regulatory network without the mechanistic details that *may* not be important.

It is however possible to add transcription and translation as shown in Figure 1.12.

Figure 1.13 illustrates a range of gene regulatory patterns. We will cover the diagrammatic notation in more detail in Chapter 3, but (Figure 1.3) blunt ends are used to indicate inhibition while round circle ends represent activation. We can now construct gene regulatory



**Figure 1.10** ChIP-chip and ChIP-seq methods for identifying transcriptional binding sites. Adapted from [108].



**Figure 1.11** Representing a single gene.  $I$  represents the inducer and  $P$  the expressed protein. Increasing  $I$  will increase the rate of protein  $P$  expression.



**Figure 1.12** Representing a single gene with the addition of explicit translation and transcription.  $I$  represents the inducer and  $P$  the expressed protein.

networks by connecting single gene units together. Figure 1.14 illustrates two typical gene regulatory networks. The first (a) shows a relatively simple sequence of gene regulatory steps where on the left of the figure, an inducer,  $I$ , activates the expression of protein  $p_1$ . The gene is represented as a horizontal line with an emerging arrow on the right indicating the expression of either mRNA or in this case protein.  $p_1$  in turn inhibits a second gene that reduces the expression of protein,  $p_2$ .

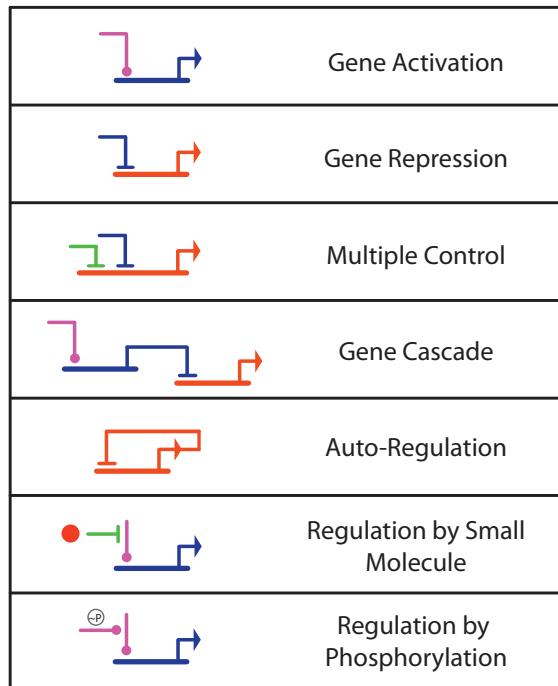
It should be evident that these diagrams hide an enormous amount of detail. In these examples even the mRNA intermediates are absent. These diagrams are therefore at a very high level and may be sufficient to answer particular questions when the detail is unnecessary.

The network in Figure 1.14(b), shows additional features including negative feedback and a transcription factor dimerizing ( $v_4$ ) to form the active regulatory protein,  $p_2$ . The mechanistic nature of the repression and activation is not specified in these diagrams.

Although the description of the three main network types may give the impression that they act independently of each other, this is definitely not the case. For example, Figure 1.15 is an example taken from *Caulobacter* [18] showing a mixed gene regulatory and protein network.

## 1.6 Genome Sizes

How big are cellular networks? We can try to answer this question by looking at whole genomes. The sizes of genomes vary considerably from the minuscule 159,662 bases of the symbiotic bacterium called *Carsonella ruddii*, which lives off sap-feeding insects, to the Whisk fern comprised of  $2.5 \times 10^{11}$  bases. Some of this size difference is related to the complexity of the organism, simpler organisms requiring fewer genes. However the correlation, although on the whole positive, is not entirely linear. For example, *E. coli* has roughly 4,300 genes on a genome of size 4.6 Mb, while humans have roughly 25,000 genes



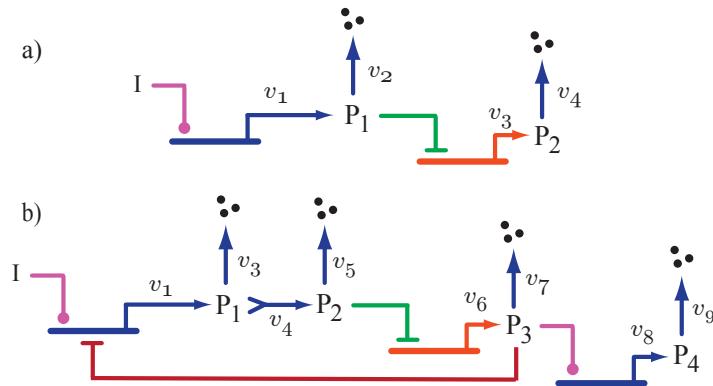
**Figure 1.13** Simple gene regulatory patterns.

on a genome of about 3000 Mb. The *E. coli* genome is quite dense with roughly 88% of the genome coding for proteins [172] with the remainder being made up of RNA coding, promoter sequences and other common segments. The human genome on the other hand is very sparse with only about 2% of the genome actually coding for protein. There is ongoing speculation as to why the human genome is so sparse and what role the other 98% might play. Some evidence suggests an RNA based regulatory network [111] that is coded in at least some of the non-coding sequences (the so-called junk DNA).

Figure 1.16 shows an example of a small genome from *Mycoplasma genitalium*. This organism is a small parasitic bacteria that lives in primate genital and respiratory tracts and is the smallest known free-living bacteria. The genome of this organism has 521 genes in total, 482 of these code for protein while the remaining 39 reading frames code for tRNA and rRNA.

The 482 genes that encode proteins in *Mycoplasma genitalium* include a wide variety of functions (Figure 1.17) covering areas such as energy metabolism, replication and the cell envelope. Even for such a small organism, there are still eight genes of unknown function.

Eukaryotic genes, especially human, are also fragmented into segments called exons (coding) and introns (non-coding). This segmentation allows different forms of protein to be derived from the same gene by splicing together different exons. Although the number of

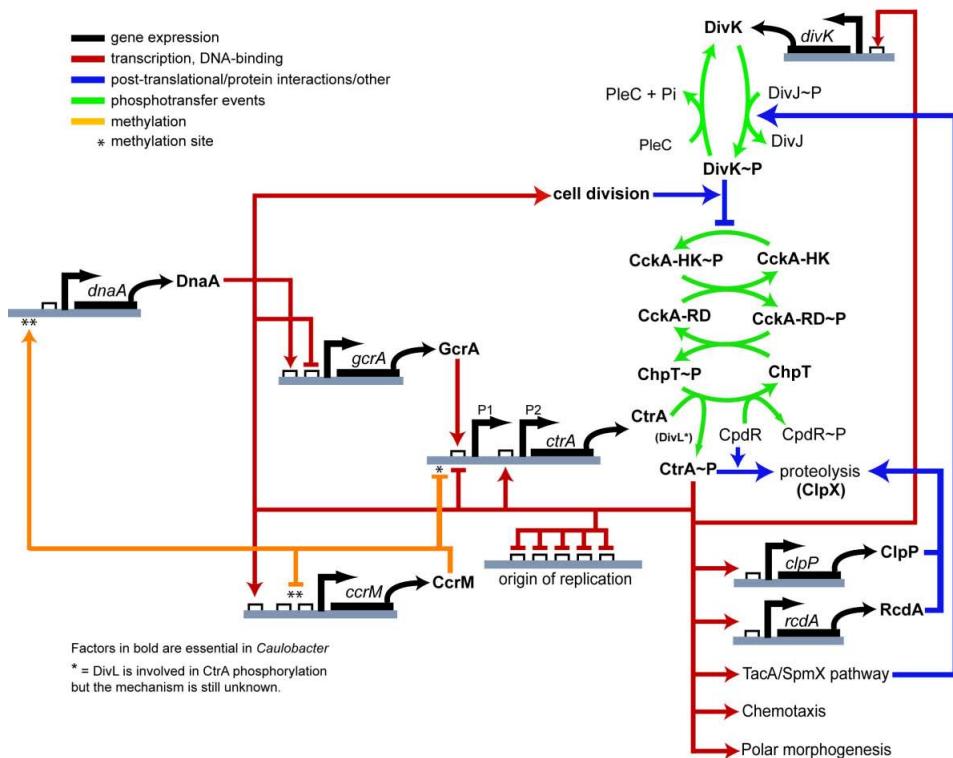


**Figure 1.14** Two examples of gene regulatory networks.  $v_1$  to  $v_9$  represent either gene expression or protein degradation rates (for example  $v_2$ ).  $I$  represents the concentration of some inducer, assumed to be a constant value. Figure b) illustrates dimerization of  $p_1$  and negative feedback from  $p_3$ .

genes is roughly 25,000, alternative splicing likely increases this number [17, 180]. Finally, many proteins, particularly those involved in signaling pathways, also have alternative forms due to covalent modification such as phosphorylation or methylation. This again increases the actual number of states. In other words, the number of genes in a genome gives a lower limit to the size of a cellular network, particularly in eukaryotic organisms. The size of a given genome is therefore a poor indicator of organism complexity. To give a better idea of the size and complexity of a small genome, let's look more closely at a specific one, *E. coli*.

**Table 1.2** A comparison of genome sizes (base pairs) and estimated number of genes. Data from Taft and Mattick [172].

Organism	Genome Size	Est. Number of Genes
<i>E. coli</i>	4,639,221	4,316
<i>Bacillus subtilis</i>	4,214,810	4,100
<i>Saccharomyces cerevisiae</i>	12,100,000	6,000
<i>Caenorhabditis elegans</i>	97,000,000	19,049
<i>Arabidopsis thaliana</i>	115,409,949	25,000
<i>Drosophila melanogaster</i>	120,000,000	13,600
<i>Mus musculus</i>	2,500,000,000	37,000
<i>Homo sapiens</i>	3,000,000,000	30,000



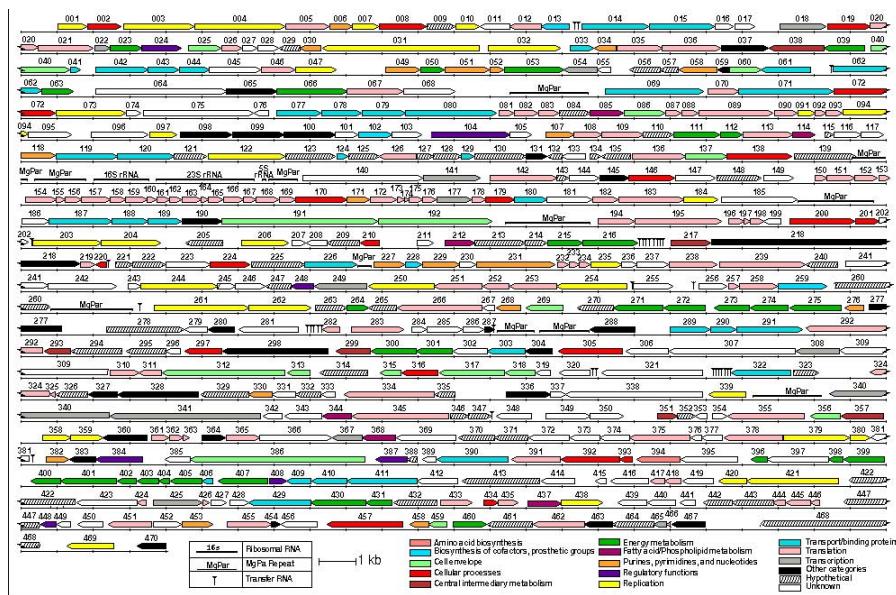
**Figure 1.15** Example of a mixed network involving gene regulatory and protein phosphorylation networks in *Caulobacter*. Blunt ends to regulatory arcs indicate inhibition while arrow ends indicate activation. Image from BioMed Central [18].

## 1.7 *E. coli*

The bacterium *E. coli* is probably one the best understood organisms so is worth considering some of its features. Much of the information provided here comes from the EcoCyc and RegulonDB online databases and their respective publications [84, 46].

*E. coli* is a cylindrical body, with a length of about  $2\mu m$  and diameter of about  $0.8\mu m$ . These dimensions offer a convenient translation between concentration and number of molecules in *E. coli*. Thus 1 nM concentration roughly translates to one molecule per *E. coli* cell (See exercises at end of chapter). For example, ATP is present at a concentration of approximately 2 mM, meaning there are roughly 2,000,000 molecules of ATP in a single *E. coli* cell.

The *E. coli* circular genome is composed of 4,639,221 base pairs ( $490\mu m$  in diameter) encoding at least 4,472 genes. 4,316 code for proteins with the remainder coding for various RNA products such as tRNAs and rRNAs. The genes in *E. coli*, like other prokaryotes, are



**Figure 1.16** Small genome with 521 genes from *Mycoplasma genitalium*. Image taken from BioCyc.

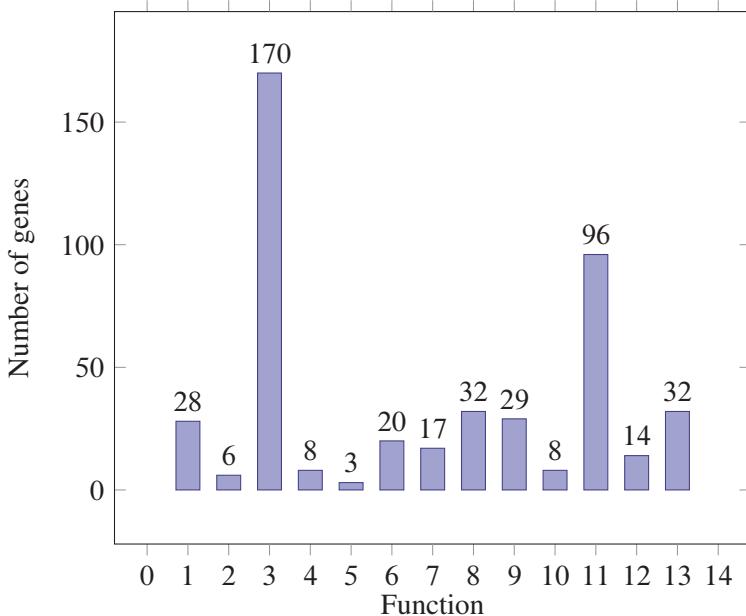
not segmented (genes made of introns and exons); that is, a gene in *E. coli* is a contiguous sequence of DNA translated into the final protein without editing. In addition, there is very little non-coding DNA in *E. coli* with almost 88% of the genome coding for proteins.

Almost one quarter of all proteins produced by gene expression in *E. coli* form multimers, proteins composed of multiple subunits. Many of these multimers are homomultimers, meaning they are made up of the same subunits. Some of these proteins can also be covalently modified by phosphorylation, methylation or other means. There are estimated to be at least 171 transcription factors that directly control gene expression. This number provides insight into the size of the *E. coli* gene regulatory network. The EcoCyc database reports at least 48 small molecules and ions that regulate these transcription factors.

Of the 4,316 genes in *E. coli*, 3,384 (76%) have been assigned a biochemical function. There are at least 991 genes involved directly in metabolism with a further 355 genes involved in transport. Other gene functions include DNA replication, recombination and repair, protein folding, transcription, translation and regulatory proteins. An inventory of small molecules has not been thoroughly made, but EcoCyc records at least 1,352 unique small organic molecules, but there are likely many more.

These statistics suggest large numbers of interactions among many thousands of cellular components forming extensive networks.

Given the size of a single *E. coli* cell, the concentration of protein in the cytoplasm, and the average diameter of a protein (5 nm), it is estimated that the average spacing (center to



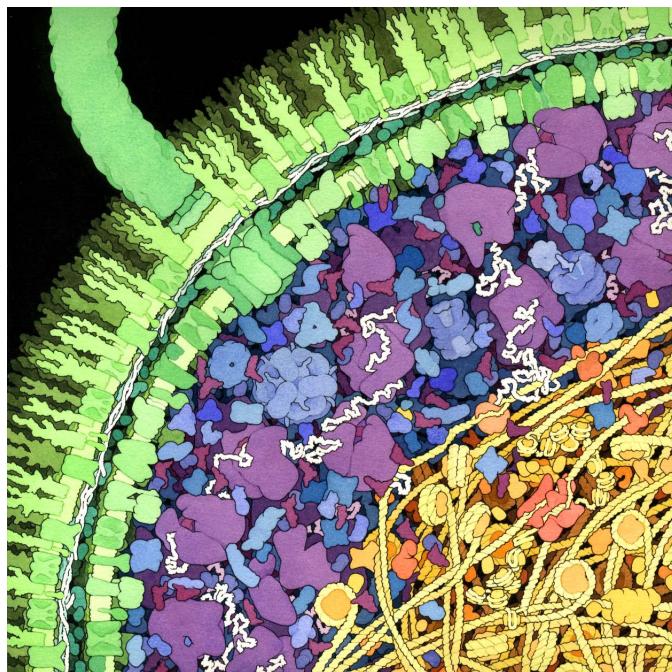
**Figure 1.17** 1: Cell Envelope; 2: Regulatory; 3: Unknown; 4: Central Metabolism; 5: Cofactor Biosynthesis; 6: Purine/Pyrimidine metabolism; 7: Transcription; 8: Transport; 9: Replication/Repair; 10: Lipid Metabolism; 11: Translation; 12: Cellular Processes; 13: Energy Production.

center) between proteins is about 7 nm. This suggests that the cytoplasm is quite dense. David Goodsell (<http://mgl.scripps.edu/people/goodsell>) is well known for his evocative illustrations of subcellular spaces. Figure 1.18 illustrates his rendition of a cross-section through *E. coli* and gives a vivid impression of how packed the cytoplasm is.

There are two useful websites for obtaining basic operating information on *E. coli*. The first is the *E. coli* statistics site at Alberta ([http://gchelpdesk.ualberta.ca/CCDB/cgi-bin/STAT\\_NEW.cgi](http://gchelpdesk.ualberta.ca/CCDB/cgi-bin/STAT_NEW.cgi)). The other is a more generic and community based website called BIONUMBERS (The Database of Useful Biological Numbers). Publications from the project also supply many useful statistics on *E. coli* [84, 86].

The number of molecules in a typical *E. coli* varies with the molecule type. For example, there are approximately 2,000,000  $\text{Na}^+$  ions while only 300,000 tryptophan molecules. The larger the molecule, the fewer their number (Table 1.4). For example, transcription factors are only present in numbers ranging from 10s to 100s, whereas ions are present in the millions.

A significant study by Bennett et al. [9] measured over 100 metabolite levels in the main metabolic pathways of glucose-fed, exponentially growing *E. coli*. The average concentration was found to be 0.22 mM. We can compare this with the average  $K_m$  (concentration of substrate that gives half maximal activity) of approximately 0.1 mM as reported by the



**Figure 1.18** Artists impression (With permission, Goodsell) of a cross-section through *E. coli* illustrating the high density of proteins and other molecules in the cytoplasm, drawn roughly to scale <http://mgl.scripps.edu/people/goodsell/illustration/public>.

database. This suggests that on average enzymes operate above their half maximal activity. However, a more detailed analysis revealed considerable variability among different metabolite types. For example, cofactors such as ATP and NAD<sup>+</sup> were at concentrations significantly above their  $K_{ms}$ . In contrast, substrate-enzyme pairs where the concentration was below the  $K_m$  were dominated by enzymes catalyzing nucleotide, nucleoside, nucleobase and amino acid degradation reactions. On the other hand, the glycolytic pathway, tricarboxylic acid cycle, and the pentose-phosphate pathways all showed substrate concentration that were similar to their  $K_m$  values.

We can also consider how fast processes occur in *E. coli*. As suggested earlier in the chapter, metabolic responses are the fastest followed by protein signaling networks and gene regulatory networks. Table 1.7 lists some estimated response times for various biological processes.

The number of molecules and the rate of various processes gives some idea of the magnitude of systems we are dealing with. However, the economy of a typical cell, how ATP is distributed to different processes and how supply and demand are maintained, is largely not understood since many of these processes are difficult to measure. Moreover, there is

Property	Dimensions
Length	2 to 3 $\mu\text{m}$
Diameter	$\approx 1 \mu\text{m}$
Volume	$1 \times 10^{-15} \text{ L}$
Optimal generation time	20 to 30 mins
Translation rate	40 amino acids per sec
Transcription rate	70 nucleotides per sec
Number of ribosomes per cell	18,000
Average protein diameter	5 nm
Average concentration of protein	5-8 mM
Average number of proteins	3,600,000

**Table 1.3** Basic Information on *E. coli*.

Molecule	Estimated Number
Ions	Millions
Small Molecules	10,000 - 100,000
Metabolic Enzymes	1000 - 10,000s
Signaling Molecules	100 - 1,000s
Transcription factors	10s to 100s
DNA	1 - 10s

**Table 1.4** Orders of magnitude for various *E. coli* molecule types.

no economic theory that describes the life of a cell. This is a significant omission in our understand of organisms, particularly when we attempt to engineer them.

## 1.8 Network Motifs

At first glance the complex biochemical maps we see on lab walls and in text books appear to have little order. However on closer examination, patterns emerge. One way to discern these patterns is to compare real biochemical networks with random networks and to look for a given pattern in each. For example, let's say we identify a pattern of regulation which we label  $p_1$ . We look for the occurrence of  $p_1$  in both the real biochemical network and the randomly generated network. If we find that the pattern is statistically enriched in the real biochemical network compared to the randomly generated one, we say we have found a **network motif**.

A motif is a subgraph within a network that occurs more often than one would expect by

Ions	Estimated Numbers
Na	3,000,000
Ca	2,300,000
Fe	7,000,000
Small Molecules	Estimated Numbers
Alanine	350,000
Pyruvate	370,000
ATP	2,000,000
ADP	70,000
NADP	240,000

**Table 1.5** Small molecule estimates in *E. coli*.

random chance alone. Such subgraphs can be simple triangles, squares etc. It is assumed that such motifs occur more frequently because they confer some functional advantage; their identification is therefore considered important. Locating motifs in a large network entails a three step process:

- Estimate the frequency of each isomorphic subgraph in the target network.
- Generate a suitable random graph to test the significance of the frequency data.
- Compare the target network with the random graph.

The critical stage is generating a suitable random model for comparison. The approach is to generate a random network which has a degree distribution<sup>1</sup> that is the same as the degree distribution of the real target network [116, 117, 118]. One way to accomplish this is by starting with the target network itself and randomizing edges in such a way that the original degree distribution is preserved. This is carried out multiple times in order to generate a population of random networks. Once the random and target networks are ready, additional algorithms are invoked to count the number of given motifs. The frequency distribution of the motif in the random networks is then compared to the frequency distribution of the target network. A simple significance test can be carried out using the z-score (1.1). The z-score is computed by subtracting the number of a given motif in the target network from the mean number of the same motif in the randomized networks. This difference is then normalized by dividing by the standard deviation of the motif count in the random population. If the z-score is greater than zero, then the observed number of motifs is greater than the mean, while a negative z-score indicates that the observed number of motifs is below the mean. A z-score of two indicates that the observed value is two standard deviations above the mean

<sup>1</sup>The degree of a node is the number of edges incident on the node.

Signaling Proteins	Estimated Numbers
LacI	10 to 50
CheA kinase	4,500
CheB	240
CheY	8,200
Chemoreceptors	15,000
Metabolic Enzymes	Estimated Numbers
Phosphofructokinase	1,550
Pyruvate Kinase	11,000
Enolase	55,800
Phosphoglycerate kinase	124,000
Malate Dehydrogenase	3,390
Citrate Synthase	1,360
Aconitase	1,630

**Table 1.6** Estimated numbers for larger molecules in *E. coli*.

which can also be roughly interpreted as the 95% confidence level. That is, if a z-score is two or above, the number of motifs is significantly different from a random network suggesting that the motif has some functional significance.

$$z = \frac{n - n_r}{\sigma_r} \quad (1.1)$$

The definition of a motif, while useful, has important restrictions. For example, consider a large electronic circuit containing transistors, resistors and capacitors. A motif search in such a circuit may find an overabundance of amplifier like motifs compared to a completely random circuit. However, such an analysis will not find specialist circuits such as

Process	Rate
Cell Division Time	50 minutes
Rate of Replication	2,000 bp/s
Protein Synthesis	1,000 proteins/s
Lipid Synthesis	20,000 lipids/s
Ribosome Rates	25 amino acids per sec per ribosome
Number of ATP to make one cell	55 billion ATPs

**Table 1.7** *E. coli* grown on minimal media plus glucose. Data from Phillips et al. (2010) and *E. coli* stats reference: <http://ccdb.wishartlab.com>.

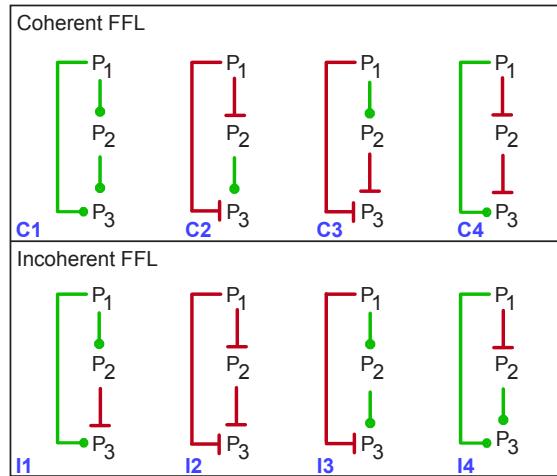
a resonance filter, which may only occur once in the circuit. The motifs located using this approach therefore need to be fairly common in the network. The operational definition of motifs excludes motifs which may only appear once in a network but whose role is critical to the network's function [162].

One motif that has been both theoretically and experimentally studied is the feedforward loop (FFL). We will discuss this motif in more detail in Chapter 13. Here we will briefly mention its relative abundance in real networks. Figure 1.19 illustrates motif findings in part of yeast data using the MAVisto software [159]. The software has picked up a number of feedforward loop motifs.

**Figure 1.19** Occurrences of feedforward loop motifs as generated by the software MAVisto [159]. The displayed network is part of yeast data supplied with the MAVisto software. The software is very straightforward to use and will identify a wide variety of motifs. Other similar tools include FANMOD [182] and the original motif tool mFinder [85].

The FFL has a simple structure; there is a single input,  $P_1$ , and a single output,  $P_3$ . There are two routes from the input to the output nodes, one is direct and the other goes via an intermediate node,  $P_2$ . Figure 1.24b shows a generic FFL. Given this basic structure we can imagine various combinations of activation and repression on the edges for a total of eight combinations. These are shown in Figure 1.20. We can further categorize the eight FFLs into two groups of four, coherent and incoherent. Incoherent FFLs are those where

the two routes have opposite effects on the output. Coherent FFLs are where the routes have the same effect on the output. A motif search for all eight types in *E. coli* and yeast reveals an asymmetry in the relative abundance in the different types. Most noticeably from Figure 1.21 we see that two types predominate in both organisms, Coherent Type 1 (C1) and Incoherent Type 1 (I1).

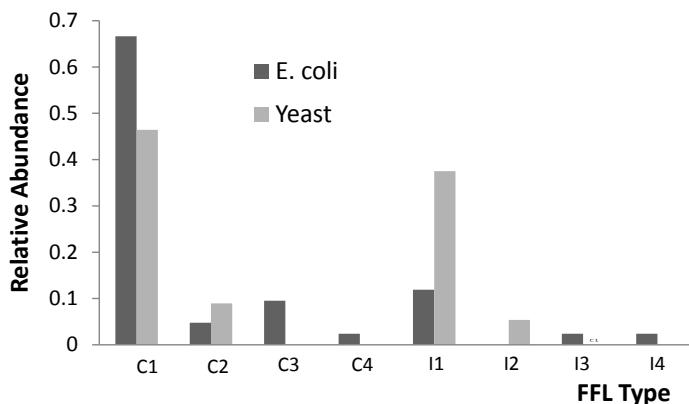


**Figure 1.20** Full complement of feedforward motifs, classified into coherent and incoherent types.

More interesting is that these two types have distinct behavioral properties. We will return to the question of what dynamics these networks can display in Chapter 13 where we will use simulation as a guide.

### Menagerie of Motifs

The feedforward network described in the previous section is one of many different kinds of identified motifs. It would take an entire book to describe them all. Instead we will summarize them here, together with their basic dynamic properties. Figures 1.22, 1.23 and 1.24 show a variety of motifs. No doubt many more natural patterns remain to be discovered, in addition to new motifs that have and will be designed by the synthetic biology community [45].



**Figure 1.21** Relative abundance of different FFL types in yeast and *E. coli*. Labels on the x-axis refer to the particular FFL motifs seen in Figure 1.20. Data taken from [105].

## Further Reading

---

### General

- Bray D (2011) Wetware: A Computer in Every Living Cell. Yale University Press. ISBN: 978-0300167849
- Goodsell D S (2009) The Machinery of Life. Springer, 2nd edition. ISBN 978-0387849249
- Phillips R, Kondev J and Theriot J (2010) Physical Biology of the Cell. Garland Science. ISBN 978-0-8153-4163-5

### Specific

- Alberts et al., (2002) General Principles of Cell Communication <http://www.ncbi.nlm.nih.gov/books/NBK26813/>
- Brown TA (2006) Genomes 3, Garland Science, 3rd edition. ISBN: 978-0815341383
- Gerhard M and Schomburg D (2012) Biochemical Pathways: An Atlas of Biochemistry and Molecular Biology, Wiley, 2nd edition. ISBN: 978-0470146842
- Hancock J (2010) Cell Signalling, Oxford University Press, 3rd edition. ISBN: 978-0199232109
- Hartl DL (2008) Genetics: Analysis Of Genes And Genomes. Jones & Bartlett Learning, 7th edition. ISBN: 978-0763772154

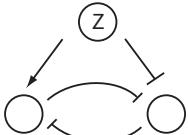
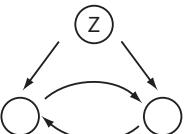
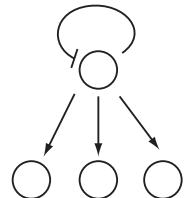
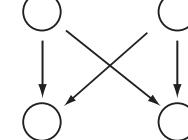
Motif Structure	Name	Dynamic Properties
a) 	Negative Autoregulation	1. Noise suppression 2. Accelerated response 3. High fidelity amplifier 4. Feedback oscillator
b) 	Positive Autoregulation	1. Bistability 2. Memory unit 3. Op Amp component
c) 	Relaxation Oscillator	Adaptable oscillator
d) 	Double Positive Feedback	Memory unit where both units are either on or off
e) 	Double Negative Feedback	Memory unit where one unit is off, the other is on

**Figure 1.22 Motifs.**

6. Nelson DL and Cox MM (2008) Wetware: Lehninger Principles of Biochemistry. W. H. Freeman, 5th edition. ISBN: 978-0716771081
7. Salway JG (2004) Metabolism at a Glance, Wiley-Blackwell, 3rd edition. ISBN: 978-1405107167

## Motifs

1. Alon U, (2006) An Introduction to Systems Biology: Design Principles of Biological Circuits, Chapman & Hall/Crc Mathematical and Computational Biology Series.
2. Sauro, HM and Kholodenko, BN, (2004), Quantitative analysis of signaling networks, Progress in Biophysics and Molecular Biology, 86, 5–43.

Motif Structure	Name	Dynamic Properties
a) 	Regulated Double Negative Feedback	Memory unit where nodes switch in opposite directions due to an event Z
b) 	Regulated Double Positive Feedback	Memory unit that records an event in Z
c) 	Single Input Module (SIM)	1. Master/Slave regulator 2. Temporal sequencer - last gene activated is the first gene deactivated
d) 	Bi-Fan	1. Neural network type computations 2. Synchronizers 3. Filters

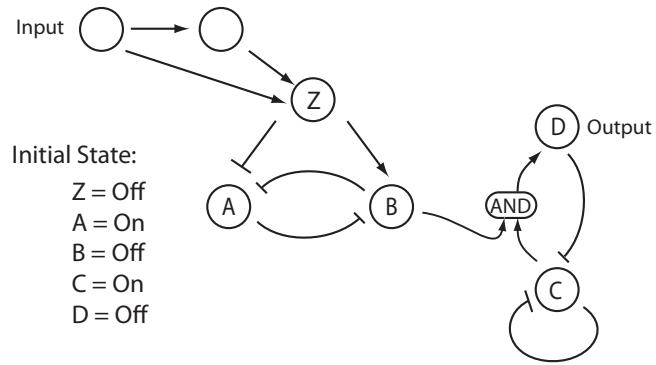
**Figure 1.23 Motifs.**

3. Tyson JJ, Chen, KC and Novak, B, Sniffers, buzzers, toggles and blinkers: dynamics of regulatory and signaling pathways in the cell, *Current Opinion in Cell Biology*, 2003, 15, 221–231.
4. Yosef N and Regev A (2011), Impulse Control: Temporal Dynamics in Gene Expression. *Cell* 144, 886–896.

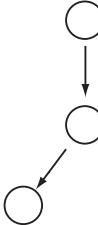
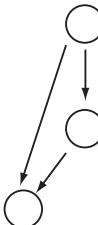
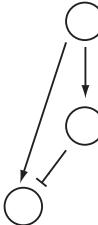
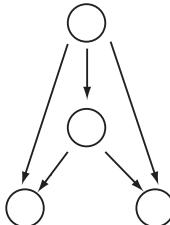
## Exercises

In the following exercises use the data given in the main text along with Tables 1.3, 1.4, 1.5, and 1.6.

1. How many *E. coli* cells laid end to end would fit across the full stop at the end of this sentence? Assume the diameter of the full stop is 0.5 mm.
2. Estimate the volume of an *E. coli* cell.
3. Calculate the surface area of an *E. coli* cell. If a typical membrane protein is 5 nm in diameter, estimate the number of membrane proteins that can be laid out on the membrane if the center-center distance between each protein is 6 nm.
4. Show that a 1 nM concentration is roughly equivalent to 1 molecule in a volume of 1 *E. coli* cell.
5. Estimate the number of protein molecules a typical *E. coli* cell can make per second assuming the average protein is 360 amino acids long. Assume that the number of proteins in a cell is 3,000,000. How long would it take to make 3,000,000 proteins?
6. If it takes 1,500 ATP molecules to make an average protein, how long would it take before all the ATP is used up? Assume the ATP is not being replaced.
7. *E. coli* can be considered a cylindrical volume with length 2  $\mu\text{m}$  and diameter 1  $\mu\text{m}$ . A reaction is known to occur in *E. coli* with an intensive rate of  $0.5 \text{ mmol s}^{-1} \text{ l}^{-1}$ . What is the rate of reaction per volume of *E. coli*? If Avogadro's number is  $6.022 \times 10^{23}$ , express the rate in terms of molecules converted per second per *E. coli*.
8. What are the visual symbols often used to represent activation and repression in biochemical networks?
9. Draw a similar diagram to the glycolysis regulatory diagram (Figure 1.4) but for the lysine, threonine and methionine biosynthesis pathway from *E. coli*.
10. Why is the size of an organism's genome a poor indicator of the organism's complexity?
11. Describe the basic approach used to find network motifs.
12. Looking at motif b) in Figure 1.23, try to explain how it might operate as a memory unit.
13. Study the network shown below and try to figure out its function. Use Figures 1.22, 1.23, and 1.24 as guides.



The AND block on the left of the network represents an AND gate, that is the output of the block is only active if *both* inputs are also active.

Motif Structure	Name	Dynamic Properties
a) 	Cascade Unit	1. Noise filter 2. Nonlinear amplifier
b) 	Coherent Feedforward	1. Noise rejection 2. Pulse shifter
c) 	Incoherent Feedforward	1. Pulse generator 2. Concentration detector 3. Response time accelerator
d) 	Multi-Output FFL	Pulse train generator

**Figure 1.24** Feedforward Network Motifs.

# 2

## *Kinetics in a Nutshell*

---

### 2.1 Introduction

---

Understanding chemical kinetics is at the heart of building biochemical models. This chapter gives a minimal introduction to some of the essential concepts of elementary chemical kinetics. A fuller account is given in the companion book, ‘Enzyme Kinetics for Systems Biology’. This chapter may be omitted by those already familiar with this topic.

---

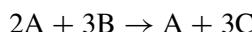
### 2.2 Definitions

---

Reaction kinetics is the study of how fast chemical reactions take place, what factors influence the rate of reaction, and what mechanisms are responsible.

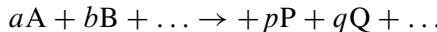
#### **Stoichiometric Amount**

The **stoichiometric amount** is the number of molecules for a particular reactant or products takes part in a given reaction reaction. For example:



In the above example the stoichiometric amount for reactant A is 2 and for B is 3. The stoichiometric amount for product A is 1 and for C is 3.

## Depicting Reactions



where  $a, b, \dots, p, q, \dots$  are stoichiometric amounts.

## Rates of Change

The rate of change is defined as the rate of change in concentration or amount of a designated molecular species.

$$\text{Rate of Change} = \frac{dA}{dt}$$

## Stoichiometric coefficients

The **stoichiometric coefficient**,  $c_i$ , for a molecular species  $A_j$ , is the difference between the molar amount of species,  $i$  – also called the **stoichiometric amount** – on the product side and the molar amount of the same species on the reactant side.

$$c_i = \text{Molar Amount of Product} - \text{Molar Amount of Reactant}$$

In the reaction,  $2 A \rightarrow B$ , the molar amount of A on the product side is zero while on the reactant size it is two. Therefore the stoichiometric coefficient of A is given by  $0 - 2 = -2$ . In many cases a particular species will only occur on the reactant or product side but it is not uncommon to find situations where a species occurs simultaneously as a product and a reactant. As a result, reactant stoichiometric coefficients tend to be *negative* while product stoichiometric coefficients tend to be *positive*.

## Reaction Rates

The **reaction rate**, often denoted by the symbol  $v$ , is measured with respect to a given molecular species normalized by the species stoichiometric coefficient. This definition ensures that no matter which molecular species in a reaction is measured, the reaction rate is uniquely defined for that reaction. More formally, the reaction rate for the given reaction is:

$$aA + bB + \dots \rightarrow pP + qQ + \dots$$

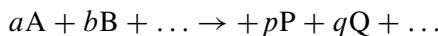
$$v = \frac{1}{c_a} \frac{dA}{dt} = -\frac{1}{c_b} \frac{dB}{dt} \dots = \frac{1}{c_p} \frac{dP}{dt} = \frac{1}{c_q} \frac{dQ}{dt} \dots$$

where  $c_x$  are the stoichiometric coefficients. Alternatively, we can express the rate of change in terms of the reaction rate as:

$$\frac{dA}{dt} = c_a v \quad (2.1)$$

## 2.3 Elementary Mass-Action Kinetics

An elementary reaction is one that cannot be broken down into simpler reactions. Such reactions will often display simple kinetics called mass-action kinetics. For a reaction of the form:



the mass-action kinetic rate law is given by:

$$v = k_1 A^a B^b \dots - k_2 P^p Q^q \dots \quad (2.2)$$

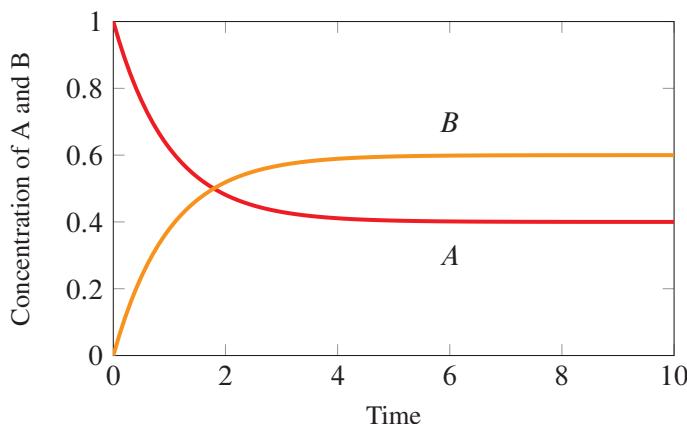
$k_1$  and  $k_2$  are the forward and reverse rate constants, respectively.

## 2.4 Chemical Equilibrium

In principle, all reactions are reversible, meaning transformations can occur from reactant to product or product to reactant. The net rate of a reversible reaction is the difference between the forward and reverse rates. Given a reversible reaction such as:



we can observe the concentrations of A and B approach equilibrium (Figure 2.3).

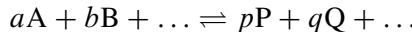


**Figure 2.1** Approach to equilibrium for the reaction  $A \rightleftharpoons B$ ,  $k_1 = 0.6$ ,  $k_2 = 0.4$ ,  $A(0) = 1$ ,  $B(0) = 0$ . Progress curves calculated from the solution to the differential equation  $dA/dt = k_2 B - k_1 A$ .

At chemical equilibrium the forward and reverse rates are equal and is described by the relation:

$$\frac{k_1}{k_2} = \frac{B}{A} = K_{eq} \quad (2.3)$$

This ratio has special significance and is called the **equilibrium constant**, denoted by  $K_{eq}$ . The equilibrium constant is also related to the ratio of the rate constants,  $k_1/k_2$ . For a general reversible reaction such as:

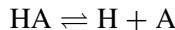


and using arguments similar to those described above, the ratio of the rate constants can be easily shown to be:

$$K_{eq} = \frac{P^p Q^q \dots}{A^a B^b \dots} = \frac{k_1}{k_2} \quad (2.4)$$

where the exponents are the stoichiometric *amounts* for each species.

For a bimolecular reaction such as:



chemists and biochemists will often distinguish between two kinds of equilibrium constants called association and dissociation constants. Thus the equilibrium constant for the above bimolecular reaction is often called the **dissociation constant**,  $K_d$ :

$$K_d = \frac{H \cdot A}{HA} \quad (2.5)$$

to indicate the degree that the complex is dissociated into its component molecules at equilibrium. As a reminder, italicized symbols such as  $H$ ,  $HA$ , etc, represent the concentration of the particular species. The **association constant**,  $K_a$ , though less commonly used, describes the equilibrium constant for the reverse process  $H + A \rightleftharpoons HA$ , that is the formation of a complex from component molecules:

$$K_a = \frac{HA}{H \cdot A} \quad (2.6)$$

It should be evident that:

$$K_d = \frac{1}{K_a} \quad (2.7)$$

The equilibrium constant is also related to the standard free energy change,  $\Delta G^\circ$ , such that:

$$\Delta G^\circ = -RT \ln K_{eq}$$

where  $R$  is the gas constant, and  $T$  the temperature. Rearranged we can also see that:

$$K_{eq} = e^{-\Delta G^\circ / RT} \quad (2.8)$$

## 2.5 Mass-action and Disequilibrium Ratio

Although in closed systems reactions tend to equilibrium, reactions occurring in living cells are generally out of equilibrium and the ratio of the products to the reactants *in vivo* is called the **mass-action ratio**,  $\Gamma$ . The ratio of the mass-action ratio to the equilibrium constant is called the **disequilibrium ratio**:

$$\rho = \frac{\Gamma}{K_{eq}} \quad (2.9)$$

At equilibrium the mass-action ratio will be equal to the equilibrium constant, that is  $\rho = 1$ . If the reaction is away from equilibrium then  $\rho \neq 1$ .

For a simple unimolecular reaction it was previously shown that the equilibrium ratio of product to reactant,  $B/A$ , is equal to the ratio of the forward and reverse rate constants. Substituting this into the disequilibrium ratio gives:

$$\rho = \Gamma \frac{k_2}{k_1} = \frac{B}{A} \frac{k_2}{k_1}$$

Therefore:

$$\rho = \frac{v_r}{v_f} \quad (2.10)$$

That is, the disequilibrium ratio is the ratio of the reverse and forward rates. If  $\rho < 1$ , the net reaction must be in the direction of product formation. If  $\rho$  is zero, the reaction is as out of equilibrium as possible with no product present.

## 2.6 Modified Mass-Action Rate Laws

A typical reversible mass-action rate law will require both the forward and the reverse rate constants to be fully defined. Often however, only one rate constant may be known. In these circumstances it is possible to express the reverse rate constant in terms of the equilibrium constant.

For example, given the simple unimolecular reaction,  $A \rightleftharpoons B$ , it is possible to derive the

following:

$$\begin{aligned} v &= k_1 A - k_2 B \\ v &= k_1 A \left(1 - \frac{k_2 B}{k_1 A}\right) \\ \text{Since } K_{eq} &= \frac{k_1}{k_2} \\ v &= k_1 A \left(1 - \frac{\Gamma}{K_{eq}}\right) \end{aligned} \quad (2.11)$$

where  $\Gamma$  is the mass-action ratio. This can be generalized to an arbitrary mass-action reaction to give:

$$v = k_1 A^a B^b \dots \left(1 - \frac{\Gamma}{K_{eq}}\right) = k_1 A^a B^b \dots (1 - \rho) \quad (2.12)$$

where  $A^a B^b \dots$  represents the product of all reactant species,  $a$  and  $b$  are the **corresponding** stoichiometric amounts, and  $\rho$  is the disequilibrium ratio. The advantage of this expression is that equilibrium constants are more experimentally assessable than rate constants. For the reaction:



where  $k_1$  is the forward rate constant, the modified reversible rate law is:

$$v = k_1 A^2 B (1 - \rho) \quad (2.13)$$

The modified formulation demonstrates how a rate expression can be divided up into functional parts to include both kinetic and thermodynamic components [73]. The kinetic component is represented by the term  $k_1 A^a B^b \dots$ , while the thermodynamic component is represented by the expression  $1 - \rho$ .

We can also derive the modified rate law in the following way. Given the net rate of reaction  $v = v_f - v_r$ , we can write this expression as:

$$v = v_f \left(1 - \frac{v_r}{v_f}\right)$$

That is:

$$v = v_f (1 - \rho) \quad (2.14)$$

## Further Reading

---

1. Sauro HM (2012) Enzyme Kinetics for Systems Biology. 2nd Edition, Ambrosius Publishing ISBN: 978-0982477335

# 3

## *Stoichiometric Networks*

### **3.1 Stoichiometric Networks**

---

Almost all cellular events involve some kind of chemical process that includes binding, unbinding, or transformation of compounds in specific stoichiometric amounts. The binding of the yeast cell cycle proteins cdc2 and cdc13 to form a cdc2-cdc13 complex, or the isomerization of glucose-6-phosphate to fructose-6-phosphate are two notable examples. When we put a collection of these processes together, we form a **stoichiometric network**.

One of the key characteristics of stoichiometric networks is that mass is conserved at each transformation step. For example, the transformation  $S_1 \rightarrow S_2$  means that when one molecule of  $S_1$  disappears, one molecule of  $S_2$  is formed. An example of a very simple and minimal stoichiometric network is the two step pathway shown below:



In this system mass is conserved at every stage. In more sophisticated models where electric charge is also considered, charge will be conserved as well.

#### **Elementary Reactions**

Chemical reactions that involve no reaction intermediates other than a single transition state are called **elementary reactions**.

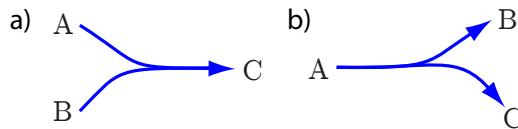
Elementary reactions have been depicted in a number of ways in the literature. For example, the transformation of one species into another can be represented by a simple line with an arrow at the tip. The direction of the arrow indicates the direction of the *positive* reaction rate (Figure 3.1). If a reaction rate is  $-0.75 \text{ mol l}^{-1}$ , this means the reaction proceeds in the *opposite* direction indicated by the arrow at a rate of  $0.75 \text{ mol l}^{-1}$ . Most if not all



**Figure 3.1** Simple Transformations. a) A single arrow, indicates positive rate direction. b) Two arrows showing explicit reversibility. c) Common barb style used to indicate reversibility. d) Reversibility with dominant arrow indicating positive direction.

reactions are in principle reversible, that is, the reaction can only go in both directions. Unless otherwise stated by the author, the reversibility is defined by the rate law attached to the reaction. Sometimes reversibility is explicitly indicated by using multiple arrows. These come in various forms. One approach is to use two lines and add arrowheads to both the reactant and product line as shown in Figure 3.1b. Other authors add a smaller reverse arrow as shown in Figure 3.1d, or more commonly use a barbed style as shown in Figure 3.1c. In example (d) and (c) it is not possible to know which direction represents the positive reaction rate unless it is assumed left to right.

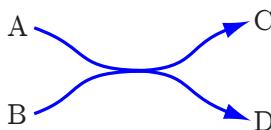
For a bimolecular reaction that depicts dissociation or association, the notation is shown in Figure 3.2 (a) and (b). This style makes it clear that there is a stoichiometric constraint



**Figure 3.2** Dissociation and Association Reactions. (a) Equal stoichiometric proportions of compounds A and B combine to form a complex, C. (b) Likewise, complex A dissociates into equal proportions of B and C.

between A and B and B and C. One molecule of A reacts with one molecule of B to form one molecule of C. This notation can be misused for example where lines departing from a branch point are joined, thereby implying a stoichiometric constraint when none actually exists.

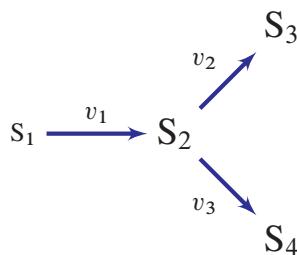
The simple association and dissociation reactions can be naturally extended to depict situations where both association and dissociation occur in the same reaction as shown in Figure 3.3.



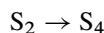
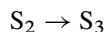
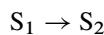
**Figure 3.3** A bimolecular interaction, coupling one process, A to C, to another, B to D. Equal proportions of A and B combine to form equal proportions of C and D.

*Example 3.1*

Write out the individual reactions for the following network, taking care to indicate the correct stoichiometries.



Answer:

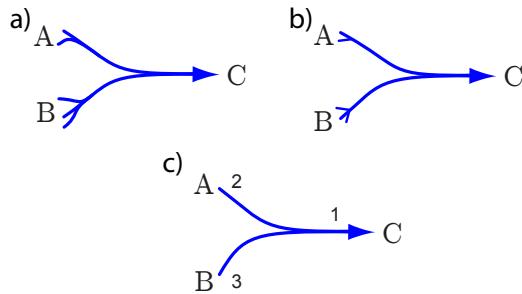


One area that is sometimes problematic is visually depicting reactions with non-unity stoichiometry. The previous examples assumed that each molecular species had a stoichiometry of one. However, what if species A in Figure 3.2 has a stoichiometry of 2 and B a stoichiometry of 3. How should these be represented? Figure 3.4 shows three depictions that have been used by authors in the past. Sometimes simple arc extensions are used to indicate the stoichiometry, as seen in Figure 3.4a. A variation of (a) is to use small barbs at the tips of the reaction arcs [29] where the number of barbs indicate the stoichiometry as seen in Figure 3.4b. Finally, stoichiometric numbers may be placed near the tips of the arcs, as shown in Figure 3.4c.

*Example 3.2*

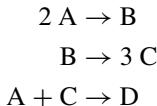
The following network is made from elementary reactions. Write out the individual reactions, taking care to indicate the correct stoichiometries.





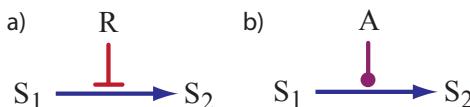
**Figure 3.4** Alternative ways for visually depicting non-unit stoichiometries. The use of numbers in (c) makes it possible to depict fractional stoichiometries.

Answer:



## Non-Elementary Reactions

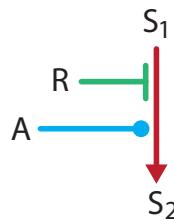
Non-elementary reactions include all reactions that have hidden reaction intermediates. The most familiar is the enzymatic reaction where the enzyme-substrate complex and free enzyme are rarely shown in network diagrams. The effect of hiding intermediates is that it is possible to include regulatory links. For example, an enzyme may be regulated by an allosteric effector where the mechanism is quite complex. Very often this mechanism will be hidden and instead, the action of the effector will be represented by a simple regulatory line in the diagram. For example, if an enzyme that catalyzes the conversion of species  $S_1$  to  $S_2$  is inhibited by a repressor molecule  $R$ , or activated by a activator  $A$ , then we can depict this situation as shown in Figure 3.5.



**Figure 3.5** Depicting regulation: a) Repression; b) Activation.

The blunt end representing inhibition is fairly well established in the literature, while the activation symbol is more variable. Here we will employ a filled circle at the end point to

indicate activation. If a non-elementary reaction is regulated by multiple inputs, we would use a depiction similar to what is shown in Figure 3.6.



**Figure 3.6** Multiple regulators on one reaction.

In hiding detailed mechanisms we also invoke certain assumptions when converting the diagrams to a mathematical model. In the case of a simple enzyme mechanism, we will often assume the rapid-equilibrium or steady state assumption for the formation of enzyme-substrate complex (See Appendix D). Sometimes these assumptions are reasonable, other times they are not. For a more comprehensive discussion of these issues see the companion book ‘Enzyme Kinetics for Systems Biology’.

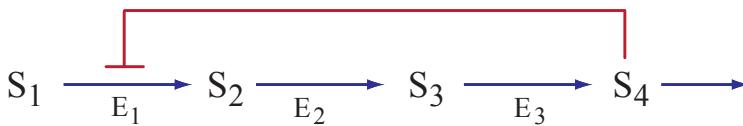
In Chapter 1 the diagrammatic notation shown in Figure 3.7 was used for representing gene regulatory networks. This single genetic unit is certainly non-elementary as it hides a considerable amount of detail. We can treat the unit as if it were a reaction step whose rate of reaction is the rate of protein expression. It is important to remember that whenever



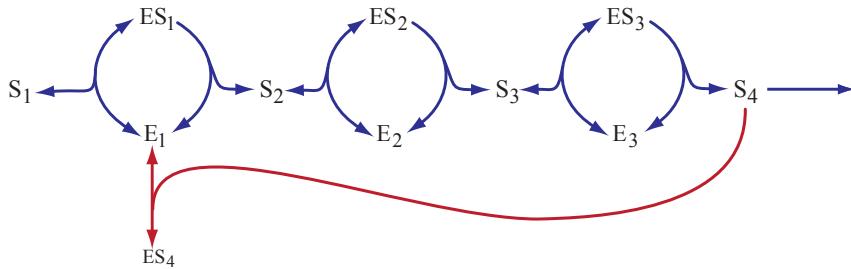
**Figure 3.7** Representing a single gene. I represents the inducer and P the expressed protein.

one sees a regulatory link in a reaction step, it always means that the reaction is non-elementary and hides other mechanistic details. The use of non-elementary reactions is a high level representation because unwrapping every non-elementary reaction into its full set of elementary reactions would make the network overly complex to view, understand and parameterize. Figure 3.8 illustrates an example of a simple pathway drawn using non-elementary reactions together with a feedback inhibition step and the equivalent unwrapped view of the same system. The exploded view is clearly more complex. The mechanism chosen for the inhibition is the simplest possible, and therefore the unwrapped view could potentially be even more complex. There will be many instances where we will not know how an effector acts mechanistically and therefore unwrapping an elementary reaction is not an option.

### A) Network made from Non-elementary Steps



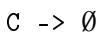
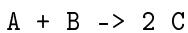
### B) Equivalent Network made from Elementary Steps



**Figure 3.8** Equivalent networks made from non-elementary and elementary components.

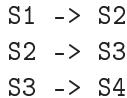
## Text Representation

Although representing biochemical networks using pictures is very common, it is also possible to represent networks using a text notation. Text representations are particularly easy for computers to read. For example, a linear chain of four reactions is shown in Figure 3.9. If a species is converted to a waste product such as degradation fragments, then the symbol  $\emptyset$  is typically used to represent the empty species set. For example:



Three software tools that support text based input are Jarnac [150], Antimony [165], and PySCeS [125]. The Python based application called Tellurium ([tellurium.analogmachine.org](http://tellurium.analogmachine.org)) integrates Antimony and the simulator libRoadRunner ([libroadrunner.org](http://libroadrunner.org)) and will be used to illustrate all simulations in this book. There are also rule-based text notations but these are beyond the scope of this book and are supported by tools such as BioNetGen or PySB. See Maus et al. for a review [112].

In this book models will be expressed in the Antimony syntax which was itself derived and improved from the Jarnac syntax. For example, to represent the above model, we would write the script in Antimony as shown in Listing 3.1.



**Figure 3.9** Simple textual representation of a linear chain of three reactions and four molecular species.

```
// Example Antimony script
$A + B -> 2 C; k1*A*B;
C -> ; k2*C;
```

### Listing 3.1 Example Antimony model.

The \$ sign in front of species A means that the species concentration is fixed. This means that when compiled by a simulator such as libRoadRunner no differential equation for this species will be generated<sup>1</sup>.

The \$ sign in front of a species A means that the concentration of A is fixed.

Also note that Antimony permits empty reactants or products in a reaction. In this case the second reaction that consumes C does not specify what C is converted to, only its rate,  $k2*C$ . The implication here is that the products of reaction from C emerge into a large volume such that their concentrations remain approximately unchanged during the process. It also implies that the products do not affect the reaction, also evident in the rate law.

We can also use Antimony to initialize concentrations and parameters, Listing 3.2.

```
1 // Example Antimony script with value initialization
2   $A + B -> C; k1*A*B;
3   C -> ; k2*C;
4
5 // Initialize values
6 k1 = 0.34; k2 = 4.5;
7 A = 10; B = 0; C = 0;
```

### Listing 3.2 Example Antimony model with initialization.

Antimony offers a host of other features including models composed of other models, and the ability to specify discrete events directly in the model.

For example:

```
// Example Antimony script
```

<sup>1</sup>We will return to the idea of ‘fixed species’ in the next chapter.

```

A + B -> C; k1*A*B;
C -> ; k2*C;

// When time reaches 5 time units, halve the k2 rate constant
at (time >= 5) : k2 = k2/2;

k1 = 0.34; k2 = 4.5;
A = 10; B = 0; C = 0;

```

**Listing 3.3** Example Antimony script with events.

## 3.2 Standard Visualization Notation

Cellular networks have been depicted on wall charts for many decades using a variety of informal notations which we have briefly reviewed. With the increased interest in protein and gene regulatory networks, the variety of notations has proliferated. As a result, there have been some efforts, most notably the Systems Biology Graphical Notation (SBGN), to define a standard set of node and edge symbols to represent stoichiometric networks. Another visual notation is employed by Biotapestry [100] which provides a concise and easy to read notation for representing gene regulatory networks.

SBGN can represent stoichiometric networks using a notation called SBGN process description. For example, Figure 3.10 illustrates the SBGN approach to representing an enzyme catalyzed reaction. Round shaped nodes or a stadium shape (pill shaped) represent small molecules such as DHAP, ATP and F6P. Rounded rectangles are used to represent macromolecules, in this case enzymes TPase (Triose phosphate Isomerase) and PFK (phosphofructokinase). In the second reaction (Figure 3.10) ATP negatively regulates the reaction.

Full details of this visual specification can be found at the SBGN web site [www.sbgn.org](http://www.sbgn.org); Figure 3.11 summarizes the main symbols.

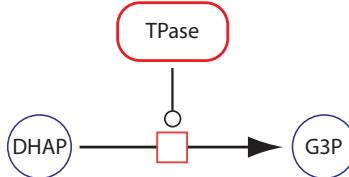
## 3.3 Mass-Balance Equations

Consider a simple network made up of two reactions,  $v_1$  and  $v_2$ , with a common species, S.  $v_1$  and  $v_2$  are the rates of reaction such that  $v_1$  is the rate at which S is produced and in the second reaction,  $v_2$  is the rate at which S is consumed (Figure 3.12).

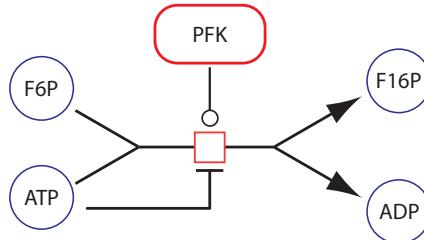
According to the law of conservation of mass, any observed change in the amount of species S must be due to the difference between the inward rate,  $v_1$ , and outward rate,  $v_2$ . That is, the change in S is the difference in the two rates, leading to the differential equation:

$$\frac{dS}{dt} = v_1 - v_2 \quad (3.1)$$

a) Simple uni-uni reaction



b) More complex reaction with regulation

**Figure 3.10** SBGN notation for enzyme catalyzed reactions.

This equation is called a **mass-balance equation**. We can reexpress equation (3.1) as:

$$\frac{dS_a}{dt} \frac{1}{V} = v_1 - v_2$$

where  $S_a$  is the amount in moles and  $V$  is the volume. Alternatively, we note that:

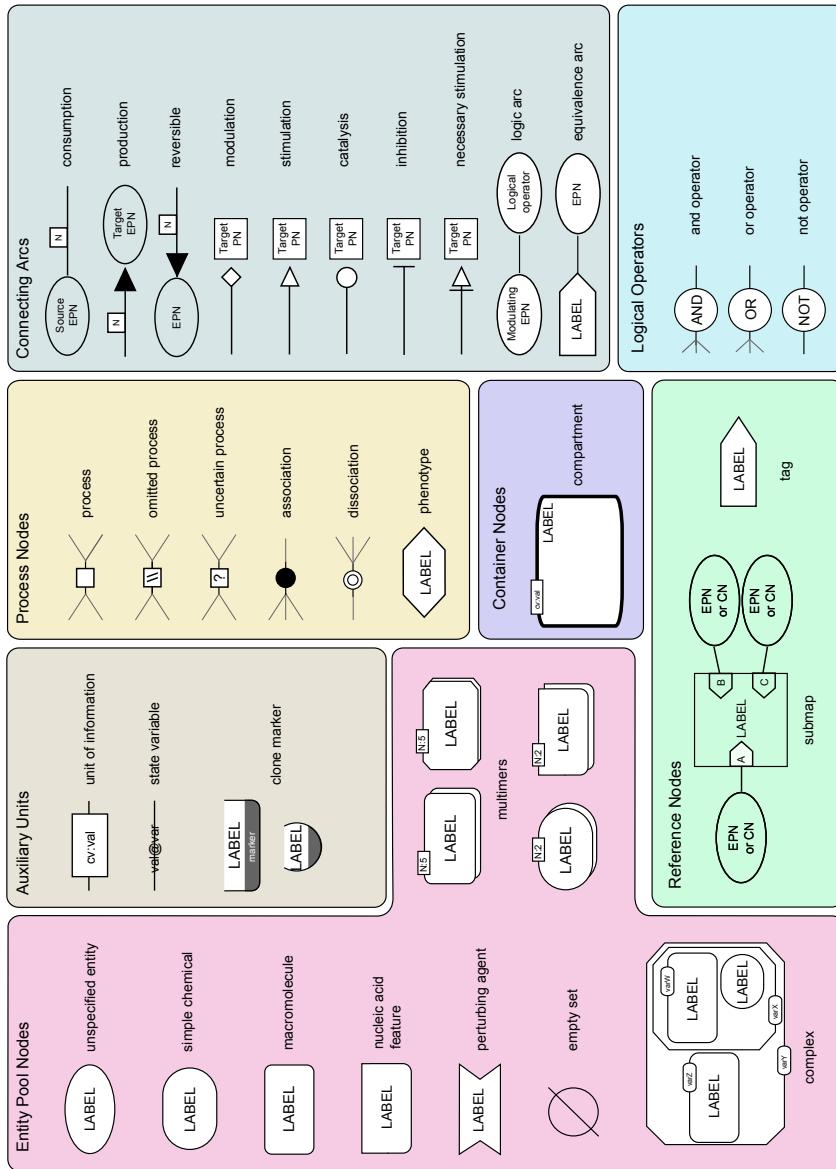
$$\frac{dS_a}{dt} = V(v_1 - v_2)$$

This assumes the reaction rates are expressed in  $\text{mol l}^{-1} \text{ t}^{-1}$ . Biochemical models will sometimes assume a constant unit volume so that numerically:

$$\frac{dS}{dt} = \frac{dS_a}{dt}$$

Although we will express the rate of change in terms of concentration, it is implied that we are dealing with a constant unit volume so that the change in concentration is the same as the change in amount. It is important to note that it is amounts that are mass conserved, not concentration. For example, if movement is from one compartment to another compartment with a different volume, it is necessary to factor in the volume difference and explicitly express the rate of change in amounts (We will consider this in more detail in Chapter 8).

Unless otherwise stated, the following assumptions should be made about models in this book:



**Figure 3.11** SBGN notation Reference Card, reused but modified with color shading from [www.sbgn.org](http://www.sbgn.org).



**Figure 3.12** Simple two step pathway.

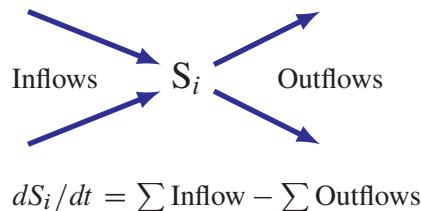
1. *Well-Stirred Reactor.* Many biochemical models assume that the volume in which reactions take place is well-stirred. This means there are no spatial inhomogeneities. For small cells such as *E. coli*, this is a reasonable assumption. The diffusion rate of molecules in the cytoplasm is so fast that a given molecule will, on average, sample every location in the *E. coli* cell in one second. In larger eukaryotic cells spatial homogeneity may occur as there are known mechanisms to restrict diffusion of important molecules from a given location. As such, the assumption of a well-stirred reactor may still apply. Ultimately, the validity of the assumption rests with whether the model generates useful and verifiable predictions.

2. *Large number of molecules.* In the last chapter we reviewed the range of molecule and ion numbers found in biological cells. The numbers varied from a few copies for the LacI repressor protein to many millions in the case of ions. When there are large numbers of molecules or ions, concentrations can be approximated using a value that continuously changes. In such cases we can use differential equations to model the rates of change. When dealing with small numbers of molecules however, concentration as a continuous variable may no longer make sense. For example, given that 1 nM roughly equates to one molecule per *E. coli* cell, it doesn't make much sense to quote a figure of 1.5 nM since that implies 1.5 molecules per cell. When dealing with small numbers of molecules, it is not possible to have a continuous range of concentrations. Under these circumstances a discrete probabilistic approach is best. We will come back to this important topic later.

3. *Unit Volume.* Unless otherwise indicated, we will assume *fixed unit volumes*.

### Model Complex Networks

For more complex systems such as the one shown in Figure 3.13 where there are multiple inflows and outflows, the mass-balance equation is given by:



**Figure 3.13** Mass Balance: The rate of change in species  $S_i$  is equal to the difference between the sum of the inflows and the sum of the outflows.

$$\frac{dS_i}{dt} = \sum \text{Inflows} - \sum \text{Outflows} \quad (3.2)$$

For an even more general representation, we can reexpress the mass-balance equations by taking into account the stoichiometric coefficients. The rate at which a given reaction,  $v_j$ , contributes to change in a species  $S_i$ , is given by the stoichiometric coefficient of the species,  $S_i$  with respect to the reaction,  $c_{ij}$  multiplied by the reaction rate,  $v_j$  (See equation (2.1)). That is, a reaction  $j$  contributes  $c_{ij} v_j$  rate of change in species  $S_i$ . For example, with the reaction  $A \rightarrow B$  which has a reaction rate  $v$ , and  $c_A$  is -1, we can say that the reaction contributes  $-1v$  to the rate of change in A. For a species  $S_i$  with multiple reactions producing and consuming  $S_i$ , the mass-balance equation (assuming constant unit volume) is given by:

$$\frac{dS_i}{dt} = \sum_j c_{ij} v_j \quad (3.3)$$

where  $c_{ij}$  is the stoichiometric coefficient for species  $i$  with respect to reaction,  $j$ . For reactions that consume a species, the stoichiometric coefficient is often *negative*; otherwise the stoichiometric coefficient is *positive* (See Chapter 2). In considering the simple example in Figure 3.12, the stoichiometric coefficient for S with respect to  $v_1$  is +1 and for  $v_2$  is -1. That is:

$$\frac{dS}{dt} = c_{s1} v_1 + c_{s2} v_2$$

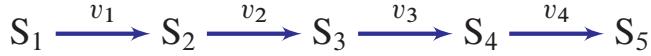
or

$$\frac{dS}{dt} = (+1)v_1 + (-1)v_2 = v_1 - v_2$$

How we describe the construction of the mass-balance equation may seem overly formal, however the formality allows us to write software that can automatically convert network diagrams into mass-balance differential equations.

**Example 3.3**

Consider a linear chain of reactants from  $S_1$  to  $S_5$  shown in Figure 3.14. Write out the mass-balance equations for this simple system.



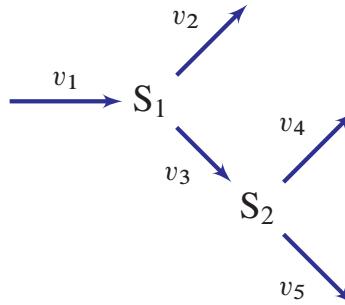
**Figure 3.14** Simple straight chain pathway.

$$\begin{aligned}\frac{dS_1}{dt} &= -v_1 & \frac{dS_2}{dt} &= v_1 - v_2 \\ \frac{dS_3}{dt} &= v_2 - v_3 & \frac{dS_4}{dt} &= v_3 - v_4 \\ \frac{dS_5}{dt} &= v_4\end{aligned}\tag{3.4}$$

Each species in the network is assigned a mass-balance equation which accounts for the flows into and out of the species pool.

**Example 3.4**

Write out the mass-balance equation for the branched system shown in Figure 3.15:



**Figure 3.15** Multi-branched pathway.

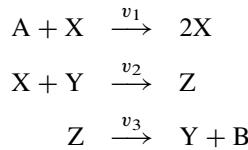
The mass-balance equations are given by:

$$\frac{dS_1}{dt} = v_1 - v_2 - v_3$$

$$\frac{dS_2}{dt} = v_3 - v_4 - v_5$$

**Example 3.5**

Write out the mass-balance equation for the more complex pathway:



This example is more subtle because we need to take into account the stoichiometry change between the reactant and product side in the first reaction ( $v_1$ ). In reaction  $v_1$ , the stoichiometric coefficient for  $X$  is  $+1$  because two  $X$  molecules are made for every one consumed. Taking this into account, the rate of change of species  $X$  can be written as:

$$\frac{dX}{dt} = -v_1 + 2v_1 - v_2$$

or more simply as  $v_1 - v_2$ . The full set of mass-balance equations can therefore be written as:

$$\begin{array}{ll} \frac{dA}{dt} = -v_1 & \frac{dX}{dt} = v_1 - v_2 \\ \frac{dY}{dt} = v_3 - v_2 & \frac{dZ}{dt} = v_2 - v_3 \\ \frac{dB}{dt} = v_3 & \end{array}$$

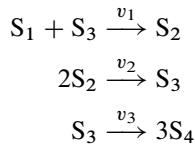
The last example (3.5) illustrates a very important aspect of converting a network diagram into a set of differential equations. The process is potentially **lossy**. That is, it is *not always possible* to fully recover the original network diagram from the set of derived differential equations. This is because in one or more of the reactions, the stoichiometries may cancel. In example (3.5) the reaction  $A + X \longrightarrow 2X$  is not recoverable from the final set of differential equations. Instead, if we reverse engineered the differential equations, the first reaction would be:



which is not like the original. This is not a common occurrence although in protein signaling pathways it might be more common than other kinds of networks. What it means however is that sharing models by exchanging differential equations is *not* recommended. This is one reason why standard exchange formats such as SBML [76] store models explicitly as a set of reactions, not as a set of differential equations. Many models are exchanged using Matlab which means that much of the biological information, particularly information on the underlying network, is lost. *Exchanging models* via computer languages such as Matlab is therefore not recommended.

**Example 3.6**

Write out the mass-balance equation for pathway:



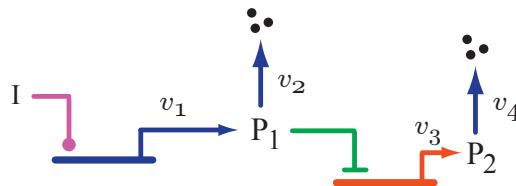
In this example we have non-unity stoichiometries in the second and third reaction steps. The mass-balance equations are given by:

$$\begin{aligned} \frac{dS_1}{dt} &= -v_1 & \frac{dS_2}{dt} &= v_1 - 2v_2 \\ \frac{dS_3}{dt} &= v_2 - v_3 - v_1 & \frac{dS_4}{dt} &= 3v_3 \end{aligned}$$


---

**Example 3.7**

Write out the mass-balance equations for  $P_1$  and  $P_2$  for the following gene regulatory network:



The key to this problem is that the network diagram suggests that the regulation from  $P_1$  to  $v_3$  results in no consumption of  $P_1$ .  $P_1$  acts only as a regulator. That being the case, the two mass-balance equations are:

$$\begin{aligned} \frac{dP_1}{dt} &= v_1 - v_2 \\ \frac{dP_2}{dt} &= v_3 - v_4 \end{aligned}$$


---

From the previous examples we see that it is fairly straightforward to derive the mass-balance equations from a visual inspection of the network. Many software tools exist to assist in this effort by converting network diagrams, either represented visually on a computer screen (for example, PathwayDesigner), or by processing a text file that lists the reactions in the network (for example via Tellurium) into a set of differential equations (See Appendix H).

### 3.4 Stoichiometry Matrix

When describing multiple reactions in a network, it is convenient to represent the stoichiometries in a compact form called the **stoichiometry matrix**. Traditionally the matrix is denoted by **N**, where the symbol **N** refers to ‘number’<sup>2</sup>. The stoichiometry matrix is a  $m$  row by  $n$  column matrix, where  $m$  is the number of species and  $n$  the number of reactions:

$$\mathbf{N} = m \times n \text{ matrix}$$

The columns of the stoichiometry matrix correspond to the individual chemical reactions in the network. The rows correspond to the molecular species, with one row per species. Thus, the intersection of a row and column in the matrix indicates whether a certain species takes part in a particular reaction or not. The sign of the element determines whether there is a net loss or gain of substance, and the magnitude describes the relative quantity of substance taking part in the reaction.

The elements of the stoichiometry matrix *do not* concern themselves with the rate of reaction. This latter point is particularly important because various stoichiometric analyses can be carried out purely on the stoichiometry without *any* reference to reaction rate laws.

The stoichiometric matrix is not concerned with describing reaction rates. Reaction rates are given by rate laws in a separate vector (See section 3.9).

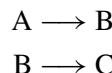
In general, the stoichiometry matrix has the form:

$$\mathbf{N} = \begin{matrix} & \xleftarrow{\quad} & v_j & \xrightarrow{\quad} \\ S_i & \uparrow & \begin{bmatrix} c_{ij} & \dots & \dots \\ \vdots & & \\ \vdots & & \end{bmatrix} \end{matrix}$$

where  $c_{ij}$  is the stoichiometry coefficient for the  $i^{\text{th}}$  species and  $j^{\text{th}}$  reaction. As mentioned before, the stoichiometry matrix is generally a lossy representation. That is, it is not always possible to revert back to the original biochemical network from which the matrix was derived. For example, consider the simple stoichiometry matrix:

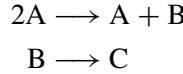
$$\mathbf{N} = \begin{bmatrix} -1 & 0 \\ 1 & -1 \\ 0 & 1 \end{bmatrix}$$

The most obvious network that this matrix could have been derived from is:



<sup>2</sup>Some recent flux balance literature uses the symbol **S**; the traditional symbol **N** will be used here.

But an equally plausible network is:



It is not possible from the stoichiometry matrix alone to determine the original network.

---

*Example 3.8*

Write out the stoichiometry matrix for the simple chain of reactions which has five molecular species and four reactions as shown below. The four reactions are labeled,  $v_1$  to  $v_4$ .



The stoichiometry matrix for this simple system is given by:

$$N = \begin{bmatrix} v_1 & v_2 & v_3 & v_4 \\ -1 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{array}{l} S_1 \\ S_2 \\ S_3 \\ S_4 \\ S_5 \end{array}$$

The rows and columns of the matrix have been labeled for convenience. Normally labels are absent.

---

*Example 3.9*

Write out the stoichiometry matrix for the multibranched pathway shown in Figure 3.15.

$$N = \begin{bmatrix} v_1 & v_2 & v_3 & v_4 & v_5 \\ 1 & -1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 & -1 \end{bmatrix} \begin{array}{l} S_1 \\ S_2 \end{array}$$


---

## 3.5 Reversibility

---

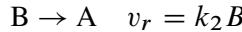
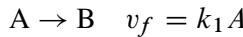
Up to this point we have not discussed whether a given reaction is reversible or not. When dealing with kinetic models, reversibility often manifests itself as a **negative reaction rate** in the rate law. For example, the rate law for the simple mass-action reversible reaction  $A \rightleftharpoons B$  is given by:

$$v = k_1 A - k_2 B$$

When this reaction goes in the reverse (right to left) direction, the reaction rate,  $v$ , will be negative. This may not be apparent from the stoichiometry matrix, which in this case is:

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

Information on reversibility is therefore traditionally found in the rate law. In this example the rate law could equally have been  $k_1 A$ , suggesting an irreversible reaction. Depending on the modeling problem, reversibility can be made more explicit in the stoichiometry matrix by specifying a *separate* reaction path for the reverse reaction. For example, in the previous example we might instead represent the system by two separate rate laws:



The stoichiometry matrix now becomes:

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

Splitting a reaction into separate forward and reverse steps might not always be possible however. For example, an enzyme catalyzed reversible reaction such as  $A \rightleftharpoons B$  *cannot* be represented using:

$$\frac{dB}{dt} = v_f - v_r$$

where  $v_f$  is the forward rate and  $v_r$  the reverse rate. At first glance we might choose to model the forward and reverse rates using irreversible Michaelis-Menten rate laws (D.2). However, the forward and reverse reactions are not independent. They are connected by the shared free enzyme pool so that when the forward rate rises, the reverse rate falls due to competition for free enzyme. If the modeler insists on separating the forward from the reverse rate, then the full enzyme mechanism in terms of elementary steps must be used (See the companion text book Enzyme Kinetics for Systems Biology for more details). Alternatively, and more commonly, a reversible enzyme catalyzed reaction is expressed using the reversible Michaelis-Menten equation ( D.4):

$$v = \frac{V_f / K_S (S - P / K_{eq})}{1 + S / K_S + P / K_P}$$

where S and P are the substrate and product concentrations respectively.  $V_f$  is the maximal forward rate. Some modelers will choose to express all reactions using elementary reactions but this poses its own problems, particularly when trying to set values for the many elementary rate constants that result. Ultimately, the decision has to be made on a case by case basis and will depend on the model's purpose.

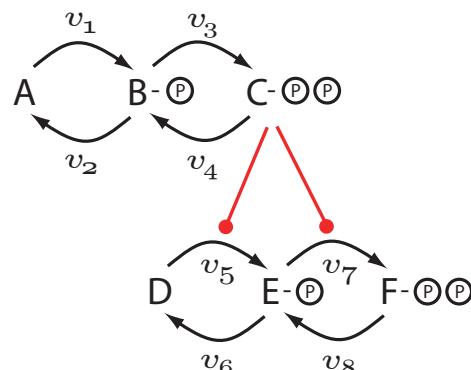
Before leaving the topic of reversibility, it is worth mentioning product inhibition. This occurs when the product binds to an enzyme without resulting in any reverse reaction rate.

However, binding of product competes with substrate which in turn represses the forward rate. Reactions that are often considered irreversible can still be affected by product. More details are provided in section D.5.

To illustrate how we apply the stoichiometry matrix to different kinds of networks, let's look at a simple signaling network and two simple gene regulatory networks.

## 3.6 Signaling Networks

Figure 3.16 illustrates a simple protein signaling network comprised of two double phosphorylation cycles coupled through activation by protein C on the lower double cycle (D, E and F). In this model all species are proteins and we assume that protein A and D are unphosphorylated, B and E singly phosphorylated, and C and F doubly phosphorylated. C acts as a kinase and phosphorylates D and E. The reverse reactions,  $v_2$ ,  $v_4$ ,  $v_7$  and  $v_8$  are assumed to be catalyzed by phosphatases.



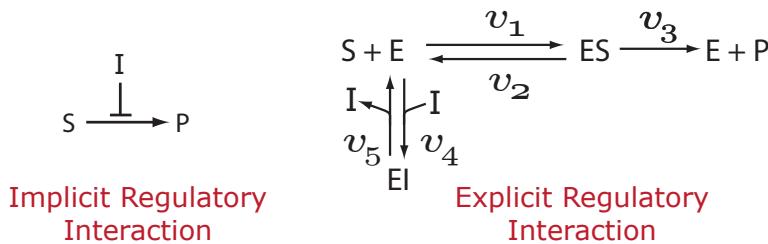
**Figure 3.16** Simple signaling network. Protein C activates the activity of reactions  $v_5$  and  $v_6$ .

There is no specified stoichiometric mechanism for the activation on  $v_5$  and  $v_6$ . Therefore, the stoichiometric matrix will contain no information about this. The stoichiometric matrix for this system is:

$$\mathbf{N} = \begin{matrix} & v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 \\ \begin{matrix} \mathbf{A} \\ \mathbf{B} \\ \mathbf{C} \\ \mathbf{D} \\ \mathbf{E} \\ \mathbf{F} \end{matrix} & \left[ \begin{matrix} -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{matrix} \right] \end{matrix} \quad (3.5)$$

The matrix is composed of two separate blocks corresponding to the two cycle layers. It is important to emphasize again that whenever there are *regulatory* interactions in a pathway diagram, these *do not* appear in the stoichiometry matrix. Instead, such information will reside in the rate laws that describe the regulation. If however the mechanism for the regulation is made explicit, then details of the regulation will appear in the stoichiometry matrix. Figure 3.17 shows a simple example of an inhibitor, I, regulating a reaction, S to P. The left displays an implicit regulatory interaction. All we see is a blunt ended arrow indicating inhibition. In this case details of the regulation will be found in the rate law governing the conversion of S to P. On the right is an explicit mechanism, a simple competitive inhibition. In this case details of the inhibition mechanism will find its way into the stoichiometry matrix, although from an inspection of the matrix, the type of regulation may not be obvious.

Figure 3.18 shows a comparison of the implicit and explicit models in terms of the stoichiometry matrix. In each case the rate laws also change. In the implicit form, the rate law will be a Michaelis-Menten competitive inhibition model whereas in the explicit model, the rates laws (now multiplied in number) will be simple mass-action rate laws. The choice of what to use, an implicit or explicit model, will depend entirely on the type of question that the model is attempting to answer. *There is no right or wrong way to do this*, the details of a model will depend on the type of question being asked.



**Figure 3.17** Example of implicit and explicit depiction of a regulatory interaction. The left-hand mechanism involving inhibitor, I, will not appear in the stoichiometry matrix whereas in the explicit mechanism, it will.

### 3.7 Gene Regulatory Networks

Consider a transcription factor  $P_1$  that represses a gene with expression rate  $v_3$  shown in Figure 3.19, left panel. In this model we have production of  $P_1$  from reaction  $v_1$ , and degradation of  $P_1$  via  $v_2$ . The construction of the stoichiometry matrix will depend on how we represent the regulated step,  $v_3$ . If regulation is implied, meaning there is no explicit kinetic mechanism, then the regulation will not appear in the stoichiometry matrix. For the

$$\mathbf{N} = \begin{matrix} \text{S} \\ \text{P} \\ \text{I} \end{matrix} \begin{bmatrix} v_1 \\ -1 \\ 1 \\ 0 \end{bmatrix} \quad \mathbf{N} = \begin{matrix} \text{S} \\ \text{P} \\ \text{I} \\ \text{E} \\ \text{ES} \\ \text{EI} \end{matrix} \begin{bmatrix} v_1 & v_2 & v_3 & v_4 & v_5 \\ -1 & 1 & 0 & -1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 \\ -1 & 1 & 1 & -1 & 1 \\ 1 & -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 \end{bmatrix}$$

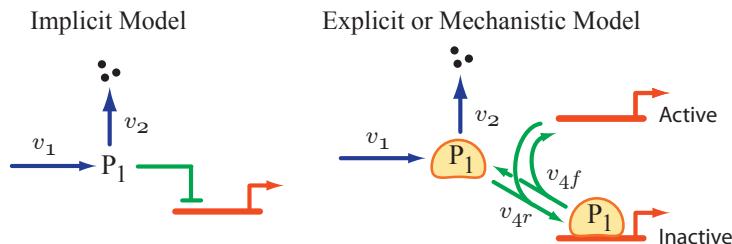
**Figure 3.18** Stoichiometry matrices corresponding to the two models in Figure 3.17.

network on the left in Figure 3.19, the stoichiometry matrix is:

$$\mathbf{N} = \mathbf{P}_1 \begin{bmatrix} v_1 & v_2 \\ 1 & -1 \end{bmatrix} \quad (3.6)$$

The stoichiometry matrix has only one row indicating that there is only one species in the model,  $P_1$ , and there is no hint in the stoichiometry matrix of any regulation. In this model  $P_1$  is not explicitly sequestered by the operator site upstream of the gene. We make the significant assumption that when  $P_1$  regulates, its own state is not affected in any way.

Consider now that the interaction between  $P_1$  and  $v_3$  is made mechanistically explicit. The right-hand network in Figure 3.19 shows one possible way in which to represent the interaction of the transcription factor,  $P_1$  with gene  $v_3$ . In the explicit model the transcription factor  $P_1$  is assumed to bind to a repressor site preventing gene expression. In the



**Figure 3.19** Two simple gene regulatory networks involving gene repression. On the left side is the implicit model where  $P_1$  represses  $v_3$ , on the right side is the explicit model showing a more detailed mechanism for the regulation.

explicit model there are two new species, designated active gene and inactive gene. The stoichiometry matrix will therefore include two additional rows corresponding to these two

new species. The stoichiometry matrix for the explicit model is shown here:

$$\mathbf{N} = \begin{matrix} & \begin{matrix} v_1 & v_2 & v_{4r} & v_{4f} \end{matrix} \\ \begin{matrix} P_1 \\ P_1(\text{Active}) \\ P_1(\text{InActive}) \end{matrix} & \left[ \begin{matrix} 1 & -1 & -1 & 1 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 1 & -1 \end{matrix} \right] \end{matrix} \quad (3.7)$$

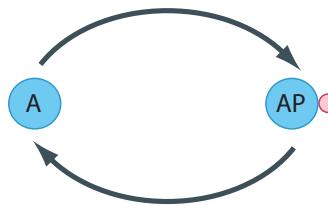
In this case  $P_1$  is actively sequestered onto the operator site and therefore appears in the stoichiometry matrix. Processes such as consumption, production, or sequestration by some binding mechanism will appear as columns in the stoichiometry matrix.

In conclusion, regulation does not appear explicitly in a stoichiometry matrix unless the regulation is represented as an explicit mechanistic scheme. The choice of implicit or explicit representations depends on the question being asked and the availability of suitable data.

### 3.8 Moiety Conserved Cycles

Many cell processes operate on different time scales. For example, metabolic processes tend to operate on a faster scale than protein synthesis and degradation. Such time scale differences have a number of implications to model builders, software designers, and model behavior. In this chapter we will briefly examine some of these aspects in relation to species conservation laws. We will return again to the topic in Chapter 5.

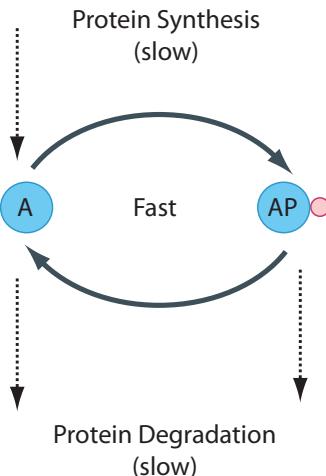
To introduce this topic, consider a simple protein phosphorylation cycle such as the one shown in Figure 3.20. This shows a protein undergoing phosphorylation (upper limb) and dephosphorylation (lower limb) via a kinase and phosphatase, respectively.



**Figure 3.20** Phosphorylation and dephosphorylation cycle forming a moiety conservation cycle between unphosphorylated (left species, A) and phosphorylated protein (right species, AP).

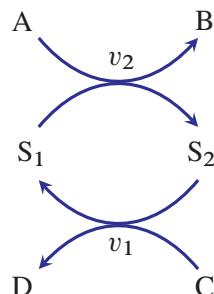
The depiction in Figure 3.20 is a simplification. The ATP used during phosphorylation and the release of free phosphate during the dephosphorylation event are not shown. In addition, synthesis and degradation of protein is also absent. In many cases we can leave these aspects out of the picture. ATP for instance is held at a relatively constant level by strong homeostatic forces from metabolism so that within the context of the cycle, changes

in ATP isn't something we must worry about. More interesting is that within the time scale of phosphorylation and dephosphorylation, we can assume that the rate of protein synthesis and degradation is negligible (Figure 3.21). This assumption is more significant and leads to the emergence of a new property called **moiety conservation** [144].



**Figure 3.21** Phosphorylation and dephosphorylation cycle that also includes the slower process of protein synthesis and degradation. We assume that the phosphorylated and unphosphorylated protein can be degraded but only the unphosphorylated protein is synthesized.

In chemistry a **moiety** is described as a subgroup of a larger molecule. In this case the moiety is a protein. During the interconversion between the phosphorylated and unphosphorylated states, the amount of moiety (protein) remains constant. More abstractly we can draw a cycle in the following way (Figure 3.22), where  $S_1$  and  $S_2$  are the cycle species:



**Figure 3.22** Simple conserved cycle where  $S_1 + S_2 = \text{constant}$ .

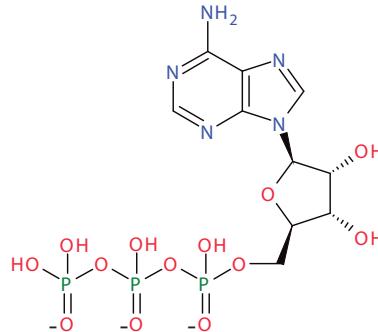
The two species  $S_1$  and  $S_2$  are conserved because the total  $S_1 + S_2$  remains constant over time (at least over a time scale shorter than other processes that may be involved). Such

cycles are collectively called **moiety conserved cycles**.

**Moiety:** A subgroup of a larger molecule.

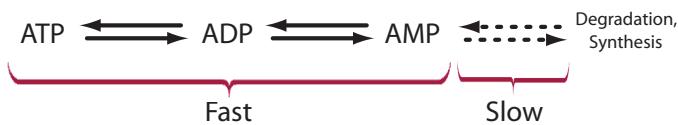
**Conserved Moiety:** A subgroup whose interconversion through a sequence of reactions leaves it unchanged.

Protein signalling pathways abound with conserved cycles such as these although many are more complex and may involve multiple phosphorylation reactions. In addition to protein networks, other pathways also possess conservation cycles. One of the earliest conservation cycles to be recognized was the adenosine triphosphate (ATP) cycle. ATP is a chain of three phosphate residues linked to a nucleoside adenosine group as shown in Figure 3.23.



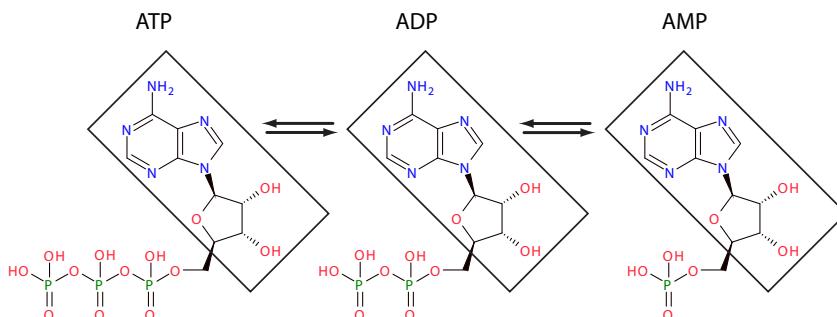
**Figure 3.23** Adenosine Triphosphate: Three phosphate groups plus an adenosine subgroup.

The linkage between the phosphate groups involves an unstable phosphoric acid anhydride bond. These bonds can be cleaved by hydrolysis one at a time leading to the formation of adenosine diphosphate (ADP) and adenosine monophosphate (AMP), respectively. The hydrolysis provides much of the free energy to drive endergonic processes in the cell. Given the insatiable need for energy, there is a continual and rapid interconversion between ATP, ADP and AMP as energy is released or captured. One constant during these interconversions is the amount of adenosine group (Figure 3.25). Adenosine is a conserved moiety. Over longer time scales there is also the slower process of AMP degradation and biosynthesis via the purine nucleotide pathway; but for many models, we assume that this process is very slow compared to ATP turnover by energy metabolism.



**Figure 3.24** The interconversion of ATP, ADP and AMP is generally considered fast in comparison to the slow process of synthesis and degradation of AMP.

There are many other examples of conserved moieties such enzyme/enzyme-substrate complexes, NAD/NADH, phosphate and coenzyme A. In all these cases the basic assumption is that the interconversions of the subgroups is rapid compared to their net synthesis and degradation. We should emphasize that in reality, conserved moieties do not exist since all molecular subgroups will at some point be subject to synthesis and degradation. However, over sufficiently short time scales, the sum total of these groups can be considered constant.



**Figure 3.25** The adenosine moiety, indicated by the boxed molecular group, is conserved during the interconversion of ATP, ADP and AMP.

### 3.9 The System Equation

Equation (3.3), which describes the mass-balance equation, can be reexpressed in terms of the stoichiometry matrix to form the **system equation**:

$$\frac{d\mathbf{s}}{dt} = \mathbf{N}v \quad (3.8)$$

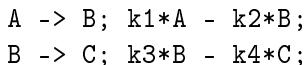
where  $\mathbf{N}$  is the  $m \times n$  stoichiometry matrix and  $\mathbf{v}$  is the  $n$  dimensional rate vector, whose  $i$ th component gives the rate of reaction  $i$  as a function of the species concentrations.  $\mathbf{s}$  is the  $m$  vector of species. This is a key equation for describing a network of processes inside

a cell. Of particular significance is that the equation explicitly separates the network, in the form of  $\mathbf{N}$ , from the process rates,  $\mathbf{v}$ .

Looking again at the simple chain of reactions in Figure 3.14, the system equation can be written as:

$$\frac{ds}{dt} = \mathbf{N}\mathbf{v} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix} \quad (3.9)$$

If the stoichiometry matrix is multiplied into the rate vector, the mass-balance equations shown earlier (3.4) are recovered. To illustrate what the system equation might look like for a simple system, consider the following model expressed in Antimony format:



The system equation for this model is:

$$\frac{ds}{dt} = \mathbf{N}\mathbf{v} = \begin{bmatrix} -1 & 0 \\ 1 & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} k_1A - k_2B \\ k_3B - k_4C \end{bmatrix} \quad (3.10)$$

## 3.10 Tellurium

The modeling platform Tellurium [150] provides facilities to extract the stoichiometry matrix from a model. The command for generating the stoichiometry matrix is `getFullStoichiometryMatrix()`. The short-hand version for this command is `sm`. The script and results of a run are given below:

```
import tellurium as te

r = te.loada '''
J1: A -> B; k1*A - k2*B;
J2: B -> C; k3*B - k4*C;

k1 = 0.1; k2 = 0.02;
k3 = 0.3; k4 = 0.04;
A = 10; B = 0; C = 0;
'''

print r.getFullStoichiometryMatrix()
print r.getReactionIds()
print r.getFloatSpeciesIds()
```

If this script is run, the output is:

```
[[ 1. -1.]
 [-1.  0.]
 [ 0.  1.]]
['J1', 'J2']
['B', 'A', 'C']
```

The row and column order in the stoichiometry matrix can be obtained from calls to  
`rr.getReactionIds()`  
for the column order and  
`rr.getFloatingSpeciesIds()`  
for the row order. Using the supported shortcuts (See appendix), the script becomes:

```
import tellurium as te

r = te.loada '''
J1: A -> B; k1*A - k2*B;
J2: B -> C; k3*B - k4*C;

k1 = 0.1; k2 = 0.02;
k3 = 0.3; k4 = 0.04;
A = 10; B = 0; C = 0;
''')

print r.sm()
print r.rs()
print r.fs()
```

## Further Reading

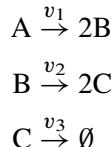
1. Palsson BO (2006) Systems Biology Systems Biology: Properties of Reconstructed Networks. Cambridge University Press, ISBN: 978-0521859035
2. Sauro HM (2012) Enzyme Kinetics for Systems Biology. 2nd Edition, Ambrosius Publishing ISBN: 978-0982477335
3. Stephanopoulos G, Aristidou A, and Nielsen J (1998) Metabolic engineering: principles and methodologies. Academic Press, ISBN: 978-0126662603

## Exercises

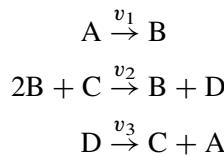
---

1. Explain the difference between the terms: Stoichiometric amount, stoichiometric coefficient, rate of change ( $dX/dt$ ), and reaction rate ( $v_i$ ). Refer to Chapter 2 to answer this question.
2. Determine the stoichiometric amount and stoichiometric coefficient for each species in the following reactions (Refer to Chapter 2 to answer this question):
 
$$\begin{aligned} A &\longrightarrow B \\ A + B &\longrightarrow C \\ A &\longrightarrow B + C \\ 2A &\longrightarrow B \\ 3A + 4B &\longrightarrow 2C + D \\ A + B &\longrightarrow A + C \\ A + 2B &\longrightarrow 3B + C \end{aligned}$$

3. Derive a set of differential equations for the following model in terms of the rate of reaction,  $v_1$ ,  $v_2$ , and  $v_3$ :



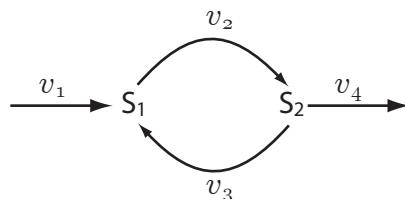
4. Derive the set of differential equations for the following model in terms of the rate of reaction,  $v_1$ ,  $v_2$  and  $v_3$ :



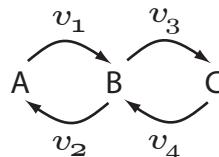
5. Write out the stoichiometry matrix for the networks in question 3 and 4.
6. Enter the previous models, 3 and 4, into Tellurium and confirm that the stoichiometry matrices are the same as those derived manually in the previous question.
7. Derive the stoichiometry matrix for each of the following networks. In addition, write out the mass-balance equations in each case.
  - (a)



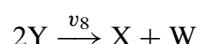
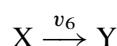
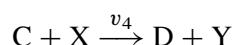
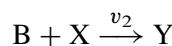
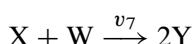
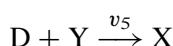
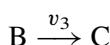
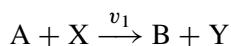
(b)



(c)



(d)



8. For the irreversible enzyme catalyzed reaction,  $A \rightarrow B$ :

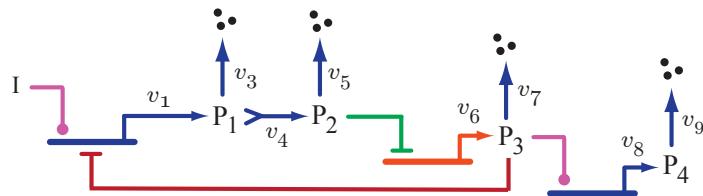
(a) Write out the stoichiometry matrix.

(b) Write out the stoichiometry matrix in terms of the elementary reactions that make up the enzyme mechanism.

9. A gene  $G_1$  expresses a protein  $p_1$  at a rate  $v_1$ .  $p_1$  forms a tetramer (4 subunits), called  $p_1^4$  at a rate  $v_2$ . The tetramer negatively regulates a gene  $G_2$ .  $p_1$  degrades at a rate  $v_3$ .  $G_2$  expresses a protein,  $p_2$  at a rate  $v_9$ .  $p_2$  is cleaved by an enzyme at a rate  $v_4$  to form two protein domains,  $p_2^1$  and  $p_2^2$ .  $p_2^1$  degrades at a rate  $v_5$ . Gene  $G_3$  expresses a protein,  $p_3$  at a rate  $v_6$ .  $p_3$  binds to  $p_2^2$  forming an active complex,  $p_4$  at a rate  $v_{10}$ , which can bind to gene  $G_1$  and activate  $G_1$ .  $p_4$  degrades at a rate  $v_7$ . Finally,  $p_2^1$  can form a dead-end complex,  $p_5$ , with  $p_4$  at a rate  $v_8$ .

- (a) Draw the network represented in the description given above.
- (b) Write out the differential equation for each protein species in the network in terms of  $v_1, v_2, \dots$
- (c) Write out the stoichiometric matrix for the network.
- 10.** Write out the differential equations for the system depicted in equation (3.9).
- 11.** Given the following stoichiometry matrix, write out the corresponding network diagram. Why might this process not fully recover the original network from which the stoichiometry matrix was derived?
- $$\begin{array}{c} v_1 & v_2 & v_3 & v_4 & v_5 \\ \hline A & -1 & 0 & -1 & 0 & 0 \\ B & 1 & -1 & 0 & 0 & 3 \\ C & 0 & 2 & -1 & 0 & 0 \\ D & 0 & 0 & 1 & -1 & 0 \\ E & 0 & 0 & 0 & 1 & -1 \\ F & 0 & 0 & 0 & 0 & 1 \\ G & 0 & 0 & 0 & -1 & 0 \end{array} \quad (3.11)$$

- 12.** Derive the mass-balance equations for the following gene regulatory network:



- 13.** Why is it better to store a model as a list of reactions rather than a set of differential equations?

# 4

## *Introduction to Modeling*

### 4.1 Introduction

---

The universe is a very large place and to study it in its entirety would not be practical. Instead, we always study a small portion of the universe, often under very controlled conditions which we call a **system**. Everything else other than the system is called the **surroundings**. Between the system and the surroundings we, as scientists, try to enforce strict rules on how the system interacts with the surroundings. These interactions occur at the system **boundary**.

The **system** is a defined region of the universe that we wish to study.

The **surroundings** is everything else other than the system.

The **boundary** is the interface between the system and the surroundings.

The word **system** derives from a Greek term that means “place together”, suggesting a system is one or more parts working together.

In order to make the study of a particular system possible, we will often impose strict conditions on how the system interacts with the rest of the universe. If the system were allowed to freely interact with its surroundings, then we’re effectively back to studying the entire universe again. When we study a system, we make sure that we know exactly how the system interacts with its surroundings and in ways that we can control.

The actual boundary of the system is however entirely at the discretion of the experimenter and depends on practical as well and scientific considerations. The important point is that the boundary is under *our strict control*, at least in principle. The nature of this control also determines whether our system is open, closed or isolated.

In general, the experimenter decides the location of the boundary that exists between the system and the surroundings. Once set, the experimentalist will usually impose constraints on how the surroundings and system are allowed to interact with each other.

## 4.2 Open, Closed, and Isolated Systems

---

When considering systems it is helpful to distinguish between three types of boundary conditions that exist between the system and the surroundings. These types are called *isolated*, *closed* and *open* systems. Each of these systems represent an idealized state. In practice we try to approximate them as well as possible. An isolated system, as the name suggests, is completely cut off from the rest of the universe, that is neither energy nor matter can be transferred across the isolated system's boundary. A closed system is one that only transfers energy, for example heat, work or light. An open system is one that can exchange both energy and mass with the surroundings.

System	Property
Isolated	No transfer of energy or matter
Closed	No transfer of matter
Open	Transfer of matter and energy

The distinction between a closed and open system in biology is very important. Open systems are characteristic of biological systems. For example, glycolysis is a pathway for converting an external nutrient source such as glucose, into available energy, such as ATP or heat and waste products lactate or ethanol. That is, it exchanges mass and energy with the surroundings. Without mass and energy exchange, biological systems would eventually run to thermodynamic equilibrium and cease to function. All models of living biological systems are therefore open.

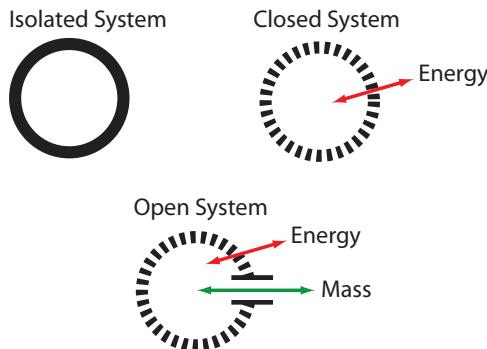
---

### Example 4.1

---

For each of the following systems, decide whether the system is isolated, closed or open. Comment on the nature of the surroundings.

- A system represented by a mechanical clock slowly winds down in a room controlled by a thermostat.



**Figure 4.1** Open, Closed, and Isolated Systems.

The clock starts with an amount of potential energy in the wound spring which slowly dissipates, ultimately as heat which is transferred to the surroundings. No mass is exchanged with the room. The clock is therefore a closed system. Because the clock is in a temperature controlled room, the temperature of the room appears constant to the clock even though the clock dissipates heat.

ii) A car engine running idle in the open air.

The car engine is burning fuel that generates both waste gases and heat. The heat and waste gases are lost to the surroundings. At the same time the car engine takes in oxygen. The car system is therefore open since it exchanges both matter and energy with the surroundings. In addition, given that the exchange takes place in the open, the surrounding temperature, oxygen and carbon dioxide levels appear constant because the large volume of the atmosphere acts as a buffer. We assume that the fuel tank is part of the system.

iii) A bacterial culture is grown in batch and kept in a sealed and insulated chamber.

The batch vessel is isolated and therefore the culture itself is an isolated system. There is no exchange of mass or energy with the surroundings. However, if we focus our attention on a single bacterium, we would have to conclude that a single cell is an open system which consumes nutrients, produces waste, and generates heat. However, the bacterial surroundings are not kept constant and the temperature, as well as waste products, rise with the loss of nutrients. Eventually the nutrients are used up, the culture dies, and the system tends to thermodynamics equilibrium.

## 4.3 What is a Model?

There are many ways to describe systems, ranging from pictures or cartoons to verbal and mathematical representations. Collectively, these descriptions are called **models**. A model is our way of describing a particular system. The Oxford English Dictionary defines a model in the following way:

“A simplified or idealized description or conception of a particular system,

situation, or process, often in mathematical terms, that is put forward as a basis for theoretical or empirical understanding, or for calculations, predictions, etc.”

This definition embodies a number of critical features that define a model, the most important being that a model represents an **idealized description**, a simplification, of a real world process. This may at first appear to be a weakness, but simplification is usually done intentionally. Simplification allows us to comprehend the essential features of a complex process without being burdened and overwhelmed by unnecessary detail.

A more interesting way to describe models is to use mathematics, a language created for logical reasoning. Mathematical models are useful in biology for a number of reasons, but the three most important are increased precision, prediction, and the capacity for analysis. Analysis is carried out either by simulation or by mathematical analysis. Although visual models can be used to make predictions, the kinds of predictions that can be made are limited. The use of mathematical models opens up whole new vistas of study which visual models simply can not match.

A **model** is a simplified description of a system. A model can be used to represent known facts about the system and hypotheses concerning the system’s operation. Models can be described using pictures, plain text, mathematics or computer software.

Models come in various forms including verbal, written text, visual, mathematical (4.1), and others. Molecular biology has a long tradition of using visual models to represent cellular structure and function; one need only look through a modern textbook to see instances of visual models on every page. Visual models have been immensely useful at describing complicated biological processes but are limited in scope.

Like visual models, mathematical models can serve at least two important roles in systems biology:

- Heuristic Models

Heuristic models serve as test beds for investigating basic principles, for example the effects of feedback or sequestration. Heuristic models are employed to aid reasoning about particular aspects of biological networks. Heuristic models are frequently used to illustrate properties of biological networks.

- Particular Models

Particular models are constructed to model a specific real system, such as glycolysis [173], apoptosis [158], or the sporulation circuit in *Bacillus subtilis* [80]. They represent a **working hypothesis** for a particular biological system, and allow us to generate predictions about the real system and falsifiable statements about the model.

## Particular Models

What makes a good particular model? There are a range of properties that a good model should exhibit, but the most important are **accuracy**, **predictability** and **falsifiability**.

- ▷ A model is considered **accurate** if the model is able to describe current experimental observations by reproducing the current state of knowledge.
- ▷ A **predictive** model should be able to generate insight and/or predictions beyond current knowledge. Without this ability, a model is considerably less useful, some would even suggest useless.
- ▷ Finally, a model should be **falsifiable**. By this we mean that a model cannot be proven true, only disproved. The only discipline where statements can actually be proven true or false is mathematics. Starting with a set of axioms, mathematicians derive theorems that can be shown beyond any doubt to be true or false. In contrast, scientific models based on observations *cannot be proven correct*. This is because it is simply not possible to test every possible circumstance in which the model may apply and be able to make the necessary measurements error free. Instead, we are left with two options: model falsification and model validation.

## Model Falsification

We can falsify a model by finding observations that the model fails to predict. In this case the model must be changed or abandoned. Although the idea of falsifying a model is appealing, in practice it is not often used since even partially correct models can at times be useful. This leads to the second option, model validation which is the most commonly used approach.

## Model Validation

Model validation is based on the idea that predictions made by a model are verified by experimentation. The word validate may imply that once a model is ‘validated’, the model can now be considered a true representation of the real system, but this is not accurate. A validated model is simply one where our *confidence* in a model’s ability to predict and provide insight has *improved*. As already suggested, no model is correct. The utility of a model is based on how well it can make useful predictions and how well it fits existing knowledge. Models will have a certain scope within which they are useful. For example, Newtonian mechanics is useful for describing objects traveling at speed much slower than the speed of light. Objects traveling close to the speed of light cannot be described by Newtonian mechanics. Michaelis-Menten kinetics is useful for describing steady state systems, but is less useful for describing transient behavior if enzyme concentration is higher or comparable to substrate concentration. One role of model validation is therefore to delineate its scope.

Thus, validation serves two purposes: to describe the scope of a model and to increase con-

fidence in the ability of the model to make useful predictions. It is important to understand that validation does not ‘prove’ that a model is correct since no such statement can be made.

Models cannot be proved to be true. We can only improve our confidence in a model’s utility through experiment.

## Other Attributes

There are other desirable model attributes including **parsimonious** and **selective**. A parsimonious model is a model that is as simple as possible, but no simpler. Occam’s infamous razor states that “Entities should not be multiplied beyond necessity” and argues that given competing and equally good models, the simplest is preferred. Finally, since no model can represent every single detail of a system, a model must be *selective* and represent those things most relevant to the task at hand.

## 4.4 Building a Model

---

In this section we will introduce modeling by building a simple water tank model. We will wait until Chapter 5 before discussing how to use software to run a model simulation.

### Water Tank Model

Figure 4.2 shows two water tanks. This is a pictorial model of our system. Our aim it to attempt to make quantitative predictions on the height of water in the tanks. We cannot easily do this with just the picture diagram. Instead we must convert the picture into a mathematical model.

Let us first verbally describe the model. The first tank is fed with water at a rate  $Q_1$  ( $\text{m}^3 \text{s}^{-1}$ ). This tank drains into a second tank at a rate  $Q_2$ , which in turn drains to waste at a rate  $Q_3$ . The second tank has an additional feed of water flowing in at a rate  $Q_4$ . The height of the water level in each tank is given by  $h_1$  and  $h_2$ , respectively and the volumes by  $V_1$  and  $V_2$ . Each tank has a cross sectional area,  $A$ . In building the mathematical model we will make some assumptions, most notably:

- Mass is conserved as water moves from one tank to another.
- External environment is constant, for example the temperature.
- We will assume that the rate of flow out of a tank is proportional to the height of water.<sup>1</sup>

---

<sup>1</sup>Strictly speaking the flow follows Torrielli’s Law,  $Q \propto \sqrt{h}$  but we’ll keep things simple as assume direct

With the assumptions in place, we can now start to construct a mathematical model of the tank system.

The rate of change in the volume of water in a given tank is the rate at which the water enters, minus the rate at which it leaves. For the first tank that rate of change of volume,  $V_1$ , is:

$$\frac{dV_1}{dt} = Q_1 - Q_2$$

This uses the assumption of conservation of mass. If we want the equation in terms of the rate of change of height, then we need to recall that  $V = Ah$ , that is:

$$\frac{dV}{dt} = A \frac{dh}{dt} \quad \text{or} \quad \frac{dh}{dt} = \frac{1}{A} \frac{dV}{dt}$$

so that:

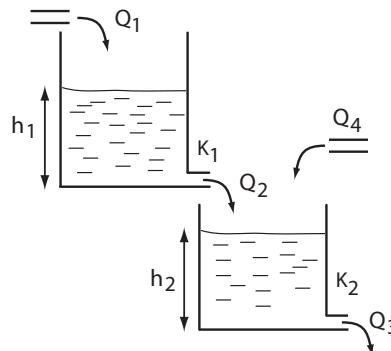
$$\frac{dh_1}{dt} = \frac{Q_1 - Q_2}{A}$$

Assuming that flow out of a tank is proportional to the height of water, we assuming in our model that the rate of water flowing out of a given tank,  $i$ , is equal to:

$$Q_i = K_i h_i$$

Where  $K_i$  is a constant related to the resistance of the output pipe. Therefore, for the first tank we have:

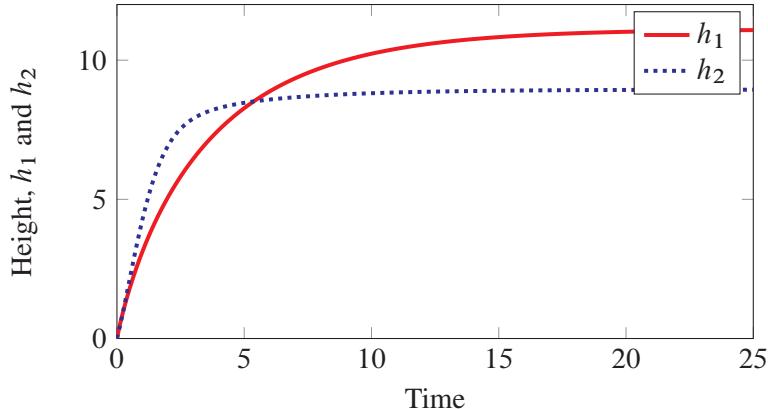
$$\frac{dh_1}{dt} = \frac{Q_1 - K_1 h_1}{A}$$



**Figure 4.2** Water Tank Model.

Describing the height of the second tank is slightly more complicated because we also have the additional flow  $Q_4$ . Again we will invoke the law of mass conservation to state that any

proportionality. At low flow rates  $Q$  is approximately proportional to the height of water. For more details see Torrielli's Law: [http://en.wikipedia.org/wiki/Torricelli%27s\\_law](http://en.wikipedia.org/wiki/Torricelli%27s_law)



**Figure 4.3** Simulation of the tank model.

change in volume must be due to the difference in water flow into the tank and out of the tank. That is, the rate of change of the second tank volume is given by:

$$\frac{dV_2}{dt} = Q_2 + Q_4 - Q_3$$

Using the relationship between volume, area and the height  $h_2$ , we can write:

$$\frac{dh_2}{dt} = \frac{Q_2 + Q_4 - Q_3}{A}$$

$Q_2$  was given previously as  $K_1 h_1$ , likewise,  $Q_3$  is given by  $K_2 h_2$ . Therefore the full differential equation is:

$$\frac{dh_2}{dt} = \frac{K_1 h_1 + Q_4 - K_2 h_2}{A} \quad (4.1)$$

With the differential equations in hand, the next stage is to assign values to the various parameters in the model. For example, the cross-sectional areas of the tanks, the flow  $Q_4$  into the second tank, the flow  $Q_1$  into the first tank, and the two tank constants,  $K_1$  and  $K_2$ . Once the parameters are assigned, we initialize the two heights  $h_1$  and  $h_2$ , then enter the equations into a computer program to solve the differential equations. We will leave the actual simulation task to Chapter 5. For now we just show the results from a simulation of the tank model using Tellurium (Figure 4.3). The plot displays the heights,  $h_1$  and  $h_2$  as water fills the tanks one at a time.

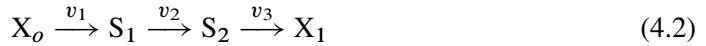
To summarize, we have learned a number of things from this exercise. First, models include assumptions and simplifications, and it is the careful selection of these that marks a good model from a bad one. The assumptions in this model include: 1) We assume low flow rates such that Torrielli's Law is a reasonable approximation; 2) The flows  $Q_1$  and  $Q_4$  are constant. The other thing we have learned is that there are at least four different types of quantities in the model. We will go into more detail in the next section but for now, we can briefly indicate what these quantities are:

1. Inputs to the system such as  $Q_1$ .
2. Model variables which include the two heights,  $h_1$  and  $h_2$ , which change in time as the system evolves.
3. A number of physical parameters which are fixed during the study of the model but which we could in principle change. In the tank model these include the volume, cross-sectional area of each tank, and the diameter of the outflow pipes.
4. Finally, there are parameters which we cannot change such as the force of gravity.

## 4.5 Variables, Parameters and Absolute Constants

---

Figure 4.4 classifies the different kinds of quantities we find in a model. These include absolute constants, parameters, inputs, dependent variables, independent variables and outputs. This is a long list so a concrete example will help better explain each quantity. Consider the following simple pathway model:



The rate laws are given by:

$$\begin{aligned} v_1 &= k_1 X_o \\ v_2 &= k_2 S_1 \left( 1 - \frac{S_2/S_1}{e^{-\Delta G^o/RT}} \right) \\ v_3 &= k_3 S_2 \end{aligned}$$

where  $\Delta G^o$  is the standard free energy,  $R$  the gas constant, and  $T$  the temperature. Note that  $e^{-\Delta G^o/RT}$  equals the equilibrium constant,  $K_{eq}$ , see equation (2.8). We will make a number of assumptions: i) The reactions take place in a constant unit volume at a constant temperature. ii) Species  $X_o$  and  $X_1$  are fixed by some external and unspecified process. iii) Reactions occur in well-stirred volumes. Let us list each type of quantity in this model:

**Absolute Constants** The absolute constants in a model include Napier's constant  $e$ , and the gas constant,  $R$ . Absolute constants cannot typically be changed by the experimenter.

**Parameters** The parameters of a model are those quantities which could, in principle, be changed by the experimenter but which *remain constant* when the model is used to make predictions. In the pathway model one can imagine that the  $\Delta G^o$  and the reaction rate constants are parameters. However, these particular parameters are not easily changed. It

might be possible to change them by altering the ionic composition, the solvent, or temperature. Usually however we treat kinetic and thermodynamic parameters as absolute constants. The exception to this is if the reactions are enzyme catalyzed. In this case one could change the enzyme concentration or through site-direct mutagenesis, change the enzyme kinetic properties.

**Inputs** The inputs to the system are those quantities which are under direct control of the experimenter and can conceivably be changed by the experimenter during the course of a model simulation. In the pathway model, the inputs include  $X_o$  and  $X_1$ . Other examples of inputs include nutrient sources, temperature, enzyme concentrations, and any kind of external effector such as a drug or inhibitor.

In biology the inputs are often clamped to some fixed values (*cf.* voltage clamp), but can also be varied in some controlled way by the experimenter. The clamping mechanism can simply be a large external reservoir so that any exchange of mass between the system and the external reservoir has a negligible effect on the external concentration. Alternatively, there may be active mechanisms maintaining an external concentration. A classic example of active maintenance of an external variable is the voltage clamp used in electrophysiology.

External concentrations may also change slowly in time compared to the timescale of the model so that over the study period, the external concentrations change very little. A typical example is the study of a metabolic response over a timescale that is shorter than change in gene expression. This permits a modeler to study a metabolic pathway without considering the effect of changes in gene expression.

The external species inputs such as  $X_o$  are also called **boundary variables** because they are considered to be at the boundary of the system.

Molecular species that are not dependent on the action of the model are sometimes called **boundary species**. Often boundary species are fixed by the modeler but it is possible for the modeler to impose a particular change in a boundary species to simulate, for example the administration of a drug as a bolus or as a continuous infusion.

**Dependent Variables** The dependent variables, also called the **state variables**, are the minimum set of variables to describe the state of a system. In biochemical modeling these variables often include the concentrations of molecular species or voltages across membranes. In the pathway model (Figure 4.2), the two dependent variables are  $S_1$  and  $S_2$ . The distinguishing feature that separates the input variables from the dependent variables is that while the inputs can be directly controlled by the model observer, the only way the dependent variables can change is through the operation of the model itself, i.e. they depend on the model. In biochemical modeling the dependent variables are also called **floating species**.

Internal Variable	External Variable
State variable	Inputs
Dependent variable	Independent variable
Floating variable (species)	Boundary variable (species)

**Table 4.1** Synonyms for internal and external variables.

Molecular species that change in time as a result of the action of the model are sometimes called **floating species**.

The distinction between the inputs and the dependent variables is important. Once the choice is made, the separation is strictly adhered to during the course of a study. This means for example that the environment surrounding the physical system will, *by definition*, be *unaffected* by the behavior of the system. If for some reason parts of the environment do change as a result of the system and can in turn affect the system in some way, then these parts must now be considered part of the system.

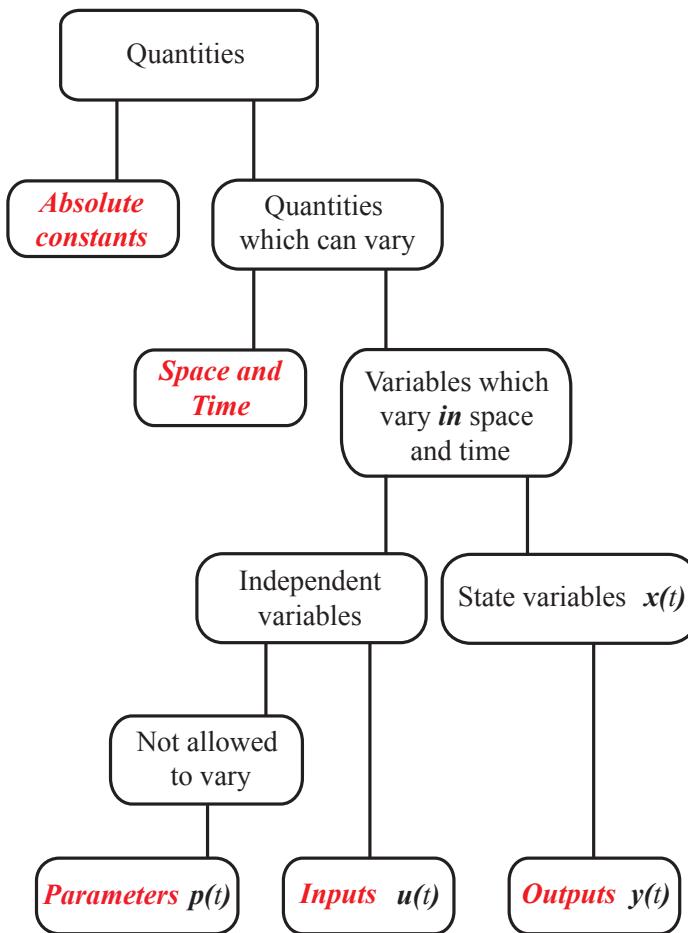
The state of a system at time  $t$  is described by a set of **state variables**:

$$\mathbf{x}(t)$$

They are the smallest set of variables that define the state of the system.

**Independent Variables** There are two main independent variables in biochemical modeling: time and space. In this book we will be mainly concerned with time dependent and not space dependent models.

**Outputs** The outputs are the readouts from the model, and are the quantities that an experimenter can actually measure. However the outputs are sometimes no different from the dependent variables, particularly in a computer model. Experimentally however, there are times when it is not possible to measure a particular dependent variable or when a derived measurement is required or measured. For example, we will often report the pH rather than the actual hydrogen ion concentration. In the case when we cannot make a direct measurement, we instead use a proxy, for example a fluorescence measurement or another molecular marker that follows the variable of interest. In the case of derived quantities, a very common one is the pathway flux. We will not cover this in great detail in this book, but separating the outputs from the independent variables is an important part of classical control and metabolic control theory.



**Figure 4.4** Classification of quantitative terms.

## 4.6 Mathematical Descriptions of Models

There are many different ways to represent models using mathematics. We must describe for example how the variables and parameters in the system will be represented. Two common representations include **discrete** or **continuous** variables. The change in the level of water in a tank is reasonably described using a continuous variable such as height. On the other hand, it might be more realistic to describe the dynamics of lion predation on the Serengeti using a discrete model where individual lions are represented. It does not make much sense to refer to 8.67 lions in a model. The choice of whether to use a discrete or continuous description depends entirely on the system being studied and the questions posed.

Another important categorization is whether the model should be represented in a **deter-**

**deterministic** or **stochastic** form. A deterministic model is one where if we repeated the simulation using the same starting conditions, we would get exactly the same result again. That is, the future state of the model is completely determined by its initial starting point. The model of the water tanks filling up is an example of a deterministic model.

A **discrete variable** is one that cannot take on all values within a given numeric range. For example, the number of airplanes in the sky at any one time is a discrete number. In statistics this is generalized further to a finite set of states, such as true/false or combinations in a die throw.

**Continuous variables** can assume all values within a given numeric range. For convenience we will often represent a measurement as a continuous variable. For example, we may use a continuous variable such as the mole to represent the concentration of a solute as it is unwieldy to refer to the concentration of a solute as 5,724,871,927,315,193,634,656 molecules per liter.

A stochastic model is not deterministic, that is running a simulation of a stochastic model with the same initial conditions will *not* lead to the same outcome. The reason for this is that processes in a stochastic model are probabilistic. For example, whether a chemical reaction will occur or not during a set time period is given by a probability. This is reasonable since at the molecular level, collisions between molecules are unpredictable.

Each step in a stochastic simulation is determined by one or more random processes. To give an example, modeling lion predation on the Serengeti could be modeled as a stochastic process. It is not guaranteed that a lion will catch its prey every time, instead there is a probability it will succeed. To model this process a computer simulation would throw a die to determine whether the lion had succeeded or not. Repeatedly running such a simulation would naturally give a slightly different outcome because the die throws would be different for each run. In systems biology stochastic models have been shown to be very important in reproducing certain behaviors.

A deterministic model based on ordinary differential equations assumes a continuum of values for concentration. This ignores the fact that cellular processes operate at the molecular level and concentrations can be described using discrete values representing the number of molecules. However, because we often deal with systems containing tens of thousands of particles, we assume that we can describe concentration as a continuous variable and therefore differential equations are an appropriate choice. For systems where the particulate number is very low, of the order of tens of particles, the use of a continuum measure might be unreasonable.

A **deterministic model** is one where a given input will always produce the same output. For example, in the equation  $y = x^2$ , setting  $x$  to 2 will always yield the output 4.

A **stochastic model** is one where the processes described by the model include a random element. This means that repeated runs of a model will yield slightly different outcomes.

However, an additional and more important problem arises when dealing with low particulate numbers. At low concentrations, Brownian motion becomes a significant factor in determining reaction rates. The time when a molecule binds or is transformed becomes a probabilistic property. Models of systems containing low particulate numbers are therefore better modeled using a stochastic, discrete approach [185, 149].

We can now classify a model as a combination of attributes. The water tank model uses a deterministic, continuous approach. The model of the lion population on the Serengeti uses a discrete and stochastic approach. Table 4.2 shows four possible combinations and examples where each combination might be appropriately used.

Type	Example
Continuous/Deterministic	Projectile motion
Continuous/Stochastic	Brownian motion
Discrete/Deterministic	Large population dynamics
Discrete/Stochastic	Small population dynamics

**Table 4.2** Examples of different kinds of model.

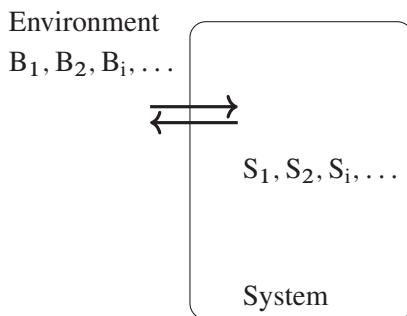
## Forcing Functions

As described earlier, it is common to ensure that the surroundings do not change during the duration of the study. For example, we might make sure that the pH remains constant by using a buffer solution. The key point is that the experimenter has complete control over the experiment. In some cases it is useful for an experimenter to change the surrounding conditions in a controlled fashion. For example, he/she might slowly increase the concentration of an administered drug or make a step change in a variable such as enzyme concentration. In systems theory such controlled changes are often called **forcing functions**.

## 4.7 Example

---

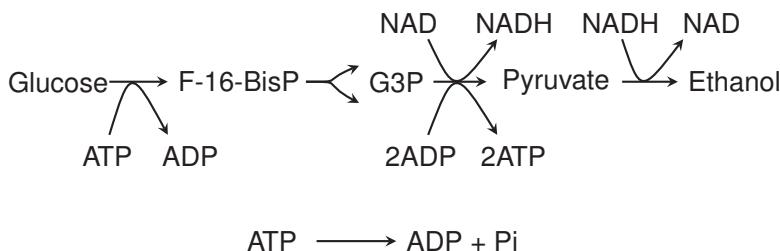
Figure 4.6 illustrates a simplified model of glycolysis. The corresponding Table 4.3 lists the different variables and parameters identified in the model. The concentration of glucose and ethanol are assumed to be boundary variables, controlled by the observer and classified



**Figure 4.5** System and Environment:  $S_1, S_2, S_i, \dots$  are state variables that may change during the evolution of the system;  $B_1, B_2, B_i, \dots$  are boundary variables that are clamped to certain values by the observer. The exchange arrows represent the exchange of mass between the environment and system.

as inputs. Control can be arranged by supplying glucose from a large volume compartment so that during its consumption there is only a negligible change in concentration. Likewise, we assume that ethanol is discharged into a large volume.

Another set of concentrations assumed to be constant are the NAD and NADH cofactors. This may be an unreasonable assumption to make however, because we know that the redox potential can change. We must assume that the model builder has good reason for making this assumption and will make this explicit when the model is formally published. The model builder must be specific about these decisions and explain why they were made. Such choices are necessary when building models and great care should be made when making them. One simple way to justify this assumption is that if the model adequately predicts experiments of interest to the experimenter, then it seems reasonable that a floating redox potential is not important. However as demands on the model to make further predictions increase, there may come a time when the model fails to make a correct prediction, and assumptions such as the fixed redox potential need to be revisited.



**Figure 4.6** A simplified glycolytic pathway. Many reactions have been condensed and ATP consumption has been simplified to a single process,  $\text{ATP} \rightarrow \text{ADP} + \text{Pi}$ .

State Variables	System Parameters	Boundary Variables/Inputs
F-16-BisP	Kinetic Constants	Glucose
G3P	Enzyme Activities	Ethanol
Pyruvate	Volume	NAD
ATP	Temperature	NADH
ADP		Pi

**Table 4.3** Variables and parameters for the simplified glycolytic model 4.6. We assume that glucose and ethanol are clamped by the observer using large volume sinks. We assume that during the period of study, the concentrations of NAD and NADH remain essentially unchanged. F-16-BisP = Fructose-1,6-bisphosphate; G3P = Glyceraldehyde-3-Phosphate; Pi = Phosphate.

The modeler also makes an assumption about ATP. Since glycolysis is an important pathway for generating ATP, some way to simulate ATP consumption is necessary. This is achieved by including a single step that hydrolyzes ATP to ADP, even though we know that ATP consumption is a complex process involving many separate reactions. The response of the pathway to changing ATP demand can be simulated by perturbing the ATP demand step.

We know that the number of molecules involved in glycolysis is huge, of the order of 100,000 to millions. We can therefore safely use a continuous, deterministic model, most likely based on a set of differential equations. The assumptions made in building this model may appear to be completely unreasonable, but one sure test is to determine how well the model reproduces what is currently known about the system and whether it makes useful predictions that can be further tested. If either of these tests fail, then we know that the assumptions about the model need amendment.

There is one final and important point to make. It is easy to look at a model and suggest that it is unrealistic because it misses out certain features. However a model should only be judged by how useful it is, not by how many details it incorporates. This is a common error made by many who are new to modeling.

The realism of a model can only be judged with respect to its purpose and utility.

## Steps in Building a Model

To summarize, we can break down the approach to building a model into five stages:

- Define the system boundaries.

- Define the simplifying assumptions.
- Invoke physical laws to describe the system processes.
- Test (validate) the model against experimental data.
- Alter model if necessary and repeat.

## 4.8 Dimensions and Units

The variables and parameters that go into a model are expressed in some standard of measurement. In science the recognized standard for units are the SI units. These include units such as the *meter* for length, *kilogram* for mass, *second* for time, *Joules* for energy, *kelvin* for temperature and the *mole* for amount. The mole is of particular importance because it is a means to measure the number of particles of substance irrespective of substance mass. Thus 1 mole of glucose has the same number of molecules as 1 mole of the enzyme glucose-6-phosphate isomerase even though the mass of each type of molecule is quite different. The actual number of particles in 1 mole is defined as the number of atoms in 12 grams of carbon-12 which has been determined empirically to be  $6.0221415 \times 10^{23}$  (Avogadro's constant). This definition means that 1 mole of substance will have a mass equal to the molecular weight of the substance, making it easy to calculate the number of moles using the following relation:

$$\text{moles} = \frac{\text{mass}}{\text{molecular weight}}$$

The concentration of a substance is expressed in moles per unit volume and is usually termed molarity. Thus a 1 molar solution means 1 mole of substance in 1 liter of volume.

### Dimensional Analysis

Dimensional analysis is a simple but effective method for uncovering mistakes when formulating kinetic models.

Amounts of substance is usually expressed in moles and concentrations in moles per unit volume ( $\text{mol } l^{-1}$ ). Reaction rates can be expressed either in concentrations or amounts per unit time depending on the context ( $\text{mol } t^{-1}$ ,  $\text{mol } l^{-1} t^{-1}$ ).

Rate constants are expressed in differing units depending on the form of the rate law. The rate constants in simple first-order kinetics are expressed in per unit time ( $t^{-1}$ ), while in second-order reactions the rate constant is expressed per concentration per unit time ( $\text{mol}^{-1} t^{-1}$ ).

In dimensional analysis, units on the left and right-hand sides of expressions must be the same units (or dimensions). There are certain rules for combining units when checking consistency in units. Only like units can be added or subtracted, thus the expression  $S + k_1$  cannot be summed because the units of  $S$  are likely to be  $\text{mol } l^{-1}$ , and the units for  $k_1$ ,

$t^{-1}$ . Even something as innocent looking as  $1 + S$  can be troublesome because  $S$  has units of concentration but the constant value ‘1’ is unitless.

#### Example 4.2

---

Determine the overall units for the expression  $k_1 S/K_m$  where the units for each variable are  $k_1(t^{-1} l)$ ,  $S(\text{mol } l^{-1})$ , and  $K_m(\text{mol } l^{-1})$ .

We first write out the expression in terms of the individual units:

$$t^{-1} l \text{ mol } l^{-1}/(\text{mol } l^{-1})$$

By treating the symbols as algebraic variables, we see that the symbol  $\text{mol } l^{-1}$  will cancel leaving just:

$$t^{-1} l$$


---

The term in the exponential must be dimensionless. The term  $e^{kt}$  is permissible, but  $e^k$  is not if, for example,  $k$  is a first-order rate constant. Trigonometric functions will always resolve to dimensionless quantities because the argument will be an angle. Angles can always be expressed as a ratio of lengths which will, by necessity, have the same dimension.

## 4.9 Classification of Models

---

In addition to classifying models as discrete/continuous and deterministic/stochastic, there are additional properties of models that can be used for further categorization (Table 4.4).

---

Linear or Nonlinear
Dynamic or Static
Time invariant or time dependent
Lumped or distributed parameter models

---

**Table 4.4** Additional categories for classifying models.

### Dynamic and Static Models

A static model is one where the variables of the system do not change in time. For example, a circuit made up of only resistors can be modeled as a static system because there are no elements in the circuit that can store or dissipate charge. The currents and voltages are considered instantaneous without any time evolution. Static systems are therefore unaffected by time and as such they are simpler to model. A flux balance model [130] is an example of a static model in biochemical modeling.

## Time Invariant Systems

All the models we will consider in this book will be dynamic models, that is proteins or metabolite levels change over time. In these cases time is acting as an independent variable and means that running the model at a start time of  $t = 0$  or  $t = 10$  makes no difference to the time evolution of the model. All that matters are the initial conditions we set to the state variables and the values we assign to the parameters and inputs. Such models are called **time invariant**.

If a parameter of the system depends on time, then the model is called time dependent. This means that the system will behave differently if the same input is applied at different times. An example of a time dependent model is where we apply a drug in the form of a pulse and the duration of the pulse depends on when the drug was administered. An example of a time dependent non-biological model is a parking lot where the price of a ticket depends on the time of day. Those systems which are linear and time invariant represent a special category of system called linear time invariant systems (LTI). Such systems will be covered in greater detail in a subsequent book.

## Lumped and Distributed Parameter Models

Many complex models can be approximated with a single number. For example, we often describe a resistor using a single value, its resistance. In reality the resistor has a length, a diameter, and a chemical composition. The resistance is a function of all these properties that make up the resistor. We could model the resistor by slicing up the resistor into many small compartments and compute the resistance as a systemic property. In the former case we have what is called a lumped parameter model, in the second case a distributed parameter model.

## 4.10 Linear and Nonlinear Models

---

When we use mathematics to describe physical systems, there is a great divide that separates **linear** from **nonlinear models**. This separation is fundamental and places hard limits on what we can and cannot do with mathematical analysis.

Inputs to a linear system result in their weighted sum appearing in the outputs. The output is a superposition of the inputs. The simplest linear system is given by the relation  $y = ax$ , where  $x$  is the input and  $y$  the output. We know this is linear for the following reason. Let us apply two separate inputs,  $x_1$  and  $x_2$  to this system. This gives us outputs  $ax_1$  and  $ax_2$ , respectively. If we now apply the sum of the inputs,  $x_1 + x_2$ , we get  $a(x_1 + x_2)$  as the output, which is simply the sum of the separate inputs.

$$ax_1 + ax_2 = a(x_1 + x_2)$$

This is called the property of **additivity** and can be generalized as follows. A mathematical model,  $f(x)$ , shows additivity if the following is true:

$$f(x_1 + x_2 + \dots) = f(x_1) + f(x_2) + \dots$$

This states that the sum of multiple inputs applied simultaneously is equivalent to applying the inputs separately. Nonlinear systems do not follow this rule. Strictly speaking, a linear system also needs to satisfy **homogeneity** (or scaling), that is,  $f(ax) = af(x)$ . Combining additivity and homogeneity gives us the general rule of linearity called **superposition**:

$$f(ax_1 + bx_2 + \dots) = f(ax_1) + f(bx_2) + \dots$$

Any system that satisfies superposition is a linear system. Any system that does not is a nonlinear system. Table 4.5 illustrates some functions that are nonlinear.

$x^n$	$\sqrt[n]{x}$
$xy$	$\sin(x)$
$e^x$	$\log(x)$
$(dy/dy)^n$	$V_m S/(S + K_m)$

**Table 4.5** Examples of nonlinear functions.

### Example 4.3

Show that the function  $e^x$  is nonlinear.

We first apply separate inputs,  $x_1$  and  $x_2$ , to the function and compute the sum of the output, that is:

$$e^{x_1} + e^{x_2}$$

We next take the sum of the inputs,  $x_1 + x_2$ , and apply the sum to the function, that is:

$$e^{x_1+x_2}$$

To obey additivity the two expressions must be equal. However,  $e^{x_1+x_2} = e^{x_1}e^{x_2}$  which is not the same as  $e^{x_1} + e^{x_2}$ . Therefore  $e^x$  is a nonlinear function.

Similarly, we can also easily show that homogeneity ( $f(ax) = af(x)$ ) is not true because it should be evident that:

$$ae^x \neq e^{ax}$$

---

To appreciate the difference between linear and nonlinear functions, consider the system  $y = x^2$ . Let us apply two separate inputs,  $x_1$  and  $x_2$ , to give outputs  $x_1^2$  and  $x_2^2$ . If we

now apply the inputs simultaneously, that is  $y = (x_1 + x_2)^2$ , we obtain  $x_1^2 + x_2^2 + 2x_1x_2$ . We see that the output is not simply  $x_1^2 + x_2^2$  but includes an additional term,  $2x_1x_2$ . This term is the nonlinear contribution. Imagine that this difference now enters further nonlinear processes, leading to further changes. Eventually the output looks nothing like the input. This makes most nonlinear systems difficult to understand.

Unless the system has an infinite number of solutions (degenerate) or has the trivial solution (where the solution is zero), linear systems will admit only one solution. In contrast, it is possible for nonlinear systems to admit multiple solutions, that is given a single input, a nonlinear system can regurgitate one of a number of possible distinct outputs. To make matters worse, in the majority of mathematical models found in biochemical networks, it is not even possible to find the solutions analytically. That is, we cannot mathematically describe how an output depends on an input other than by doing a brute-force computer simulation. Understanding nonlinear systems in biology or elsewhere is a huge unresolved problem. While there is a complete theory of linear systems, no such equivalent exists for nonlinear systems. When dealing with nonlinear systems we are often forced to use computer simulations.

There is one useful approach to help address nonlinear models. If we were to draw a nonlinear curve on a graph and zoom in closer to a particular point on the graph, the curve would eventually look like a straight line. We can essentially turn a nonlinear system into a linear one but only in small regions of the system's behavior where linearity dominates. This process is called **linearization** and is a powerful technique for studying nonlinear systems.

## 4.11 Linearization

---

When modeling nonlinear systems we have two options, to simulate or to linearize. Simulation will be considered later, here we will look closely at a technique called linearization. To linearize a model means replacing the nonlinear version with a linear approximation which is easier to understand. It should be emphasized that in the process, we loose valuable information, but enough information is preserved to make linearization an extremely useful and popular tool.

One of the most useful results in mathematics is the **Taylor series** (See Appendix E for a review). This is a way of approximating a mathematical function by using an infinite polynomial series such as the following:

$$f(x) = c_0 + c_1x + c_2x^2 + c_3x^3 + \dots \quad (4.3)$$

We can represent any continuous function using such a polynomial. For example, we can represent  $\sin(x)$  using the formula:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \dots \quad (4.4)$$

Without going into the details, the Taylor series is a means for defining the  $c_i$  terms in the polynomial series (4.3) given any continuous function. The Taylor series is always defined around some operating point,  $x_o$ , and a point near the operating point,  $x$ . The Taylor series is given by:

$$f(x) = f(x_o) + \frac{df}{dx} \Big|_{x_o} (x - x_o) + \frac{1}{2!} \frac{d^2 f}{dx^2} \Big|_{x_o} (x - x_o)^2 + \dots + \frac{1}{n!} \frac{d^n f}{dx^n} \Big|_{x_o} (x - x_o)^n + \dots \quad (4.5)$$

All derivatives must be elevated at the operating point  $x_o$ .

The various derivatives in the Taylor series **must** be evaluated at  $x_o$ . The function  $f(x)$  must be continuous so there are no holes or sudden breaks (discontinuities) in the curve described by the function. The number of terms in the Taylor series determines how well the series approximates the function: the fewer terms, the more approximate the series is. For example, the most approximate expression is given by using only the first term,  $f(x_o)$ . However,  $f(x_o)$  is a constant so this represents a very poor approximation. To make the approximation more useful we include the first two terms of the Taylor series:

$$f(x) \approx f(x_o) + \frac{df}{dx} \Big|_{x_o} (x - x_o) \quad (4.6)$$

Provided  $x$  is close to  $x_o$ , the approximation is good. Note that the derivative must be computed at the operating point,  $x_o$ . For example, let us form the Taylor series for the function  $y = \sin(x)$  around  $x_o = 0$ . Recall that  $\sin(0) = 0$  and  $\cos(0) = 1$ , then write out the Taylor series:

$$\begin{aligned} y &\approx \sin(0) + \frac{d\sin(x)}{dx} \Big|_{x_o} (x - 0) + \frac{1}{2!} \frac{d^2 \sin(x)}{dx^2} \Big|_{x_o} (x - 0) + \dots \\ y &\approx 0 + 1x + 0 - \frac{1}{3!}x^3 + 0 + \frac{1}{5!}x^5 + \dots \end{aligned}$$

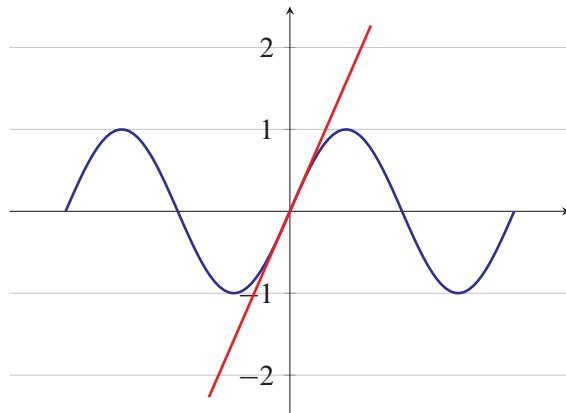
That is:

$$y \approx x - \frac{x^3}{3!} + \frac{x^5}{5!} - \dots$$

Note this is the same as equation (4.4). The linear approximation is given by the first two terms:

$$y \approx \sin(0) + \frac{d\sin(x)}{dx} \Big|_{x_o} (x - 0)$$

Since  $\sin(0) = 0$  and  $d\sin(x)/dx = \cos(0) = 1$ , the linear approximation is therefore  $y = x$ , a straight line running through the origin (Figure 4.7). We have linearized the sin



**Figure 4.7** Linearized  $\sin(x)$  function represented by the straight line through zero.

function and Figure 4.7 shows how good our approximation is. With only two terms the linear approximation only matches a region near  $x_o$  and fails to capture the periodic nature of the sin curve.

To illustrate linearization with another example, consider the simple nonlinear function,  $y = x^2$ . To linearize we must first choose an operating point around which to linearize, for example,  $x_o = 2$ . According to the second term in the Taylor series we need to find the derivative,  $df/dx$  so that the first two terms of the Taylor series (Equation 4.6) become:

$$f(x) = f(2) + 2x_o(x - 2)$$

To obtain the linear approximation we evaluate the derivative at the operating point ( $x_o = 2$ ), that is  $df/dx = 2x_o = 4$  so that the final linear approximation is:

$$f(x) = 4 - 4(x - 2) = 4x - 4$$

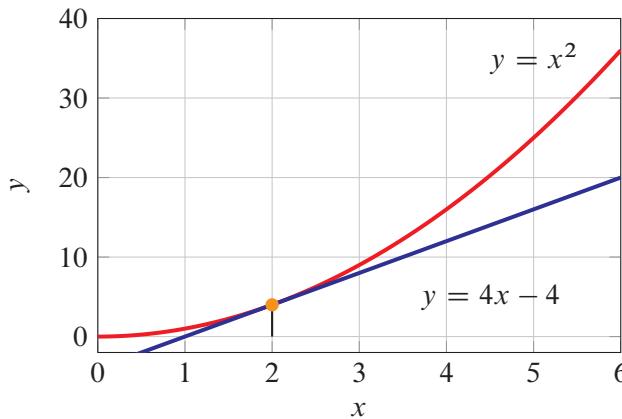
Figure 4.8 shows the original nonlinear function together with the linear approximation.

Equation (4.6) is also commonly written in the form:

$$f(x) \simeq f(x_o) + \frac{df}{dx} \Big|_{x_o} \delta x$$

where  $\delta x = (x - x_o)$ . If the equation  $f$  is a function of more than one variable, then additional terms appear. For example, the linearization of  $f(x, y)$  near  $x_o$  and  $y_o$  will give:

$$f(x, y) \approx f(x_o, y_o) + \frac{\partial f}{\partial x} \Big|_{x_o, y_o} \delta x + \frac{\partial f}{\partial y} \Big|_{x_o, y_o} \delta y \quad (4.7)$$



**Figure 4.8** Taylor series approximation of  $y = x^2$  at the operating point,  $x_o = 2$ . The linear approximation is  $y = 4x - 4$ .

As before, the derivatives must be evaluated at the operating point.

#### Example 4.4

---

Linearize the following equation at  $x_o = 2$ :

$$y = \frac{x^3}{x + 1}$$

To linearize we must apply equation (4.6). We first compute,  $f(x_o)$ . Since  $x_o = 2$ , then:

$$f(x_o) = 8/3$$

Next we form the derivative  $\partial f / \partial x$ :

$$\frac{df}{dx} = \frac{2x^3 + 3x^2}{(x + 1)^2}$$

At  $x_o = 2$  the derivative is given by:

$$\frac{df(2)}{dx} = \frac{28}{9}$$

Inserting  $f(x_o)$  and the derivative into:

$$f(x) \approx f(x_o) + \frac{df}{dx}(x - x_o)$$

yields:

$$f(x) \approx \frac{8}{3} + \frac{28}{9}(x - x_o) = \frac{8}{3} + x \frac{28}{9} - \frac{56}{9} = \frac{28x - 32}{9}$$

***Example 4.5***

Linearize the following equation at  $x_o = 1$  and  $y_o = 0$ :

$$f(x, y) = x^2 - 2xy - \sin(y)$$

To linearize a two dimensional system we must apply equation (4.7). We first compute,  $f(x_o, y_o)$ . Since  $x_o = 1$  and  $y_o = 0$ , then:

$$f(x_o, y_o) = 1$$

Next we form the two derivatives  $\partial f / \partial x$  and  $\partial f / \partial y$ :

$$\frac{\partial f}{\partial x} = 2x - y \quad \frac{\partial f}{\partial y} = -2x - \cos(y)$$

At  $x_o = 1$  and  $y_o = 0$  the derivatives are given by:

$$\frac{\partial f(1, 0)}{\partial x} = 2 \quad \frac{\partial f(1, 0)}{\partial y} = -3$$

Inserting  $f(x_o, y_o)$  and the derivatives into:

$$f(x, y) \approx f(x_o, y_o) + \frac{\partial f}{\partial x}(x - x_o) + \frac{\partial f}{\partial y}(y - y_o)$$

yields:

$$f(x, y) \approx 2x - 3y - 1$$


---

## 4.12 Approximations

By their very nature, models involve making assumptions and approximations. The best modelers are those who can make the most shrewd and reasonable approximations without compromising a model's usefulness. There are however some kinds of approximations which are useful in most problems, these include:

- Neglecting small effects.
- Assuming that the system environment is unchanged by the system itself.
- Replacing complex subsystems with lumped or aggregate laws.
- Assuming simple linear cause-effect relationships where possible.
- Assuming that the physical characteristics of the system do not change with time.

- Neglecting noise and uncertainty.

Let's review each of these in greater detail.

**Neglecting small effects.** This is the most common approximation to make. In many studies there will always be parts of the system that have a negligible effect on the properties of the system, at least during the period of study. For example, the rotation of the earth, the cycle of the moon, or the rising and setting of the sun will most likely have a negligible influence when studying the action of an enzyme. Assuming of course we are not studying circadian rhythms.

**Assuming that the system environment is unchanged by the system itself.** This is a basic assumption in any study. The minute a system starts to affect the environment in an uncontrolled way, we have effectively extended the system boundaries to include more of the environment. It will often be the case that the interface between the environment and the system will not be perfect so that there will be some effect that the system has on the environment. So long as this effect is small, we can assume that the environment is not affected by the system.

**Replacing complex subsystems with lumped or aggregate laws.** Lumping subsystems is a commonly used technique in simplifying cellular models. The most important is the use of aggregate rate laws, such as Michaelis-Menten or Hill like equations to model cooperativity. Sometimes entire sequences of reactions can be replaced with a single rate law. Certain assumptions are invoked in making the aggregations, in particular it will often be assumed that the processes inside the aggregate are much faster than the processes external to the aggregate. We will return to this topic in Chapter 5.

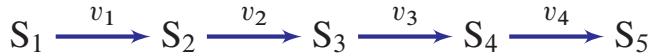
**Assuming simple linear cause-effect relationships.** In some cases it is possible to assume a linear cause-effect between an enzyme reaction rate and the substrate concentration. This is especially true when the substrate concentration is below the  $K_m$  of the enzyme. Linear approximations make it much easier to understand a model.

**Physical characteristics do not change with time.** A modeler will often assume that the physical characteristics of a system do not change, for example the volume of a cell, the values of the rate constants or the temperature of the system.

**Neglecting noise and uncertainty.** Most models make two important approximations. The first is that noise in the system is either negligible or unimportant. In many nonbiological systems such an approximation might be quite reasonable. However cellular phenomena operate at the molecular level. Biological systems are susceptible to noise generated from thermal effects as a result of molecular collisions. For many systems the large number of particles ensures that the noise generated in this way is insignificant and in most cases can be safely ignored. For some systems such as prokaryotic organisms, the number of particles can be very small. In such cases the effect of noise can be significant and therefore must be included as part of the model.

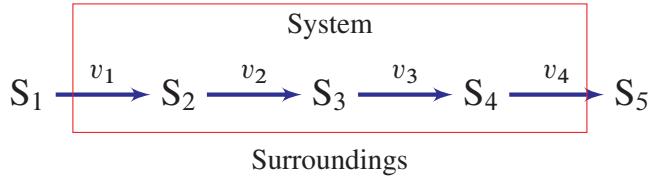
## 4.13 Example Model

Before we leave this chapter, let us look at building a model of a simple chain of four enzyme catalyzed reactions (Figure 4.9). Begin by constructing a mathematical model of this system.



**Figure 4.9** Simple Straight Chain Pathway.

First, we must decide where the boundary of the pathway is, assuming there is one. A convenient place to have a boundary is the start and end metabolites of the pathway, that is  $S_1$  and  $S_5$ . We will assume that these two metabolites are **fixed** and are unaffected by the system (Figure 4.10). In modeling language these are the boundary species or inputs to the model.



**Figure 4.10** Simple Straight Chain Pathway with system shown in a box and  $S_1$  and  $S_5$  outside the system. We assume  $S_1$  and  $S_5$  are fixed.

The metabolites that can change in time include  $S_2$ ,  $S_3$ , and  $S_4$  and are known as the dependent variables, the state variables, or floating species. We can write the differential equations that represent the rates of change of  $S_2$ ,  $S_3$  and  $S_4$ . Note that there will be no differential equations assigned to  $S_1$  and  $S_5$  because these are fixed and unchanging. According to mass-balance, the following differential equations must be true:

$$\frac{dS_2}{dt} = v_1 - v_2$$

$$\frac{dS_3}{dt} = v_2 - v_3$$

$$\frac{dS_4}{dt} = v_3 - v_4$$

Next we must decide on the rate laws,  $v_1$ ,  $v_2$ ,  $v_3$ , and  $v_4$ . This is possibly the most difficult part to building a model and a detailed examination of the literature is necessary to decide

which rate laws are the most appropriate to use. The companion text book, “Enzyme Kinetics for Systems Biology” [148] gives much more detail on rate laws in general. Here a variety of rate laws will be used to illustrate the kinds of rate laws that one might employ. For example, a simple reversible mass-action rate law may be best for the first reaction  $v_1$ , that is:

$$v_1 = k_1 S_1 - k_2 S_2$$

This rate law introduces two new parameters, the rate constants,  $k_1$  and  $k_2$ . These are fixed and unaffected by the model. For the second reaction, let us use a simple allosteric regulated rate law. Assume that the reaction  $v_2$  is allosterically inhibited by  $S_4$ . For this we can apply the simplest exclusive Monod, Wyman, Changeaux model [120]:

$$v_2 = V_m \frac{S_2 (1 + S_2/K_m)^4}{(1 + S_2/K_m)^4 + L (1 + S_4/K_I)^4}$$

where the Hill coefficient is equal to four,  $L$  is the allosteric constant,  $K_I$  is the inhibition constant,  $K_m$  is the substrate concentration at half-maximal activity, and  $V_m$  the maximal velocity.

The third rate law will be a simple irreversible but product inhibited Michaelis-Menten rate law, that is:

$$v_3 = V_m \frac{S_3}{S_3 + K_m (1 + S_3/K_p)}$$

where  $V_m$  is the maximal velocity of the reaction,  $K_m$  is the Michaelis constant, and  $K_p$  is the product inhibition constant. The last reaction,  $v_4$  will be assigned a simple irreversible mass-action rate law:

$$v_4 = k_3 S_4$$

where  $k_3$  is the rate constant. In total the model has ten parameters, two boundary species and three state (or floating) species. The model can be completed by assigning values to all the parameters, boundary species, and initial conditions to the state variables. Once the model is described, it can be entered into a simulation tool such as Tellurium (See Appendix H) or PathwayDesigner [155, 11] and the evolution of the system studied. We will discuss running simulations in Chapter 5 and 6.

## **4.14 Where to get Data for Building Models**

---

The perennial problem that confronts the biochemical pathway modeler is where to get the data to build the first version of the model. We should first distinguish two kinds of data, network connectivity and data related to the kinetics of individual reaction steps. The former is well supported in the literature and various databases. An entire field called metabolic network reconstruction has emerged in the last ten years as a result of the availability of genome-scale data sets.

## Metabolic Reconstruction

Metabolic network reconstructions [68, 174] describe an organism's metabolism through the analysis of genomic data. The scale of network reconstruction may range from individual pathways to whole genomes. Analyzing and annotating genomic sequences, storing and retrieving metabolic network information, and representing network data are key tasks associated with metabolic network reconstruction. A common first approach to reconstructing metabolic networks is to compare the unknown network with already well characterized networks. After that, further experimental data is collected to validate or fill in any missing gaps. As a result of these efforts, there are now many hundreds of metabolic reconstructions available. Model SEED [69] is a resource for genome-scale metabolic models. Of particular interest is that metabolic reconstructions can be download in standard SBML, thus allowing a wide range of tools to import the reconstructions.

Other significant sources of networks are the KEGG<sup>2</sup> and MetaCyc<sup>3</sup> repositories. KEGG in particular has a wide range of networks including both vertebrates and invertebrates. The SuBliMInaL Toolbox<sup>4</sup> provides facilities to download and manage network models from KEGG and MetaCyc in the form of SBML.

The data are less easily obtained for protein signaling and gene regulatory networks. For both of these network types, one has to trawl through the literature. Although there have been many attempted efforts to use high-throughput data to generate networks, these are generally unreliable [167, 7]. The current most reliable way to generate protein and gene regulatory network is to read the source literature.

## Kinetic Data

The real problem however is collecting kinetic data for the individual reaction steps. BREND<sup>5</sup> is an enzyme database that contains details on the kinetics of many different enzymes. The main problem is that the data reported in BREND was often collected under non-physiological conditions. It has been shown several of times in recent years [40, 97] that reliable models require kinetic data to be measured under physiological conditions. If reliable kinetics data is not available, then an alternative is to employ generalized or approximate rate laws. There are a variety of these (covered in more detail in the companion book Enzyme Kinetics for Systems Biology), but one in particular will be mentioned here, the lin-log approximation.

Without going into the derivation, the simplest *linear* approximation is given by:

---

<sup>2</sup><http://www.genome.jp/kegg/>

<sup>3</sup><http://metacyc.org/>

<sup>4</sup><http://www.mcisb.org/resources/subliminal/>

<sup>5</sup><http://www.brenda-enzymes.org/>

$$v = v_o \left( 1 + \sum_i \varepsilon_{S_i}^v \frac{\delta S_i}{S_i^o} \right) \quad (4.8)$$

In the linear approximation (4.8) the species term is given by:  $\delta S / S_o$ , or  $(S - S_o) / S_o$ . Recall the Taylor expansion (E.6) for the natural logarithmic function ( $\ln$ ) around  $y_o$  to the first (linear) approximation is given by:

$$\ln(y) \simeq \ln(y_o) + \frac{y - y_o}{y_o}$$

Note that  $\partial \ln(y_o) / \partial y_o = 1/y_o$ . Rearranging the linear approximation yields:

$$\frac{y - y_o}{y_o} \simeq \ln(y) - \ln(y_o) = \ln\left(\frac{y}{y_o}\right)$$

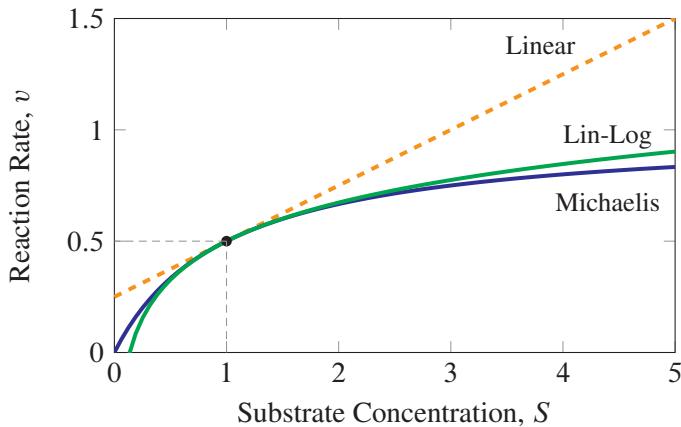
Now substitute  $\delta S_i / S_i^o$  for  $\ln(S_i / S_i^o)$ . This simple change leads to a significantly improved approximation over the linear equation and is called the linear-logarithmic approximation or lin-log for short [183, 64, 179, 66].

One of the chief advantages of this approximation is that at high substrate concentration the response approximates the saturation by substrate (See Figure 4.11). The general form of the lin-log equation is given by:

$$v = v_o \left[ \frac{e}{e_o} \right] \left( 1 + \sum_i \varepsilon_{S_i}^v \ln\left(\frac{S_i}{S_i^o}\right) \right) \quad (4.9)$$

where  $S$  is the reactant concentration and  $\varepsilon$  the elasticity (D.17). The summation is over all reactants and effectors that might modulate the reaction rate (except the enzyme concentration). The rate law is always defined around some reference state where  $v_o$  is the reference reaction rate and  $S_i^o$  is the reference reactant concentration.

As with the linear approximation (4.8), the utility of this method is that the elasticity values (D.17) (kinetic orders) can be estimated from the known thermodynamic properties of the reaction, especially if the reaction is operating below saturation. If no thermodynamic information is available, the elasticities may be set to the stoichiometries of the respective reactants if necessary. In either case it is important to note the lin-log approximation is only valid around the chosen reference state, but is much better (See Figure 4.11) than the linear approximation. One possible drawback to the lin-log approximation is that at zero reaction rate, the reactant levels are not necessarily at equilibrium (Figure 4.11). This can lead to reverse reaction rates when the prevailing metabolite levels suggest otherwise. The lin-log approximation is therefore not suitable when a reaction is close to equilibrium or when metabolites levels are very low.



**Figure 4.11** Linear, power law, and lin-log approximations to a Michaelis-Menten curve around the reference point  $S_o = 1$ ;  $V_m = 1$ ;  $K_m = 1$ . Dashed: Linear law. The drawback of the lin-log approximation is that the curve does not go through zero.

## 4.15 Of Exactitude in Science

And finally a lesson to all model builders:

“On Exactitude in Science...In that Empire, the Art of Cartography attained such Perfection that the map of a single Province occupied the entirety of a City, and the map of the Empire, the entirety of a Province. In time, those Unconscionable Maps no longer satisfied, and the Cartographers Guilds struck a Map of the Empire whose size was that of the Empire, and which coincided point for point with it. The following Generations, who were not so fond of the Study of Cartography as their Forebears had been, saw that that vast Map was Useless, and not without some Pitilessness was it, that they delivered it up to the Inclemencies of Sun and Winters. In the Deserts of the West, still today, there are Tattered Ruins of that Map, inhabited by Animals and Beggars; in all the Land there is no other Relic of the Disciplines of Geography.

Suarez Miranda, *Viajes de varones prudentes*, Libro IV,Cap. XLV, Lerida, 1658

From Travels of Praiseworthy Men (1658) by J.A. Suárez Miranda, 1946, translated by Andrew Hurley

## Further Reading

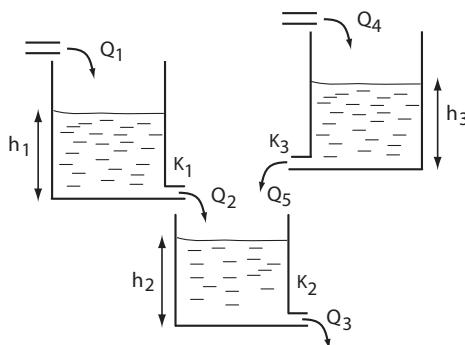
1. Davis PJ and Hersh R (1981) The Mathematical Experience. Houghton Mifflin Company. ISBN: 0-395-32131-X

2. Riggs DS (1979) Control Theory and Physiological Feedback Mechanisms. Waverly Press, SBN: 683-07244-7
3. Sauro HM (2011) Enzyme Kinetics for Systems Biology. ISBN: 978-0982477311

## Exercises

---

1. Which of the following best describes what a model is:
  - (a) an attempt to form an exact replica of reality.
  - (b) the truth about the real system.
  - (c) a simplification of the real world.
2. State the difference between a deterministic and stochastic model.
3. State the difference between a discrete and continuous model.
4. Suggest what modeling approach you would use for the following systems, i.e. continuous or discrete and deterministic or stochastic:
  - (a) The spread of a forest fire.
  - (b) Growth and spread of sand dunes.
  - (c) A line of people waiting at cash tills in a store.
  - (d) AM radio electrical circuit.
  - (e) A chess game where both players are computer programs.
  - (f) A tumor where individual cells secrete growth factors.
5. Figure 4.12 shows a three tank system similar to the two tank system in Figure 4.2. Derive the differential equations that describes the rate of change of the heights,  $h_1$ ,  $h_2$ , and  $h_3$ .
6. State any assumptions or approximations you made in the previous question relating to the water tank model.
7. List the three most desirable attributes of a model.
8. When we “validate” a model, which of the following do we most likely mean:
  - (a) We show that the model represents the truth about the real system.
  - (b) We increase our confidence in the model’s predictive power.
  - (c) We prove that the model is correct.
9. Two scientists are arguing about a model, one claims that the model is correct but the other suggests that it is the best. Who is making the most reasonable claim and why?



**Figure 4.12** Three tank model.

10. Explain the difference between accuracy and predictability of a model.
11. The authors of a published biochemical model claim that their model has been validated. What do they mean by this?
12. The author George Box is said to made a statement similar to: “all models are wrong, but some are useful.”. What does he mean by this?
13. The transport of a solute across a membrane is given by the equation  $J = P_A(S_{\text{in}} - S_{\text{out}})$ . If  $P_A$  is expressed in  $\text{cm s}^{-1}$  and the transport rate in moles  $\text{cm}^{-2}\text{s}^{-1}$ , what should the concentrations,  $S_{\text{in}}$  and  $S_{\text{out}}$  be expressed in?
14. What is the difference between a state variable and a boundary variable in a biochemical model?
15. Describe the state variables and types of parameter in the following model of a biochemical pathway:

$$\frac{dS_1}{dt} = k_1 X_o - k_2 S_1$$

$$\frac{dS_2}{dt} = k_2 S_1 - (k_4 S_2 - k_4 X_1)$$

List the approximations you think were made in the above model.

16. Show that the following functions are nonlinear with respect to  $x$ :
  - (a)  $\sin(x)$
  - (b)  $e^x$
  - (c)  $V_m x / (x + K_m)$

17. Linearize the following functions:
- $4x^2 + 6x - 10$  at  $x = 1$
  - $V_m x / (x + K_m)$  at  $x = 0$  and  $x = K_m$
18. In the equation  $v = V_m S / (K_m + S)$  where  $S$  is expressed in units of  $\text{mol l}^{-1}$ ,  $V_m$  in  $\text{mol l}^{-1} \text{s}^{-1}$ , and the reaction velocity,  $v$  in  $\text{mol l}^{-1} \text{s}^{-1}$  what are the units for  $K_m$ ?
19. In the previous question, if only the units for  $S$  are known, what can one say about the units of  $K_m$ ?

# 5

## Differential Equation Models

### 5.1 Introduction

---

In this chapter we will discuss how to solve the differential equations generated when we build a biochemical model (3.8). At first this may seem unnecessary given that modern software hides all the details and makes the effort so easy. However, it is still useful to know the basic approach and the limitations of black box solvers so that if problems do arise, one is in a better position to make an informed judgement on how to proceed.

### 5.2 Differential Equation Models

---

To begin, consider the simplest possible model, the first-order irreversible degradation of reactant, S into product P:



The differential equation for this reaction is given by the familiar form:

$$\frac{dS}{dt} = -k_1 S \quad (5.1)$$

Our aim is to solve this equation so that we can describe how S changes in time. There are at least two ways to do this: we can either solve the equation using algebraic methods, or we can use a computer to obtain a numerical solution. To solve the equation using algebra we first divide both sides by S:

$$\frac{dS}{dt} \frac{1}{S} = -k_1 \quad (5.2)$$

In differential calculus, the derivative of  $\ln y$  with respect to  $t$  is:

$$\frac{d \ln y}{dt} = \frac{dy}{dt} \frac{1}{y}$$

This means we can rewrite equation (5.2) in the following form:

$$\frac{d \ln S}{dt} = -k_1$$

Now integrate both sides with respect to  $dt$ , that is:

$$\int \frac{d \ln S}{dt} dt = -k_1 \int dt$$

$$\ln S = -k_1 t + C$$

where  $C$  is the constant of integration. If we assume that at  $t = 0$ ,  $S = S_o$ , then  $\ln S_o = C$ . Substituting this result into the solution yields:

$$\ln S = -k_1 t + \ln S_o$$

$$\ln \left( \frac{S}{S_o} \right) = -k_1 t$$

Raising both sides to the power of  $e$  and multiplying both sides by  $S_o$  gives:

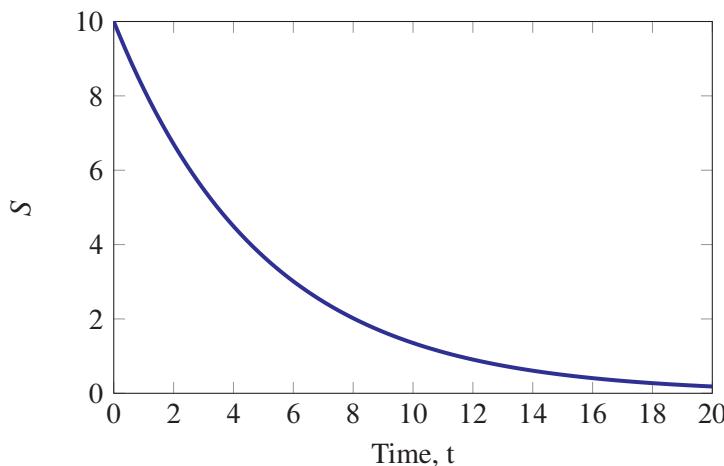
$$S = S_o e^{-k_1 t} \quad (5.3)$$

Figure 5.1 illustrates one solution given a specific initial condition and value for the rate constant. For simple systems or nonlinear system we linearize, it is possible to obtain analytical solutions. However we are quickly confronted with the fact that for the vast majority of real problems, no analytical solution exists. In such cases we must use computers to obtain numerical solutions.

## Numerical Solutions

In the last section we saw how it was possible to solve a differential equation algebraically. If the system of differential equations is linear, there are systematic methods for deriving a solution. Most of the problems we encounter in biology however are non-linear and for such cases algebraic solutions rarely exist. Because of this, computer simulation is often used instead. Since the 1960s, almost all simulations have been carried out using digital computers. Before the advent of digital computers, analog computers were frequently used. In these cases an analog of the system was built using either mechanical or more commonly, electrical analogs of concentrations. Here we will focus on methods used for digital computers.

The general approach to obtain a solution by computer:



**Figure 5.1** Exponential decay from the equation:  $S = S_o e^{k_1 t}$  where  $S_o = 10, k_1 = 0.2$ .

1. Construct the set of ordinary differential equations, with one differential equation for every molecular species in the model.
2. Assign values to all the various kinetic constants and boundary species.
3. Initialize all floating molecular species to their starting concentrations.
4. Apply an integration algorithm to the set of differential equations.
5. If required, compute the fluxes from the computed species concentrations.
6. Plot the results of the simulation.

Step four is the most important and there exists a great variety of integration algorithms that may be used. We will describe three common approaches to give a flavor of how they work. Other than for educational purposes, it is rare for a modeler to write their own integration computer code because many sophisticated libraries and applications already exist. As such, we will focus on some of the approaches themselves and the various software tools now available.

An integration algorithm approximates the behavior of what is, strictly speaking, a continuous system on a digital computer. Since digital computers can only operate in discrete time, the algorithms convert the continuous system into a discrete time system. This is why digital computers can only generate approximations. In practice a particular discrete time step size,  $h$ , is chosen, and solution points are generated at intervals of  $h$  until an upper time limit is reached. As we will discover, the approximation generated by the simplest method is dependent on the step size and in general, the smaller the step size the more accurate the solution. However, since computers can only represent numbers to a given precision (usually 15 to 16 digits on modern computers), it is not possible to continually reduce the step

size in hopes of increasing accuracy. First, the algorithm will soon reach the computer's limits of precision and secondly, the smaller the step size the longer it will take to compute the solution. As a result, we often make a trade-off between accuracy versus computation time.

Let us first consider the simplest method, the Euler method, where the trade-off between accuracy and computer time can be clearly demonstrated.

### Euler Method

The Euler method is the simplest possible way to solve a set of ordinary differential equations (See E.5). Consider the following differential equation that describes the degradation rate of a species, S:

$$\frac{dS}{dt} = -k_1 S$$

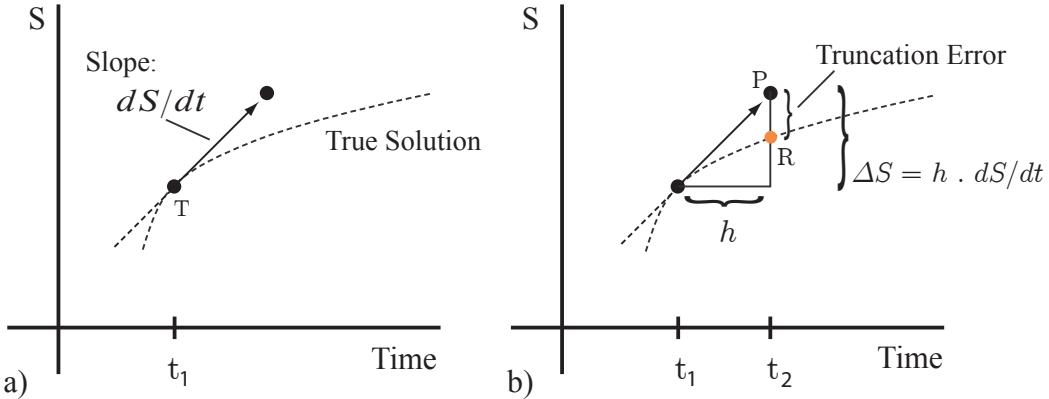
The Euler method uses the rate of change of S to predict the concentration at some future point in time. Figure 5.2 describes the method in detail. At time  $t_1$  the rate of change in S is computed from the differential equation using the known concentration of S at  $t_1$ . The rate of change is used to compute the change in S over a time interval,  $h$ , using the relation,  $h \frac{dS}{dt}$ . The current time,  $t_1$  is incremented by the time step,  $h$ , and the procedure is repeated again, this time starting at  $t_2$ . The method can be summarized by the following two equations which represent one step in an iteration that is repeated until the final time point is reached:

$$\begin{aligned} y(t + h) &= y(t) + h \frac{dy(t)}{dt} \\ t_{n+1} &= t_n + h \end{aligned} \tag{5.4}$$

Figure 5.2 also highlights a problem with the Euler method. At every iteration there will be an error between the change in S we predict, and what the change in S should have been. This error is called the **truncation error** and will accumulate at each iteration. If the step size is too large, this error can make the method numerically unstable resulting in wild swings in the solution.

Figure 5.2 also suggests that the larger the step size, the larger the truncation error. This would imply that the smaller the step size, the more accurate the solution. This is indeed the case, up to a point. If the step size becomes too small, there is the risk that roundoff error will begin to have a significant effect and will propagate at each step. In addition, if the step size is too small, it will require a large number of iterations to simulate even a small time period. The final choice for the step size is therefore a compromise between accuracy and effort. A theoretical analysis of error propagation in the Euler method indicates that the error accumulated over the entire integration period (called the **global error**) is proportional to the step size. Therefore, halving the step size will reduce the global error by half. This

means that to achieve even modest accuracy, small step sizes are necessary. As a result, the method is rarely used in practice. The advantage of the Euler method is that it is very easy to implement in computer code or even on a spreadsheet.



**Figure 5.2** Euler Method. Starting at  $t_1$ , the slope  $dS/dt$  at  $T$  is computed (Panel A). The slope is used to project forward to the next solution in time step,  $h$ , to  $t_2$  (Panel B). The new solution at  $t_2$  is indicated by  $P$ . However the true solution is point  $R$ , located on the solution curve at  $t_2$ . Reducing the step size  $h$  will reduce the error between the exact and projected solution, but will simultaneously increase the number of slope projections necessary to compute the solution over a given time period.

The Euler method can also be used to solve systems of differential equations. In this case all the rates of change are computed first, followed by the application of the Euler equation (5.4). As in all numerical integration methods, the computation must start with an initial condition for the state variables at time zero. The algorithm is described using pseudo-code in Algorithm 1.

### Example 5.1

Solve the decay differential equation (5.1) using the Euler method. Assume  $k_1 = 0.2$  and the concentration of  $S_o$  and  $P$  are time = 0 is 10 and 0, respectively. Assume a step size  $h$ , of 0.4. Form a table of four columns, write out the solution to three decimal places. The 4<sup>th</sup> column should include the exact solution (5.3) for comparison.

**Algorithm 1** Euler Integration Method.  $f_i(y)$  represents the  $i^{th}$  differential equation from the system of ordinary differential equations.

---

$n$  = Number of state variables

$y_i$  =  $i^{th}$  variable

$f_i(y)$  = rate of change at  $y$

Set timeEnd

currentTime = 0

$h$  = stepSize

Initialize all  $y_i$  at currentTime

**while** currentTime < timeEnd **do**

**for**  $i$  = 1 to  $n$  **do**

$dy_i = f_i(y)$

**for**  $i$  = 1 to  $n$  **do**

$y_i(t + h) = y_i(t) + h \ dy_i$

    currentTime = currentTime +  $h$

**end while**

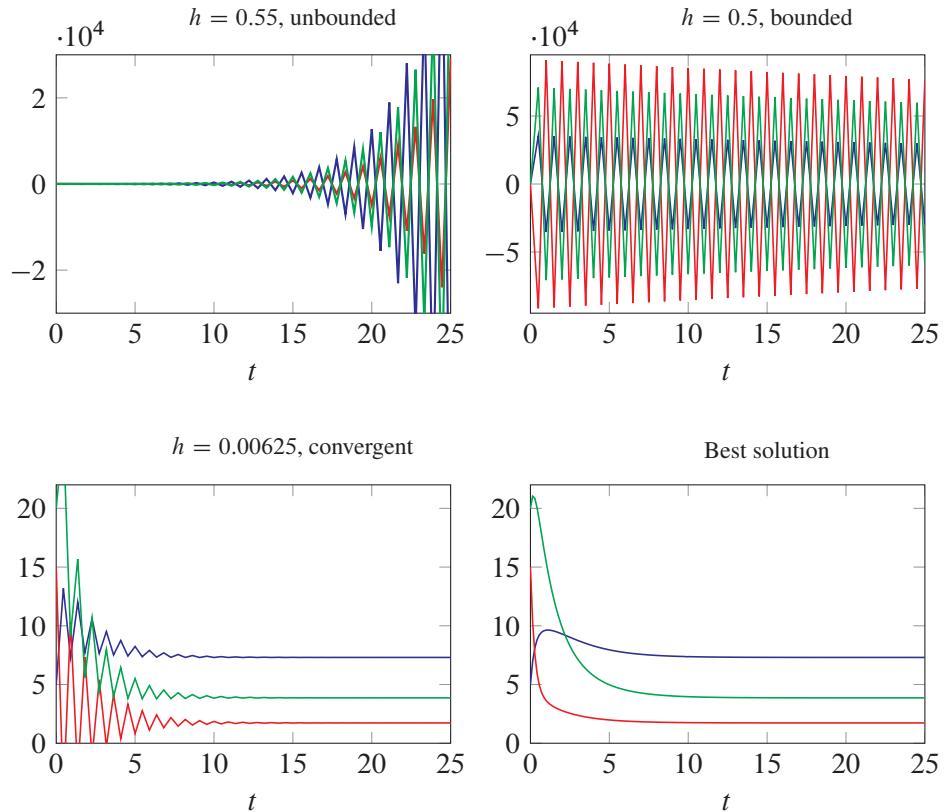
---

Time	Numerical Solution (S)	$dS/dt$	Exact Solution
0	10	2	10
0.4	9.2	1.84	9.23
0.8	8.464	1.6928	8.52
1.2	7.787	0.01	7.87
...			

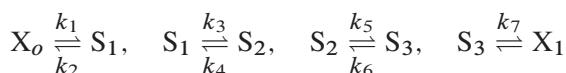
**Table 5.1** Solution to equation (5.1) using a step size of  $h = 0.4$ .

---

Figure 5.3 shows the effect of different step sizes on the Euler method. Four cases are shown, in the worse case ( $h = 0.55$ ) the solution is unbounded and the computer will eventually crash with an overflow error. The second case ( $h = 0.5$ ) is where the result is bounded, but the solution bears no resemblance at all to the actual solution. The third case ( $h = 0.00625$ ) shows that the solution is beginning to resemble the actual solution, but irregularities appear near the start of the integration. The final case shows the actual solution generated from a specialized integrator.



**Figure 5.3** Effect of different step sizes on the Euler method using a simple linear chain of reactions where each reaction follows reversible mass-action kinetics:



where  $k_1 = 0.45, k_2 = 0.23, k_3 = 0.67, k_4 = 1.2, k_5 = 2.3, k_6 = 0.3, k_7 = 0.73, X_o = 10, X_1 = 0, S_1 = 5, S_2 = 15, S_3 = 20$ . See text for details.

## Modified Euler or Heun Method

As indicated in the last section, the Euler method, though simple to implement, tends not to be used in practice because it requires small step sizes to achieve reasonable accuracy. Furthermore, the small step size makes the Euler method computationally slow. A simple modification however can be made to significantly improve its performance. This approach can be found under a number of headings, including the modified Euler method, the Heun, or the improved Euler method.

The modification involves improving the estimate of the slope by averaging two derivatives, one at the initial point ( $dy(t)/dt$ ) and another at the end point ( $dy(t + h)/dt$ ). In order to calculate the derivative at the end point, the first derivative is used to predict the end point. The two slopes are then averaged and the average is used to predict the final predicted  $y$  value. (Figure 5.4). This method is quite simple to implement in computer software and is summarized by equations (5.5).

Figure 5.4 describes the Heun method graphically. A theoretical analysis of error propagation in the Heun method shows that it is a second-order method; that is, if the step size is reduced by a factor of 2, the global error is reduced by a factor of 4. However to achieve this improvement, two evaluations of the derivatives is required per iteration compared to only one for the Euler method.

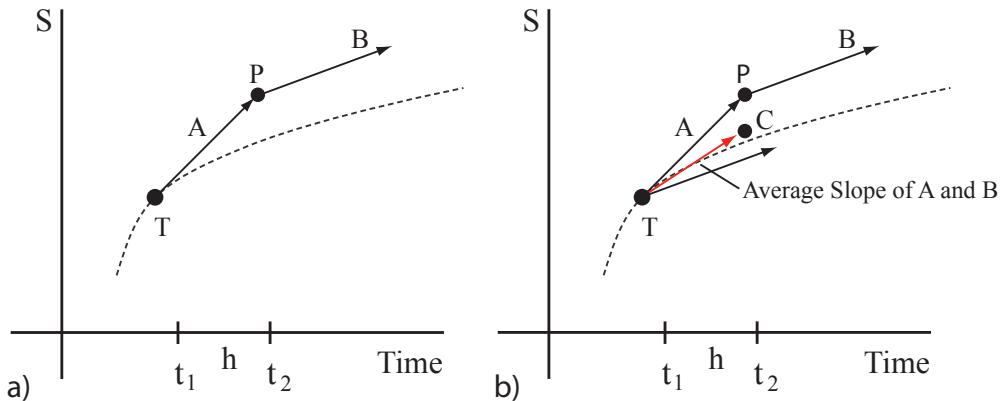
$$\begin{aligned} y(t + h) &= y(t) + h \frac{dy(t)}{dt} \\ y(t + h) &= y(t) + \frac{h}{2} \left( \frac{dy(t)}{dt} + \frac{dy(t + h)}{dt} \right) \end{aligned} \quad (5.5)$$

$$t_{n+1} = t_n + h$$

## Runge-Kutta

The Heun method described in the previous section is sometimes called the RK2 method where RK2 stands for 2nd order Runge-Kutta method. The Runge-Kutta methods are a family of methods developed around the 1900s by the German mathematicians, Runge and Kutta. In addition to the 2nd order Heun method, there are also 3rd, 4th, and even 5th order Runge-Kutta methods. For hand-coded numerical methods, the 4th order Runge-Kutta algorithm (often called RK4) is probably the most popular among modelers. The algorithm is a little more complicated because it involves the evaluation and weighted averaging of four slopes.

In terms of global error however, RK4 is considerably better than Euler or the Heun method and has a global error on the order of four. This means that halving the step size will reduce



**Figure 5.4** Heun Method. Starting at  $t_1$ , the slope A at T is computed. The slope is used to predict the solution at point P using the Euler method. From point P, the new slope, B, is computed (Panel A). Slopes A and B are now averaged to form a new slope, C (Panel B). The averaged slope is used to compute the final prediction.

the global error by a factor of 16. In other words, the step size can be increased 16 fold over the Euler method and still have the same global error. The method can be summarized by the equations (5.6) which have been simplified by removing the dependence on time.

$$\begin{aligned}
 k_1 &= h f(y_n) \\
 k_2 &= h f\left(y_n + \frac{k_1}{2}\right) \\
 k_3 &= h f\left(y_n + \frac{k_2}{2}\right) \\
 k_4 &= h f\left(y_n + k_3\right) \\
 y(t+h) &= y(t) + \frac{1}{6} \left( k_1 + 2k_2 + 2k_3 + k_4 \right) \\
 t_{n+1} &= t_n + h
 \end{aligned} \tag{5.6}$$

Figure 5.5 shows a comparison of the three methods, Euler, Heun, and RK4 in solving the Van der Pol equations. The Van der Pol systems of equations is a classic problem set often used when comparing numerical methods. The equations model an oscillating system, originally inspired from modeling vacuum tubes. At a later date it also formed the basis for developments in modeling action potentials in neurons. Figure 5.5 shows that the Heun and RK4 methods are very similar, at least for the Van der Pol equations, though this will not

always be the case. For this particular model the solution generated by the RK4 method is very similar to the best possible solution obtained numerically. Notice how bad the Euler method is in comparison.

---

**Algorithm 2** Heun Integration Method.  $f_i(y)$  is the  $i^{th}$  ordinary differential equation.

---

$n$  = Number of state variables

$y_i$  =  $i^{th}$  variable

Set timeEnd

currentTime = 0

$h$  = stepSize

Initialize all  $y_i$  at currentTime

**while** currentTime < timeEnd **do**

**for**  $i$  = 1 to  $n$  **do**

$a_i = f_i(y)$

**for**  $i$  = 1 to  $n$  **do**

$b_i = f_i(y + h a)$

**for**  $i$  = 1 to  $n$  **do**

$y_i(t + h) = y_i(t) + \frac{h}{2} (a_i + b_i)$

    currentTime = currentTime +  $h$

**end while**

---

## Variable Step Size Methods

In the previous discussion of numerical methods for solving differential equations, the step size,  $h$ , was assumed to be fixed. This makes implementation straight forward but also makes the methods inefficient. For example, if the solution is at a point where it changes very little, then the method could increase the step size without loosing accuracy while at the same time achieve a considerable speedup in processing time. Likewise, if at a certain point in the integration the solution starts to change rapidly, it would be prudent to lower the step size to increase accuracy. Such strategies are implemented in the **variable step size methods**.

The approach used to automatically adjust the steps size may be simple or very sophisticated depending on what level of performance is desired. The simplest approach is to carry out two integration trials, one at a step size of  $h$ , and another trial using two steps of size  $h/2$ . The software then compares the solutions generated by the two trials. If the solutions are significantly different, the step size must be reduced. If the solutions are about the

---

**Algorithm 3** 4th Order Runge-Kutta Integration Method.

---

$n$  = Number of state variables

$y_i$  =  $i^{\text{th}}$  variable

timeEnd = 10

currentTime = 0

$h$  = stepSize

Initialize all  $y_i$  at currentTime

**while** currentTime < timeEnd **do**

**for**  $i = 1$  to  $n$  **do**

$k_{1i} = hf(y_i)$

**for**  $i = 1$  to  $n$  **do**

$k_{2i} = hf(y_i + k_{1i}/2)$

**for**  $i = 1$  to  $n$  **do**

$k_{3i} = hf(y_i + k_{2i}/2)$

**for**  $i = 1$  to  $n$  **do**

$k_{4i} = hf(y_i + k_{3i})$

**for**  $i = 1$  to  $n$  **do**

$y_i(t + h) = y_i(t) + \frac{h}{6} (k_{1i} + 2k_{2i} + 2k_{3i} + k_{4i})$

    currentTime = currentTime +  $h$

**end while**

---

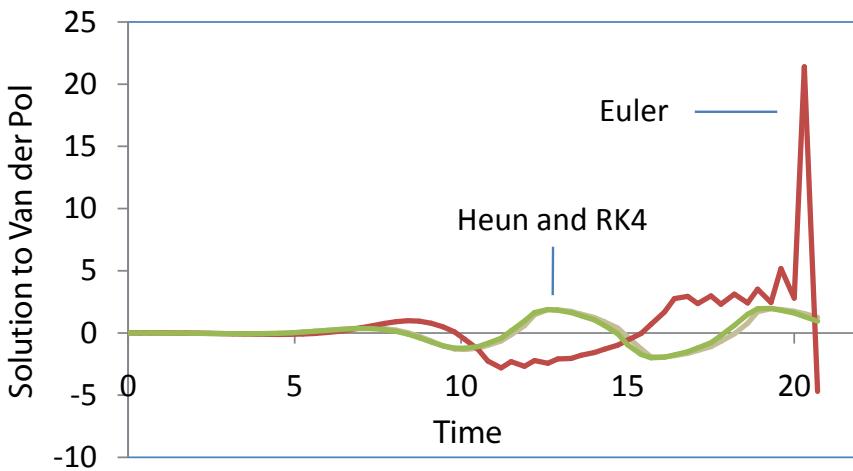
same, then it might be possible to increase the step size. These tests are repeatedly carried out, adjusting the step size as necessary as the integration proceeds. This simple variable step size approach can be easily incorporated into some of the more straightforward algorithms, particularly the fourth order Runge-Kutta which is called the variable step-size Runge-Kutta.

Another approach to adjusting the step size is called the **Dormand-Prince method** [34]. This method carries out two trials based on the fourth and fifth order Runge-Kutta. Any difference between the trials is used to adjust the step size. Matlab's ode45 implements the Dormand-Prince method. Similar methods to Dormand-Prince include the Fehlberg<sup>1</sup> and more recently the Cash-Karp method [23].

Many of these simple adjustable step size solvers are quite effective although they can be slow especially for the kinds of problem we find in biochemical models. Specifically, there

---

<sup>1</sup> [http://en.wikipedia.org/wiki/Runge-Kutta-Fehlberg\\_method](http://en.wikipedia.org/wiki/Runge-Kutta-Fehlberg_method)



**Figure 5.5** Comparison of Euler, Heun and RK4 numerical methods at integrating the Van der Pol dynamic system:  $dy_1/dt = y_2$  and  $dy_2/dt = -y_1 + (1-y_1y_1)y_2$ . The plots show the evolution of  $y_1$  in time. The RK4 solution is almost indistinguishable from solutions generated by much more sophisticated integrators. Step size was set to 0.35.

is a class of problem called stiff problems which is common in biochemical modeling. Stiff models require highly specialized solvers developed over the past four decades.

## Stiff Models

Many differential equations encountered in biochemical models are referred to as **stiff** systems. The word stiff comes from earlier studies on spring and mass systems where the springs had large spring constants and were therefore difficult to stretch. A stiff system is often associated with widely different time scales, for example when the rate constants are widely different in a biochemical model. Such systems may have molecular species whose decay rates are very fast compared to other components. This means the step size must be very small to accommodate the fast processes even though the rest of the system could be accurately solved using a much larger step size. The overall result is the need for very small steps sizes at a significant computational cost, in addition to rounding error as a result of the small step sizes. Roundoff errors in turn can be amplified by the large time constants. The net result are solutions which bear no resemblance to the true solution.

Most modern simulators will employ specific stiff algorithms for solving stiff differential equations. Of particular importance is the SUNDIALS suite [31] and ODEPACK [71]. Sundials includes a number of very useful, well written, and documented solvers. In particular, the CVODE solver is very well suited for solving stiff differential equations. Sundials is therefore widely used in the biochemical modeling community (for example by Jarnac and roadRunner). Before the advent of SUNDIALS, the main workhorse for solving stiff

systems was the suite of routines in ODEPACK. Of particular note was LSODA which in the 1990s was very popular and is still a valuable set of software (currently used in COPASI).<sup>2</sup> The original stiff differential equation solver was developed by Gear [52] in the 1970s and is still used in Matlab in the form of ode15s.

## 5.3 Matlab Solvers

---

Although this isn't a book about Matlab, it is worth mentioning how Matlab can be used to solve differential equations. Matlab offers a range of solvers with the two most commonly used being ode45 and ode15s.

The ode45 solver implements a variable step size Runge-Kutta method by using the Dormand-Prince method. The basic syntax for ode45 is:

```
[t,y] = ode45(@myModel, [t0, tend], yo, [], p);
```

where

myModel is the function containing the differential equations.

t0, tend are the initial and final values for the independent variable,  $t$ .

yo is a vector of initial conditions.

p is the set of parameters for the model, and can be any size.

The empty vector in the call is where additional options can be placed.

For example, to solve the set of ODEs:

$$\frac{dy_1}{dt} = v_o - k_1 y_1$$

$$\frac{dy_2}{dt} = k_1 y_1 - k_2 y_2$$

We would write the following .m file and load it into Matlab:

```
function dy = myModel(t, y, p)
dy = zeros (2,1);
vo = p(1);
k1 = p(2);
k2 = p(3);
dy(1) = vo - k1*y(1);
dy(2) = k1*y(1) - k2*y(2);
```

We would then call the solver as follows:

---

<sup>2</sup>In some of our own work, we have noticed that LSODA can be much faster than CVODE.

```
p = [10, 0.5, 0.35]
y0 = [0, 0]
[t, y] = ode45 (@myModel, [0, 20], y0, [], p)
```

Although many problems can be solved using `ode45`, some stiff models will fail to give the correct solution using this method. In these cases `ode15s` is recommended. `ode15s` is a variable order solver and uses the well known Gear method [52]. Like `ode45`, `ode15s` is also a variable step size method. `ode45` might be faster than `ode15s` on simple problems, but with today's fast computers the difference is not great. Therefore `ode15s` is recommended for all problems unless computing time is critical.

## 5.4 Python Solvers

---

Like Matlab, Python is a general purpose computing language. However, unlike Matlab, Python is open source and freely available for anyone to use. Python offers a variety of ODE solvers via the `scipy` package<sup>3</sup>. These include LSODA [71], an implicit Adams method [61] (for non-stiff systems), 4th order adaptive step size Dormand-Prince and an eight order adaptive step size Dormand-Prince [34]. The code below shows the Matlab code shown in the previous section expressed using Python. The example uses the default LSODA integrator.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint

vo = 10
k1 = 0.5
k2 = 0.35

# Declare the model
def myModel(y, t):

    dy0 = vo - k1*y[0]
    dy1 = k1*y[0] - k2*y[1]
    return [dy0, dy1]

time = linspace(0.0, 20.0, 100)
yinit = array([0.0, 0.0])
y = odeint (myModel, yinit, time)
```

---

<sup>3</sup><http://scipy.org/>

```
plt.plot(time, y[:,0], time, y[:,1]) # y[:,0] is the first column of y
plt.xlabel('t')
plt.ylabel('y')
plt.show()
```

## 5.5 Other Software

---

Matlab and Python aren't the only software that can be used to solved differential equations. Mathematica is an example of commercial tool that can be used to solve differential equations.

For those who require more control or who are unable to purchase a commercial tool, there are many free applications and professionally developed open source software libraries that can be used very effectively. Octave (<http://www.gnu.org/software/octave/>) is an open source tool that is very similar to Matlab. SciLab (<http://www.scilab.org/>) is another free Matlab like application. If you like programming in Python then Sage (<http://www.sagemath.org/index.html>) is a good option. There are therefore many alternative and free options to using Matlab.

For those who require much more control and higher performance, it is possible to write your own code around the SUNDIALS C/C++ library which is available under the unrestricted BSD open source licence. Within SUNDIALS is the CVODE library used by many of the commercial tools. CVODE implements an advanced Gear like algorithm using a variable order and variable step size approach. It is well suited for stiff systems and is the preferred method for those who need to write their own code. One final library worth mentioning is the GPL (GNU General Public License) licensed GSL library (<http://www.gnu.org/software/gsl/>). Although very comprehensive, the GPL license unfortunately puts critical restrictions on how the library can be used. Unless one has a real need to use the GSL library, it is recommend that one employ the unrestricted SUNDIALS suite.

### Specialized Software

Simulating biochemical networks has a long history dating back to the 1940s [24]. The earliest simulations relied on building either mechanical or electrical analogs of biochemical networks. It was only in the late 1950s, with the advent of digital computers and the development of specialized software tools [48], that the ability to simulate biochemical networks became more widely available. In the intervening years up to 1980, a handful of other software applications were developed [19, 20, 132] to help the small community of modelers. In more recent years, particularly since the early 1990s, there has been a significant increase in interest in modeling biochemical processes and a wider range of tools is now available to the budding systems biologist. Many open source tools have been developed by practicing

scientists and are therefore freely available.

In this book we will be using the author's modeling tool Tellurium [157]. Tellurium is well suited for our purpose. It is a script based modeling application which makes it easy to illustrate a modeling exercise.

Many tools do not offer readable text based renderings of models because they use either a visual approach to modeling, such as PathwayDesigner [10] or CellDesigner [89], or have a graphical user interface such as COPASI [75] or iBiosim [123]. All these tools export and import the standard modeling language SBML (See section G.1). However, because SBML is written in XML, it is also difficult to display a model using SBML in a textbook.

## Tellurium

Tellurium [157] is an integrated Python based environment for modeling in systems biology. The current version (July 2014) integrates a number of libraries including libRoadRunner (Simulator), libSBML (SBML support), libAntimony (Antimony support) and SBML2-Matlab (SBML to Matlab converter). In addition Tellurium distributes a number of standard Python packages such as Matplotlib (plotting) and NumPy (array support). All packages are integrated using spyder2 (<https://code.google.com/p/spyderlib/>) which offers a Matlab like experience for modelers.

Visually, Tellurium has two main windows (Figure H.1): a console where commands can be issued and results returned, and an editor where control scripts and models are written. The application also has a plotting window which is used when graphing commands are issued.

Tellurium uses Antimony to let users describe biochemical pathways and Python coupled with libRoadRunner to do simulations and other analyses. Models can also be imported or exported as SBML. Many other capabilities are offered through libRoadRunner including support for metabolic control analysis, structural analysis of networks, and stochastic simulation. It has no explicit support for fitting as of yet. A more detailed description of Tellurium is given in Appendix H. The following code shows the model we used previously expressed using Tellurium:

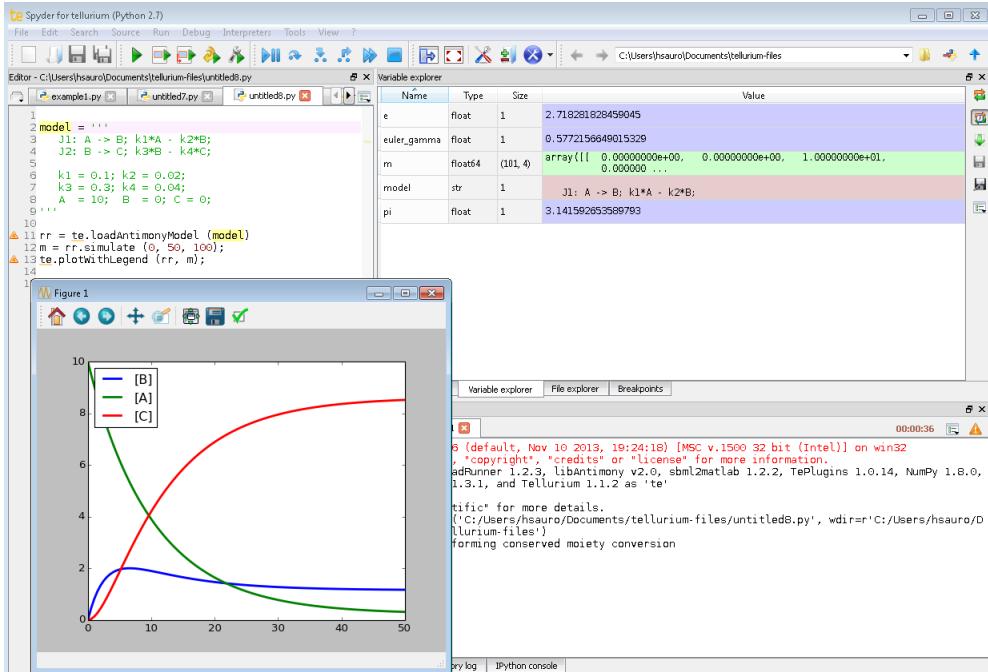
```
import tellurium as te

r = te.loada ('''
$Xo -> y1; vo;
y1 -> y2; k1*y1;
y2 -> $waste; k2*y2;

vo = 10; k1 = 0.5; k2 = 0.35;
y1 = 0; y2 = 0;
''' )
```

```
m = r.simulate (0, 20, 100);
r.plot();
```

The first part of the code shows the model expressed using Antimony and loaded into roadrunner while the second part show two commands to simulate and plot the results via libRoadRunner.

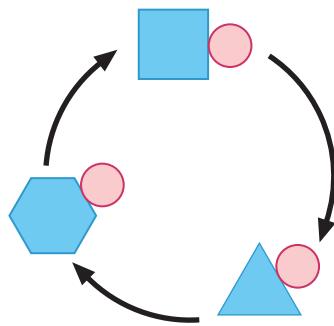


**Figure 5.6** Screen shot of Tellurium with simulation results.

## 5.6 Moiety Conserved Cycles

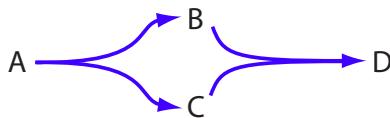
Any chemical group that is preserved during a cyclic series of interconversions is called a conserved moiety (See section 3.8), Figure 5.7. Examples of conserved moiety subgroups include species such as phosphate, acyl, nucleoside groups, or covalently modifiable proteins. As a moiety gets redistributed through a network, the *total amount* of the moiety is constant and does not change during the time evolution of the system. For any particular subgroup, the total amount is determined solely by the initial conditions imposed on the model.

There are rare cases when a ‘conservation’ relationship arises out of a non-moiety cycle. This does not affect the mathematical analysis, but only the physical interpretation of the



**Figure 5.7** Conserved Moiety in a Cyclic Network. The blue species (larger symbols) are modified as they traverse the reaction cycle, but the red subgroup (small circle) remains unchanged. This creates a conserved cycle where the total number of moles of moiety (small circle, red subgroup) stays constant.

conservation relationship. For example, in Figure 5.8 the constraint  $B - C = T$  applies even though there is no moiety involved.



**Figure 5.8** Conservation due to stoichiometric matching. In this system,  $B - C = \text{constant}$ .

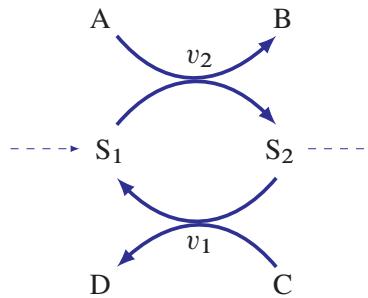
The presence of conserved moieties is an approximation introduced into a model, however, over the time scale in which the conservation may hold, their existence can have a profound effect on the dynamic behavior of the model. For example, the hyperbolic response of a simple enzyme (in the form of enzyme conservation between E and ES) or the sigmoid behavior observed in protein signalling networks is due in significant part to moiety conservation laws [102, 78].

Figure 5.9 illustrates the simplest possible network which displays a conserved moiety. The total mass,  $S_1 + S_2$ , is constant during the evolution of the network.

The system equations for the simple conserved cycle are easily written as:

$$\frac{dS_1}{dt} = v_1 - v_2$$

$$\frac{dS_2}{dt} = v_2 - v_1$$



**Figure 5.9** Simple Conserved cycle. The dotted lines signify negligible levels of synthesis and degradation. Thus, over short time scales,  $S_1 + S_2 = \text{constant}$ .

From these equations it should be evident that the rate of appearance of  $S_1$  must equal the rate of disappearance of  $S_2$ , that is  $dS_1/dt = -dS_2/dt$ . This means that whenever  $S_1$  changes,  $S_2$  must change in the opposite direction by *exactly* the same amount. During a simulation the sum of  $S_1$  and  $S_2$  will therefore remain unchanged. This is a characteristic of a moiety-conserved cycle.

Computationally, we only need to explicitly evaluate one of the differential equations because the other can be computed from the conservation relation. The system can therefore be reduced to one differential and one linear algebraic equation compared to the two differential equations in the original formulation.

$$S_2 = T - S_1$$

$$\frac{dS_1}{dt} = v_1 - v_2$$

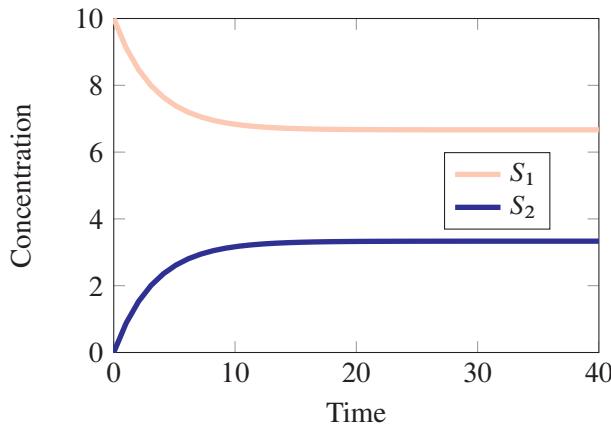
The term  $T$  in the algebraic equation shown above refers to the total amount of  $S_1$  and  $S_2$ . This value is computed from the initial amounts of  $S_1$  and  $S_2$  at the start of a simulation.

The conservation can be seen in the stoichiometry matrix as linear dependencies among the matrix rows. Let us look at an example where there are dependencies between the rows. Consider the cyclic pathway shown in Figure 5.11 with the corresponding stoichiometry matrix shown in equation (5.7).

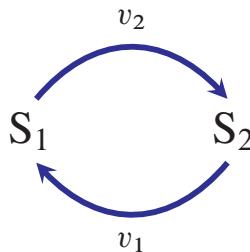
$$\mathbf{N} = \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} \quad (5.7)$$

Note that there is one row dependency in the stoichiometry matrix. Multiplying the second row by -1 gives the first row.

What this means is that given the amount for either  $S_1$  or  $S_2$ , it is possible to compute



**Figure 5.10** Simulation of the simple cycle shown in Figure 5.9. The total moiety remains constant at 10 concentration units. Model:  $S_1 \rightarrow S_2$ ;  $k_1 \cdot S_1$ ;  $S_2 \rightarrow S_1$ ;  $k_2 \cdot S_2$ ;  $S_1 = 10$ ;  $k_1=0.1$ ;  $k_2=0.2$ .



**Figure 5.11** Simple cycle.

the other. That is, the total mass in the cycle is fixed,  $S_1 + S_2 = T$ . Conservation constraints such as these can have profound effects on network behavior [109]. In addition, they affect certain numerical procedures<sup>4</sup> and should be eliminated by computer software whenever possible [153]. Most modern software apply this operation before proceeding to solve model equations. For certain types of analysis, such as computing the steady state, bifurcation analysis, and certain optimization methods, eliminating the redundant species is critical. These topics will be covered in more detail in a separate book.

## 5.7 Exploiting Fast Processes

Chapter 4 discussed some of the simplifications that are often made when we construct a computer model. One simplification involves aggregating reaction steps. Typical examples

<sup>4</sup>In particular, the Jacobian matrix becomes singular thereby preventing the calculation of Bifurcation curves and computing the steady state.

include the use of Michaelis-Menten or Hill rate type kinetics. In the majority of these cases, the implicit assumption is that the processes inside the aggregate are much faster than processes outside. As noted already in this chapter, numerical instabilities can arise when a model has very different times scales, where some parts of the model are much slower or faster than other parts. There are different ways to take advantage of fast processes to simplify models; here we will consider two.

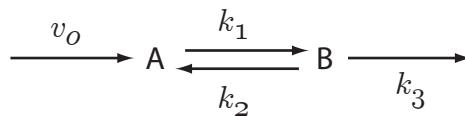
### Equilibrium Assumption

The first approach assumes that a reaction with fast forward and reverse rates is always very close to equilibrium. We can illustrate this with a simple example. Consider the pathway shown in Figure 5.12. The two differential equations for this system are:

$$\frac{dA}{dt} = v_o - k_1 A \left( 1 - \frac{\Gamma}{K_{eq}} \right)$$

$$\frac{dB}{dt} = k_1 A \left( 1 - \frac{\Gamma}{K_{eq}} \right) - k_3 B$$

The rate law representing the middle reaction between A and B is a modification of the usual mass-action rate law,  $k_1 A - k_2 B$ , where  $k_2$  has been replaced by the equilibrium constant,  $K_{eq}$ , and the mass-action ratio,  $\Gamma$ . See section 2.6 for the derivation.



**Figure 5.12** Fast reaction sandwiched between two slower reactions.

Let us assume that the middle reaction between A and B is very fast, that is,  $k_1$  and  $k_2$  are much larger compared to  $v_o$  and  $k_3$ . In this situation the middle reaction can be considered to be very close to equilibrium. In other words, the mass-action ratio  $\Gamma$  approaches the equilibrium constant such that the ratio of A and B is largely tied to the equilibrium constant. Rather than integrating A and B individually, we can define the dynamics of the system in terms of a new variable, the total,  $T = A + B$ , which can change. Part of the justification for this is that any change in T will result in an equal proportional change to A and B (See Exercise).

The differential equation for the total,  $T$ , can be obtained by summing the two separate differential equations:

$$\frac{dT}{dt} = \frac{d(A + B)}{dt} = v_o - k_3 B$$

Note that the concentration of B (and A) is no longer a state variable and must be computed separately from the equilibrium ratio and total. Since we are assuming that A and B are in

equilibrium, we can compute the equilibrium value for B given the relations:

$$\begin{aligned} K_{eq} &= \frac{B}{A} \\ A + B &= T \end{aligned}$$

such that when combined, we get the following:

$$B = T \frac{K_{eq}}{1 + K_{eq}}$$

Once we have computed B, A can be computed using  $T - B$ . To implement this numerically, we solve the following set of equations:

$$B = T \frac{K_{eq}}{1 + K_{eq}}$$

$$A = T - B$$

$$\frac{dT}{dt} = v_o - k_3 B$$

The equilibrium models can now be compared with the full model. When  $k_2$  and  $k_3$  are large, we expect the equilibrium approximation to closely match the full model. The Tellurium script shown in Listing 5.1 implements two models, one using the equilibrium assumption and another which does not. Both models are run at the same time to compare them.

```
import tellurium as te
import pylab

# Comparing the full model with an approximation
# based on the equilibrium assumption
r = te.loada ('''
    // Model using the equilibrium assumption
    // Note the use of := which represents a simulation rule
    B := T*Keq/(1+Keq);
    A := T - B;
    $s -> T; vo - k3*B;

    // The full model
    $s -> Af; vo;
    Af -> Bf; k1*Af - k2*Bf;
    Bf -> $w; k3*Bf;

    T = 10;
```

```

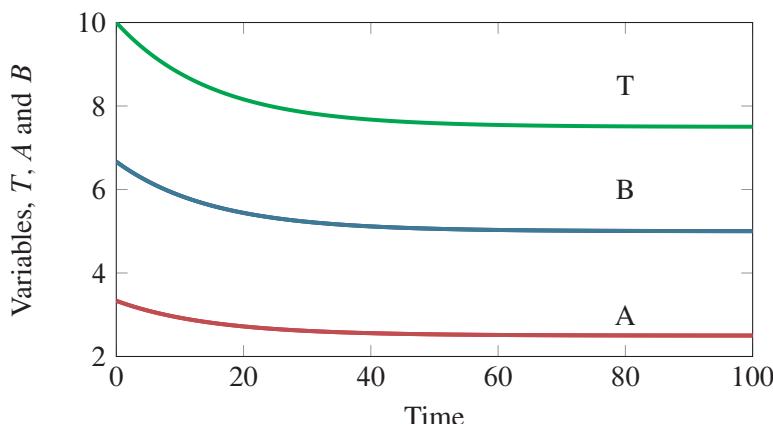
Af = 3.33333; Bf = 6.66666;
Keq = 2; vo = 0.5;
k3 = 0.1; k1 = 1;
k2 = k1/Keq
''')

result = r.simulate(0, 100, 200, ["time", "Af", "Bf", "A", "B", "T"])
r.plot(ylim=(0,10), xlim=(0,100))

```

**Listing 5.1** Script for Figure 5.12.

Figure 5.13 shows the results of running the Tellurium script (Listing 5.1) assuming that the middle reaction is so fast, we can treat it as if it were constantly very close to equilibrium. Notice that the two model simulations are almost indistinguishable. Only three lines are shown, the top curve is the total,  $T$ . The middle line plots species B and is in fact two overlapping lines, one for the approximation *and* another for the full model. The bottom line follows species A and again consists of two overlapping lines. In this case the equilibrium assumption can be used without compromising accuracy in the simulation.



**Figure 5.13** Simulation for model in Figure 5.12 where the rate constants,  $k_1$  and  $k_2$  are high and equal 1000 and 500, respectively. In this case both models coincide indicating that the approximation is good. Upper curve is T, middle curve B, and lower curve A. Tellurium model in Listing 5.1.

What if the middle reaction is not very fast, but comparable to the other steps in the pathway? In this situation we can no longer assume it is at equilibrium so if we run the simulation, we obtain the graphs shown in Figure 5.14. Interestingly, the concentration of B is

unaffected. This is because B is independent of  $k_1$  and  $k_2$  since at steady state:

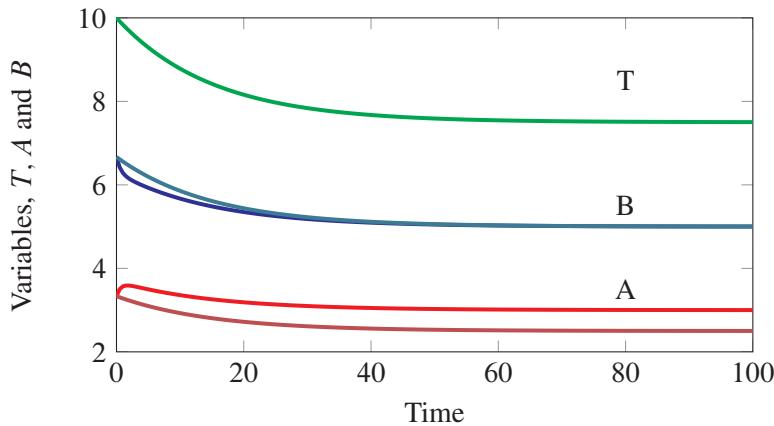
$$\frac{dA}{dt} = v_0 - Ak_1 + Bk_2 = 0$$

$$\frac{dB}{dt} = Ak_1 - Bk_2 - Bk_3 = 0$$

Solving for the steady state solution for A using  $dA/dt = 0$  and inserting this into the second equation,  $dB/dt = 0$ , yields:

$$B_{ss} = \frac{v_0}{k_3}$$

In contrast, A as computed by the approximation, diverges from the expected trajectory. Two important points are worth making. The first is that if we use the equilibrium approximation inappropriately, the time trajectories diverge. More problematic is that the final steady state value for A is also different. The equilibrium approximation should therefore be used carefully.



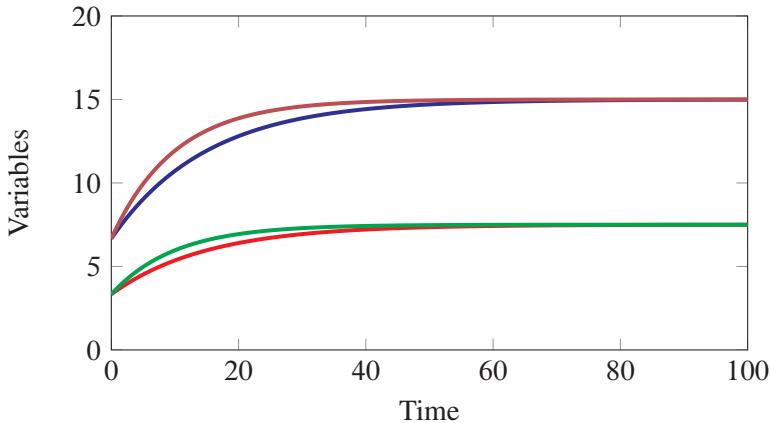
**Figure 5.14** A rerun of the simulation shown in Figure 5.13 but this time we slowed down the middle reaction so we no longer assume it is in equilibrium. Notice the concentration of A diverges from the expected solution so that the approximation is no longer valid.  $k_1 = 1; k_2 = 0.5$ . Curves from the top, T, B (equilibrium solution), B (true solution), A (equilibrium solution), B (true solution). Tellurium model in Listing 5.1.

### Quasi-Steady-State Assumption

Another way to model fast reactions is to use the **quasi-steady-state assumption** instead of the equilibrium assumption. In this approximation we assume that the dynamics of A is

fast compared to B. For example, the processes that affect A maybe faster than those that affect B. We can express the rate of change of A in Figure 5.12's model:

$$\frac{dA}{dt} = v_o - Ak_1 + Bk_2$$



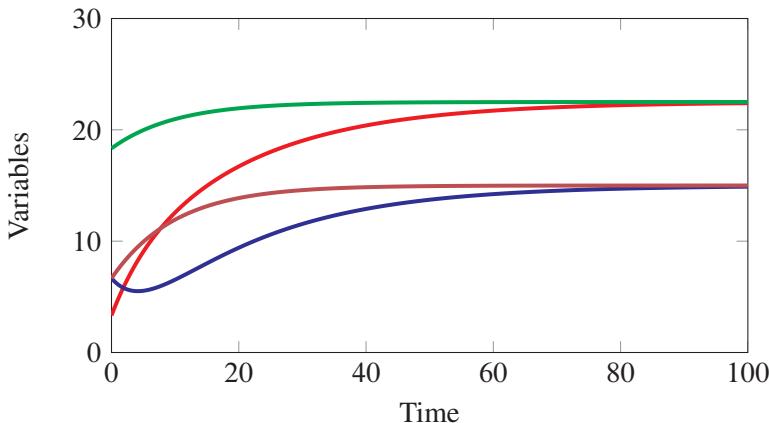
**Figure 5.15** Simulation of the model shown in Figure 5.12 assuming the quasi-steady-state assumption for A. Note that the trajectories converge, a characteristic when using the steady state assumption. From the top, curves represent: B (quasi-steady-state solution), B (true solution), A (quasi-steady-state solution), A (true solution).  $k_1 = 1000$ , Tellurium model in Listing 5.2.

```
import tellurium as te
import pylab

# Comparing the full model with an approximation
# based on the quasi-steady-state assumption
r = te.loada ('''
    Ass := (vo + k2*B)/k1;
    $s -> B; vo - k3*B
    $s -> Af; vo;

    Af -> Bf; k1*Af - k2*Bf;
    Bf -> $w; k3*Bf;

    B = 6.66666;
    Af = 3.33333; Bf = 6.66666;
    vo = 1.5;
    k3 = 0.1;
    // Use k1 = 1000 to obtain a better approximation
    k1 = 0.1;
    Keq = 2;
```



**Figure 5.16** Same simulation as in Figure 5.15 except that the reactions surrounding A are too slow to assume a quasi-steady-state approximation. In this case we see considerable divergence in the transients but still see correct convergence at steady state. From the top, curves represent: A (quasi-steady-state solution), B (quasi-steady-state solution), B (true solution), A (true solution).  $k_1 = 0.1$ , note that  $k_2 = k_1/K_{eq}$ , Tellurium model in Listing 5.2.

```

k2 = k1/Keq;
''')

result = r.simulate(0, 100, 200, ["time", "Af", "Bf", "Ass", "B"])
r.plot(ylim=(0,30), xlim=(0,100))

```

**Listing 5.2** Script for Figure 5.12.

Let us assume that A reaches steady state much faster than B such that we can set the equation  $dA/dt$  to zero and solve for A. We call this assumption the **quasi-steady state assumption**.<sup>5</sup> The steady state solution to A is:

$$A_{ss} = \frac{v_o + k_2 B}{k_1}$$

Given that the rate of change of B is:

$$\frac{dB}{dt} = Ak_1 - Bk_2 - Bk_3$$

we can rewrite this equation by inserting the steady state concentration,  $A_{ss}$  to obtain:

$$\frac{dB}{dt} = \frac{v_o + k_2 B}{k_1} k_1 - Bk_2 - Bk_3$$

<sup>5</sup>Those familiar with Michaelis-Menten kinetics will have seen this approximation used in deriving the Briggs-Haldane relationship.

which upon simplification reduces to the remarkably simpler solution:

$$\frac{dB}{dt} = v_o - k_2 B$$

To model this system we need to solve the two equations:

$$A_{ss} = \frac{v_o + k_2 B}{k_1}$$

$$\frac{dB}{dt} = v_o - k_2 B$$

In Figure 5.15 where the reactions are fast enough that the quasi-steady state assumption is reasonable, we see that both simulations follow each other closely. In contrast, Figure 5.16 shows that when the reaction reactions are too slow, the quasi-steady state assumption breaks down and both simulations diverge quite considerably. In both cases however, all trajectories coverage to the *same* steady state. This highlights one of the characteristics of the quasi-steady state assumption; the steady state is faithfully reproduced when assuming quasi-steady-state. This also suggests that using Michaelis-Menten like kinetics when the quasi-steady-state assumption is applied is reasonable, especially if the modeler is only interested in the final steady state.

## Further Reading

---

The two most popular books on numerical analysis are by Press and Burden and are included here for reference. Both books can be purchased second-hand at reasonable prices. The content in the Press book has not changed significantly between editions so any edition is useful. The main problem with the Press book is that the source code has very strict licensing rules making the code difficult to reuse for projects. However, the book is excellent at explaining how various algorithms work and for this reason alone it is worth purchasing.

1. Burden and Faires (2010) Numerical Analysis. Brooks Cole, 9th Edition. ISBN-10: 0538733519
2. Ingalls B (2013) Mathematical Modeling in Systems Biology: An Introduction, MIT Press. ISBN: 978-0262018883
3. Pahle (2009) J Biochemical simulations: stochastic, approximate stochastic and hybrid approaches. *Briefings in Bioinformatics* 10(1), 53-64
4. Reich, J.G. & Sel'kov, E.E. (1981) Energy metabolism of the cell. A theoretical treatise. Academic Press, London, ISBN-13: 978-0125859202
5. Press, Teulolsky, Vetterling and Flannery (2007) Numerical Recipes. Cambridge University Press, 3rd Edition ISBN-10: 0521880688

6. Sauro, HM, and Bergmann FT (2009) Software Tools for Systems Biology in Systems Biomedicine: Concepts and Perspectives, edited by Edison T. Liu, Douglas A. Lauffenburger. ISBN 978-0-12-372550-9

For the budget conscious buyer I highly recommend the Dover edition:

1. Dahlquist and Björck (2003) Numerical Methods. Dover Publications ISBN-10: 0486428079

---

## Exercises

---

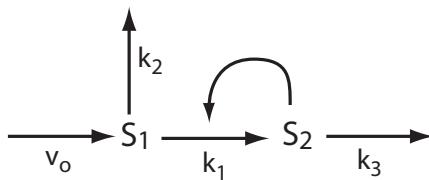
1. Implement the Euler method in your favorite computer language and use the code to solve the following two problems. Set initial conditions:  $S_1 = 10, S_2 = 0$ . Set the rate constants to  $k_1 = 0.1; k_2 = 0.25$ . Investigate the effect of different steps sizes,  $h$ , on the simulation results.
  - a)  $dS_1/dt = -k_1 S_1$
  - b)  $dS_1/dt = -k_1 S_1; dS_2/dt = k_1 S_1 - k_2 S_2$
2. In section 4.4 a model of two water tanks with water flowing from one tank to another was described. The model comprised of two differential equations that described how the heights of water in each tank changed in time. Given the equations for the model, enter them into a software tool of your choice to answer the following questions:
  - a) Plot the rate of outflow,  $Q_2$ , as a function of the height of water,  $h_1$  at a given resistance,  $K_1$ .
  - b) Assuming that  $Q_1$  and  $Q_4$  are fixed and we start with both tanks empty, what do you expect to happen over time as water flows in?
  - c) Write out the differential equations (ODEs) that describe the rate of change in the tank water levels,  $h_1$  and  $h_2$ .
  - d) Build a computer model of the tank system. Assign suitable values to the parameters in the model and run the simulation to plot the height of water in the tanks over time. Assume both tanks are empty at time zero.
  - e) Investigate the effect of increasing and decreasing the resistance parameters,  $K_1$  and  $K_2$  on the model.
3. The following model shows oscillations in  $S_1$  and  $S_2$  at a step size of  $h = 0.044$  when using the Euler method. By using simulation, show that these oscillations are in fact an artifact.

```

cell = '''
$Xo -> S1; k1*Xo;
S1 -> S2; k2*S1;
S2 -> $X1; k3*S2;

Xo = 10; S1 = 0; S2 = 0;
k1 = 23.4; k2 = 45.6; k3 = 12.3;
'''
```

4. Find out what differential equation solvers the Python SciPy Package supports.
5. Construct a model of the following system using Tellurium.



Let the reaction associated with the positive feedback ( $k_1$ ) be governed by the following rate law:

$$k_1 S_1 (1 + c S_2^q)$$

All other reactions are governed by first-order kinetics except the first reaction which has a constant rate of  $v_o$ . Set the constants to the following values:  $v_o = 8; c = 1.0; k_1 = 1; k_2 = 5; k_3 = 1$  and  $q = 3$ . Study the effect of changing  $v_o$  on the dynamics of the system.

6. Download the model BIOMD0000000010 from Biomodels and load it into Tellurium. Run a simulation of the model.
7. Given a system at equilibrium,  $A \rightleftharpoons B$ , with equilibrium constant,  $K_{eq}$ , and total mass in the system to be  $T = A + B$ , show that a change  $\delta T$  in the total results in equal proportional changes to A and B.
8. Using the model in Figure 5.12, compare the full model with an approximation that assumes the middle reaction has higher rate constants than the rate constants in the surrounding reactions.



# 6

## *Stochastic Models*

### **6.1 Stochastic Kinetic Models**

---

So far we have learned how to run simulations of continuous models described using ordinary differential equations. However, as mentioned in Chapter 4, there is a strong case in some situations to model reaction systems using a discrete stochastic approach.

It has been shown [38, 128] experimentally that stochasticity is endemic in gene regulatory networks, particularly in prokaryotic organisms where the number of transcription factors can be in the low tens of copies or fewer. The stochasticity in protein expression is a result of the many molecular events that occur between transcription and translation. For example, when the number of transcription factors (or RNA polymerase) is very small, the random binding and unbinding to the operator and promoter sites leads to random transcription events such that transcription acts in an on/off manner with *bursts* of mRNA production followed by periods of silence. This is called the random telegraph model [93]. Translation itself contributes to stochasticity given that the number of mRNA transcripts may be small. A single mRNA strand can result in many proteins being produced before it degrades. The kinetics of protein synthesis may therefore be quite different compared to what is described by a simple deterministic model.

Individual reaction events can play a dominant role in determining the evolution of the system. Since the time at which a reaction occurs is a random event, simulating reaction events becomes stochastic. In addition, since we're dealing with individual reaction events, we must keep track of exact molecule numbers rather than a number representing a continuous concentration value. The companion book, ‘Enzyme Kinetics for Systems

Biology' [149], provides a formal and informal description of the theory and algorithms behind stochastic simulations. Here we will only cover the formal description. The most common approach to stochastic simulation is to use the Gillespie method for simulation which we will describe here.

## 6.2 Stochastic Kinetics

### Basic Definitions

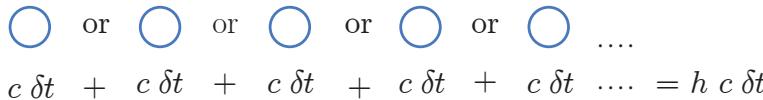
In stochastic reaction kinetics the symbol  $c\delta t$  is often used to describe the probability that a reactant molecule will react in the next time interval,  $\delta t$ . This means that  $c$  (without the  $\delta t$ ) is the average probability that a reactant will react per unit time, and can be considered a probability or stochastic rate constant.

$c\delta t$  = the average probability that a particular reactant molecule will react in the next time interval,  $\delta t$ , where  $c$  is the average probability that a reactant molecule will react per unit time. If the reaction is second or third-order, then  $c\delta t$  refers to the probability of a particular *combination* of reactant molecules reacting in the next time interval,  $\delta t$ .

Note that the  $c\delta t$  refers to a particular reactant combination that can react. For a first-order reaction, a single molecule can react, or for a second-order reaction, two molecules must meet to react. When we have a population of molecules,  $c\delta t$  must be multiplied by the total number of reactant combinations in order to obtain the probability of a reaction occurring within the entire population in  $\delta t$ . For a simple first-order reaction, the number of combinations is the number of reactant molecules. Thus, the probability that a reaction will occur in a population in the next time interval,  $\delta t$ , is:

$$hc\delta t$$

where  $h$  is the number of distinct molecular reactant combinations, or in the case of a first-order reaction, the actual number of reactants (Figure 6.1). To illustrate this further, let's say we have four reactant molecules and they react via a second-order reaction. The number of ways (combinations) the four pairs can meet up is  $4 \times 4 = 16$ . Therefore  $h = 16$ .



**Figure 6.1** Number of combinations for a simple first-order reaction. Upper row represents individual molecules.

$h$  = the number of distinct molecular reactant combinations for a given reaction at time  $t$ .

$hc\delta t$  = the probability that a reaction will occur in a population of molecules in the next time interval,  $\delta t$ .

For reactions other than first-order,  $h$  is more complicated. For a second-order reaction involving two distinct species, say X and Y, the number of combinations is  $XY$ . For a second-order reaction that is a dimerization of identical molecules, X, the number of combinations is  $X(X - 1)/2$ . For a zero-order reaction, the number of combinations is one. Table 6.1 gives a list of different reactions and the corresponding number of combinations.

Reaction	Number of reactant combinations ( $h$ )
$\rightarrow X$ (zero order)	1
$X \rightarrow Y$	$x_a$
$X + Y \rightarrow$	$x_a y_a$
$X + X \rightarrow$	$x_a (x_a - 1)/2$
$X + Y + Z \rightarrow$	$x_a y_a z_a$
$X + 2Y \rightarrow$	$x_a y_a (y_a - 1)/2$
$3X \rightarrow$	$x_a(x_a - 1)(x_a - 2)/6$

**Table 6.1** Number of combinations ( $h$ ) for different elementary reactions.  $x_a$ ,  $y_a$ , and  $z_a$  represent the number of molecules.

The term  $hc$  is analogous to the deterministic reaction rate. In the stochastic literature it is often called the **propensity function** or the average rate.  $hc$  itself is not a probability since its value can exceed one, and its units are molecules per unit time. As indicated before, the term  $c$  is analogous to the deterministic rate constant and is sometimes called the **stochastic rate constant**.

The average rate for a first-order reaction is therefore given by:

$$x_a c$$

where  $h$  has been substituted with  $x_a$  (Table 6.1).

### Relationship of $c$ to Deterministic Rate Constants

For a first-order reaction the deterministic rate,  $v$ , is given by:

$$v = kX$$

where  $X$  is the concentration of molecule X. In the following calculations we must recall that for a species with a given concentration,  $X$ , in a given volume,  $V$ , the number of molecules is represented as:

$$x_a = X N_A V$$

where  $x_a$  is the number of molecules and  $N_A$  is Avogadro's number. With this in mind, the rate of reaction in molecules per unit time is given by:

$$k X N_A V = k x_a$$

Since the stochastic rate is also in units of molecules per unit time,  $c x$ , this means:

$$c x_a = k x_a \quad \therefore \quad c = k$$

In other words for a first-order reaction, the deterministic rate constant and the stochastic rate constant are identical.

For a second-order reaction a similar analysis yields a different result. Consider a second-order reaction involving two dissimilar molecules, X and Y. The deterministic rate is given by:

$$v = k X Y$$

where X and Y are the concentration of species X and Y, respectively. The rate in terms of molecules per unit time is given by:

$$k N_A X V Y = k x_a Y = k x_a \frac{y_a}{N_A V}$$

where  $x_a$  and  $y_a$  are the molecule numbers for species X and Y. Note that  $y_a = Y N_A V$ , that is  $Y = y_a / (N_A V)$ . From Table 6.1 we know that the average rate of a stochastic process in molecules per unit time is  $c x_a y_a$ , therefore:

$$c x_a y_a = k x_a \frac{y_a}{N_A V}$$

and thus:

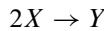
$$c = \frac{k}{N_A V}$$

Note that the stochastic rate constant is inversely related to the volume. This makes sense because the larger the volume, the less likely a molecule of X and one of Y will meet.

### **Example 6.1**

---

What is the relationship between the deterministic and stochastic rate constant for the irreversible reaction:



The deterministic rate law is given by  $v = kX^2$  where  $X$  is the concentration of species X. The rate in terms of molecules per unit time is given by:

$$kN_AXVX = kx_aX = kx_a^2/(N_AV)$$

From Table 6.1 we see the stochastic rate is given by  $cx_a(x_a - 1)/2$  so that

$$kx_a^2/(N_AV) = cx_a(x_a - 1)/2$$

Therefore:

$$c = \frac{2kx_a}{(x_a - 1)N_AV}$$

For large  $x_a$ , the term  $(x_a - 1)$  can be approximated by  $x_a$  so that:

$$c = \frac{2k}{N_AV}$$

Although the previous definitions give us the probability of a reaction occurring in a  $\delta t$  window, they do not tell us when a reaction is likely to occur. That is, if we start a clock at time zero, when is a reaction likely to take place? This question is addressed in the next two sections.

## 6.3 Time to Reaction

Assume we have ten molecules than can undergo a decomposition reaction. How can we describe this system knowing that individual molecules will react at random times? If we start a clock at time,  $t_0$  (Figure 6.2), at what point in the future might one of the molecules react?

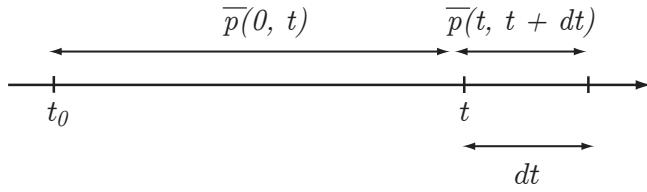
Consider a time line starting at time zero that extends into the future (Figure 6.2). With this in mind, what is the chance of a reaction occurring in the interval time  $t + dt$  if we started monitoring the system at time zero?

To answer the question it is easier to first ask the opposite question. What is the probability that a reaction will *not* occur in both intervals,  $0 \rightarrow t$  and  $t$  to  $t + dt$ ? To answer, let us split the question into two parts. In order for there to be no reaction in the interval  $0 \rightarrow t + dt$ , no reactions should occur in the two subintervals  $0 \rightarrow t$  and  $t \rightarrow t + dt$ .

Let us designate the probability of no reaction in the first interval by  $\bar{p}(0, t)$ , and for the second interval,  $\bar{p}(t, t + dt)$ . Since we assume independent events, the probability of no reaction in the entire interval,  $0 \rightarrow t + dt$  is the product:

$$\bar{p}(0, t + dt) = \bar{p}(0, t) \bar{p}(t, t + dt)$$

To continue further we can expand the second term  $\bar{p}(t, t + dt)$  and assume that the reaction occurs at a propensity of  $a$ . Over a time interval  $T$ , the reaction will occur  $aT$  times. If



Probability of no event occurring between 0 to  $t + dt$  =

$$\bar{p}(0, t) \times \bar{p}(t, t + dt)$$

**Figure 6.2** Stochastic time line.

the interval is made shorter by dividing it up into  $N$  subintervals, the chance of a reaction occurring in any one of these intervals is  $aT/N$ . If we make the time interval small enough, say  $dt$ , then the probability of a reaction occurring in  $dt$  time will be  $adt$ . The probability of a reaction *not* occurring in this interval is therefore  $1 - adt$ . We can now write the probability equation as:

$$\bar{p}(0, t + dt) = \bar{p}(0, t) (1 - adt)$$

Rearranging gives:

$$\bar{p}(0, t + dt) - \bar{p}(0, t) = -\bar{p}(0, t) adt$$

Dividing both sides by  $dt$ , we obtain the differential equation:

$$\frac{d\bar{p}(0, t)}{dt} = -\bar{p}(0, t) a$$

Now solve for  $\bar{p}(0, t)$  by integrating the differential equation and using the observation that at time zero, no reaction has occurred so  $\bar{p}(0) = 1$ :

$$\bar{p}(0, t) = e^{-at}$$

Recall this is the probability of a reaction *not* occurring in the interval 0 to  $t$ . That is, if we let  $t$  increase,  $\bar{p}(0, t)$  tends to zero; that is the longer we wait, the more likely the reaction will occur. The probability of a reaction occurring at  $t$ ,  $p(t)$ , is the probability of it *not* occurring in the interval 0 to  $t$ , times the probability that the reaction *will* occur in the interval  $t$  to  $dt$ :

$$p(t)dt = \bar{p}(0, t) p(t, t + dt) = \bar{p}(0, t)adt$$

That is:

$$p(t) = ae^{-at} \tag{6.1}$$

This equation describes a **probability distribution function** (pdf) for  $t$  (note that  $p(t)$  is positive and the area under the curve is one). As with all pdfs, the area under the curve is the probability. For a given probability we could in principle sweep out the corresponding area,

left to right, and locate the time on the  $x$ -axis where the sweep ends. It is simpler however to integrate the equation (6.1) to generate the cumulative probability function,  $P(t)$ , which is the probability as a function of time. Integration of (6.1) yields:

$$P(t) = 1 - e^{-at} \quad (6.2)$$

Equation (6.2) can be rearranged so that  $t$  is on the left-hand side:

$$t = -\frac{1}{a} \ln(1 - P(t)) \quad (6.3)$$

We can now assign a uniform random number to  $P(t)$  and compute a possible time when the reaction will occur. If we did this repeatedly, the times would be distributed exponentially. Since the value of  $P(t)$  is drawn from a uniform random number distribution, the distribution of numbers in  $1 - P(t)$  is no different from  $P(t)$ . Therefore we can simplify equation (6.3) and rewrite it as:

$$t = -\frac{1}{a} \ln(P(t))$$

Using the log addition rule, remove the negative sign and rearrange the equation to:

$$t = \frac{1}{a} \ln\left(\frac{1}{P(t)}\right)$$

or more simply, there  $P(t)$  is replaced with  $r$ :

$$t = \frac{1}{a} \ln\left(\frac{1}{r}\right) \quad (6.4)$$

This is the equation that is often presented in the literature but for computational purposes, the former is more efficient as it avoids the division operation.

$$t = -\frac{1}{a} \ln(r) \quad (6.5)$$

In practice, to simulate a simple decomposition reaction, we would first generate a uniform random number, set it to  $P(t)$ , and compute the corresponding time,  $t$ . After this, we reduce the number of reactant molecules by one to signify a decomposition has occurred. We'd also increment our time line by  $t$ , then continue by computing another  $t$  from the equation. The reaction rate (or propensity function),  $a$ , must be recomputed for each iteration. Note that the smaller the propensity  $a$ , the larger the  $t$  computed from the equation. This makes sense because a lower reaction rate corresponds to fewer reactants and since there are fewer reactants, we will likely have to wait longer before another reaction occurs. Likewise, if there are many reactants, the reaction rate is likely to be higher and correspondingly, the time until the next reaction shorter. If there are large numbers of reactants, say in the

thousands, this simulation method is quite slow because the time intervals at each iteration will be very small, requiring many steps for the simulation to evolve over time.

Equations (6.4) and (6.5) describe part of a well known algorithm called the Gillespie Stochastic Simulation Algorithm or the **Gillespie SSA** for short [56, 55]. The method described here is also called the **Direct Method** because it calculates the time to the next simulation directly. Other implementations include the First Reaction Method [56] and the more efficient Next Reaction Method [54], however these are beyond the scope of this chapter. An excellent article that describes all three approaches is provided by McCollum *et al.* [113].

The full Gillespie SSA can also address systems with multiple reactions. The extension from one to multiple reactions is surprisingly simple and will be described briefly in the following section.

## 6.4 Running Stochastic Simulations

---

There are many variants on the Gillespie method, some are faster and some are approximate. The review by Pahle [129] covers many of these variants and is well worth consulting. There are a number of software tools that support stochastic simulation, examples include COPASI [75], Dizzy [141] and Jarnac [150]. For Python users StochPy<sup>1</sup> is highly recommended and comprehensive, offering many facilities for stochastic simulation. Of these, COPASI, Jarnac, and StochPy have releases in 2014 and actively support SBML or text based languages such as Jarnac or PySCeS. Manninen *et al.* [106] reviews some of these software tools. In the deterministic world we have a multitude of easy to use software libraries (such as sundials and odepak), but libraries for stochastic solvers are virtually nonexistent. One possible option is StochKit<sup>2</sup>. Although quite comprehensive, using the StochKit library is not easy, certainly not as simple as the differential equation solvers. There is an interesting blog<sup>3</sup> by Mario Pineda-Krhc that describes his experience with using StochKit.

Given the theory presented in the last section, we summarize the basic Gillespie algorithm as:

- 
1. Initialize  $t = t_0$ ,  $x = x_0$ ,  $c$ , and end of simulation time,  $t_{max}$
  2. Compute  $h$  depending on the nature of the reaction (Table 6.1)
  3. Evaluate propensity function,  $a = hc$
  4. Draw one uniform random number,  $r_1$
  5. Determine the time,  $\tau$  when the next reaction will take place using:

---

<sup>1</sup><http://stochpy.sourceforge.net/>

<sup>2</sup><http://engineering.ucsb.edu/~cse/StochKit/>

<sup>3</sup><http://www.r-bloggers.com/vanilla-c-code-for-the-stochastic-simulation-algorithm/>

$$\tau = -\frac{1}{a} \ln(r_1)$$

6. Determine the new state:  $t = t + \tau$  and  $x = x - 1$
  7. Reached end of simulation,  $t > t_{max}$ ?
  8. No, got to step 2
  9. Yes, finished
- 

#### **Algorithm for Simulating the Stochastic Kinetics of a Single Reaction.**

Extending the Gillespie SSA to systems with multiple reactions is straight forward. In addition to computing when a reaction will fire, the full Gillespie SSA also considers *which* reaction will fire. If we have a system of  $v$  reactions and wish to determine which reaction will fire, first sum up the propensity functions,  $h_{v_i} c_{v_i}$ :

$$R = \sum_{i=1}^v h_{v_i} c_{v_i}$$

Now normalize each propensity function dividing by  $R$ :

$$n_i = h_i c_i / R$$

Intuitively the method arranges the normalized propensities in the form of a pie chart, where the size of an individual pie wedge,  $n_i$ , is equal to the size of the normalized propensity. Now imagine throwing a dart at the pie chart. Whatever pie wedge the dart lands on is the reaction to be fired in the next iteration. This means larger pie wedges are more likely to be selected than thinner ones, or in other words, reactions with larger propensities are more likely to fire than those with smaller propensities. A possible algorithm for implementing this scheme is shown below (written in a pseudo code). The code works its way through each pie wedge. The function `uniformRandomNumber()` returns a uniform random number between zero and one. In this code the reactions are numbered from one, but it could easily be modified to index reactions from zero if deemed more convenient.

```
r = uniformRandomNumber()
Reactions index from 1 to numberOfReactions
Propensities index from 1 to numberOfReactions

if r <= n(1) then
    Reaction[1] fires
else
    begin
```

```

for i = 2 to number0fReactions do
    if (r > n[i-1]) and (r <= n[i]) then
        begin
            Reaction[i] fires
        exit
    end
end

```

Although the above algorithm is relatively easy to understand, it is not very efficient. A more efficient alternative is given below. This version doesn't normalize the propensities but instead scales the random number to the sum,  $R$ , of the propensities, then walks through the propensity list until the scaled random number exceeds the accumulating propensity value. This method avoids normalizing every propensity value and instead only involves simple additions. This code could be easily implemented in a language such as Python ([www.python.org](http://www.python.org)).

```

accumulate = 0
i = 0
scaledRandomNumber = uniformRandomNumber()*R
while accumulate < scaledRandomNumber do
    begin
        i = i + 1
        accumulate = accumulate + n[i]
    end;
Fire the ith reaction

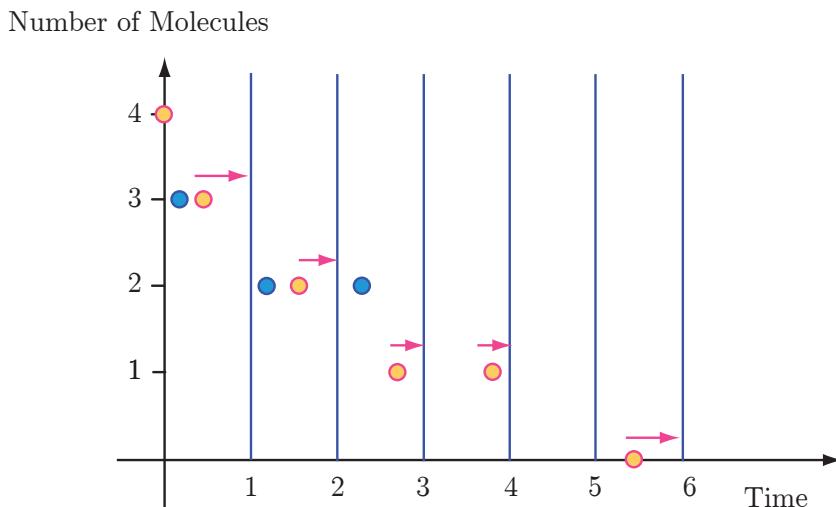
```

## 6.5 Events at Regular Intervals

---

Equation (6.5) describes how time intervals for reaction events are computed. The intervals are irregular due to the selection of a uniform random number each time a time interval is computed. Generating data on an irregular grid can however be inconvenient. It is possible to modify the algorithm so than the time axis increases in a more regular fashion. Figure 6.3 illustrates how stochastic events at irregular intervals can be arranged to lie on a regular time grid. The method works by moving simulated events nearest a vertical grid line to the grid line itself. Other reaction events are ignored. Reaction events nearest a grid line can be moved to the grid line because we know with certainty that there are no events between the nearest event and the grid line, hence at the grid line, the number of molecules will correspond to the nearest event in the past. This way an irregular set of events can be realigned onto a regular grid.

An example will make this more clear. The rows of data (Table 6.2) correspond to data generated from the Gillespie SSA Algorithm starting with 20 molecules. Note that the time intervals are irregular as expected from the Gillespie SSA.



**Figure 6.3** Technique for generating stochastic events on a regular grid. Simulated points nearest a grid line (light markers) are moved (arrow) to the nearest grid line in the future. Other data points (dark markers) are ignored. Such a technique is useful when averaging a large number of trajectories. The average number of molecules can be calculated at each grid point.

A regular grid is set to an interval of 0.5 and the raw data shown in Table 6.2 is grided. For example at grid point 1.0, the event closest to this grid point in the past is the reaction event that occurred at time 0.81, corresponding to 9 molecules. We therefore assert that at time 1.0, there were 9 molecules remaining. This method is applied to each grid point, with the results summarized in Table 6.3.

## 6.6 Stochastic Trajectories

One crucial point regarding the Gillespie SSA method is that a trajectory produced from one simulation represents only one of many possible trajectories. If we were to run the same simulation a second time, we would observe a slightly different but related trajectory. This is the nature of stochastic processes. Figure 6.4 shows ten repeated simulations of exactly the same system, with the same initial conditions and stochastic rate constant. As one can see from the graph, each trajectory is slightly different. It is possible to combine these trajectories into a single averaged trajectory. To do this easily we arrange the output from the simulation onto a regular grid.

Once on a grid, it is now easy to compute the average and variance of a series of repeated simulations. As an example, the trajectories shown in Figure 6.4 are redrawn in Figure 6.6

Time	0	0.052	0.1	0.16	0.21	0.24	0.31
Molecules	20	19	18	17	16	15	14
Time	0.41	0.5	0.52	0.56	0.81	1.2	1.47
Molecules	13	12	11	10	9	8	7
Time	1.58	1.59	1.71	2.22	2.75	3.42	4.84
Molecules	6	5	4	3	2	1	0

**Table 6.2** Data from a Gillespie SSA run.

Time	0	0.5	1.0	1.5	2.0	2.5	3.0	4.0	5.0
Molecules	20	12	9	7	4	3	2	1	0

**Table 6.3** Result of gridding data from Table 6.2.

using a grid interval of 1.0.

```
import tellurium as te

r = te.loada ('''
    $X0 -> S1; k1*X0;
    S1 -> S2; k2*S1;
    S2 -> $X1; k3*S2;

    X0 = 50; S1 = 0; S2 = 0;
    k1 = 0.2; k2 = 0.4; k3 = 2;
''')

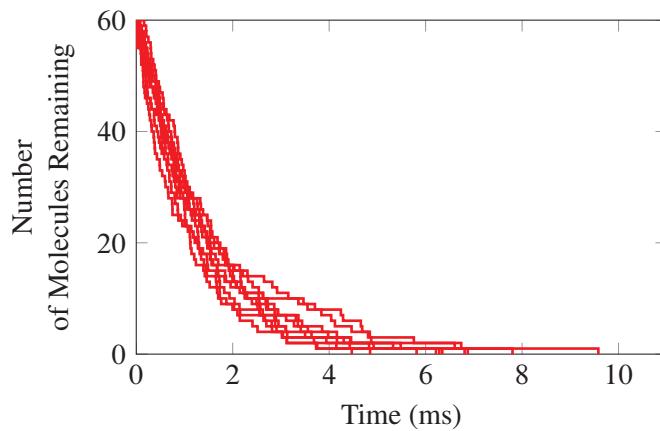
result = r.gillespie (0, 30)
r.plot()
```

**Listing 6.1** Script for Figure 6.7.

Listing 6.1 shows a Tellurium script that generated the plots shown in Figure 6.7. The key line in the script is:

```
m = r.gillespie (0, 30)
```

This takes two arguments. The first and second arguments set the time start and time end for the simulation. There is an optional third argument which can be used to set a fixed time step (output is then on a grid) and an optional forth argument which sets the columns in the matrix that will be returned. Note that species amounts have been set to integer values because we are now dealing with discrete molecules.



**Figure 6.4** Multiple simulations of the same irreversible decomposition reaction,  $X \rightarrow Y$  showing different trajectories. The stochastic rate constant is  $0.3 \text{ ms}^{-1}$ . Initial number of molecules is 60.

---

```

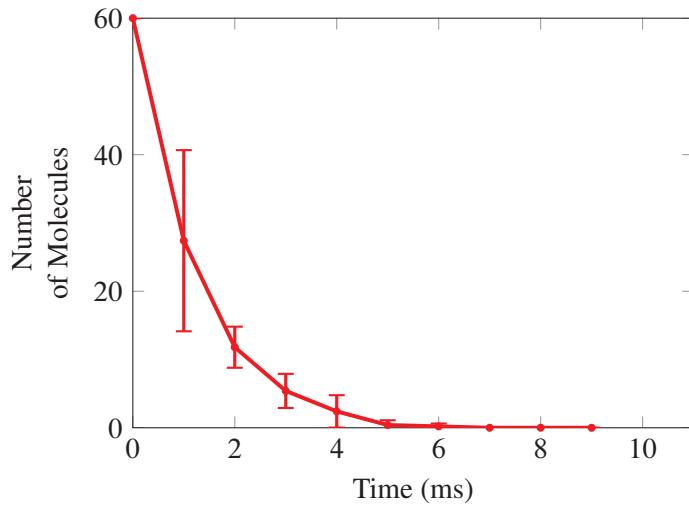
start = 0; dt = 1;
// Retrieve number of rows in input matrix
events = rows (m);
endTime = m[events,1];
// Compute length of the output matrix, out
len = trunc(endTime - start)/dt + 1;
out = matrix (len, 2);
gridTime = 0;
j = 1;

for i = 2 to events do
begin
while m[i,1] >= gridTime do
begin
out[j,1] = gridTime;
out[j,2] = m[i-1,2]
j = j + 1;
gridTime = gridTime + dt;
end;
end;

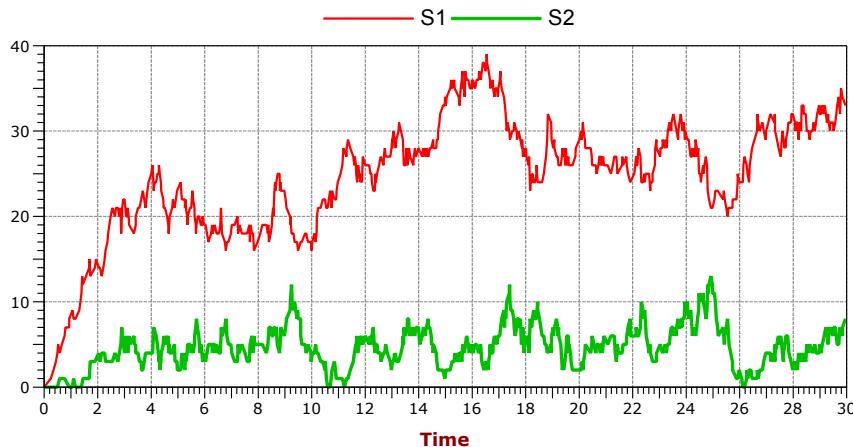
```

---

**Figure 6.5** Gridding code: Modified from Wilkinson, Stochastic Modeling for Systems Biology, Figure 6.10, ISBN: 978-1584885405. The code accepts a matrix,  $m$  from a Gillespie SSA run, where the first column is time and the second column the number of molecules. Indexing of matrices is from 1.



**Figure 6.6** Mean of 10 trajectories together with the standard deviation indicated around each time point. Computed using a grid size of 1.0.



**Figure 6.7** Stochastic simulation using Tellurium. Upper curve S<sub>1</sub>, lower curve S<sub>2</sub>

As with continuous simulations, it is possible to carry out a number of separate runs where other events are imposed in between the runs. For example, we might want to decrease one of the rate constants by a factor of six at a certain time point in the simulation and then carry the simulation on as before. Listing 6.2 shows one simulation being carried out from 0 to time 30. At time 30 one of the rate constants is decreased six fold, then the simulation is started up again, but this time setting the time start to the end time of the previous simulation. Finally, both matrices from the two runs are merged and the entire simulation plotted.

```
import tellurium as te
import numpy

r = te.loada ('''
$Xo -> S1; k1 * Xo;
S1 -> S2; k2*S1;
S2 -> $X1; k3*S2;

Xo = 50; S1 = 0; S2 = 0;
k1 = 0.2; k2 = 0.4; k3 = 2;
''')

m1 = r.gillespie(0, 30)
r.k1 = r.k1/6
m2 = r.gillespie(30, 60)

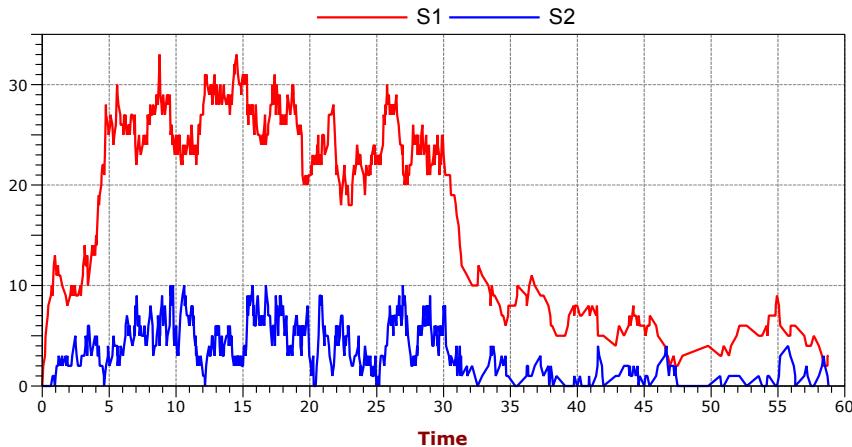
# Merge the two data sets
result = numpy.vstack((m1, m2))
te.plotWithLegend(result)
```

**Listing 6.2** Script for Figure 6.8

## Further Reading

In recent years a small number of books have emerged and are geared towards stochastic modeling for the average modeler. Brian Ingall's new book Mathematical Modeling in Systems Biology: An Introduction, is a very good start, followed by the 2nd edition of Wilkinson's book Stochastic Modeling for Systems Biology. The text Stochastic Approaches for Systems Biology by Ullah and Wolkenhauer covers other areas such as experimental aspects in cellular noise. Both the Ingalls and Ullah books are quite reasonably priced. Wilkinson's book is more mathematically orientated, but I recommend all three.

1. Ingalls B (2013) Mathematical Modeling in Systems Biology: An Introduction, MIT Press. ISBN: 978-0262018883



**Figure 6.8** Stochastic simulation using Tellurium (Script 6.2) showing how an event can be superimposed between two consecutive simulations. Upper curve S<sub>1</sub>, lower curve S<sub>2</sub>.

2. Wilkinson D (2011) Stochastic Modeling for Systems Biology 2nd Edition, Chapman & Hall/CRC Mathematical & Computational Biology (Book 44), ISBN: 978-1439837726
3. Ullah M and Wolkenhauer O (2011) Stochastic Approaches for Systems Biology, Springer, ISBN: 978-1461404774

---

## Exercises

---

1. Write a Python script to implement the Gillespie direct method.
2. Modify the Python script to allow simulations to be generated on a regular grid. Run multiple trajectories and compute the average trajectories using the grid data.
3. The deterministic rate constant for the reaction  $2X \rightarrow$  is equal to  $0.5 \text{ mM}^{-1} \text{ s}^{-1}$ . If the volume of the compartment in which the reaction takes place is  $10 \text{ mm}^3$ , what is the value for the equivalent stochastic rate constant?

# 7

## *How Systems Behave*

### 7.1 System Behavior

---

How do systems behave and how does that behavior come about? As we proceed through the book we will encounter many different types of system behavior. At this stage however, it is worth describing the states that are fundamental to all systems. These states fall into three groups: **(Thermodynamic) equilibrium, steady state, and transients**. In the literature the terms equilibrium and steady state are often used interchangeably, but here they will describe two very different states.

The simplest and arguably the least interesting state is equilibrium, or more precisely, thermodynamic equilibrium.

### 7.2 Equilibrium

---

Thermodynamic equilibrium, or simply equilibrium, refers to the state of a system when all forces are balanced. In chemistry, thermodynamic equilibrium is when all forward and reverse rates are equal. This also means that the concentration of chemical species are unchanging and all net fluxes are zero. Equilibrium is easily achieved in a closed system. For example, consider the simple chemical isomerization:



Let the net forward rate of the reaction,  $v$ , be equal to  $v = k_1 A - k_2 B$ . The rates of change of A and B are given by:

$$\frac{dA}{dt} = -v \quad \frac{dB}{dt} = v$$

The equilibrium constant for this system is:

$$K_{eq} = \frac{B_{eq}}{A_{eq}} = \frac{k_1}{k_2}$$

At equilibrium,  $dA/dt$  and  $dB/dt$  equal zero, that is  $Ak_1 = Bk_2$ , or  $v = 0$ . We can derive the analytical solution for A and B as follows. Given that the system is closed, we know that the total mass in the system,  $A + B$ , is constant. This constant is given by the sum of the initial concentrations of A and B which we define as  $A_o + B_o$ . Note that  $A_o + B_o = A(t) + B(t)$  is always true. We assume the volume is constant and set to unit volume, allowing us to state that the sum of concentrations is conserved. The differential equation for A is given by:

$$\frac{dA}{dt} = k_2 B - k_1 A$$

Before solving this equation, let us replace B by the term  $A_o + B_o - A$ . This yields:

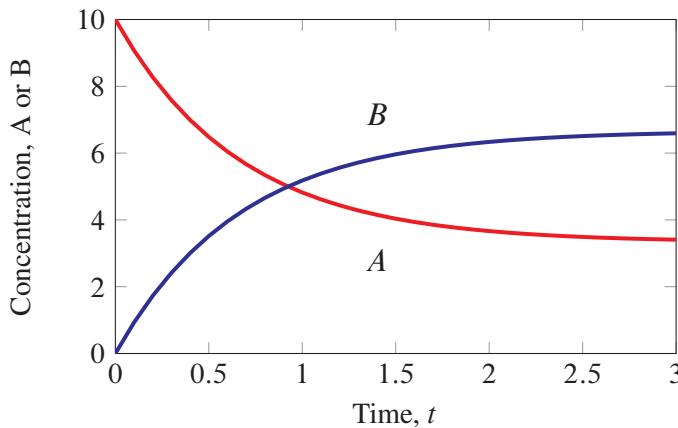
$$\frac{dA}{dt} = k_2 A_o + k_2 B_o - k_2 A - k_1 A = k_2(A_o + B_o) - A(k_1 + k_2)$$

The easiest way to solve this equation is to use Mathematica or Maxima. The Mathematica command is `DSolve[A'[t] == k2 (Ao + Bo) - A[t] (k1 + k2), A[0] == Ao, A[t], t]`, where  $A[0] == Ao$  sets the initial condition for the concentration of A to be  $A_o$ . By implication, the initial condition for  $B_o$  is  $(A_o + B_o) - A_o = B_o$ . The result of applying the Mathematica command yields the following solution:

$$A(t) = \frac{(A_o + B_o)k_2}{k_1 + k_2} + \frac{e^{-(k_1+k_2)t} v_{\text{initial}}}{k_1 + k_2}$$

The first term on the right-hand side of the equation is independent of time and equals the equilibrium concentration of A. This term is also a function of the total mass in the system ( $A_o + B_o$ ), which means that the equilibrium solution is *independent* of the starting concentrations so long as the total remains the same. Starting conditions such as  $A_o = 1; B_o = 9$  or  $A_o = 6; B_o = 4$  will lead to the same equilibrium concentrations.

The second term is time dependent and describes the evolution of the system when the initial concentrations of A and B are not set to the equilibrium concentrations. The initial concentrations are set in the term  $v_{\text{initial}}$  which is the reaction rate,  $v$ , at  $t = 0$ . The second term also has an exponential component which approaches zero as time goes to infinity. Given this, at infinite time we are left with the first term which equals the concentration of A when  $dA/dt = dB/dt = 0$ .



**Figure 7.1** Time course for equilibration of the reversible reaction in model 7.1 where  $k_1 = 1$ ,  $k_2 = 0.5$ ,  $A_o = 10$ ,  $B_o = 0$ . The ratio of the equilibrium concentration is given by  $k_1/k_2$ . Tellurium Listing: 11.1.

At equilibrium the reaction rate can be computed by substituting the equilibrium concentration of A and B into the reaction rate,  $v = k_2B - k_1A$ . Note that the equilibrium concentration of A is given by:

$$A_{eq} = \frac{(A_o + B_o)k_2}{k_1 + k_2}$$

and for B, by subtracting  $A_{eq}$  from  $A_o + B_o$ . When the  $A_{eq}$  and  $B_{eq}$  relations are substituted into  $v$ , the result is:

$$v = 0$$

From this long-winded analysis, it has been determined for the closed reversible system, at infinite time, the concentrations of A and B reach some constant values and that the net rate,  $v$  is zero. Note also that the ratio of the final concentration for A and B equals the equilibrium constant. The system is therefore at thermodynamic equilibrium.

In biochemical models it is often assumed that when the forward and reverse rates for a particular reaction are very fast compared to the surrounding reactions, the reaction is said to be in **quasi-equilibrium**. That is, although the entire system may be out of equilibrium, there may be parts of the system that can be approximated as though they were in equilibrium. This is often done to simplify the modeling process.

Living organisms are not themselves at thermodynamic equilibrium; if they were, they would technically be dead. Living systems are open so that there is a continual flow of mass and energy across the system's boundaries.

## 7.3 Steady State

The **steady state**, also called the stationary state, is where the rates of change of all species,  $dS/dt$ , are zero while at the same time the net rates are non-zero, that is  $v_i \neq 0$ . This situation can only occur in an open system, one capable of exchanging matter with the surroundings.

### Thermodynamic Equilibrium versus Steady State

Thermodynamic equilibrium (or equilibrium for short) and the steady state are distinct states of a chemical system. If we consider a system where every part is in equilibrium, we can be sure of two things: the species concentrations are not changing, and there are no net flows of mass or energy within the system or between the system and the environment. A system in equilibrium must therefore have the following properties:

$$\text{for all } i: \frac{dS_i}{dt} = 0$$

$$v_i = 0$$

where  $v_i$  is the net reaction rate for the  $i^{\text{th}}$  reaction step. When a biological system is at equilibrium, we say it is dead. Thermodynamically we can also say that entropy production is at zero and has reached its *maximum value*.

The steady state has some similarities with the equilibrium state. Species concentrations are still unchanging, however *there are net flows* of energy and mass within the system and with the environment. Systems at steady state must therefore be open and must continuously dissipate any gradients between the system and the external environment. This means that one or more  $v_i$ 's must be non-zero.

The steady state is defined when all  $dS_i/dt$  are equal to zero while one or more reaction rates are non-zero:

$$\text{for all } i: \frac{dS_i}{dt} = 0$$

$$v_i \neq 0$$

Thermodynamically, we can also say that entropy production of the system at steady state is lower than the entropy production in the environment. In some of the literature the terms equilibrium and steady state are used interchangeably resulting in possible confusion. In this book, the word equilibrium will be used to refer to a system at thermodynamic equilibrium, not at steady state.

To convert the simple reversible model described in the last section into an open system,

we need only add a source reaction and a sink reaction as shown in the following scheme:



In this case simple mass-action kinetics is assumed for all reactions. It is also assumed that the source reaction, with rate  $v_o$ , is irreversible and originates from a boundary species,  $X_o$ , where  $X_o$  is *fixed*. In addition, it is assumed that the sink reaction with rate constant  $k_3$ , is also irreversible. For the purpose of making it easier to derive the time-course solution, the reverse rate constant,  $k_2$  will be assumed to equal zero. We will also set the initial conditions for A and B to both equal zero. The mathematical solution for the system can again be obtained using Mathematica:

$$A(t) = v_o \frac{1 - e^{-k_1 t}}{k_1}$$

$$B(t) = v_o \frac{k_1 (1 - e^{-k_3 t}) + k_3 (e^{-k_1 t} - 1)}{k_3 (k_1 - k_3)}$$
 (7.3)

As  $t$  tends to infinity,  $A(t)$  tends to  $v_o/k_1$ , and  $B(t)$  tends to  $v_o/k_3$ . In addition, the reaction rate through each of the three reaction steps tends to  $v_o$ . This is confirmed by substituting the solutions for A and B into the reaction rate laws. Given that  $v_o$  is greater than zero and that A and B reach constant values within sufficient time, we conclude that this system eventually settles to a steady state rather than thermodynamic equilibrium.

The system displays a continuous flow of mass from the sink to the source. This can only continue undisturbed so long as the source material,  $X_o$ , never runs out, and that the sink is continuously emptied. Figure 7.2 shows a simulation of this system.

At steady state, the rate of mass transfer across a reaction is called the flux, or  $J$ .

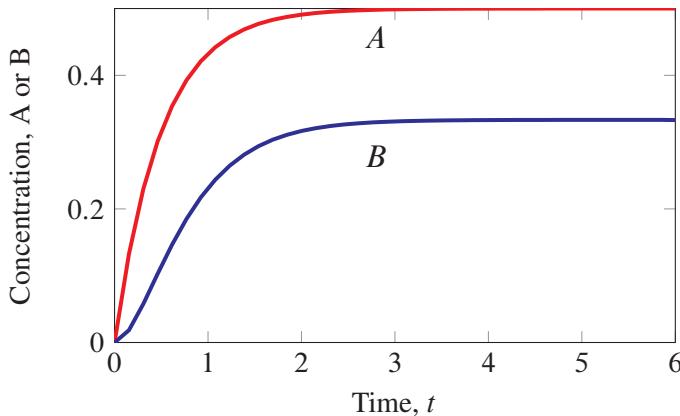
In some cases we can calculate the steady state in a different way. For example, in Figure 7.2 we used the simplified model:



The differential equations for this system are:

$$\frac{dA}{dt} = v_o - k_1 A$$

$$\frac{dB}{dt} = k_1 A - k_3 B$$



**Figure 7.2** Time course for an open system reaching steady state in model 7.4 where  $v_o = 1, k_1 = 2, k_2 = 0, k_3 = 3, A_o = 0, B_o = 0$ .  $X_o$  is assumed to be fixed. The Tellurium model: 11.2.

Now set the rates of change to zero:

$$\begin{aligned} 0 &= v_o - k_1 A \\ 0 &= k_1 A - k_3 B \end{aligned}$$

With two equations and two unknowns, A and B, we can solve for A and B to obtain:

$$\begin{aligned} A &= v_o / k_1 \\ B &= v_o / k_3 \end{aligned}$$

In most cases we cannot solve the equations because they will be nonlinear and so must revert to computer simulation or specialist software (such as Tellurium) to compute the steady state. The script below shows a model in Tellurium where we solve for the steady state using the command `p.ss.eval`.

```
import tellurium as te

rr = te.loada ('''
$Xo -> A;    vo;
A -> B;    k1*A;
B -> $X1; k3*B;

// Set up the model initial conditions
Xo = 1; X1 = 0;
vo = 0.5; k1 = 0.2; k3 = 0.3;
''')
```

```
# Evaluate the steady state
# Evaluate returns a number indicating how far we are
# from the steady state solution. A number less than 1E-6
# is a good indicator that it has found the steady state.
rr.steadyState()
print rr.A, rr.B

# Output follows:
Steady State values:  2.5  1.66667
```

Another special characteristic of steady states is that they can be classified as either stable or unstable. We will revisit this concept in much more detail in later chapter. Suffice to say that stable steady states are those where transients converge on to the steady state, while an unstable steady state is where transients diverge. We will talk more about the properties of the steady state in Chapter 11.

## 7.4 Transients

---

Another simple behavior that a system can show is a transient. A transient is usually the change that occurs in the species concentrations as the system moves from one state, often a steady state, to another. Equation (7.3) shows the solution to a simple system that describes the transient behavior of species A and B. Figure 7.2 illustrates the transient from an initial condition, in this case from a non-steady state condition, to a steady state. A periodic (such as an oscillation) or a chaotic system may be considered a transient, one that is unable to settle to a fixed steady state. In the case of a system showing periodic behavior, the transient repeats itself indefinitely at regular intervals. In a chaotic system, the transient never repeats the exact same trajectory and will continue indefinitely.

## 7.5 Setting up a Model in Software

---

There are many software tools both free (including open source) and commercial that one can use to build models of cellular networks. In this book we use Tellurium [115], a software tool written by the author and collaborators. As we have seen Tellurium is a script-based tool where one enters a model as a text file. The model is then compiled, run, and the results displayed. Because Tellurium is based on Python more advanced users can develop their own sophisticated analysis. A brief introduction on how to use Tellurium is given in Appendix H. For those who wish to use other tools such as COPASI (<http://www.copasi.org>), CellDesigner (<http://celldesigner.org/>), or even Matlab (<http://www.mathworks.com>), it is easy to convert Tellurium files into standard Systems Biology Markup Language (SBML) or Matlab scripts (See Appendix H) so that models can be loaded into the simulation tool of choice.

## 7.6 Robustness and Homeostasis

---

Biological organisms are continually subjected to perturbations. These perturbations can originate from external influences such as changes in temperature, light, or the availability of nutrients. Perturbations can also arise internally due to the stochastic nature of molecular events, or by natural genetic variation. One of the most remarkable and characteristic properties of living systems is their ability to resist such perturbations and maintain very steady internal conditions. For example, the human body can maintain a constant core temperature of  $36.8^{\circ}\text{C} \pm 0.7$  even though external temperatures may vary widely. The ability of a biological system to maintain a steady internal environment is called **homeostasis**, a phrase introduced by Claude Bernard almost 150 years ago. Modern authors may also refer to this behavior as **robustness**, although this word is used in many other contexts.

There are a number of mechanisms used in biology to maintain homeostasis. Perhaps the most common is negative feedback. This is where the difference between the desired output, and the actual output is used to modulate the process that determines the output. For example, if the output is lower than the desired output then the process will increase the output. Such systems are found at multiple levels in a living organism, including subcellular processes such as metabolism and multicellular processes that control, for example, the level of glucose in the blood stream. The field of control and regulation in biochemical systems is large and growing, and the topic will be reserved for a separate book.

## Further Reading

---

1. Klipp E, Herwig R, Kowald A, Wierling C and Lehrach H (2005) Systems Biology in Practice, Wiley-VCH Verlag.
2. Steuer R, Junker BH (2008) Computational Models of Metabolism: Stability and Regulation in Metabolic Networks, Advances in Chemical Physics, Volume 142, (ed S. A. Rice), John Wiley & Sons, Inc.
3. Stucki JW (1978) Stability analysis of biochemical systems—a practical guide. *Prog Biophys Mol Biol.* 33(2):99-187.

## Exercises

---

1. Describe the difference between thermodynamic equilibrium and a steady state.
2. Write out the differential equations for the system  $A \rightarrow B \rightarrow C$  where the reactions

rates are given by:

$$\begin{aligned}v_1 &= k_1 A - k_2 B \\v_2 &= k_3 B - k_4 C\end{aligned}$$

Find the concentrations of A, B, and C when the rates of change are zero:  $dA/dt = 0$ ,  $dB/dt = 0$ ,  $dC/dt = 0$ . Show that this system is at thermodynamic equilibrium when the rates of change are zero.

3. What do we mean by the phrase quasi-equilibrium?
4. Find the mathematical expression that gives the steady state levels of A and B in the following network:



Assume that  $X_o$  is fixed, and that all reactions are governed by simple mass-action kinetics.

5. Consider the following model, use a software tool of your choice to visualize the time evolution for the following system, simulate for 5 time units. At time zero, set  $x = 1$  and  $y = 2$ . Simulate for 30 time units.

$$\begin{aligned}\frac{dx}{dt} &= -0.25x - 0.4y \\ \frac{dy}{dt} &= 0.5x + 1.5y\end{aligned}$$

Given the model from the previous question, compute the steady state in two ways:  
1) Simulating the model for a very long time; 2) Determine algebraically the steady state. Compare the two solutions.

6. Given the model from the previous question, explore how perturbations in  $x$  and  $y$  at steady state behave.
7. Use a software tool of your choice to visualize the time evolution for the following system, simulate for 5 time units.

$$\begin{aligned}\frac{dx}{dt} &= 2.55x - 4.4y \\ \frac{dy}{dt} &= 5x + 2.15y\end{aligned}$$

## Appendix

---

See Appendix H for more details of Tellurium.

```
// Simulation of a simple closed system
import tellurium as te

# Simulation of a simple closed system
r = te.loada ('''
    A -> B; k1 * A;
    B -> A; k2 * B;

    A = 10; B = 0;
    k1 = 1; k2 = 0.5;
''')

result = r.simulate(0, 6, 100)
r.plot()
```

**Listing 7.1** Script for Figure 7.1.

```
// Simulation of an open system
import tellurium as te

# Simulation of an open system
rr = te.loada ('''
    $Xo -> S1; vo;
    S1 -> S2; k1*S1 - k2*S2;
    S2 -> $X1; k3*S2;

    vo = 1
    k1 = 2; k2 = 0; k3 = 3;
''')

result = rr.simulate(0, 6, 100)
r.plot()
```

**Listing 7.2** Script for Figure 7.2.

# 8

## *Multicompartment Systems*

---

### 8.1 Multicompartment Systems

---

It is easy to think of a biological cell as a well mixed compartment and base our models around that premise. However, anyone who has looked through a microscope at a drop of pond water and observed swimming protists will quickly realize that many cells are highly structured and compartmentalized. In eukaryotic cells the most obvious compartments are the nucleus, mitochondria, chloroplasts, and a wide variety of enclosed spaces serving different functions. In all these cases, movement of material occurs from one compartment to another, sometimes active (requiring energy) and sometimes passive. Additionally, all the compartments have widely different volumes. This chapter will briefly look at how to build models involving multiple compartments with differing volumes.

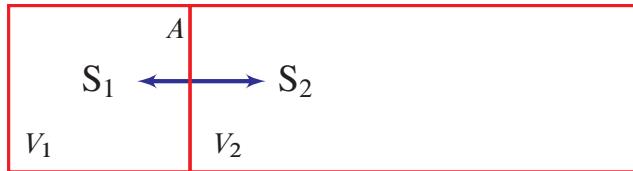
---

### 8.2 Simple Diffusion

---

Let us start by considering the simplest possible example, the reversible and passive diffusion of solute from one compartment of volume  $V_1$  to another compartment of volume  $V_2$  (Figure 8.1).

Let us assume that the volume in compartment two is ten times the volume of compartment one. This means that as mass moves from  $V_1$  to  $V_2$ , the mass will be diluted in the large volume of  $V_2$ . To illustrate this, consider that in compartment  $V_1$  and  $V_2$  we have 5 mM of solute. We will also assume that the volume of  $V_1$  is 1 liter, and the volume of  $V_2$



**Figure 8.1** Two compartment model with volumes  $V_1$  and  $V_2$ .  $S_1$  and  $S_2$  diffusion passively across the membrane with area,  $A$ .

is 10 liters. Let us now move 2 mmoles of solute from compartment  $V_1$  to  $V_2$ . The new concentration of solute in  $V_1$  will be 3 mM. In  $V_2$  the total number of moles of solute before the transfer was 50 mmoles (5 mM in 10 liters). During the transfer, we added 2 mmoles to  $V_2$  resulting in a total amount of solute of 52 mmoles in  $V_2$ . The concentration of solute in  $V_2$  is therefore  $52/10 = 5.2$  mM. This tells us that while the concentration in  $V_1$  changed by 40%, the concentration change in  $V_2$  was only 4%.

These calculations show that we must take into account the different compartment volumes when we move mass from one compartment to another. In the following we will use the symbol  $l$  to represent length, and  $t$  to represent time. One of the basic discoveries in the science of diffusion was Fick's first law. This states that the diffusion rate (or flux) of a compound,  $S$ , from a region of high concentration to a region of low concentration, is proportional to the concentration gradient:

$$J_A = -D_A \frac{dS}{dx}$$

This equation describes the rate of movement of compound across an infinitely thin window of a given area at a position  $x$  across the diffusion flow. The negative sign ensures that the flux is positive when the concentration gradient is negative, that is declining left to right.  $J_A$  is the flux in units of moles  $l^{-2} t^{-1}$  (moles per unit area per time),  $D_A$  the **diffusion coefficient** has units of  $l^2 t^{-1}$  (area per unit time),  $S$  is the concentration and  $dS/dx$  the concentration gradient in units of moles  $l^{-3} l^{-1}$ . That is, moles per volume per length, denoted moles  $l^{-4}$ .

If the zone or window of diffusion has a finite width  $\delta$ , we can approximate Fick's law using:

$$J_A = -D_A \frac{S_{\text{out}} - S_{\text{in}}}{\delta}$$

or

$$J_A = P_A (S_{\text{in}} - S_{\text{out}}) \quad (8.1)$$

where  $P_A$  equals  $D_A/\delta$  and is called the **permeability coefficient** with units of length per unit time (often  $\text{cm } s^{-1}$  in the literature). We assume here that the permeability is the same

on both sides of the membrane. The units of flux at this stage are moles per unit area per unit time (moles  $l^{-2} t^{-1}$ ).  $S_{out}$  and  $S_{in}$  refer to the concentration of solute outside and inside the compartment. To obtain the total amount of mass that moves from one compartment to another we must multiply the flux,  $J_A$ , by the cross-sectional area of the membrane, thus:

$$J = AJ_A$$

where  $J$  is the total amount of substance crossing the membrane and  $A$  the area of the membrane. If this substance is moving into a volume,  $V$ , then the rate of change of concentration in the compartment is given by:

$$\frac{dS}{dt} = -\frac{J}{V}$$

The negative sign indicates that mass is leaving the compartment. We can now write the differential equations for the two compartment model:

$$\frac{dS_1}{dt} = -\frac{J}{V_1}; \quad \frac{dS_2}{dt} = \frac{J}{V_2}$$

where the total flux,  $J$ , is given by:

$$J = AP_A(S_1 - S_2) \quad (8.2)$$

Let us define the amount of  $S_1$  and  $S_2$  as follows:

$$n_1 = S_1 V_1 \quad n_2 = S_2 V_2$$

where  $n_1$  and  $n_2$  are the amounts of  $S_1$  and  $S_2$ , respectively. We can then write the differential equations as:

$$\frac{dn_1}{dt} = -J; \quad \frac{dn_2}{dt} = J$$

Recall that  $J$  is a function of concentration so we can rewrite  $J$  as:

$$J = AP_A(n_1/V_1 - n_2/V_2)$$

The differential equations are now only in terms of amount. To get the concentration at any time during the simulation, we simply take the current amount of mass in the compartment and divide by the compartment's volume. The key then to dealing with multicompartmental systems is to describe the rates of change in terms of amounts rather than concentration, and to continuously recompute concentrations as needed by dividing the amount by volume. We can also show that the result is thermodynamically consistent. To test this we set the flux (8.2) to zero:

$$AP_A(S_1 - S_2) = 0$$

That is  $S_1 = S_2$ . Since we are dealing with simple diffusion, we expect at thermodynamic equilibrium for the two concentrations to be equal, which they are. Note also that the units are consistent, with the units for  $A$  being  $l^2$ , for  $V_1$ :  $l^3$ ,  $P_A$ :  $l t^{-1}$ , and for  $S_x$  : mol  $l^3$ .

**Example 8.1**

A thin membrane has a cross-sectional area of 1mm. On one side of the membrane is a solute of concentration 2 mM, and on the other a concentration of 0.2 mM. If the permeability coefficient for the solute is  $2 \times 10^{-4} \text{ cm s}^{-1}$ , compute the amount of mass that is likely to move across the entire surface of the membrane every second.

We will use equation (8.2) to compute the flux. It is important to ensure that all the units are consistent. Given that the permeability coefficient uses cm for length, we will use cm as the length unit. The area of the membrane is  $0.1 \times 0.1 = 0.01 \text{ cm}^2$ . The concentrations are expressed in moles per liter, and one liter is one thousandth of a cubic meter. A cubic meter equals 1,000,000  $\text{cm}^3$ , therefore a liter must be 1000  $\text{cm}^3$ . Our concentrations of 2 mM and 0.2 mM can therefore be expressed as 0.002 mmoles per  $\text{cm}^3$ , and 0.0002 mmoles per  $\text{cm}^3$ , respectively.

The flux of mass across the membrane can therefore be computed as:

$$J = 0.01 \times 2 \times 10^{-3} (0.002 - 0.0002) = 3.6 \times 10^{-8} \text{ mol s}^{-1}$$

Listing 8.1 shows how one can use Tellurium to model a simple diffusion transport across a thin membrane.

```
import tellurium as te
import pylab

r = te.loada ('''
    compartment V1 = 1, V2 = 10;
    var S1 in V1;
    var S2 in V2;

    S1 -> S2; A*k1*S1;
    S2 -> S1; A*k2*S2;

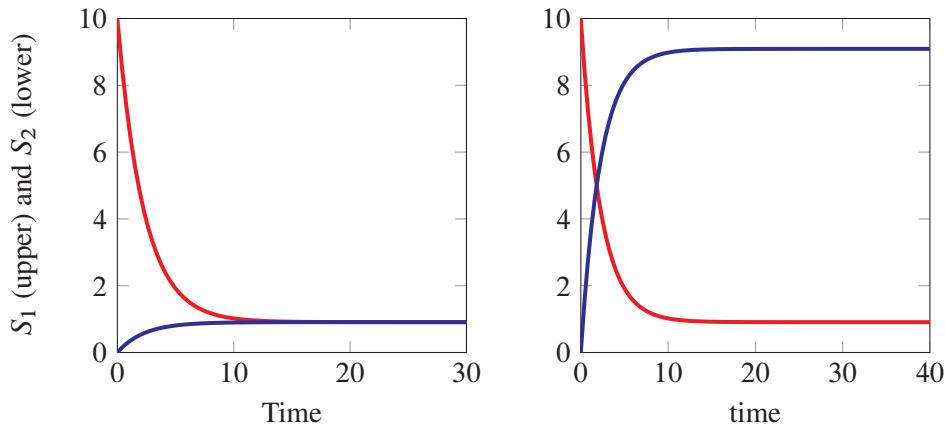
    S1 = 10; S2 = 0;
    k1 = 0.4; k2 = 0.4; A = 1;
''')

result = r.simulate(1, 40, 100)
r.plot (xlim=(0, 40))
```

**Listing 8.1** Script for multicompartment model with simple diffusion.

Figure 8.2 shows the results of the simple compartment simulation. Note that the concentrations converge to the same level because the equilibrium constant across the membrane is one. However, the total amount of mass in each compartment is different due to the difference in volumes.

In Tellurium the adjustment for compartments of different volumes is automatic. To set up a multicompartment model in Tellurium, two things must be done. One must first declare



**Figure 8.2** Simulation of simple diffusion of a solute from one compartment to another. Left graph shows changes in concentration, right graph shows changes in amounts. Upper lines on the left of each graph is S<sub>1</sub>. Tellurium script 8.1.

what compartments are present in the model, in this case  $V_1$  and  $V_2$ , line 2 in Listing 8.1. Next we specify what compartments the species are located in, line 3. After that we specify the transport rates in units of moles transported per second, which represents the total transport across the membrane. Consequently, the rate law is multiplied by the total membrane area since the base units for the transport process will be in moles per second per unit area. The units of the solute,  $S_1$  and  $S_2$  must be in concentration.

### 8.3 Membrane Transporter Protein

Let us consider a more complex example where a solute,  $S_1$ , is transported through a protein pore (and hence saturable) and appears on the other side of the membrane as  $S_2$ . Instead of using Fick's law, we must consider using a saturable Michaelis-Menten like rate law. Let us assume that the concentration of protein pores on the membrane is given by:

$$e = \frac{n_e}{A}$$

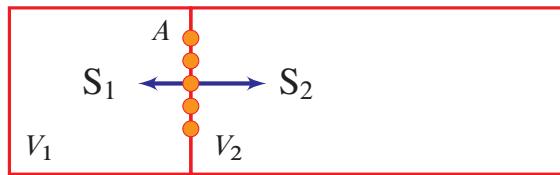
where  $n_e$  is the number of protein pores,  $A$  the area of the membrane, and  $e$  the moles of pores per unit area. The rate of catalysis will be proportional to the concentration of pores on the membrane. Since most pores are saturable, that is at high enough concentration of solute the rate of transport through the pores reaches a maximum, we can write that the rate of transformation in moles (amount) per unit area per unit time (the flux,  $J_A$ ) is given by a generic saturable rate equation such as:

$$J_A = e \frac{k_f S_1 - k_r S_2}{1 + S_1/K_{m1} + S_2/K_{m2}}$$

where  $k_f$  and  $k_r$  are the forward and reverse rate constants such that  $k_f/k_r = K_{eq}$ . As such we can write:

$$J_A = ek_f \frac{S_1 - S_2/K_{eq}}{1 + S_1/K_{m1} + S_2/K_{m2}}$$

There are many variants on this basic equation depending on the specific mechanism but for many systems it can serve as a first approximation. Given the units for  $J_A$ ,  $e$ , and the rate term, the units for  $k_f$  are  $\text{mol l}^3 \text{ t}^{-1}$ . If the transporter is simply allowing passage of solute from one side of the membrane to the other, the  $K_{eq}$  is likely to be unity.



**Figure 8.3** Two compartment model with volumes  $V_1$  and  $V_2$ .  $S_1$  and  $S_2$  move through saturable protein pores in the membrane.

The total flux across the membrane,  $J_A$ , is given as before:

$$J = AJ_A$$

The total flux will equal the following:

$$\frac{dn_1}{dt} = -J \quad \frac{dn_2}{dt} = J$$

This means that the rate of change of concentration of  $S_1$  and  $S_2$  is given by:

$$\frac{dS_1}{dt} = -\frac{J}{V_1} \quad \frac{dS_2}{dt} = -\frac{J}{V_2}$$

Figure 8.2 shows a Tellurium script that represents the transporter model. A few things are worth reviewing in greater detail. By default, Tellurium solves all differential equations in terms of amounts per unit time. This means there is no need to explicitly adjust volume sizes in any equations. Instead, we define the compartments we need using the `vol` keyword, and then indicate which species is in which compartment. All volume adjustments are automatic. Tellurium stores levels of species as amounts and converts to concentrations on an as needed basis, e.g. when a concentration is specified in a rate law. This makes it straight forward to build multicompartment models using Tellurium.

Figure 8.4 shows the results of the simulation. In this case the volume ratio is 1 to 10. Notice how the concentration of  $S_1$  starts at 21 but ends up at 1 in the first compartment, and 2 in the second compartment. We can check mass conservation by summing up the mass in each compartment. The total mass at time zero is  $21 \times 1 = 21$ . The total mass at the end of the run is:  $1 \times 1 + 2 \times 10 = 21$ . Therefore the mass has been conserved.

Name	Symbols	Units
Net Flux	J	$\text{mol } t^{-1}$
Flux	$J_A$	$\text{mol } l^{-2} \text{ } t^{-1}$
Area	A	$l^2$
Volume	V	$l^3$
Concentration	S	$\text{mol } l^{-3}$
Transporter	e	$\text{mol } l^{-2}$
Rate Constant	$k_f$	$\text{mol } l^3 \text{ } t^{-1}$

**Table 8.1** Units for transporter model.  $l$  represents length;  $S$  reactant;  $t$  time.

```
import tellurium as te

r = te.loada ('''
compartment V1, V2;
var S1 in V1, S2 in V2;
S1 -> S2; A*k*(S1-S2/Keq)/(1 + S1/Km1 + S2/Km2);

V1 = 1; V2 = 10;
S1 = 21;
A = 1; k = 1;
Km1 = 0.5; Km2 = 0.5; Keq = 2;
''')

result = r.simulate(0, 200, 100)
r.plot()
print "Total Mass = ", r.S1*r.V1 + r.S2*r.V2
```

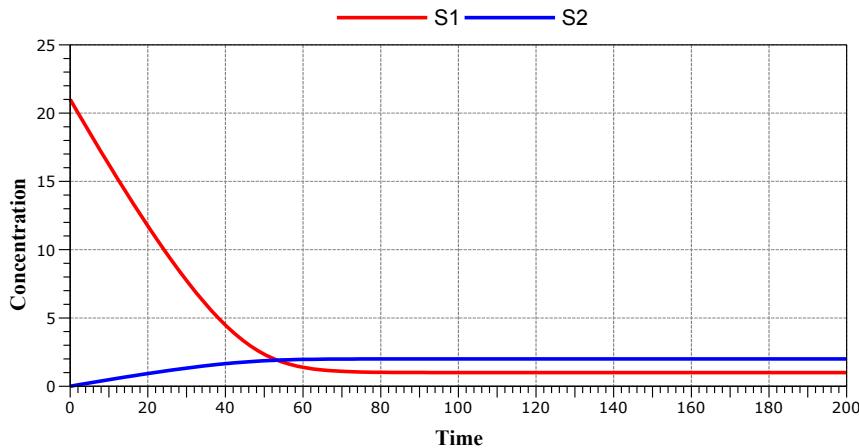
**Listing 8.2** Script for multicompartment model with transporter.

## 8.4 Three Compartment Model

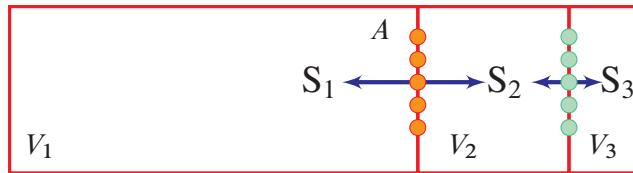
One final example uses three compartments of decreasing volume. The equilibrium constants for the transport across each membrane equal unity.

```
import tellurium as te
import pylab

r = te.loada ('''
compartment V1, V2, V3;
var S1 in V1, S2 in V2, S3 in V3;
```



**Figure 8.4** Simulation results of a membrane transporter. Upper line on left is  $S_1$  and lower line on left is  $S_2$ .



**Figure 8.5** Three compartment model with volumes  $V_1$ ,  $V_2$ , and  $V_3$ .  $S_1$ ,  $S_2$ , and  $S_3$  move through saturable protein pores in the membrane.

```

S1 -> S2; A*k1*(S1-S2/Keq)/(1 + S1/Km1 + S2/Km2);
S2 -> S3; A*k2*(S2-S3/Keq)/(1 + S2/Km1 + S3/Km2);

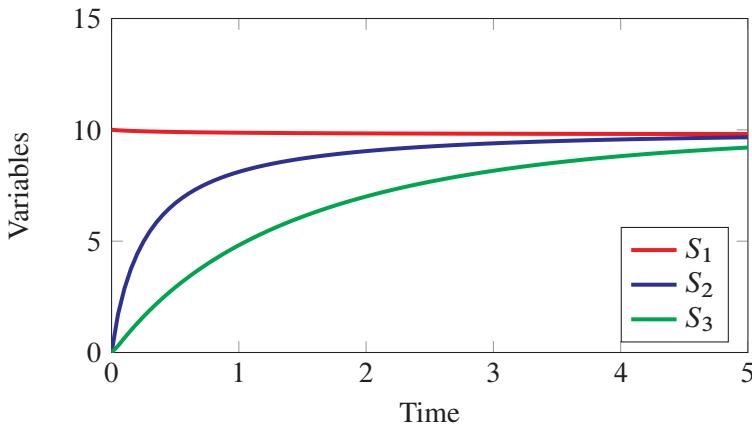
V1 = 100; V2 = 10; V3 = 1;
S1 = 10;
A = 1; k1 = 100; k2 = 25;
Km1 = 0.5; Km2 = 0.5;
Keq = 1;
''')

result = r.simulate(0, 20, 100);
r.plot(xlim=(0,20), ylim=(0,15))
print "Total Mass = ", r.S1*r.V1 + r.S2*r.V2 + r.S3*r.V3;

```

**Listing 8.3** Script for multicompartment model with three compartments, each compartment getting progressively smaller (Figure 8.5).

Figure 8.6 shows the time-course behavior for the three compartment model. Note that the



**Figure 8.6** Simulation results of a membrane transport involving three compartments. Volumes are 100, 10, and 1, respectively. Notice how the concentration in the first compartment hardly changes. Upper curve is  $S_1$ , middle curve  $S_2$ , and lower curve  $S_3$ .

concentration of  $S_1$  (upper curve) hardly changes and that all three curves converge to the same concentration. This is due to the fact that we assumed that both equilibrium constants were equal to 1.0.

## Further Reading

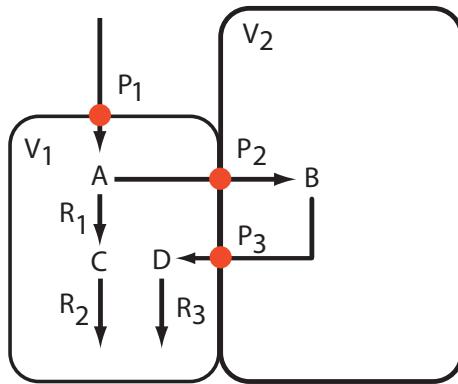
There are surprisingly few books on compartmental analysis in systems biology. Most books focus on pharmokinetic modeling, and it takes a little effort to translate the pharmokinetic formalism into a systems biology one. I list three books here, the most useful being the Neame and Richards book which can be obtained easily on the second-hand market. Atkins is a small book but one I consider a classic. For more advanced students Atkins also offers a painless introduction to the use of Laplace transforms for solving linear differential equations.

1. Atkins, GL (1969). Multicompartment models for biological systems, Methuen London, SBN: 416 13820 9 (SBN is not a typo)
2. Jacquez, JA (1985). Compartmental analysis in biology and medicine. Ann Arbor: University of Michigan Press. The third edition (1996) is available from <http://www.biomedware.com> or directly from <http://tinyurl.com/msh54u6>.
3. Neame, KD and Richards TG (1972). Elementary kinetics of membrane carrier transport. New York: Wiley. ISBN: 0-470-63078-7

## Exercises

---

1. The figure below shows a system of two compartments with volumes  $V_1$  and  $V_2$ . There are three membrane transporters,  $P_1$ ,  $P_2$ , and  $P_3$  and three cytosolic reactions,  $R_1$ ,  $R_2$ , and  $R_3$ . Write out the differential equations that describe the changes in amounts of  $A$ ,  $B$ ,  $C$ , and  $D$ . Assume simple facilitated diffusion for the transporters and irreversible first-order kinetics for the reactions. Build a computer model of the system and investigate how the output fluxes at  $R_2$  and  $R_3$  are influenced by the difference in volume between  $V_1$  and  $V_2$ .



For example, assign reasonable values to all the rate constants in the model, set the two volumes to unity ( $V_1 = V_2 = 1$ ), and compute the two output fluxes. Now increase  $V_2$  ten fold while keeping all other parameters the same. What happens to the  $R_2$  and  $R_3$ ?

# 9

## *Fitting Models*

### 9.1 Introduction

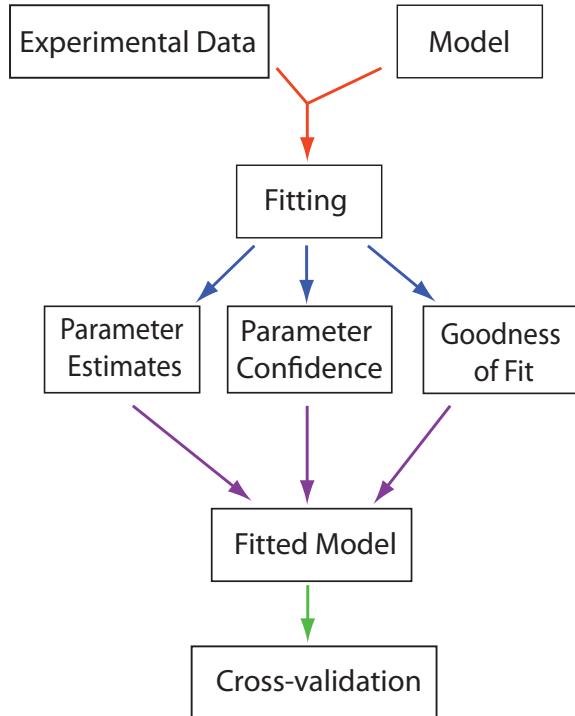
In constructing computational models (Chapter 4) of biochemical systems, we make choices about what reaction steps, regulatory interactions and molecular species to include. Given these choices, how good is the model? Does the model adequately describe existing knowledge about the system? Can the model make useful predictions? Some of the model parameters might be estimated experimentally but many will be unknown. How can we estimate these parameters and how well can they be estimated? Such questions fall under the umbrella of **model fitting**. In this chapter and the next we will consider these questions.

Questions we will consider in this and the next chapter:

1. Can we determine the values for the unknown parameters in the proposed model from the experimental data, for example using optimization methods? This is termed **system identification**.
2. Does the model reasonably represent the known experimental data, i.e. is the model a *good fit*?
3. What confidence do we have in the fitted parameters?
4. Can the fitted model make new and useful predictions?

To start, let's briefly state what it means to **fit** a model.

Fitting a model means adjusting the parameters of the model until the behavior of the model matches some known experimental data.

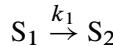


**Figure 9.1** Fitting models to data.

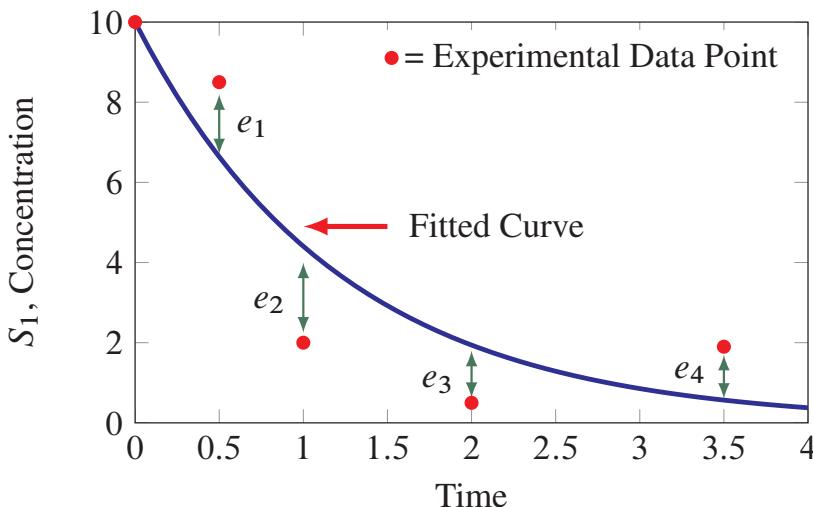
Figure 9.1 describes some of the outcomes of fitting a model to data. In particular, we can answer the question how well the model describes experimental data. We can also obtain estimates for the parameters in the model and how much confidence we have in the fitted parameters. Finally we can cross-validate. This is where we hold some data back and ask the fitted model to try to predict this data. In this chapter we will focus on fitting models to experimental data, and the use of different optimization methods. Fitting models is an active area of research and we can only cover a limited area of this important topic.

## Optimizing Parameter Values

To understand how the fitting process works, consider a simple model:



We start an experiment with an initial amount of  $S_1$  and observe the change in  $S_1$  as it reacts to form  $S_2$ . Figure 9.2 shows both a solid curve representing a simulation of the model, and four experimental data points for the concentration of  $S_1$ . The first data point at time zero represents the initial concentration of  $S_1$  which we assume is error free though. This may not be the case always however. Measurements are collected at time points 0.5, 1, 2, and 3.5. The  $e_i$  terms represent the difference between the experimental data point and the simulation curve and are called the residuals. Fitting is the process where we attempt to adjust the parameters of the model ( $k_1$  in this case), such that the differences,  $e_i$ , between the simulation curve and the data points is **minimized**.



**Figure 9.2** Model curve and experimental data plotted on the same graph. The solid line is the simulated model, the points represent experimental data. The difference between the experimental data and the simulation curve is indicated by errors,  $e_i$ , called residuals. Model fitting attempts to minimize the  $e_i$  terms by adjusting the model parameter values.

In more general terms, let us indicate the experimental data points using the symbols,  $x_i$  and  $y_i$ , where  $x_i$  is the independent variable time and  $y_i$  the dependent variable. Assume there are  $N$  data points. We will indicate the model using the expression  $f(x_i; p_1 \dots p_m)$ , where  $p_i$  is the  $i$ th parameter in the model. That is, for a given set of parameters and time point  $x_i$ , the function  $f$  will return the corresponding model  $y_i^m$  value. If the model is a set of differential equations, we would run a simulation in order to obtain the value of  $y_i^m$

at  $x_i$ . The fitting procedure will attempt to minimize the difference between the model  $f$  and the data points, that is minimize:

$$y_i - f(x_i; p_1 \dots p_m)$$

Because the difference between a data point and the model may be positive or negative depending on the error in the data point (See  $e_2$  for example in Figure 9.2), we take the square of the difference to make the term positive:

$$(y_i - f(x_i; p_1 \dots p_m))^2$$

This difference only corresponds to one data point, and we should be considering all data points when trying to fit the model. Therefore we sum up all the differences and attempt to minimize the total sum, that is:

$$\sum_{i=1}^N (y_i - f(x_i; p_1 \dots p_m))^2$$

We can take this one step further and reason that the most uncertain data points should contribute less to the sum compared to those which have been measured more precisely. We therefore weight each difference by the standard deviation,  $\sigma$ , that corresponds to that data point. This assumes that we have some measure of uncertainty, if we don't we set the weight to one:

$$\chi^2 \equiv \sum_{i=1}^N \left( \frac{y_i - f(x_i; p_1 \dots p_m)}{\sigma_i} \right)^2 \quad (9.1)$$

The above equation can also be expressed in the following equivalent form to emphasize the weighing in terms of the variance,  $\sigma^2$ :

$$\chi^2 \equiv \sum_{i=1}^N \frac{1}{\sigma_i^2} (y_i - f(x_i; p_1 \dots p_m))^2$$

This equation is called the **weighted chi-square sum of squares**<sup>1</sup> and can vary between zero and infinity. If the model is a set of differential equations, the  $f$  function is a list of data points from a simulation run. For example, using the previous model let us assume the parameter  $k_1$  is set to -0.95. Table 9.1 shows an example of computing the chi-square given some data points and results from a model run.

An important variant of the chi-square is the **reduced chi-square** (9.2) which is used when looking at the quality of the fit and estimating the confidence in the fitted parameter.

<sup>1</sup>The notation  $\chi^2$  is possibly misleading. The  $\chi^2$  is not the square of a quantity  $\chi$  and is why the term chi-square is used rather than chi-squared. The  $^2$  is simply to remind us of the square on the right-hand side of the definition.

Time	Data Point	Point from Model	Difference	Difference Squared
0	10	10	0	0
0.5	7.9	6.2	-1.68	2.8
1	2.1	3.87	1.77	3.12
2	0.5	1.5	1	1
3	0.6	0.58	-0.02	0.00046
Sum =				6.92

**Table 9.1** Calculating chi-square. Assume we have no variances with the data points, therefore the weighting is one.  $\chi^2$  is the sum of the right most column and equals 6.92. The reduced chi-square,  $\chi^2_{\text{reduced}}$ , is  $6.9/(5/1) = 1.3$ .

$$\chi^2_{\text{reduced}} \equiv \frac{1}{N - P} \sum_{i=1}^N \frac{1}{\sigma_i^2} (y_i - f(x_i; p_1 \dots p_m))^2 \quad (9.2)$$

In the above equation (9.2),  $N$  is the number of data points and  $P$  the number of parameters to be fitted in the model. The difference  $N - P$  is called the **degrees of freedom**. We will return to the topic of the reduced chi-square later, we simply want to define it here.

### Maximum Likelihood Justification - Optional

In this section we will justify the use of  $\chi^2$  as a means to estimate the unknown parameters. This section may be omitted on first reading. The previous section discussed using the difference between a data point and a simulated point, squaring the difference to eliminate negative terms and summing and weighting all data points to estimate the quality of our fit. This sounds quite reasonable, but is there a more theoretical justification for this approach? The question is, how can we be sure this particular definition of the  $\chi^2$  leads to the best parameter estimates for the experimental data? Is there another formula we could use that would give us more accurate estimates?

The answer to this question lies in using **maximum likelihood**, an approach developed by Fisher [2] between 1912 and 1922. Maximum likelihood is a method that asks the question, given a set of data and associated model with unknown parameters,  $p_i$ , what are the most likely values for the parameters? Very briefly, the method works by first computing the likelihood function which describes the likelihood of a set of parameters,  $p$ , given the data,  $x$ , often denoted:

$$L(p|x)$$

If we change the parameters for a given set of data, the likelihood,  $L$ , will change. What set of parameter values maximizes the likelihood? The way to find a maximum is to find

the point where the derivative of the function of interest is zero, and the second derivative is negative. The maximum likelihood can therefore be found by differentiating the likelihood function with respect to the parameter, setting the derivative to zero, and solving for the parameter. To make matters simpler, the log of the likelihood is often differentiated, that is we differentiate:

$$\ln(L(p|x))$$

and then determine  $p$  from:

$$\frac{\partial \ln L(p|x)}{\partial p} = 0$$

A fuller account of maximum likelihood is given in Appendix F, together with a proof that shows that the sum of squares when minimized does indeed yield the most likely estimates for the parameter values. It is therefore true that the original, intuitive reasoning matches the more formal approach. The formal derivation also gives the limits on the use of  $\chi^2$ . In particular, the maximum likelihood derivation assumes that the errors in the experimental data are *normally distributed* and *independent* (See Appendix for a refresher, F).

When using the sums of square to find the best parameters for a model, it is assumed that the experimental data points are normally distributed and independent.

## 9.2 Optimization Algorithms

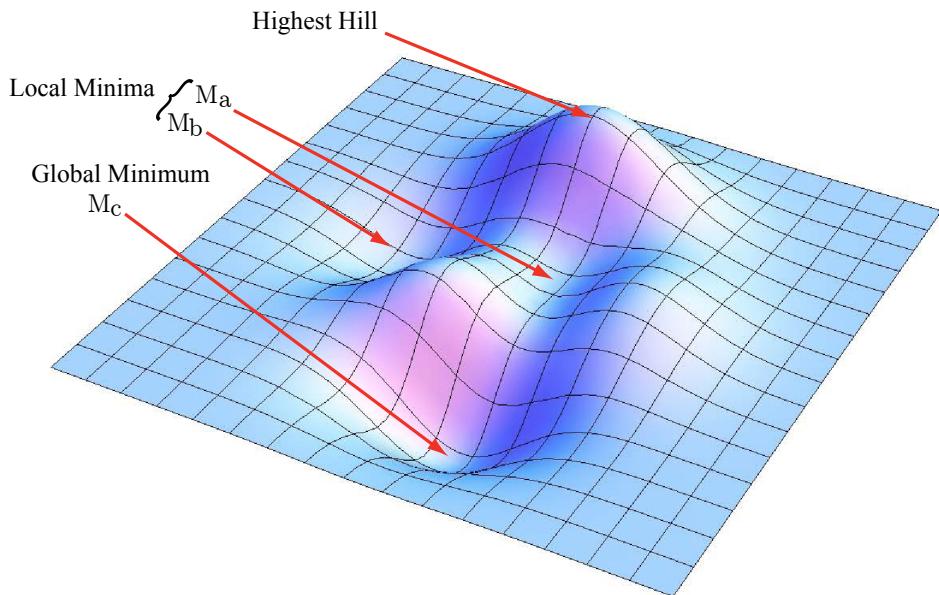
A brute force method for fitting a model is to run a simulation of the model many times with random parameter values until we find a set of parameters that gives us simulation data that matches the experimental time series. One problem with this approach is that we're likely to spend a great deal of time coming up with random parameter values in the hopes that at least one set will match the experimental data. This however is unlikely, and the brute force method is rarely used in practice. Instead, special search algorithms have been devised, called **optimization algorithms**, to search for the best set of parameters in a systematic way.

Optimization is an iterative process. It involves making an initial guess for the parameters,  $p_i$ , computing the  $\chi^2$  value, and using a rule that adjusts the parameter values such that the  $\chi^2$  is reduced in the next iteration. This procedure is repeated many times until the  $\chi^2$  can no longer be reduced, at which point the iteration stops. If the fit was successful, the model should be able to reproduce the experimental data given the final set of parameters.

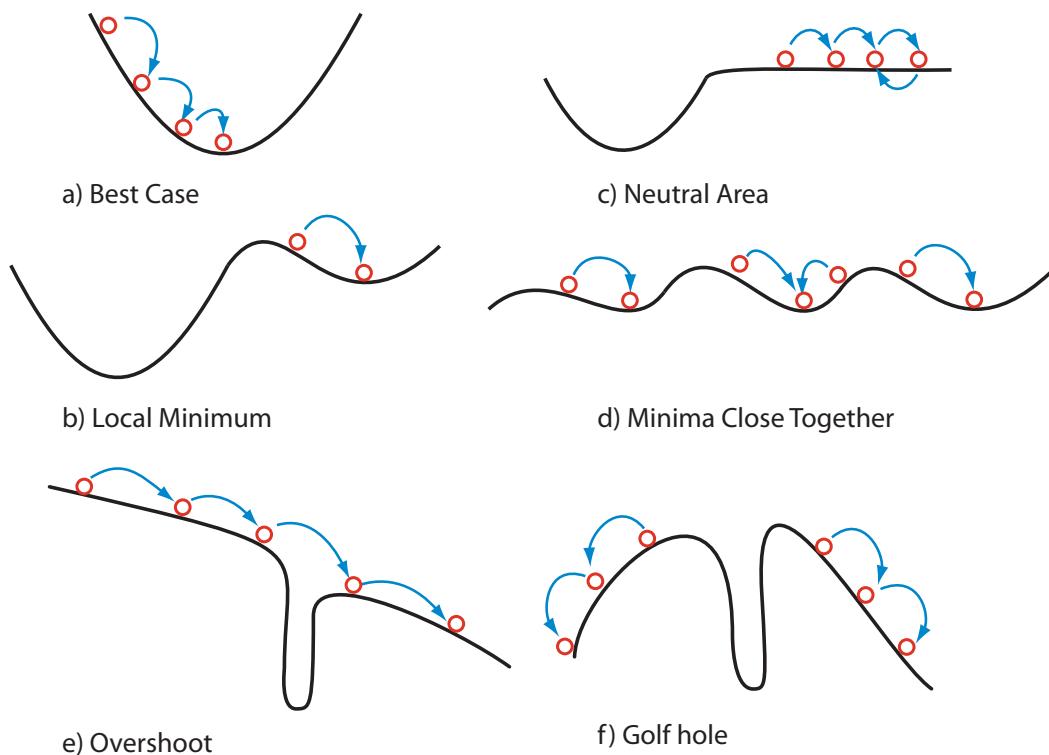
One way to imagine this process is to consider a two parameter system where the  $\chi^2$  describes a surface. Figure 9.3 shows such a surface, also called a fitness landscape. The  $z$ -axis is a measure of the  $\chi^2$ , and the  $x$  and  $y$  axes are two parameters we wish to estimate. As the two parameters are varied,  $\chi^2$  changes, sometimes to high values and sometimes to low values. The low values are the ones of interest, ideally at the lowest possible  $\chi^2$ .

value, called the **global minimum**. We can see that the surface is quite complicated with a number of hills and valleys. This is often the case when fitting a model.

To start the optimization process we select, possibly at random, values for the two parameters. Let us assume we started the optimization at the top of the tallest hill. What we seek is the lowest point on the surface. An obvious strategy is to move down the hill until we reach the lowest point. However if we did this we wouldn't necessarily reach *the* lowest point, but an intermediate low point called a local minimum (most likely point  $M_a$  or  $M_b$  in the figure). However if we started on the near side of the second peak and moved down the hill, we would reach the deepest point or global minimum at  $M_c$ . Depending on the surface complexity, it can be difficult to find the global minimum because it is easy to find a local minimum first and claim success. Depending on the landscape, a search method can encounter a range of problems while searching for the global minimum. Figure 9.4 illustrates examples of common issues when searching for the global minimum. A great variety of approaches have therefore been devised to solve this problem. The next section will describe five common methods employed to find optima.



**Figure 9.3** Example of a fitness landscape showing multiple minima ( $M_a$  and  $M_b$ ) and a global minimum at  $M_c$ . The vertical axis represents  $\chi^2$ , and the  $x$  and  $y$  axes the parameters. The plot shows how  $\chi^2$  changes for different parameter values. The function used to plot the surface is:  $3(1-x)^2 \exp(-x^2 - (y+1)^2) - 10(x/5 - x^3 - y^5) \exp(-x^2 - y^2) - 1/3 \exp(-(x+1)^2 - y^2)$ .



**Figure 9.4** Problems encountered in different fitness landscapes. Vertical axis is the objective or fitness function.

## Gradient Descent

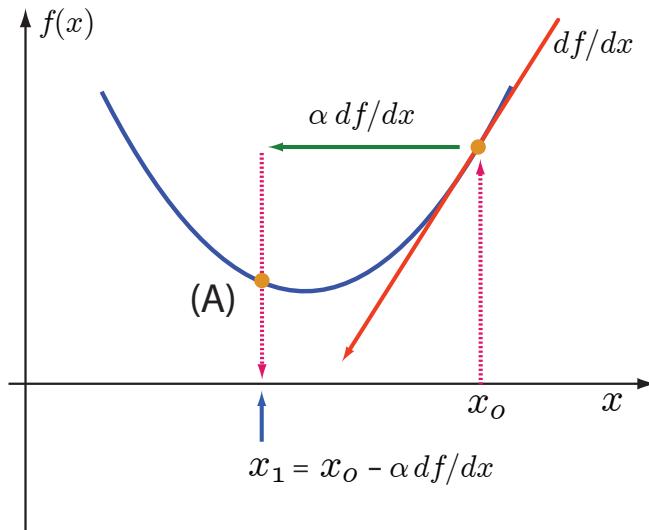
Gradient descent is one of the simplest methods for finding a minimum. It is not recommended for practical since given that it is such crude approach but it serves as a basis for building and understanding more sophisticated gradient descent methods such as the Levenberg-Marquardt method.

The gradient descent method moves in the steepest direction that reduces the sum of squares. A one dimensional problem is the easiest to explain. Consider a function,  $f(x)$ , such as  $2x^2 + x - 3$  where we wish to find the value for  $x$  than minimizes the function<sup>2</sup>. A plot of this equation yields the parabola shown in Figure 9.5.

The gradient descent method starts by picking an initial starting point,  $x_0$ , and uses the slope,  $df/dx$ , at that point to move to a lower point on the curve, (A). This can be done by computing the distance,  $-\alpha df/dx$ , we need to move. Notice that the distance must be negative. This is to ensure that we are moving in the direction that takes us closer to the minimum. We repeat until the slope,  $df/dx$ , is below some preset threshold. The key

<sup>2</sup>We're not trying to find solutions to the equation when  $f(x)$  is zero, rather the smallest value of  $f(x)$ .

to implementing a robust gradient descent is the choice of the step size factor  $\alpha$ . We will come back to this shortly. The pseudo code for a one dimensional gradient descent using a fixed  $\alpha$  is shown in Algorithm 4. Figure 9.5. already hints at one potential problem, that is overshoot. In the figure we see that point (A) is on the other side of the minimum from the original starting point. There is a potential here to oscillate back and forth across the minimum if the choice of  $\alpha$  is not made correctly.



**Figure 9.5** Gradient descent in one dimension. The method starts with an initial guess at  $x_o$  and uses  $\alpha df/dx$  to compute the new position A where  $\alpha$  is the step size factor. Point A forms the starting point for a new iteration. This is repeated until the slope is less than some small number. If the step size is too big, there is a chance the search will overshoot. The key to implementing a robust gradient descent is the choice of step size factor,  $\alpha$  and how it should be varied as the calculation proceeds.

The method can be easily scaled to multidimensional systems where the function,  $f$ , is now a function of more than one parameter (See Appendix E.6), for example,  $f(x_1, x_2, \dots, x_n)$ . We define the gradient vector as:

$$\nabla f(x_1, x_2, \dots, x_n) = \left[ \frac{\partial f(x_1, x_2, \dots, x_n)}{\partial x_1}, \frac{\partial f(x_1, x_2, \dots, x_n)}{\partial x_2}, \dots \right]^T$$

Changing the parameters in a single iteration now becomes:

---

**Algorithm 4** One dimensional gradient descent.  $x_o$  represents the starting point for the method. This algorithm uses a fixed  $\alpha$ . This is a naive description because it doesn't deal with the case where the solution does not converge.

---

Initialize starting point to  $x_o$

Initialize the step factor  $\alpha$

Initialize slope threshold  $\epsilon$

$f(x)$  is the objective function

**while**  $\alpha \text{ abs} \left( \frac{df}{dx} \right) > \epsilon$  **do**

$$x_o = x_o - \alpha \frac{df}{dx}$$

**end while**

---

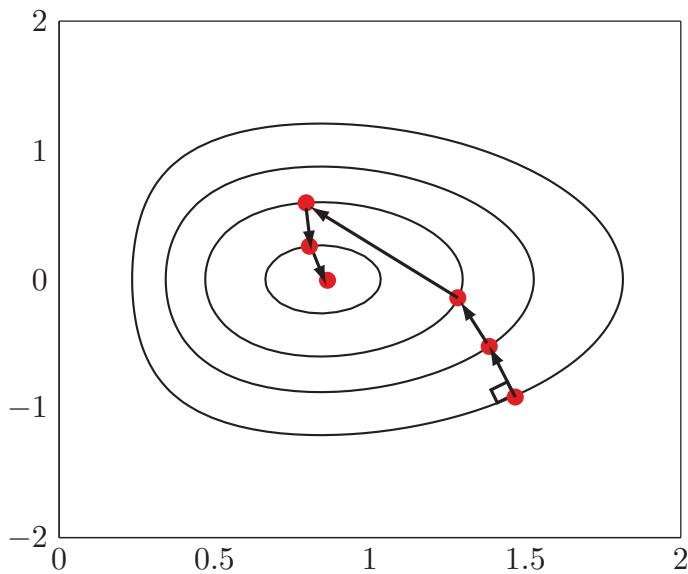
$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}_{n+1} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}_n - \alpha \begin{bmatrix} \frac{\partial f(x_1, x_2, \dots, x_n)}{\partial x_1} \\ \frac{\partial f(x_1, x_2, \dots, x_n)}{\partial x_2} \\ \vdots \\ \frac{\partial f(x_1, x_2, \dots, x_n)}{\partial x_n} \end{bmatrix}$$

In vector format the expression can be succinctly written as:

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \alpha \nabla f(\mathbf{x}_n)$$

Figure 9.6 shows the gradient descent in a two dimensional system. The search vectors are always perpendicular to the contours. When  $\alpha$  is too big, the search can overshoot as seen in the third vector.

One advantage of the gradient descent is that it is straight forward to implement. It suffers however from a number of issues. As mentioned previously, the main problem is what value to set the step factor,  $\alpha$ . If the factor is too large, the iteration will overshoot the minimum, and in the next iteration it will backtrack, possibly overshooting the minimum again. Reducing the step size can help avoid this effect but if too small, the method can take much too long to reach the minimum. A common modification to accommodate these difficulties is to add a mechanism to adjust  $\alpha$  as the search progresses. A crude but effective way to adjust  $\alpha$  is to try different values in order to find one that results in a *reduction* in the function,  $f$ . This helps avoid overshoots and reduces the number of steps required to reach the minimum. This technique and its variants are called **line searching** because the algorithms attempt to search along the gradient descent direction looking for a point that



**Figure 9.6** Gradient descent in a two dimensional system. Note that the third search vector overshoots.

reduces the value of the function,  $f$ . A modification that includes a simple line search is shown in Algorithm 5. However, these modifications are not very helpful near the minimum because the gradient is likely to be shallow resulting in very small steps. This means that although a simple gradient descent will initially converge quite rapidly, as it approaches the minimum the rate of convergence will slow considerably.

All gradient descent methods find the nearest minimum which is quite likely to be a local minimum for a complex model. Gradient descent methods, including more elaborate ones such as the Levenberg-Marquardt, should be used in conjunction with other methods that are better at finding global minima.

### Gauss-Newton Method

The gradient descent method described in the last section is general in the sense that the function can be in any form including a sums of squares. An alternative method for finding minima that is specially designed for systems where the function is a sum of squares, is the Gauss-Newton method. This method relies on the assumption that we can approximate the surface near the minimum using a quadratic function in the parameters. That is, near the minimum we assume that the objective function looks like a parabolic bowl. To obtain this approximation we use the Taylor series to expand the residual function (9.1),  $\chi^2$  to second

---

**Algorithm 5** One dimensional gradient descent with a simple line search.  $x_o$  represents the starting point for the method.

---

Initialize starting point to  $x_o$

Initialize threshold  $\epsilon$

$f(x)$  is the objective function

**while**  $\text{abs} \left( \frac{df}{dx} \right) > \epsilon$  **do**

$$d = \frac{df}{dx}$$

$$\alpha = 1$$

**while**  $f(x_o - \alpha d) > f(x_o)$  **do**

$$\alpha = \alpha/2$$

**end while**

$$x_o = x_o - \alpha d$$

**end while**

---

order about  $\mathbf{p}_0$ , where  $\delta\mathbf{p} = \mathbf{p} - \mathbf{p}_0$  (See Appendix E.6):

$$\chi^2(\mathbf{p}) = \chi_{\mathbf{p}_0}^2 + \delta\mathbf{p}^T \mathbf{d} + \frac{1}{2} \delta\mathbf{p}^T \mathbf{H} \delta\mathbf{p} \quad (9.3)$$

where  $\mathbf{d}_i = \partial\chi^2/\partial p_i$  and  $\mathbf{H}$  is called the **Hessian** and has elements defined by:

$$H_{ij} = \frac{\partial^2 \chi^2}{\partial p_i \partial p_j} \quad (9.4)$$

$\mathbf{H}$  describes the **curvature** of the surface. At the minimum the derivative of (9.3) will equal zero. The minimum of the surface can therefore be found by differentiating expression (9.3) with respect to  $\delta\mathbf{p}^T$ , and setting the result to zero:<sup>3</sup>

$$0 = \mathbf{d} + \mathbf{H} \delta\mathbf{p} \quad (9.5)$$

This is a linear system of equations which can be solved for  $\delta\mathbf{p}$ :

$$\delta\mathbf{p} = -\mathbf{H}^{-1} \mathbf{d}$$

From this we obtain an update to the parameter  $\mathbf{p}$  using  $\delta\mathbf{p}$ :

$$\mathbf{p}_{k+1} = \mathbf{p}_k - \mathbf{H}^{-1} \mathbf{d} \quad (9.6)$$

---

<sup>3</sup>Note:  $\mathbf{d}(\delta\mathbf{p}^T \mathbf{H} \delta\mathbf{p})/\mathbf{d}\delta\mathbf{p}^T = 2\mathbf{H} \delta\mathbf{p}$

As with gradient descent this is an iterative algorithm requiring repeated evaluations of the Hessian. In practice we don't compute the inverse of  $\mathbf{H}$  but determine  $\delta\mathbf{p}$  using a standard linear equation solver algorithm. One potential inefficiency is computing the Hessian. This is a matrix of second derivatives which in general are expensive in computer time to estimate accurately. To improve the efficiency let's look at this problem from a different point of view. Leaving out the weights,  $\sigma$ , and without loss of generality, let  $\chi^2$  be written as, see (9.1):

$$\chi^2(\mathbf{p}) = \sum_{i=1}^m r_i^2(\mathbf{p})$$

where  $r_i$  is the residual  $y_i - f(x_i; p_1 \dots p_m)$ . To simplify the notation let us designate  $\chi^2(\mathbf{p})$  using the symbol  $f(\mathbf{p})$ . The derivative of  $f(\mathbf{p})$  with respect to  $p_j$  is given by:

$$\frac{\partial f(\mathbf{p})}{\partial p_j} = 2 \sum_{i=1}^m r_i(\mathbf{p}) \frac{\partial r_i}{\partial p_j}$$

For all  $p_j$  we can write the above in matrix form:

$$\nabla f(\mathbf{p}) = 2 \mathbf{J}(\mathbf{p})^T \mathbf{r}(\mathbf{p}) = \mathbf{d} \quad (9.7)$$

$\nabla f(\mathbf{p})$  is a column vector of  $\partial \chi^2 / \partial p_j$ , that is  $\mathbf{d}$  in equation (9.6),  $\mathbf{J}(\mathbf{p})$  is the Jacobian matrix of derivatives  $\partial r_i / \partial p_j$  and  $\mathbf{r}(\mathbf{p})$  a column vector of  $r_i(\mathbf{p})$  terms.

The Hessian,  $\mathbf{H}$ , can be computed by differentiating  $\nabla f(\mathbf{p})$ :

$$\nabla^2 f(\mathbf{p})_{kl} = \mathbf{H} = \frac{\partial^2 f(\mathbf{p})}{\partial p_k \partial p_l} = 2 \sum_{i=1}^m \frac{\partial r_i(\mathbf{p})}{\partial p_k} \frac{\partial r_i(\mathbf{p})}{\partial p_l} + 2 \sum_{i=1}^m r_i(\mathbf{p}) \frac{\partial^2 r_i(\mathbf{p})}{\partial p_k \partial p_l}$$

This expression can be reexpressed in matrix form as:

$$\mathbf{H} = 2 \mathbf{J}(\mathbf{p})^T \mathbf{J}(\mathbf{p}) + 2 \sum_{i=1}^m r_i(\mathbf{p}) \nabla^2 r_i(\mathbf{p})$$

Near the minimum the residuals,  $r_i$ , will be small and therefore the second term can be ignored, resulting in a simplified Hessian:

$$\mathbf{H} = 2 \mathbf{J}(\mathbf{p})^T \mathbf{J}(\mathbf{p}) \quad (9.8)$$

In this form the Hessian is much easier to compute, no second derivatives are required. Only the first derivatives in the Jacobian need be estimated. We can insert the various terms (9.7) and (9.8) into equation (9.5):

$$0 = \mathbf{d} + \mathbf{H} \delta \mathbf{p}$$

$$0 = 2 \mathbf{J}(\mathbf{p})^T \mathbf{r}(\mathbf{p}) + 2 \mathbf{J}(\mathbf{p})^T \mathbf{J}(\mathbf{p}) \delta \mathbf{p}$$

$$0 = \mathbf{J}(\mathbf{p})^T \mathbf{r}(\mathbf{p}) + \mathbf{J}(\mathbf{p})^T \mathbf{J}(\mathbf{p}) \delta \mathbf{p}$$

Rearranged we obtain the version that is often to be seen in the literature (we've dropped the ( $\mathbf{p}$ ) for clarity) :

$$\mathbf{J}^T \mathbf{J} \delta \mathbf{p} = -\mathbf{J}^T \mathbf{r} \quad (9.9)$$

This is a linear set of equations which can be solved for  $\delta \mathbf{p}$  which in turn can be used to update the parameter values in (9.6). That is:

$$\delta \mathbf{p} = -[\mathbf{J}^T \mathbf{J}]^{-1} \mathbf{J}^T \mathbf{r} \quad (9.10)$$

Equation (9.9) constitutes the update strategy for the Gauss-Newton method. A number of points are worth mentioning. The method depends on an approximation of the Hessian which is only true near the minimum. Therefore the Gauss-Newton should not be used far from the minimum because it will likely fail to converge to the solution. This is in contrast to a gradient descent method where there is no requirement to be close to the minimum so long as there is a downward slope that can reach the minimum. A further problem with the Gauss-Newton method is the need for  $\mathbf{J}^T \mathbf{J}$  to have full rank, this is required so that (9.9) can be solved for  $\mathbf{p}$ . Rank deficiency will occur when there are correlations between parameters due to insufficient data. This is related to the identification problem. However, one of the chief advantages of the Gauss-Newton method is its very rapid convergence properties.

## Levenberg-Marquardt

In the last sections we discussed two approaches to finding the minimum. One approach (gradient descent) involved moving down the fitness landscape until we reached the minimum. One problem with this method is that in steep sections of the landscape the algorithm tends to move quickly while in more gradual inclines, the algorithm tends to move too slowly and convergence can take much longer. In the second approach, the Gauss-Newton method will only converge if the search is already close to the minimum, at which point convergence is very rapid.

These two methods appear complementary where each solves issues the other method has. It therefore seems natural to combine the methods, exploiting rapid descent far from the minimum using gradient descent and moving to the Gauss-Newton method when close to the solution. This is how a method called the **Levenberg-Marquardt method** works. The Levenberg-Marquardt employs a weighted mixture between the two types of searches [110, 138]. The first is a gradient search followed by a Gauss-Newton method.

In the first phase the method uses gradient descent so that we descend down the fitness landscape in a direction opposite (a positive gradient would take us uphill) to the local gradient. For the  $(k + 1)$  iteration, the parameters  $p_i$  are changed according to:

$$\mathbf{p}_{k+1} = \mathbf{p}_k - \mu \mathbf{d} \quad (9.11)$$

where the gradient  $\mathbf{d} = \frac{\partial \chi^2}{\partial \mathbf{p}}$ , and  $\mu$  is the step size.

As we get closer to the minimum, we switch strategies and use the Gauss-Newton method. The combined approach can be described by equation (9.12) where the value of  $\mu$  is used to move from one strategy to the other:

$$\mathbf{p}_{k+1} = \mathbf{p}_k - (\mathbf{H} + \mathbf{I}\mu)^{-1} \mathbf{d} \quad (9.12)$$

If  $\mu$  is large, the equation behaves as a gradient descent method (9.11):

$$\mathbf{p}_{k+1} \approx \mathbf{p}_k - \frac{1}{\mu} \mathbf{d}$$

As  $\mu$  decreases, the method moves to the Gauss-Newton method:

$$\mathbf{p}_{k+1} \approx \mathbf{p}_k - \mathbf{H}^{-1} \mathbf{d}$$

One last modification is necessary before we have the full Levenberg-Marquardt method. When using gradient decent, we are not using the curvature,  $\mathbf{H}$ . Marquardt suggested that some benefit could be obtained by incorporating the curvature during gradient decent. This means the step size can vary since the curvature changes and hence the algorithm can take longer steps in regions where the gradient is less, for example in a long shallow valley. In addition, the method is less likely to overshoot the minimum. The final Levenberg-Marquardt equation is given by:

$$\mathbf{p}_{k+1} = \mathbf{p}_k - (\mathbf{H} + \text{diag}(\mathbf{H})\mu)^{-1} \mathbf{d} \quad (9.13)$$

Where  $\text{diag}(\mathbf{H})$  represents a matrix that just contains the main diagonals of the Hessian. The algorithm starts by using gradient descent. If the error can be reduced, meaning it is successful, it decreases  $\mu$ . This starts to shift the method towards using the Gauss-Newton method. If the error increases then the  $\mu$  is increased in value and the method shifts to using the gradient descent method. This process is continued until the change in  $\chi^2$  is a very small number. A common strategy for changing  $\mu$  is:

1. Start  $\mu$  with a value of 100 (uses the gradient descent initially)
2. If the new chi-squared is bigger than the previous chi-squared, then  $\mu = \mu \times 10$  (that is the method becomes more gradient descent like)
3. If the new chi-squared is less than or equal to the previous chi-squared, then  $\mu = \mu/10$  (that is the method becomes more Gauss-Newton like)
4. Goto 2

There are more sophisticated control strategies for changing  $\mu$  that can be used if necessary [62].

Overall the Levenberg-Marquardt method has proven very successful approach. Its main drawback is that the method tends to find the nearest minimum which could easily be a local minimum. The method is therefore sensitive to starting conditions. For the surface shown in Figure 9.3, if we start on the highest hill, the Levenberg-Marquardt will most likely find the nearest minimum,  $M_a$ , which is a local minimum not the global minimum. The Levenberg-Marquardt is better suited when combined with other methods. If a good starting point can be found, the Levenberg-Marquardt can take over and rapidly find the global minimum.

There are a number of freely available open source implementations of the Levenberg-Marquardt method. There are two GPL licensed solvers, GSL<sup>4</sup> and levmar<sup>5</sup>. To avoid the distribution restriction of the GPL licence, the lmfit library<sup>6</sup> is highly recommend (licensed under FreeBSD License). There are also a variety of Java versions available on the Web, a search using Levenberg-Marquardt java will locate many of them as well as implementations in other languages such as C#. Scripting languages such as R, Python (scipy package), and Matlab also support implementations of Levenberg-Marquardt. COPASI implements the Levenberg-Marquardt method as well and Tellurium has access to the algorithm via the scipy Python package.

## Simplex or Nelder and Mead

The Levenberg-Marquardt method requires the calculation of derivatives during each iteration which can be slow and not always easy. The following and remaining methods do not require derivatives which means they can be easier to implement. Moreover they are better at avoiding local minima and are more likely to find the global minimum.

The **simplex method**, as described by Nelder-Mead [124], is a robust search method (i.e. it is generally tolerant of noisy data), in which the objective function, in our case  $\chi^2$ , is computed at several test points. The test point with the highest value for  $\chi^2$  is replaced by another point which has a lower value for  $\chi^2$ .

In a parameter space of  $P$  dimensions, a  $P + 1$  dimensional geometrical object is created, called a *simplex*, with its vertices initialized to some starting values. In two dimensions a simplex is a triangle, in three dimensions its a tetrahedron, and so on. The  $P + 1$  vertices of the simplex are the points at which the objective function is evaluated. The simplex evolves by first trying to replace the worst point with a new point using either reflection, expansion or contraction. Each of these possibilities lies along a line that passes through the centroid of the simplex (Figure 9.7). Reflection is tried first. If the reflected result is better, then the reflection is expanded. If the reflected point is worse, then instead of a reflection a contraction is employed. If all three operations fail to reduce the objective function, a contraction along all faces towards the best point is carried out (Figure 9.8). This process

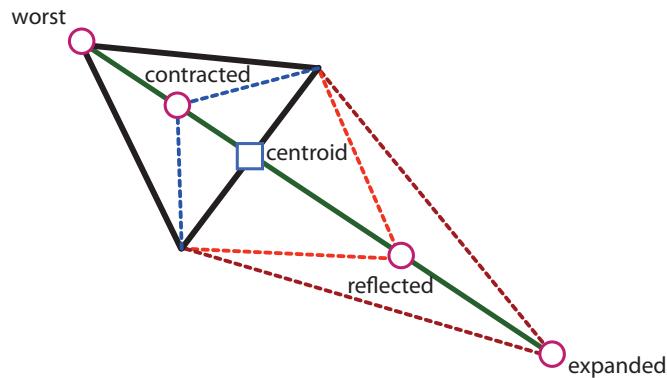
<sup>4</sup><http://www.gnu.org/software/gsl/>

<sup>5</sup><http://users.ics.forth.gr/~lourakis/levmar/>

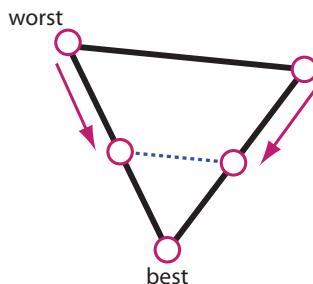
<sup>6</sup><http://joachimwuttke.de/lmfit/>

is summarized below:

- The simplex reflects the worst point through the opposite face to a new point.
- If the reflection results in a better point, i.e. lower error, it is further stretched in that direction (expansion).
- If the reflection results in a worse point, abandon the reflection and contract the worst point towards the opposite face of the simplex.
- If all the above fails, contract along all faces towards the best point.



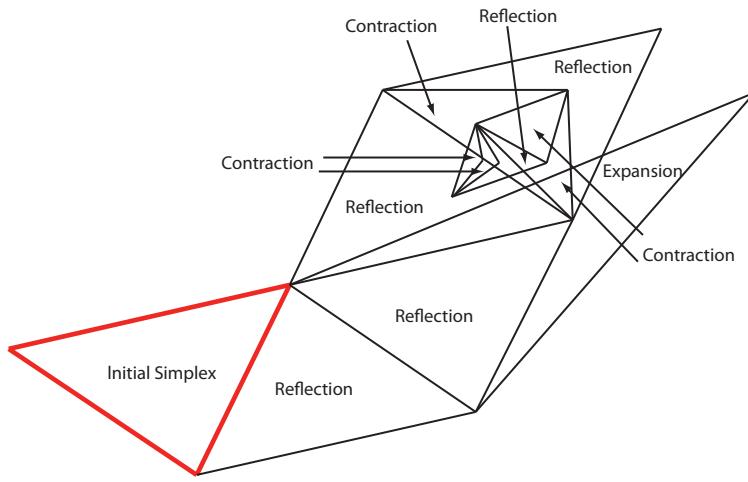
**Figure 9.7** Nelder and Mead Algorithm: The worst point is either reflected (then expanded if the reflection is successful) or contracted along a line that passes through the simplex centroid. Thick lines represent the original simplex.



**Figure 9.8** Nelder and Mead Algorithm: If reflection, expansion and contraction fail to improve the objective function, the entire simplex is contracted towards the best point.

Figure 9.9 illustrates a simplex search involving multiple reflections, extensions and contractions<sup>7</sup>.

<sup>7</sup>Modified from [http://mathfaculty.fullerton.edu/mathews/n2003/neldermead/NelderMeadMod/Links/NelderMeadMod\\_lnk\\_5.html](http://mathfaculty.fullerton.edu/mathews/n2003/neldermead/NelderMeadMod/Links/NelderMeadMod_lnk_5.html)



**Figure 9.9** Nelder and Mead Algorithm: Example trace of a simplex looking for the minimum. The initial simplex is in bold.

By successively evolving the simplex according to the previous rules, the simplex slowly makes its way along the fitness landscape<sup>8</sup>. The shape of the simplex adapts to the landscape, stretching and contracting. The simplex method can be quite successful unless the initial starting point is a very poor guess. The simplex can be assumed to have converged, either when it has converged to a very small region, or when there is no significant improvement in the error from one iteration to the next. The simplex method has the potential to find the global minimum because it can sample multiple points at once on the fitness surface. However, it can *easily* get trapped in a local minimum if the simplex is too small. Simulated annealing which is described in the next section avoids this scenario. In practice the simplex method should be repeated many times with different starting positions. The method is described in pseudocode below.

1. Let  $n$  equal the number parameters,  $x_i$  that we wish to fit
2. Set the tolerance  $\epsilon$  to decide when to exit the algorithm.
3. Initialize the parameters to random values or values that might be considered in the region of the optimum.
4.  $f(x_i)$  is the value of the objective function at  $x_i$ .
5. **Order:** Order  $f(x_i)$  low to high (low being the best):

$$f(x_1) \leq f(x_2) \leq \dots f(x_{n+1})$$

---

<sup>8</sup>See <https://www.youtube.com/watch?v=HUqLxHfxWqU> for an animated example.

6. **Centroid:** Compute the centroid of the simplex,  $\bar{x} = (\sum x_i)/(n + 1)$

7. **Reflect:** Reflect the worse vertex over the centroid to give  $x_r$  using:

$$x_r = \bar{x} + \alpha(\bar{x} - x_{n+1})$$

If  $f(x_1) \leq f(x_r) < f(x_n)$  then replace the worst:  $x_{n+1} = x_r$ . Goto step 4

8. **Expand:** if  $f(x_r) < f(x_1)$  (i.e the reflection improved things) then compute expansion:

$$x_e = \bar{x} + \gamma(x_r - \bar{x})$$

If  $f(x_e) < f(x_r)$  then (If the expansion improved things)

    replace  $x_{n+1}$  with  $x_e$ . Return to step 4

    else (Just keep the reflection)

        replace  $x_{n+1}$  with  $x_r$ . Return to step 4

9. **Contract:** If there was no improvement during the reflection, then  $f(x_r) \geq f(x_n)$  and there are two possibilities to consider.

10. **Outside:** If  $f(x_n) \leq f(x_r) < f(x_{n+1})$  then (i.e.  $f(x_r)$  is better than  $f(x_{n+1})$ )

$$x_{oc} = \bar{x} + \beta(x_r - \bar{x}), \text{ goto 13}$$

If  $f(x_{oc}) \leq f(x_r)$  then replace  $x_{n+1}$  with  $x_{oc}$  and return to step 4 otherwise goto **Shrink**

11. **Inside:** If  $f(x_r) \geq f(x_{n+1})$  then

$$x_{ic} = \bar{x} + \beta(x_{n+1} - \bar{x}), \text{ goto 13}$$

if ( $f(x_{ic}) < f(x_n)$ ), replace  $x_n$  with  $x_{ic}$  and goto 4 otherwise goto **Shrink**

12. **Shrink:** Shrink the simplex towards  $x_1$ :  $x_i = x_1 + \frac{1}{2}(x_i - x_1)$  and return to 3  
 $(i = 2, 3, \dots, n + 1)$ .

13. If  $f(x_{n+1}) - f(x_1) > \epsilon$  then return to 4, else terminate.

Implementations of the simplex algorithm are available from a number of sources. The GPL GSL library <http://www.gnu.org/software/gsl/> has an implementation. An unrestricted licence version is available from <http://www.mikehutt.com/neldermead.html>. Versions exist for most of the common computer languages.

One advantage of the simplex method is that it is not very difficult to implement (unlike the Levenberg-Marquardt algorithm, based on the author's own experience). Most scripting

languages, including Scipy for Python<sup>9</sup> and Java, have implementations available. COPASI also implements the Nelder and Mead method and Tellurium as access to the method via the Python scipi library. The method has continued to be developed with some of the more recent modifications found in Singer et al. [163].

## Simulated Annealing

The simulated annealing method derives its name from thermal physics where the minimization of  $\chi^2$  is equivalent to the way a system, such as a metal, reaches its lowest state as it slowly cools [88]. At a given temperature, the atoms of the metal collide with each other so that the energy of the system is continually being redistributed. As the temperature is slowly reduced, the atoms begin to form a crystalline structure and eventually reach the minimum energy state. The metal has to be cooled slowly, or else pockets, where the metal is in a higher energy state than neighboring regions, can form. The idea has been used to implement an optimization algorithm called simulated annealing. For optimization problems the algorithm works in the following way: given an initial state  $i$ , which in our case would be a set of parameters, the system jumps to another state  $i + 1$ , with the Boltzmann probability:

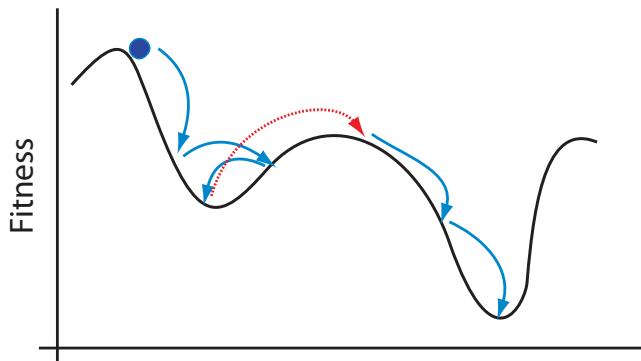
$$\exp \frac{(\chi_i^2 - \chi_{i+1}^2)}{T} \quad (9.14)$$

where  $T$  is the *temperature*. For example, if  $\chi_{i+1}^2$  is lower than  $\chi_i^2$ , the expression will always be greater than one, so we will always jump to the new solution. However if  $\chi_{i+1}^2$  is bigger than  $\chi_i^2$ , the probability of accepting the new state is less than one and it is possible to actually accept the worse state, effectively going uphill. Going uphill may seem counterproductive, but it allows the algorithm to potentially jump out from local minima and eventually find the global minimum. The higher the temperature the more likely the algorithm will move uphill, therefore the temperature is slowly lowered so that the chance of going uphill reduces.

At a given temperature the system must be given enough time to sample all the configurations which are accessible using equation (9.14). There is no simple way to design temperature scheduling (i.e. temperature as a function of time/iterations) and several methods exist depending on the problem at hand. One way is to follow the algorithm as described in [110, 138], where the authors consider an adaptation of the simplex method. The author's own experience with this approach has been successful. COPASI implements a version of simulated annealing.

---

<sup>9</sup><http://docs.scipy.org/doc/scipy/reference/tutorial/optimize.html>



**Figure 9.10** In the simulated annealing algorithm, jumps can be towards or away from a minimum. This allows the algorithm to move away from local minima.

The basic simulated annealing algorithm is described in the code below:

```

1. Initialize parameter values,  $\mathbf{p}$ 
2. Initialize the temperature,  $T$ 
3. Calculate the chi-square,  $\chi^2_i$ , at the
   current parameter values  $\mathbf{p}_i$ 
4. Make small random changes,  $\Delta\mathbf{p}$  to  $\mathbf{p}_i$ 
5. Set  $\mathbf{p}_{i+1} = \mathbf{p}_i + \Delta\mathbf{p}$ 
6. Calculate the new chi-square,  $\chi^2_{i+1}$ 
7. Calculate  $\Delta\chi^2 = \chi^2_{i+1} - \chi^2_i$ 
8. If  $\Delta\chi^2 \leq 0$  then accept the new state
9. If  $\Delta\chi^2 > 0$  then
   Generate uniform random number,  $u$ 
   If  $u < e^{-\Delta\chi^2/T}$  then
     Accept state
   else
     Restore previous state,  $\mathbf{p}_i$ 
20. Reduce the temperature,  $T = T - \varepsilon_T$ 
21. If  $T < 0$  or exceeded Max Iterations then
    exit
21. Goto to Step 3.

```

The GPL GSL library<sup>10</sup> has an implementation of simulated annealing and has been successfully used by the author. An unrestricted licensed version in C# is available<sup>11</sup> and a C version<sup>12</sup>.

<sup>10</sup><http://www.gnu.org/software/gsl/>

<sup>11</sup><http://www.codeproject.com/Articles/13789/Simulated-Annealing-Example-in-C>

<sup>12</sup><http://www.cs.sunysb.edu/~skiena/algorist/book/programs/>

## Genetic Algorithm

A genetic algorithm (GA) is an optimization technique that mimics natural evolution. GAs are motivated by natural biological processes such as selection, crossover and mutation. The Schema theorem of Holland [57] addresses these intuitive notions, and demonstrates that these operations serve to increase the fitness of a population. In our case we will consider real value coded GAs, where the ‘genes’ are real with nonnegative kinetic parameters. We start with a random population of individuals where an individual is a model with a given set of parameters, i.e. ‘genes’. The fitness of each individual in the population is measured by its chi-square value. Various approaches are employed to decide which individuals will be carried over to the next generation. Only a proportion of the population survives this transition so the population needs to be rebuilt back to its original size. It is the process of rebuilding, via replication and mutation of the survivors, that results in new individuals. Such new individuals could, by chance, have improved fitness. This process repeats over a number of generations. COPASI implements a genetic algorithm optimization method. Interestingly there doesn’t appear to be genetic algorithm implemented in the Python `scipy` package. In the authors experience, writing effective genetic algorithm software is not easy and the success is markedly dependant on how the algorithm is implemented as there are many possible variants to the approach.

## Selection

There are various ways in which selection can takes place, these include elitism, tournament selection, or roulette wheel selection. In tournament selection random pairs of individuals are made to play a tournament and the winner is decided based on which is fitter. This ensures that even bad individuals can get selected into the next generation and helps prevent premature convergence. Elitism is where the top 20% or more of the fittest individuals are passed on to the next generation. Roulette selection is where the probability of picking out an individual from the population is based on the fitness of the individual. One or more of these strategies can be used to pick the next generation.

## Crossover

Some GAs use crossover as a means to shuffle variation between individuals in a population. In crossover, two parents exchange genetic material. This mechanism offers the chance to bring two favorable traits together into one individual. Crossover also serves to spread beneficial mutations over a population.

The selected parents can be crossed over [70] using an arithmetic mean defined in the following way. Assuming we represent the parents as:

$$p_1 = (p_1^1, p_1^2, p_1^3, \dots) \quad \text{and} \quad p_2 = (p_2^1, p_2^2, p_2^3, \dots)$$

where the  $p_i^j$  term is related to the  $j^{\text{th}}$  parameter in the  $i^{\text{th}}$  parent. The crossover between  $p^1$  and  $p^2$  will generate two children,  $\beta_1$  and  $\beta_2$ , such that:

$$\begin{aligned}\beta_1^i &= \lambda_i p_1^i + (1 - \lambda_i) p_2^i \\ \beta_2^i &= \lambda_i p_2^i + (1 - \lambda_i) p_1^i\end{aligned}\quad (9.15)$$

where  $\lambda$  is a uniform random number between -0.5 and 1.5. The wider range allows a larger region of parameter space to be explored as new points may lie outside the line joining the parents.

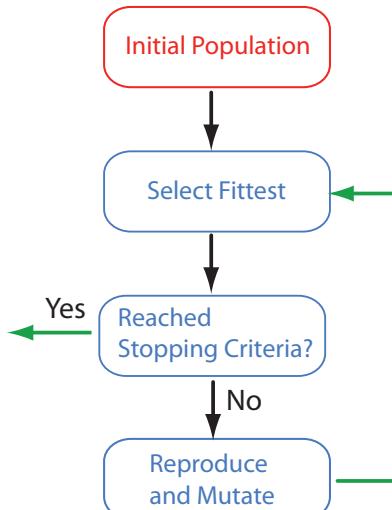
## Mutation

Mutation is a vital part of any GA as it is the one technique that allows entirely new traits to enter into the population. For a random number of individuals, one parameter,  $p^i$ , will be randomly selected and changed according to:

$$p^i = z p_{\max}^i \quad (9.16)$$

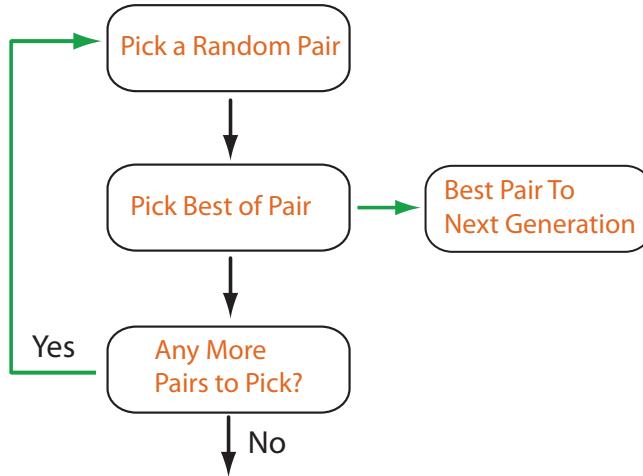
where  $z = \text{random } [0, 1]$ , is uniformly distributed, and  $p_{\max}^i$  is the maximum possible value of the  $i$  component of the parameter set. We must of course ensure that  $p_{\max}^i$  is finite.

## General Scheme



**Figure 9.11** Basic flowchart for a genetic algorithm, though many variants exist.

### Select Fitness: Tournament Selection



**Figure 9.12** Tournament selection is one strategy used for selecting individuals for the next generation.

An example scheme is shown in Figure 9.11 and a flowchart describing tournament selection is given in Figure 9.12.

The operations of crossover and mutation occur with certain adjustable probabilities. However the mutation rate is generally a small number,  $< 0.05$ . Mutations allow the system to explore new regions, whereas crossovers spread these mutations over the population. If the mutation rate is very high, large regions will be explored. However individuals may not survive into the next generation because the search is much too exploratory and not enough information about the landscape has been exploited by crossovers.

Once there is little improvement in the fitness from one generation to the next, the computation can be stopped. COPASI implements a range of genetic algorithms, including the more effective genetic algorithm with stochastic ranking. COPASI also implements variants such as evolutionary programming and evolutional strategy (SRES) approaches (consult the COPASI documentation for details). The `scipy` Python package supports one related genetic algorithm like approach called differential evolution which we describe next.

## Differential Evolution

The final optimization method to discuss is the differential evolution (DE) algorithm. Like the genetic algorithm optimization method, DE is an evolutionary type method developed by Storn and Price in 1996 [168]. DE has a number of key advantages, it is relatively simple to implement, it is fast, does not use derivatives, and is easily parallelized. DE uses mutation as the search mechanism by linearly combining individuals in a population in an

attempt to create fitter offspring. This results in a remarkably effective method.

The basic algorithm is (Also shown in Figure 9.13):

1. Create space for two populations, one a working population and a second temporary store for recording new individuals **in** the population loop.
2. Create a working population **where** individuals have randomly assigned parameter values.
3. **Start** a generation loop.
4. **Start** a loop for all  $i$  individuals **in** the working population
5. Create an  $i^{\text{th}}$  mutant linear combination **from** three randomly selected individuals.
6. Copy the  $i^{\text{th}}$  mutant into the temporary population store depending on random crossover, else copy over the  $i^{\text{th}}$  individual **from** the working population.
7. Continue to the next individual **in** the population.
8. Copy the temporary store to the working population.
9. Next generation, stop **if** fittest is better than a threshold value.

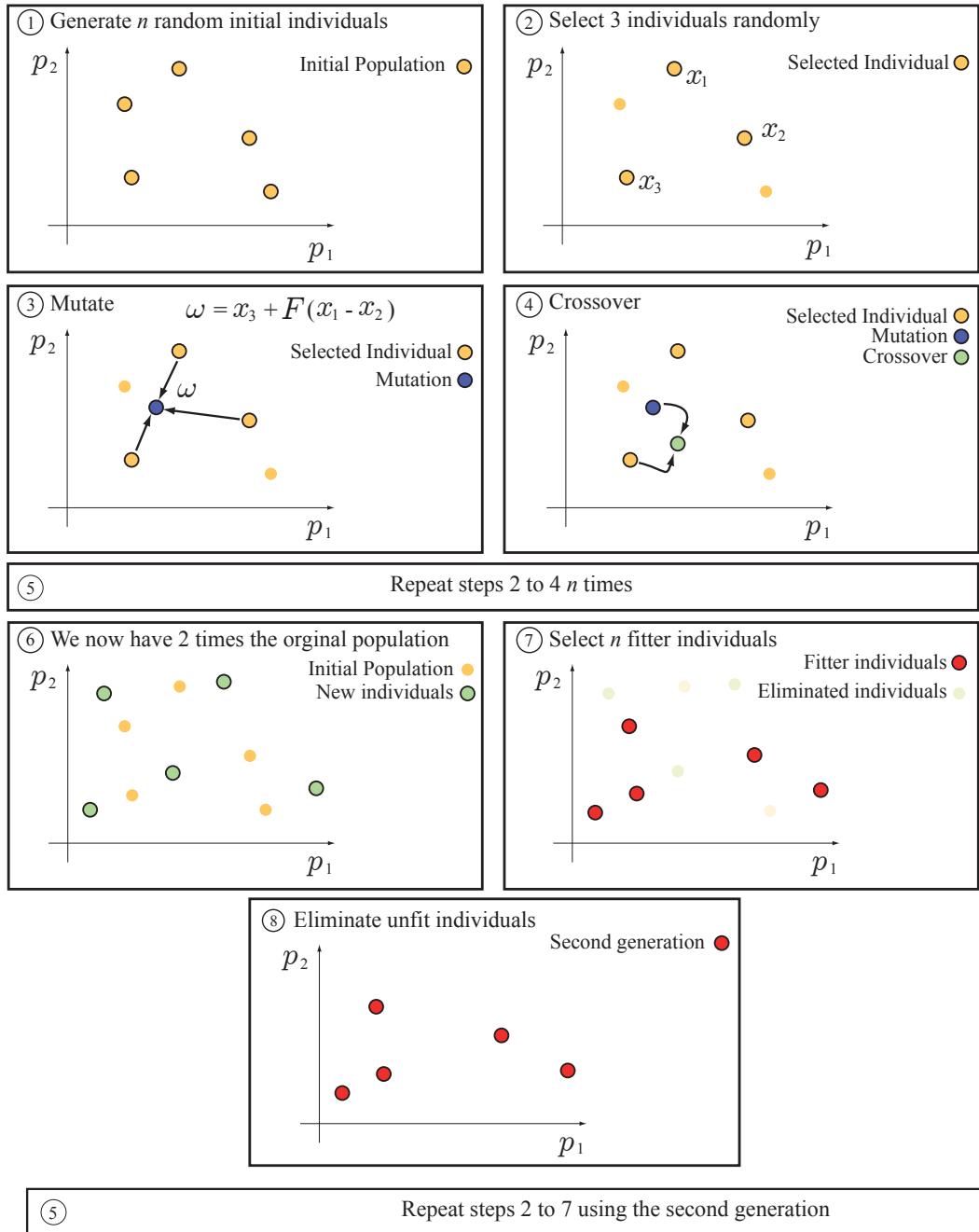
In this scheme an individual in a population is a vector containing the values for the parameters of the model. If for example a model has twelve parameters to fit, then an individual is a vector of size twelve. The key to the method is computing a linear combination of individuals to produce a new candidate, or mutant individual. The linear combination is given by:

$$\text{new candidate} = x_1 + F(x_2 - x_3)$$

where  $x_1$ ,  $x_2$  and  $x_3$  are individuals randomly drawn from the population and  $F$  is called the mixing factor. In practice the  $x$  terms are vectors so that the linear combination is a vector calculation. The second part of the algorithm must decide whether the mutant will be passed to the next generation or not. This is done by combining, via crossover, the mutant with another individual from the population. In practice the algorithm iterates through every individual in the population, for example at the  $i^{\text{th}}$  individual, create a linear combination to generate a mutant and combine the mutant via crossover with the  $i^{\text{th}}$  individual. If there is no crossover the  $i^{\text{th}}$  individual is kept for the next generation. All mutants that went through crossover are compared to the corresponding  $i^{\text{th}}$  individual in the original population, if the crossover individual is fitter then it is copied to the next generation, otherwise the original is kept.

There are a number of variations on the basic DE algorithm. One variant is to use islands. This allows multiple populations to evolve independently with a limited degree of migration between islands and allows alternative solutions to be explored while at the same time solutions may be merged leading to further improvements. In the appendix (Listing 9.4) a Python implementation of island based differential evolution is given and an example that uses the code to fit an oscillatory model is provided in the example section.

A more detailed description of the algorithm is described in the listing shown below. Rules



**Figure 9.13** Outline of the Differential Evolution Algorithm.

of thumb have been devised for setting mutation and cross-over probabilities and these are listed below and in the Python code in the appendix (Listing 9.4).

```

1. Set crossover rate: CR = 0.6
2. Set the mixing factor: F = 0.8
3. Set number of parameters = P
4. Create an initial random population of individuals
5. Create space,  $u$  to hold the potential new population
6. Iterate through each individual,  $x_i$ , in the population
7. Pick three unique individuals,  $a$ ,  $b$  and  $c$  from the population
   ( $i$  should not be a choice for the random number)
8. Compute a mutated individual using: mutant =  $c + F(a - b)$ 
9. Compute a trial individual,  $u$ , from the mutant by crossover:
   Generate a randomIndex between 1 and P
   For each parameter,  $j$ 
      Generate a uniform random number,  $r_j$ 
      if  $r_j \leq CR$  or ( $i = randomIndex$ )
          $u_{ij} = mutant_i$ 
      else
          $u_{ij} = x_i$ 
10. Continue to the next individual  $x_{i+1}$ , in the population
11. For all  $i$ , if the fitness( $u_i$ ) > fitness( $x_i$ ) then  $x_i = u_i$ .
12. Increment the generation number
13. Sort the population, is the fittest individual less than tolerance?
    No, then goto 4

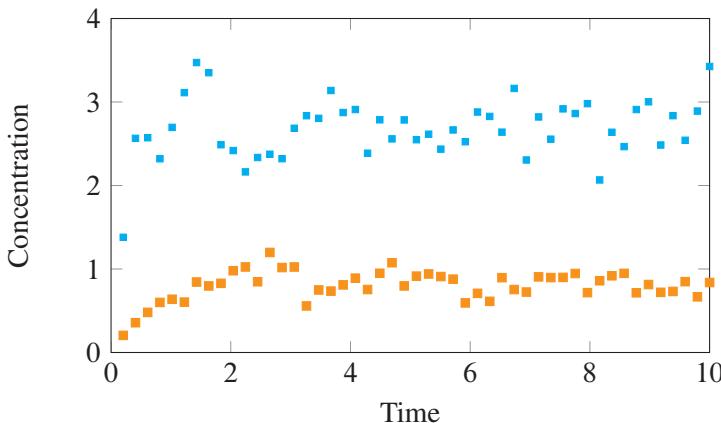
```

Although relatively simple, differential evolution has turned out to be a remarkably effective approach. Of all the methods described, differential evolution and simulated annealing are probably the most effective. As with all optimizer methods, they should be repeated many times in order to try to locate the global minimum.

## Combining Global and Local Search

Some optimization methods are considered local, whereas others are global. The Levenberg-Marquardt is considered a local method because given a starting point, it can usually only find the nearest minimum. Other methods such as genetic algorithms or simulated annealing are considered global because they tend to search across the entire fitness landscape, sampling many regions.

Combining a local search within a global search algorithm is a very attractive possibility. A global search can be used to provide an initial seed point. The search will converge to a point where a local search can then begin. One strategy is to conduct a global search and take the two best individuals and use them as initial conditions for a local search. The two individuals resulting from the local search are put back into the main population and the entire procedure starts again. This can be repeated until the population has converged.



**Figure 9.14** Simulated experimental data.

## Optimization Example

To illustrate the use of optimization, let us consider fitting data to a simple model that displays oscillatory behavior. We will use a simple oscillator model which arises from positive feedback and is based on the Heinrich model [67]. The model was simulated and random noise added to the time series to produce noisy data which we will call the experimental data. This data is shown in Figure 9.14. Noise was drawn from a Gaussian distribution with mean zero and a standard deviation of 10% of the  $y$  value.

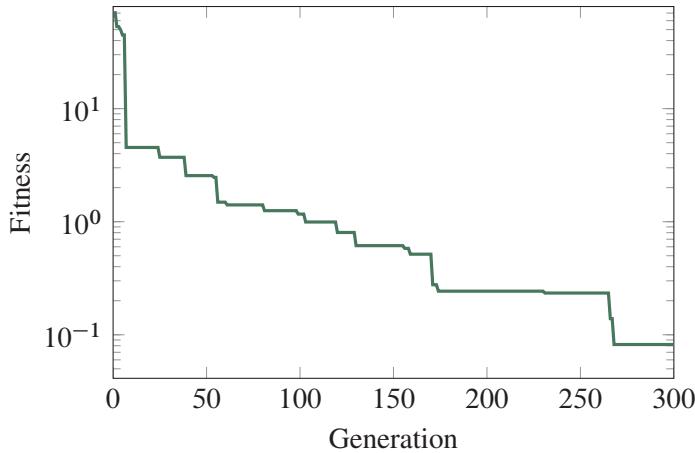
The following model was fitted to the experimental data:

$$\begin{aligned}\frac{dy_1}{dt} &= p_0 - y_1 - (p_1 y_1)(1 + p_4 (y_2)^4) \\ \frac{dy_2}{dt} &= (p_1 y_1)(1 + p_4(y_2)^4) - y_2 p_6\end{aligned}$$

Differential evolution was used to fit the model because it was found to be the most effective method for this problem. It is worth noting that the Levenberg-Marquardt method was unable to fit this model to the data. A genetic algorithm based optimizer could fit the model, but only about 20% of the time.

In the differential evolution method, parameters were randomly assigned between 0 and 10 for each individual in the population. Only thirty individuals for the population were needed to achieve a successful fit. Four parameters were fitted,  $p_0$ ,  $p_1$ ,  $p_4$  and  $p_6$ . The Python code for this computation is given in the Appendix at the end of the chapter (Listing 9.4).

Figure 9.15 shows the progress of the fitness during the optimization. The smaller the number, the better the fit. At the beginning the fitness dropped rapidly. The fitness data was plotted on a log y-axis to show the number of smaller steps taken near the end of the optimization. The result of the fit, superimposed on the simulated experimental data, is



**Figure 9.15** Plot showing the progress of the optimization using differential evolution. Smaller numbers indicate a better fit. The fitness was plotted on a log axis to illustrate the very small number of steps that were taken after the initial large drop in fitness.

shown in Figure 9.16. The fit has managed to capture the correct dynamics even though the data is very noisy. As mentioned before other optimization algorithms such as Levenberg-Marquardt and even the genetic algorithm found this model difficult to optimize. In practice, a variety of fitting methods should be tried until a satisfactory fit is obtained. Sometimes a method that works well in one data set can fail on data sets for other models.

### 9.3 Model Fitting Software

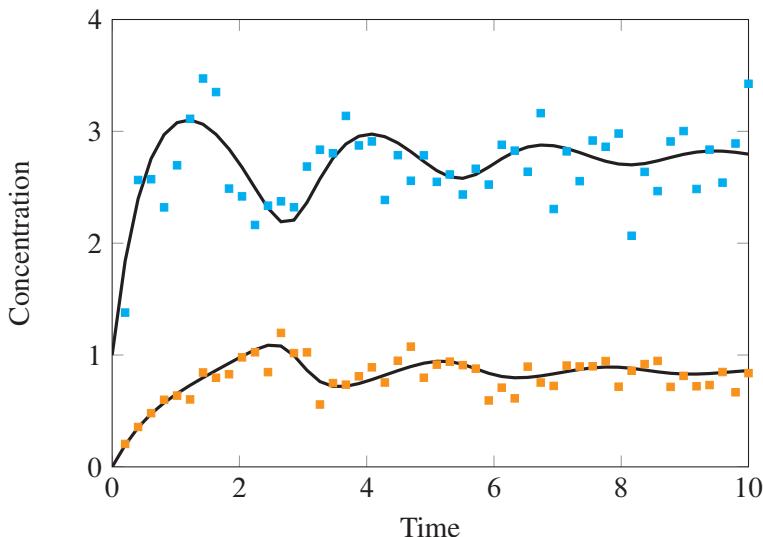
There are a number of biochemical modeling applications that support curve fitting of differential equation models. The most popular tools in this category (in alphabetic order) include COPASI<sup>13</sup>, PottersWheel<sup>14</sup>, SBSI<sup>15</sup>, and VCell<sup>16</sup>. COPASI in particular has an extensive set of parameter fitting algorithms and a number have been mentioned already. For Matlab users, PottersWheel is probably the best choice. There is also a long standing set of software and parameter optimization code by Peter Kuzmic who can provide expert consultancy on parameter fitting (<http://www.biokin.com/>).

<sup>13</sup><http://www.copasi.org/>

<sup>14</sup><http://www.potterswheel.de/>

<sup>15</sup><http://www.sbsi.ed.ac.uk/>

<sup>16</sup><http://www.vcell.org/>



**Figure 9.16** Simulated experimental data and fitted curve: Fitted parameters (actual in brackets):  $p_0 = 6.77(7)$ ,  $p_1 = 1.01(1)$ ,  $p_4 = 1.26(1)$ ,  $p_6 = 5.11(4.96)$ .

## 9.4 Using Python to Fit Data

Python has good support for optimization and data fitting via the SciPy<sup>17</sup> extension. SciPy supports the optimize package<sup>18</sup> which in turn implements a number of optimization algorithms including the Nelder and Mead simplex and Levenberg-Marquardt. A script that fits a simple model is shown in Listing 9.1 where a Michaelis-Menten equation is fitted to data.

```
from scipy import *
from scipy import optimize
# Declare the experimental data
x = array([0, 10, 20, 50, 100, 200, 400])
y = array([0, 9, 10, 17, 18, 20, 19])

# Define the objective function
def residuals (p):
    [vmax,Km] = p
    return y - vmax*x/(Km+x)

# Fit the model to the data
output = optimize.leastsq (residuals, [10, 10])
```

**Listing 9.1** Python Script to Fit Data.

<sup>17</sup> [www.scipy.org](http://www.scipy.org)

<sup>18</sup> <http://docs.scipy.org/doc/scipy/reference/tutorial/optimize.html>

When the script is executed, the variable `output` will contain the values for the fitted  $V_{\max}$  and  $K_m$  which in this case is 20.745 and 15.408, respectively. The  $y$  data was generated using a  $V_{\max} = 20$  and  $K_m = 15$  with added noise to simulate experimental error.

One important point worth noting is that the `leastsq` routine expects a routine called `residuals` to return the differences between the data and the model. In other words, there is no need to square and sum up the residuals to compute the chi-square directly. In general, the `residuals` routine will compute the following component of the sum of squares:

$$\frac{y_i - f(x_i, p)}{\sigma}$$

We can go further and plot the results of the fit using the code in Listing 9.2:

```
def peval(x, p):
    return p[0]*x/(p[1]+x)

Vmax, Km = 20, 15
yTrue = Vmax*x/(Km+x)

import matplotlib.pyplot as plt
plt.plot(x, peval(x, output[0]), '--', x, y, 'o', x, yTrue,
          'r', x, residuals(output[0]), 'r^', markersize=10)
plt.title('Least-squares fit to noisy data')
# loc=10 means center the legend
plt.legend(['Fitted Curve', 'Noisy Data',
           'Underlying Function', 'Residuals'], loc=10)
plt.show()
```

**Listing 9.2** Python Script to Fit Data.

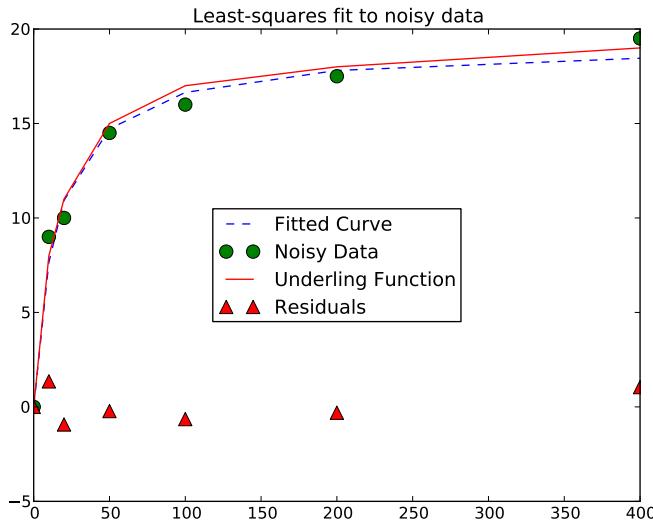
The Python `leastsq` uses a modified Levenberg-Marquardt algorithm from the minipack `lmdif` routine.

### Additional Example - Fitting a Model

The previous example showed how to fit a function to some data. This is the kind of thing you might do when fitting an enzyme kinetic rate laws to experimental data. Here we will show how to fit a complete model using Tellurium and the differential evolution algorithm provided by Scipy<sup>19</sup>

We begin with the hypothetical model shown in Figure 9.18.  $X_o$  and  $X_1$  are boundary species and are therefore fixed. The model has six reactions with two feedback regulation loops. The model which we will consider the ‘ground truth’ is given in Figure 9.3.

<sup>19</sup>I wish to acknowledge Vernonia Porubsky, a graduate student in my lab at UW, for assisting me with this example.



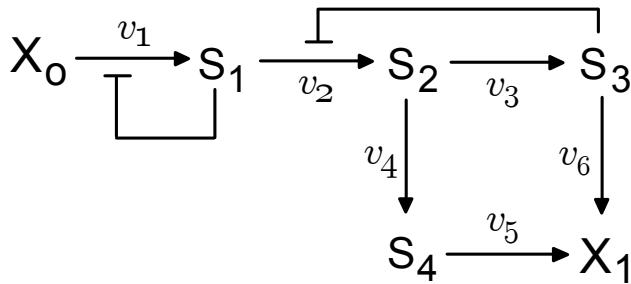
**Figure 9.17** Results from Python fitting code, comparing the fitted model to the underlying actual model.

```
r = te.loada("""
# Reactions
J1: $X0 -> S1; (Vm*X0)/((Km + X0) + (S1/KI));
J2: S1 -> S2; k2*S1/(1 + S3/KI2);
J3: S2 -> S3; k3*S2;
J4: S2 -> S4; k4*S2;
J5: S4 -> $X1; k5*S4;
J6: S3 -> $X1; k6*S3;
# Species initializations
$X0 = 10; S1 = 0; S2 = 0; S3 = 0; S4 = 0
$X1 = 10
# Parameters:
Vm = 15; Km = 5; KI = 10; k2 = 5; k3 = 10
k4 = 5; k5 = 15; k6 = 5; KI2 = 10.1
""")
```

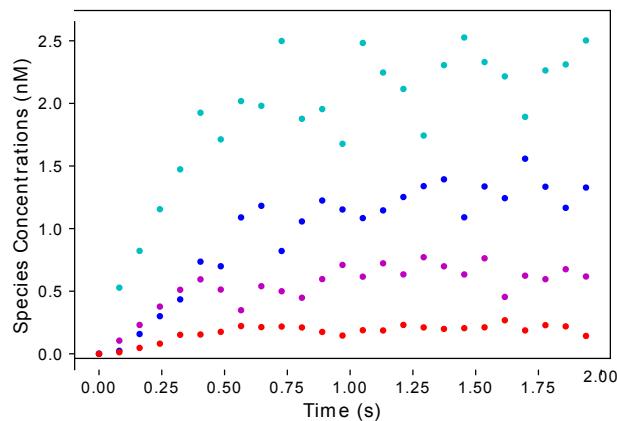
**Listing 9.3** Model using to illustrate fitting using Python and Tellurium.

Using the ground truth model a set of ‘experimental’ data was generated by adding 15% Gaussian noise to simulation results. A plot of the ‘experimental data’ can be found in Figure 9.19.

Figure 9.20 shows the fitness measures as a function of generation number. It shows a typical patterns where improvements to the fit occur in jumps followed by pauses. Even



**Figure 9.18** Model used to illustrate fitting using Python and Tellurium.  $X_o$  and  $X_1$  are boundary species.



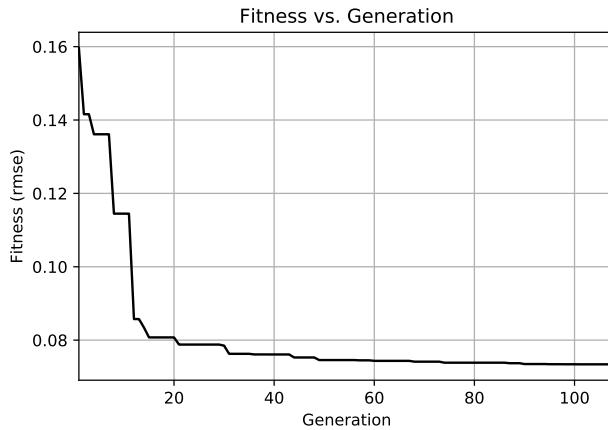
**Figure 9.19** Raw data for fitting model. Generated from the ground truth model (Listing 9.3) with 15% added Gaussian noise.

within fifty generation a reasonable fit has been achieved. Figure 9.21 plots both the ground truth, the ‘experimental data’, the fit itself (dotted lines), and the residuals. Table ??shows the resulting fitted parameters along side the ground truth values.

Average = 15.6369400988 Average = 5.42260571926 Average = 10.0716078035 Average = 5.03409905233 Average = 12.6214671201 Average = 9.93120589225 Average = 5.0147594914 Average = 15.1653422057 Average = 5.38234612932

## Further Reading and Online Resources

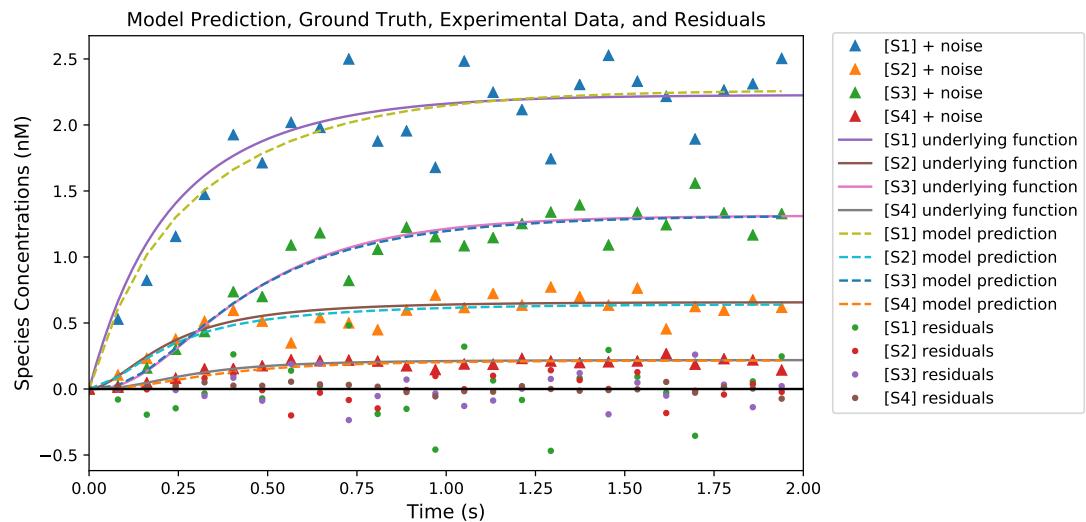
1. Berendsen HJ. (2011) A Student’s Guide to Data and Error Analysis. Cambridge University Press. ISBN: 978-0-521-13492-7



**Figure 9.20** Fitness as a function of generation time for the scipy Differential Evolution optimizer using the model shown in listing 9.3.

Parameter	Fitted value	True Value
$V_m$	12.81	15
$K_m$	2.81	5
$K_I$	14.93	10
$k_2$	4.99	5
$K_{I2}$	14.19	10.1
$k_3$	10.24	10
$k_4$	4.72	5
$k_5$	14.22	15
$k_6$	5.68	5

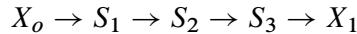
2. Draper NR and Smith H (1998) Applied Regression Analysis. 3rd edition. Wiley Series on Probability and Statistics. ISBN-13: 978-047117082
3. Johnson ML, Faunt LM (1992) Parameter estimation by least-squares methods. Methods in Enzymology, 210, 1-37.
4. Johnson ML (1994) Use of Least-Squares Techniques in Biochemistry. Methods in Enzymology, 240, 1-22.
5. Mendes P and Kell DB (1998) Parameter Estimation in Biochemical Pathways: A Comparison of Global Optimization Methods. Bioinformatics, 14(10), 869-883.



**Figure 9.21** Result of using differential evolution to fit the nine parameters found in the model shown in listing 9.3.

## Exercises

1. Create a simple linear chain model of four steps and three species:



Assume that  $X_o$  and  $X_1$  are boundary species. Choose nonlinear reversible rate laws for the reactions, assign suitable values to the parameters, and set the initial conditions for  $S_1$ ,  $S_2$ , and  $S_3$  to zero. Run a simulation to obtain time-course data for the three species. Add noise to the simulated data and treat this data as your ‘experimental data’. Fit the experimental data to the model and see how well your parameter estimates agree with the original model. Try different fitting methods such as simplex, Levenberg-Marquardt, etc. to investigate how well each method performs. In addition, investigate different starting points for the parameter values when using the Levenberg-Marquardt and Nelder and Mead Method.

## Appendix

```
# Written and supplied by Wilbert Copeland, 2014
# Differential evolution code, used to fit the Henirich model (Figure 8.15)
import random
import sys
```

```

from datetime import datetime
import scipy.integrate
import numpy
import matplotlib.pyplot as mplot

"""

Best values:
ObjFunc = Sum of Differences Squared, Generations = 2000
Fitness: 0.33 Vector: [8.0227418, 0.9762534, 98.806163,
                      732.4952511, 1.0336005, 108.808089, 5.0047913]

ObjFunc = Percent Error, Generations = 250
Fitness: 0.076 Vector: [8.0497271, 0.9955929, 2.8358877, 37.7531871,
                        1.0162701, 130.1762945, 5.0427175]

"""

# Expected fitted values:
# 8, 1, 0, 1, 1, 2.2, 5

class DiffEvo(object):
    def __init__(self, dy, y0, t, expected):
        self.DY = dy
        self.Y0 = y0
        self.T = t
        self.Expected = expected
        self.Islands = []
        return

    def CreateIsland(self, population_size, vector_length, min_val, max_val):
        island = []
        for i in range(population_size):
            island.append(self.CreateRandomMember(vector_length, min_val, max_val))
        self.Islands.append(island)
        return

    def CreateRandomMember(self, vector_length, min_val, max_val):
        v = [round(random.uniform(min_val, max_val), 7) for i in range(vector_length)]
        f = self.GetFitness(v)
        return Member(v,f)

    def CreateTrialMember(self, original, samples, CR=0.6, F=0.8):
        o = original.Vector
        a = samples[0].Vector
        b = samples[1].Vector
        c = samples[2].Vector

        new_vector = []
        for i in range(len(o)):
            if random.random() <= CR:
                v = round(a[i] + F * (b[i] - c[i]), 7)
            else:
                v = round(o[i], 7)
            new_vector.append(v)
        return Vector(new_vector)

```

```

        if v>0:
            new_vector.append(v)
        else:
            new_vector.append(o[i]/2.)
    else:
        new_vector.append(o[i])
new_fitness = self.GetFitness(new_vector)
return Member(new_vector, new_fitness)

def GetFitness(self, vector):
    obs = scipy.integrate.odeint(self.DY, self.Y0, self.T, args=(vector,), mxstep=1000)
    sum_of_squares = 0.
    for i in range(len(obs)):
        for j in range(len(obs[i])):
            if self.Expected[i][j+1] == 0:
                0.
            else:
                sum_of_squares += ((obs[i][j] -
                                     self.Expected[i][j+1])/ self.Expected[i][j+1]) ** 2
    return sum_of_squares

def SortIslandsByFitness(self):
    for island in self.Islands:
        island = sorted(island, key=lambda o: o.Fitness)
    return

class Member(object):
    def __init__(self, vector, fitness):
        self.Vector = vector
        self.Fitness = fitness
    return

# Heinrich model using used in main text
def dY(y, t, p):
    dy0 = p[0] - y[0] * 1. - (p[1] * y[0] - 0. * y[1]) * (1. + p[4] * y[1] ** 4)
    dy1 = (p[1] * y[0] - 0. * y[1]) * (1. + p[4] * y[1] ** 4) - y[1] * p[6]
    return [dy0, dy1]

def PlotResults(de):
    best_members = [x[0] for x in de.Islands]
    solution = scipy.integrate.odeint(de.DY, de.Y0, de.T, args=(best_members[0].Vector,))

    exp_t = [de.Expected[i][0] for i in range(len(de.Expected))]
    exp_y0 = [de.Expected[i][1] for i in range(len(de.Expected))]
    exp_y1 = [de.Expected[i][2] for i in range(len(de.Expected))]

    obs_y0 = solution[:,0]
    obs_y1 = solution[:,1]

    mplot.plot(exp_t, exp_y0, 'bo')
    mplot.plot(exp_t, exp_y1, 'go')
    mplot.plot(de.T, obs_y0)

```

```

mplot.plot(de.T, obs_y1)
mplot.xlabel('Time')
mplot.ylabel('Unknown')
mplot.show()
return [exp_t, obs_y0, obs_y1]

# Read file
file = open('expdata.txt','r')
data = []
for line in file:
    d = line.replace("\n").split(' ')
    d = [float(x) for x in d]
    data.append(d)

# Differential evolution
GENERATION_COUNT = 0
MAX_GENERATIONS = 200
FITNESS_THRESHOLD = 1e-6
PARAMETER_COUNT = 7

NI = 1      # Number of islands
MF = 0.45   # Migration frequency
NM = 1      # Number of migrants
SP = 3      # Selection policy = Randomly choose one of the top 3 for migration.
RP = 3      # Replacement policy = Randomly choose one of the top 3 for migration.
MT = range(NI)[1:] + [0]      # Migration topology: Circular, unidirectional

#random.seed (4532)

print('Starting DE search.')
clock = datetime.now()

# Initialize Diff Evo routine
DE = DiffEvo(dy=dY, y0=[1., 0.], t=numpy.linspace(0., 10., 50), expected=data)

# Create islands
[DE.CreateIsland(30, 7, 0., 10.) for i in range(NI)]
for island in DE.Islands:
    assert len(island) > 3

while True:
    GENERATION_COUNT += 1

    for island in DE.Islands:
        samples = [random.sample(island, k=3) for i in range(len(island))]
        trial_values = [DE.CreateTrialMember(island[i], samples[i]) for i in range(len(island))]

        for i in range(len(island)):
            if trial_values[i].Fitness < island[i].Fitness:
                island[i] = trial_values[i]

```

```
DE.SortIslandsByFitness()

top_members = [x[0] for x in DE.Islands]
for member in top_members:
    print('Fitness: {}'.format(round(member.Fitness,3)))

if GENERATION_COUNT >= MAX_GENERATIONS or
    min([x[0].Fitness for x in DE.Islands]) < FITNESS_THRESHOLD:
    top_members = [x[0] for x in DE.Islands]
    for member in top_members:
        print('Fitness: {} Vector: {}'.format(round(member.Fitness,3), member.Vector))
    break

print('Done optimizing.')
print('Optimization time: {}'.format(datetime.now() - clock))

PlotResults(de=DE)

print('Done.)
```

**Listing 9.4** Differential Evolution Code by Wilbert Copeland, Dept of Bioengineering, 2014. User will need to supply a text file containing columns that represent time and the two variables



# 10

## *Parameter Estimation*

---

### 10.1 Introduction

---

The last chapter discussed the use of various optimization techniques to fit a model to a set of data. In this chapter we will go one step further and investigate ways to assess the quality of the fit and confidence in the parameters.

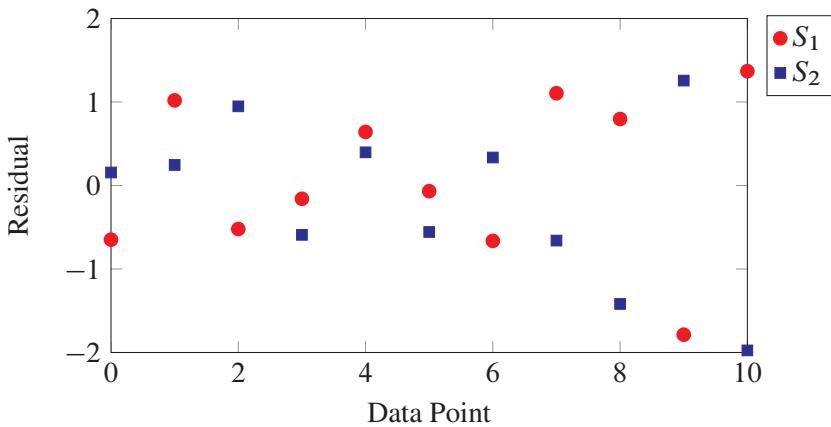
---

### 10.2 Analysis of Residuals

---

When a model has been fit to a set of data, the difference between a model prediction and the corresponding experimental data point is called the **residual**. Figure 9.2 in the previous chapter shows a plot of a fitted curve and the corresponding experimental data used to fit the model. The  $e_i$  terms are the residuals. One of the easiest ways to assess the quality of a fit is to examine the residuals. In fitting data to a model we make a number of assumptions (as required by Maximum likelihood, see section 9.1) about the experimental data. The first is that the experimental uncertainties in the data are normally distributed, and the second that the errors are uncorrelated (i.e. independent). Any departure from these assumptions means the residuals will show a pattern that is not accounted for by the model. Even assuming that the experimental data is well behaved, an incorrect model can also result in systematic trends in the residuals. Examination of the residuals is therefore an easy and informative way to assess the fit.

Figure 10.1 shows a typical residual plot. This was obtained by fitting the model  $S_1 \rightarrow S_2$ , assuming an irreversible first-order reaction kinetic law with a single unknown kinetic con-



**Figure 10.1** Residual plot data fitted to a simple irreversible decay model of  $S_1$  into  $S_2$ .

stant,  $k_1$ . For the purposes of the demonstration, the model was fitted to a set of synthetic noisy data generated from a simulation using a known value of  $k_1$ . The synthetic data was then used to recover the value of  $k_1$  by using the Levenberg-Marquardt fitting method. If the model is a good fit to the data, we expect the residuals to be normally distributed about a mean of zero.

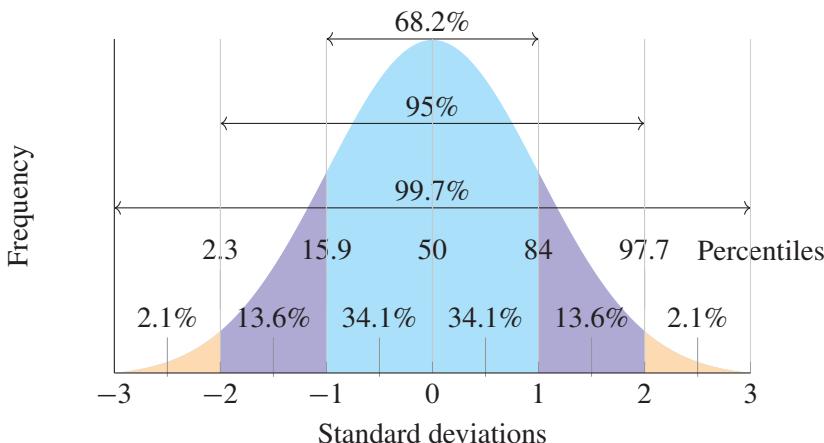
If the model is a good fit to the data, we expect the residuals to be normally distributed about a mean of zero.

If a pattern is observed in the residual plot, it can mean a number of things: 1) the model is incorrect; 2) one or more assumptions about the distribution of errors in the experimental data is violated; 3) there may be one or more very unusual outliers in the experimental data set. The plot in Figure 10.1 hints at a possible pattern with the residuals increasing, suggesting the variance in the experimental data is increasing. The most damning patterns are where the residuals follow a linear or nonlinear relationship and are not randomly distributed. This will usually imply an incorrect model.

Plotting residuals is therefore an effective way to investigate how well the model fits the data. Sometimes it can be difficult to spot trends in a residual plot and one very practical way to test whether the residuals are normally distributed (implying no systematic pattern) is to construct a **normal probability plot** [121, 169]. We can construct a probability plot by first ranking the residuals in increasing order such that:

$$e_1 < e_2 < e_3 < \dots < e_i < \dots < e_n$$

where  $e_i$  is the  $i^{th}$  residual value. We next compute the percentile value for the  $i^{th}$  residual



**Figure 10.2** The normal distribution showing percentiles, z-scores and areas. Modified from <http://johncanning.net/wp/?p=1202>.

using:

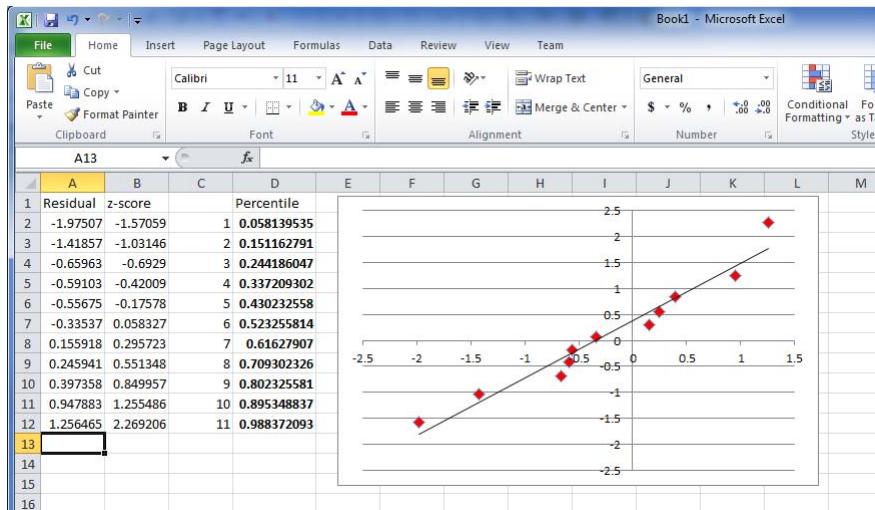
$$P_i = \frac{i - 0.5}{n} \quad (10.1)$$

For example, if we have 20 ranked residuals, then the percentile for the 10th residual is 0.475. That is, 47.5% of the residuals fall below the 10th residual. We can think of the percentiles as representing the cumulative area (or probability) under a normal curve (or any other distribution we wish to test) from which we can compute the corresponding z-values. That is, for a given area under a normal curve with mean zero, what is the value on the  $x$  axis?

For example, an area of 0.5 will yield a z-score of zero because we are at the center of the normal curve, while an area of 0.25 will give a z-score of -0.68 (See Table F.3). If the residuals are sampled from a normal distribution, we would expect the trend in the ranked residuals to follow the same trend as the z-scores. The plot should be a straight line. The easiest way to test this is to plot the ranked residuals against the z-scores. Out of interest we can plot the residuals in Figure 10.1 in a probability plot shown in Figure 10.4. Note that the points lie reasonably on a straight line with perhaps the last two points showing some deviation. Overall there doesn't appear to be any major problem.

We can generate a probability plot using Excel or Python. We will show both approaches here. If using Excel, in the first column enter the ranked residual data. In a third column, enter number 1 to  $n$  where  $n$  is the number of residuals. In the fourth column compute the percentiles using equation (10.1). In the *second column* use the built-in Excel function `NORM.S.INV` to convert the percentile values in the fourth column to z-scores. Finally, plot the first and second column as a scatter plot to yield the probability plot, see Figure 10.3.

When using Python, use the `statsmodels.api` module. The script shown in listing 10.1



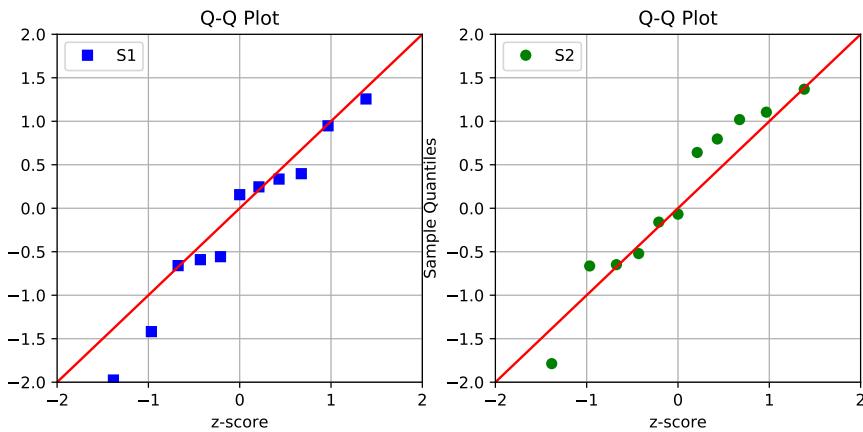
**Figure 10.3** Using Excel to compute probability plots. See main text for details.

shows how to generate probability plots for the residual data shown in Figure 10.1. The resulting plots are shown in Figure 10.4.

Since the residuals are the deviations of the observations away from fitted values, in an ideal case we would expect the residuals to vary randomly about zero, and their spread to be almost the same across the plot. If the points in the plot lie on a curve rather than fluctuating randomly, it is an indication that the zero mean assumption is invalid. If the residuals exhibit a pattern, i.e. increase or decrease in magnitude with the fitted values, this can suggest either systematic changes to the data variances or trends generated by a bad model. A plot of residuals against fitted values may sometimes also reveal points with unusually large residuals. These points are potential outliers, that is, data points for which the model is not appropriate. The presence of outliers in the data sets may significantly influence the estimation of model parameter values. It is therefore important to identify those points and correct them whenever possible, or delete them from the raw data sets. However, in some cases outliers may actually aid in improving our knowledge about the system under consideration. One should do a detailed investigation before rejecting outliers (see [8]).

## Standardized Residuals

Given that residuals can potentially vary over a wide range of values, it is sometimes convenient to normalize the residuals before plotting. If the residuals are well behaved we expect them to be distributed with a mean of zero and standard deviation,  $\sigma$ , often depicted using  $e_i \sim N(0, \sigma_i)$ . One possibility is to scale each residual by its standard deviation. This is



**Figure 10.4** Q-Q probability plots generated using Python script 10.1. See main text for details.

akin to calculating the z-score for a normally distributed variate (See section F.6) where in this case we assume the mean is zero. However, we don't have access to these standard deviations, so instead we normalize each residual with respect to the standard deviation of all the residuals. We therefore define the standardized residual as:

$$s_i = \frac{e_i}{\sigma}$$

where:

$$\sigma = \sqrt{\frac{1}{N-1} \sum_{i=1}^N e_i^2}$$

Standardized residuals are expressed in units of standard deviations. Thus a standardized residual of one indicates that the residual is one standard deviation away from the mean (zero in this case). For a well behaved set of residuals, we expect that 95% of the time, the residuals will lie roughly between the limits 2 and -2. If more than 5% of standardized residuals are greater than 2, then this should raise some concern. If only one observation is found beyond the 95% range, this could also suggest that the data point is an outlier.

## 10.3 $\chi^2$ -Goodness of Fit Test

If the residuals show no unusual trends, the next step is to carry out a goodness of fit test. Recall that the reduced  $\chi^2$  is given by:

$$\chi_r^2 \equiv \frac{1}{N-P} \sum_{i=1}^N \frac{(y_i - f(x_i; p_1 \dots p_m))^2}{\sigma_i^2} \quad (10.2)$$

where  $N - P$  is the degrees of freedom. The value of  $\chi^2$  is determined by a number of factors. The two most interesting to us are:

1. The normally distributed errors in the experimental data.
2. The choice of model used in the fitting.

The purpose of a goodness of fit test is to distinguish between these two contributions. If the  $\chi^2$  is significantly influenced by the model, then the model must be suspect. If the model is a good fit, we expect the model will contribute little to  $\chi^2$ . With a good fit we expect that  $\chi^2$  will *only* be influenced by errors in the experimental data. Given this, it should be clear that a goodness of fit test will *only* work if uncertainty in the experimental data is known.

$\chi^2$  is the ratio of the squared deviations from the expected values divided by the variance of the experimental data. The numerator is therefore a measure of the spread of the observations around the fitted value<sup>1</sup> and the denominator the expected spread. If the model makes little or no contribution to the numerator, that is the model is a good fit, then the two measures of spread should be roughly equal. That is:

$$\frac{(y_i - f(x_i; p_1 \dots p_m))^2}{\sigma_i^2} \simeq 1$$

Summing over all  $i$  terms we expect  $\chi^2$  to be approximately  $N$ .

$$\chi^2 = \sum_{i=1}^N \frac{(y_i - f(x_i; p_1 \dots p_m))^2}{\sigma_i^2} \simeq N \quad (10.3)$$

However this ignores the degrees of freedom and more precisely we expect the reduced chi-square to be equal to:

$$\chi_r^2 = \chi^2 / v \simeq 1$$

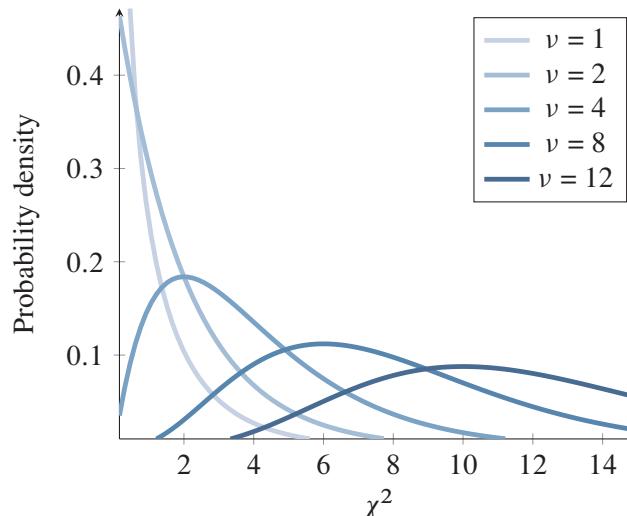
where  $v$  degrees of freedom is:  $v = N - P$ .

In summary, the closer  $\chi_r^2$  is to one, the more likely the model fits the data. In this situation most of the variation we see in  $\chi_r^2$  must originate from errors in the experimental data.

Two questions remain: what if  $\chi_r^2 < 1$ , and how can we decide whether a given value for  $\chi_r^2$  represents a good fit or not? To address the first question; if  $\chi_r^2$  is less than one, then it suggests either the errors in the experimental data are overestimated or more worrying, the data is possibly fraudulent, in a sense, too good to be true. A  $\chi_r^2 < 1$  therefore suggests a problem with the data rather than the model.

<sup>1</sup>Recall:  $\sigma = 1/N \sum (x_i - \mu)^2$

The second question is, how do we decide whether a given value of  $\chi_r^2$  means we have a good fit or not? Consider the thought experiment where we repeat the fitting process multiples times using new experimental data for each fit. In the process we will obtain multiple  $\chi_r^2$  values. Assuming that the experimental data is sampled from a normal distribution (which we have assumed so far), the  $\chi_r^2$  values will by definition be distributed according to a chi-square distribution.



**Figure 10.5** Chi-Square distribution for different degrees of freedom.

Some  $\chi^2$  values will be smaller and others larger but according to the  $\chi_2$  distribution with  $v$  degrees of freedom, the sample of  $\chi_r^2$  values will have a mean of  $v$  and variance of  $2v$ . By chance it is possible that a given experimental data set will result in a larger  $\chi_r^2$ . Using  $\chi^2$  tables we can estimate this likelihood. If the likelihood is small, it means that the  $\chi_r^2$  value is unlikely to have come only from variation in the experimental data. Instead it is more likely that the  $\chi^2$  value is *not* due to variation in the experimental data, but instead due to our application of a *bad model*. This is a subtle argument and will illustrate it further with an example.

Assume that with 10 points and two parameters ( $v = 10 - 2 = 8$ ), we obtain a reduced chi-square of 20. This is a high value and is the result of variation in the data and/or a potentially poor model. The question is which? Using a table of chi-square values, we find that the likelihood of obtaining this value or greater is 0.01. This means that the  $\chi^2$  value could only occur through random fluctuations in the experimental data 1% of the time. This is rare, therefore the major contribution to the  $\chi^2$  is likely to be a poor fit to the model. Given this we conclude that the model must be rejected.

As with all statistical tests, the fact that a goodness of fit test rejects a model does not mean we have shown the model to be incorrect. There is still a chance, albeit small, that we have made an error and rejected a model that is more than adequate at explaining the data. We

will return to this issue in a later section.

## Comparison of Models

It is possible and quite likely that two candidate models fit the same set of experimental data. Is there anyway to decide which is the most plausible? If the two models have the same number of parameters, then a simple comparison of the two  $\chi^2_r$  values is probably sufficient. In general, the model with the smallest chi-square is the better model. What if the number of parameters are different? In this case a statistical test can be carried out to determine which model to select.

## Overfitting

One could reason that of the two models, the model that results in the lowest  $\chi^2$  is the better fit. However this is not necessarily the case. Imagine a model,  $m_1$ , that has ten parameters to fit and another model,  $m_2$ , that has only two parameters to fit. Let us assume that the  $\chi^2$  for  $m_1$  was 0.5 and the  $\chi^2$  for  $m_2$  was 0.8. At first glance it would seem that  $m_1$  is the better fit because it has a lower  $\chi^2$ . The danger here is that because  $m_1$  has ten parameters to adjust, it might be possible to adjust the parameters such that the model solution will go through every experimental data point resulting in a lower  $\chi^2$ . This effect is termed **overfitting**.

It is much better to compare the reduced  $\chi^2$  (equation 9.2) value because this takes into account the number of parameters we fit. Let us assume that we had ten points to fit to the model. The reduced  $\chi^2$  for  $m_1$  will be  $0.5/(10 - 9) = 0.5$ , while the reduced  $\chi^2$  for  $m_2$  will be  $0.8/(10 - 2) = 0.1$ . After taking into account the number of parameters in each fit,  $m_2$  has the lower  $\chi^2$  and therefore we conclude that  $m_2$  is the better fit to the experimental data. To check on the plausibility of a given model, we can therefore compare the reduced chi-square.

Consider two models,  $M_1$  and  $M_2$ , where  $M_1$  has  $p_1$  parameters and  $M_2$ ,  $p_2$  parameters such that  $M_1$  has fewer parameter than  $M_2$  ( $p_1 < p_2$ ). In both cases we fit the model to the *same set of data* and obtain the  $\chi^2_r$  values which we will call  $\chi^2_1$  and  $\chi^2_2$ . A model with more parameters is likely to fit the data better than a model with fewer parameters simply because more parameters gives us more flexibility. The question to consider is whether the additional parameters lead to a significantly better fit. That is, is the likelihood of obtaining  $\chi^2_1$  the same as obtaining  $\chi^2_2$  given the known errors in the experimental data? If it is unlikely then we pick the simpler model since there is no reason to select the more complicated model. We can answer this question by comparing the difference in the variance with the variance of the more complicated model.

The usual test for comparing variances (and therefore  $\chi^2$  values) is the F-test (see sec-

	$M_1$	$M_2$
$\chi^2$	10.5	6.7
$p$	5	8

tion F.9).<sup>2</sup> In this case we will be comparing the difference in variance in the simple model to the variance of the model with the additional parameters, that is [35]:

$$f = \frac{(\chi_1^2 - \chi_2^2)/(p_2 - p_1)}{\chi_2^2/(n - p_1)}$$

The null hypothesis,  $H_o$ , is that the more complicated model  $M_2$  does not provide a significantly better fit than the simpler model,  $M_1$ , i.e. the simpler model is adequate. We will reject the hypothesis if the F statistic is greater than a critical value such as 0.05, and consider the more complicated model a better fit. For example, assume the following data for two models where the number of data points is 20 ( $n = 20$ ). The first model,  $M_1$ , has 5 parameters and the second model,  $M_2$ , 8 parameters. The test for rejecting the null hypothesis is:

$$\text{Reject } H_o \text{ if } f > F(0.95, v_1, v_2)$$

The degrees of freedom for the numerator term,  $v_1$  is  $8 - 5 = 3$  and for the denominator term,  $v_2 = 20 - 8 = 12$ . Given  $v_1$  and  $v_2$ , the critical value from a F-test table at 5% is 3.49. In order to reject the null hypothesis, the F value must be greater than 3.49. Given the data in the table, the  $f$  value can be computed as:

$$f = \frac{(10.5 - 6.7)/3}{6.7/12} = \frac{1.267}{0.446} = 2.84$$

Since  $f < 3.49$  we *accept* the null hypothesis which means that the simpler model is an adequate model to describe the data. Out of interest, what if the fit to the more complicated model was a little better with a  $\chi_2^2 = 4.5$ . In this case,  $f$  is now computed to be:

$$f = \frac{(10.5 - 4.5)/2}{4.5/13} = \frac{2}{0.3} = 6.667$$

Since  $f$  is now greater than the critical value of 3.49, we reject the hypothesis and propose that the more complicated model is a better fit to the data. What happens if we increase the complexity of the model even more, for example the new model has 12 parameters instead of 8? If we assume that the  $\chi_2^2$  is unchanged, the new  $f$  value is computed to be:

$$f = 1.22$$

We accept the null hypothesis. This shows that simply increasing the number of parameters will not necessarily increase the significance of the fit.

---

<sup>2</sup>The difference  $\chi_2^2 - \chi_1^2$  is also distributed as a  $\chi^2$  distribution with  $p_2 - p_1$  degrees of freedom.

Once again it should be emphasized that these tests do not indicate that one model is more correct than another, simply that one of the models is a less likely description of the data than the other.

## 10.4 Estimating Confidence Intervals

---

If the residuals show no unusual trend, and the model is a good fit, we can now consider how confident we are in the fitted parameters. This confidence will depend on how the errors in the experimental data propagate into the parameter estimates. There will therefore be some uncertainty in the values for the parameters and in turn in the model predictions. For example, it is important to know whether a fitted  $K_m$  with a value of 5.0 has an uncertainty of  $\pm 0.2$  or  $\pm 4.8$ .

We can describe the uncertainty using confidence limits, that is, the likelihood for a parameter value to be found within a given confidence limit such as 95% of the time. Intuitively this means if one were to repeat the same experiment many times and each time fitted the experimental data to the model, we would find that 95% of the time the fitted parameters would lie within the indicated range.

The uncertainty in a parameter  $p$ , that is the variance  $\sigma_p^2$ , can be estimated by calculating how each individual data point,  $x_i$ , influences the parameter through the data point's variance,  $\sigma_i^2$ . The following expression, derived in Bevington [12], is an approximation but can be used to compute the parameter variances:

$$\sigma_p^2 \approx \sum \left[ \sigma_i^2 \left( \frac{\partial p}{\partial x_i} \right)^2 \right] \quad (10.4)$$

By evaluating the derivative,  $\partial p / \partial x_i$ , we find that (Details in [12], page 154) the covariance matrix (See F.4) can be obtained from the inverse of the Hessian (9.4),  $\mathbf{H}$ :

$$\text{Cov} = \mathbf{H}^{-1}$$

From this an estimate for the standard deviation in the parameters can be found on the main diagonal of the covariance matrix:

$$\sigma_{p_i} \approx \sqrt{(\mathbf{H})_{ii}^{-1}} \quad (10.5)$$

Assuming a large sample size and for a confidence level of 95%, it can be shown that the quoted limits  $p_0 \pm \delta p$ , are given by:

$$\delta p_i = \pm 1.96 \sqrt{(\mathbf{H})_{ii}^{-1} \frac{\epsilon}{N - P}} \quad (10.6)$$

Recall that  $\epsilon/(N - P)$  is the reduced chi-square term 9.2. For small sample sizes ( $< 30$ ), the value 1.96 can be replaced by a value obtained from the Student's t distribution at 95%.

It is important to note that the estimates given by equation (10.6) are an approximation. Studies indicate that these estimates generally underestimate the actual confidence limits. This is due to a number of assumptions, in particular, we assume that the experimental noise is normally distributed and that the experimentally measured data points must be independent observations. In addition, we assume that the number of data points collected is sufficient to give a good random sampling of the uncertainties in the data, and that the linear approximation (10.4) when deriving (10.5) holds true. For very nonlinear models this is unlikely to be the case. The chance of inaccuracies in the uncertainty estimates is therefore quite likely.

Finally, it's worth discussing the covariances in the Hessian matrix  $\mathbf{H}$ . The confidence limits are derived from the main diagonal elements of  $\mathbf{H}$ . The off diagonal contains information on the covariances, that is how a change in one parameter can influence the change in another parameter. This indicates whether the parameter estimates are independent of each other. If parameters are correlated, it often means there is insufficient experimental data (or variety of measurements) to separate the two parameters and identify them individually. We will return to this important topic in another section ( 10.10) where we review examples of parameter correlation.

An alternative and possibly more trustworthy way to generate confidence limits and one that avoids many of the problems highlighted above, is the use of Monte Carlo simulations [151, 138, 143, 151], which we will address in the following section.

### Determining Confidence Intervals from Monte Carlo Simulations

In the last section a description was given on how to estimate the 95% confidence limits on a set of fitted parameters. Intuitively, if we were to repeat the same experiment many times and each time fitted the experimental data to the model, we would find that 95% of the time the fitted parameters would lie within the indicated range.

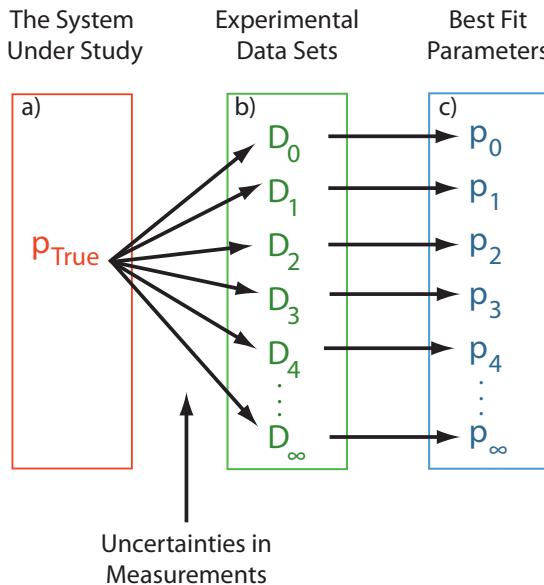
Unfortunately the approach used to estimate these confidence limits includes many assumptions which may or may not be defensible. Going back to the intuitive explanation, if we *could* repeat the experiment many times and fit the data many times, we could get many estimates for the parameters, each estimate slightly different due to errors in the experimental data. From the sample of fitted parameters we could then compute a standard deviation and thus obtain a confidence limit (Figure 10.6). Obviously repeating the experiment many times is impractical but by making two reasonable assumptions, we could do the same thing while only conducting *one real experiment*.

The two assumptions are:

1. When we repeat an experiment, the underlying biology remains the same, that is we are measuring the same thing again.
2. Whatever errors are present in the measurements, the same kind of error manifests

itself each time we repeat the experiment. What this means is that the probability distribution for the errors remains the same, meaning it may be normal, Poisson, etc.

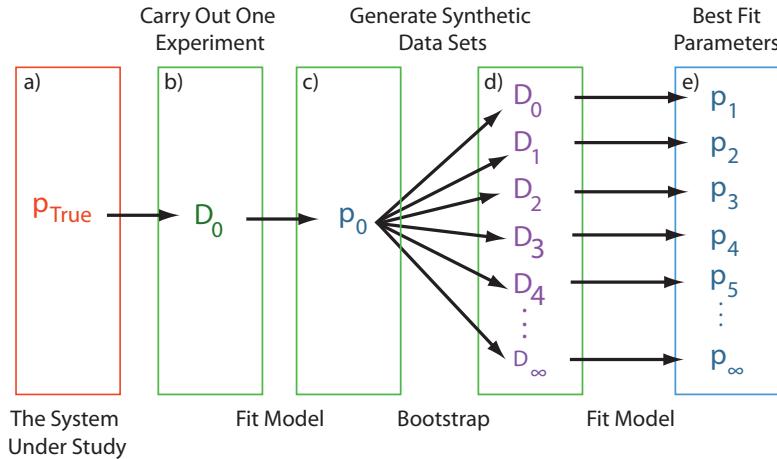
If these two assumptions hold, then we can consider creating synthetic experimental data sets if we know the probability distribution for the errors.



**Figure 10.6** a) In the real world we assume our system has a set of ‘true’ parameter values,  $p_{\text{True}}$ . b) We do experiments which give us experimental data,  $D_i$ . c) Using b) we fit our model to obtain estimates for the parameters,  $p_i$ . Because each experiment is slightly different due to measurement uncertainty, we will in turn generate slightly different sets of fitted parameters.

We need to make one further assertion before we can continue. Let us assume that the parameter estimates we obtain from fitting the real experimental data to the model are close to the true parameter values. That is,  $p_{\text{True}}$  is not far from the fitted parameter,  $p_0$ . The core concept is to generate, via a bootstrap (see next section) new synthetic data sets. The bootstrap ensures that the new data sets have the same error distribution as the data collected from the real and only experiment performed. Each synthetic data set will be fitted to the model, from which we obtain multiple estimates for the parameters. Once we have a large sample of estimated parameters, we can make statements about the uncertainty of our parameter estimates (Figure 10.7). In particular, approximate confidence intervals for the parameters can be obtained by using the  $\frac{\alpha}{2}$  and  $1 - \frac{\alpha}{2}$  sample quantiles (where  $\alpha$  is the threshold, for example, 0.05 for 95% confidence) from the Monte Carlo estimators of the parameters.

If the experimental uncertainties surrounding measured data are not known, then a proxy can be obtained by relying on the residuals that were produced as a result of the parameter estimation procedure.



**Figure 10.7** a) The actual system with true values for the parameters; b) Generate a set of measurements from one experiment a); c) Fit the data to the model to generate parameter estimates,  $p_0$ ; d) Use a bootstrap to generate synthetic data sets,  $D_i$ ; e) Fit the synthetic data sets to the model and produce a sample of parameter estimates,  $p_i$ . Use the sample of parameter estimates to gauge parameter uncertainty.

## Bootstrap

**Bootstrapping** is a method to infer the statistics of a population by sampling from a sample of the population [36, 138]. It is a means of gaining information about a population when the population itself is not available. The key assumption in a bootstrap is that the sample contains enough information to reconstruct details about the population. An example of a simple bootstrap is given in Appendix F.

The bootstrap method works as follows. Assume we have a sample of observations of size  $N$  from our population. Now generate new samples by selecting  $N$  random values with replacement, that is, returning the value back to the pool before selecting another. Since we are sampling with replacement, some of the original observations may appear more than once in the new sample sets. Repeat the sampling process until the desired number of simulated data sets are generated.

In the case of fitting a model, what do we sample? One possibility are the residuals generated from the initial fit. The residuals are the difference between the fitted data value and

the corresponding experimental value, that is:

$$r_i = (y_i) \text{ observed} - (y_i) \text{ predicted}$$

To generate a synthetic data set, sample the residuals and add them to the predicted  $y_i$  values:

$$(y_i) \text{ synthetic} = (y_i) \text{ predicted} + r_{\text{sample}} \quad (10.7)$$

For example, assume that after our initial fit we generated the residuals:  $(0.1, -0.5, 0.2)$ . Assume also that the fitted data was  $(10, 6, 3)$ . To sample residuals we randomly pick three values from the set. Each time we pick a value, we also return the value back to the set (replacement). For example, the following sets are possible sample of residuals:

$$(-0.5, 0.2, 0.2), (-0.5, 0.1, 0.2), (0.2, 0.1, 0.1), (-0.5, 0.1, -0.5)$$

With the residual samples, we now generate the synthetic experimental data by adding each set to the fitted data (10.7). For example  $(10 - 0.5, 6 + 0.2, 3 + 0.2)$ . The four new data sets will therefore be:

$$(9.5, 6.2, 3.2), (9.5, 6.1, 3.2), (10.2, 6.1, 3.1), (9.5, 6.1, 2.5)$$

It is prudent to generate at least 500 to 1,000 new synthetic data sets in this way. Taking each synthetic data set in turn, fit the data to the model to generate a ‘synthetic’ estimate for the parameters. We will thus generate 500 to 1,000 estimates for the model parameters. Using these parameters we can calculate statistics such as the standard deviation for each parameter. However, unlike the estimated statistics from the Hessian which will be symmetric, the confidence limits from the Monte Carlo method are not guaranteed to be symmetric. As a result it is best to compute confidence limits using percentile values although other approaches are possible [170]. For example, we could generate 97.5<sup>th</sup> and 2.5<sup>th</sup> percentile values.

## 10.5 Cross-validation

---

There are many models that one could propose to fit the data adequately, some complex, some simple. But just because data fits a model is no guarantee that the model will be useful. Here is a trivial example to illustrate this important point. The data in Table 10.5 fits a straight line,  $y = 1.9x - 1.2$  with  $R^2$  value of 0.9826 which suggests a very good fit. We now make a prediction that at value  $x = 30$ , the predicted response is 55.8. However, when we attempt to confirm the prediction experimentally, we find that our prediction is off by a wide margin. Instead we attempt a different fit, this time a second-order polynomial. The new fit yields a  $R^2 = 1$  and a fitted equation of  $y = 0.1x^2 + x$ . The prediction at  $x = 30$  is now 120, which is exactly the expected value. The example is trivial but it highlights a number of important points.

---

x	0	1	2	3	4	5	6	7	8	9
y	0	1.1	2.4	3.9	5.6	7.5	9.6	11.9	14.4	17.1

---

The original linear fit managed to reproduce the existing data very well, in fact there was no reason from the  $R^2$  value to think otherwise. However when tested, the model was found to be inadequate. The linear fit was only capable of making good predictions within the range of the data itself and perhaps a little beyond. The linear fit actually failed to capture the real essence of the data, that is it failed to **generalize** and recognize that the data followed a 2nd-order polynomial.

The ultimate purpose of a model is to organize our knowledge and make new and useful predictions. Even if the final model has passed all the statistical tests that we previously discussed, we cannot be sure that the model will make useful predictions. The key question is, has the model generalized? The only way to determine this is to put the model to an actual test. In the literature this is often called **cross-validation** [114].

The key idea behind cross-validation is simple. Rather than fit the model using all available data, we choose to hold back some data. We fit the model with the reduced data set and then using the fitted model we determine how well the model predicts the data we held back. The data we use to fit the model is often called the **training set** and the data we hold back the **test data**. If the fitted model fails to predict the test set, then it means that the model has failed to generalize and has probably over-fitted the training set in order to appear to be a good fit. This is one of the main functions of cross-validation.

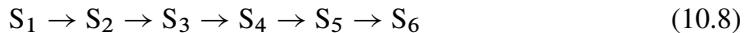
Often different training sets are used by splitting the data in different ways. The model is then subjected to every combination of training set. This is computationally expensive but ensures a thorough study. The combinations of training sets can be determined using different strategies. The simplest is to remove one data point from the available data set. The test data then constitutes one data point and we must attempt to predict the value for that data point. This can be repeated for every data point. For example, if the entire data set includes twenty data points, then twenty different training sets can be created. A generalization of this is called K-fold cross-validation [114]. This is where the data is partitioned into  $k$  equal size subsamples. One subsample is designated the training set and the remainder are used for fitting the model. This can be repeated for each  $k$  subsample and even for different sized  $k$ . Alternatively training sets can be generated by randomly selecting data to be put into a training set. For large data sets this is more practical.

Ultimately many cross-validation tests will be carried out resulting in a *distribution of predictions*. This can be examined for any patterns. For example, if all the tests yield very similar predictions, then this is a very good indicator that the model has generalized and can be trusted to make further reliable predictions. If however the tests yield markedly different predictions, then this is a clear warning sign that the model is untrustworthy and should be carefully reexamined.

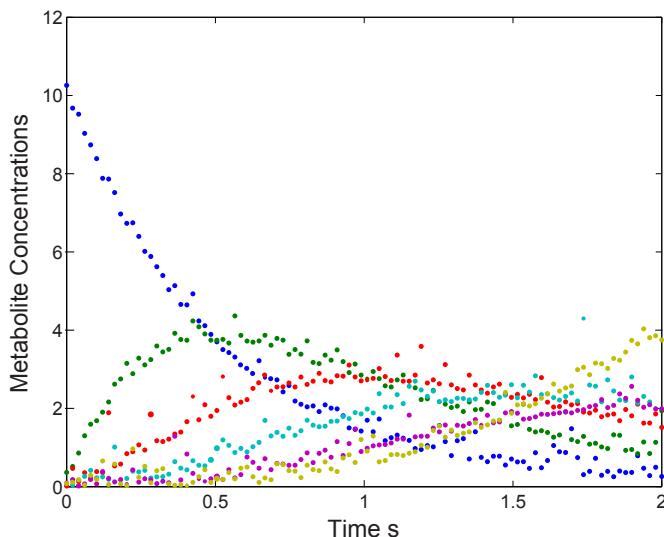
## 10.6 Case studies

### Test Example

Consider a linear chain of irreversible uni-molecular reactions that follow mass-action kinetics:



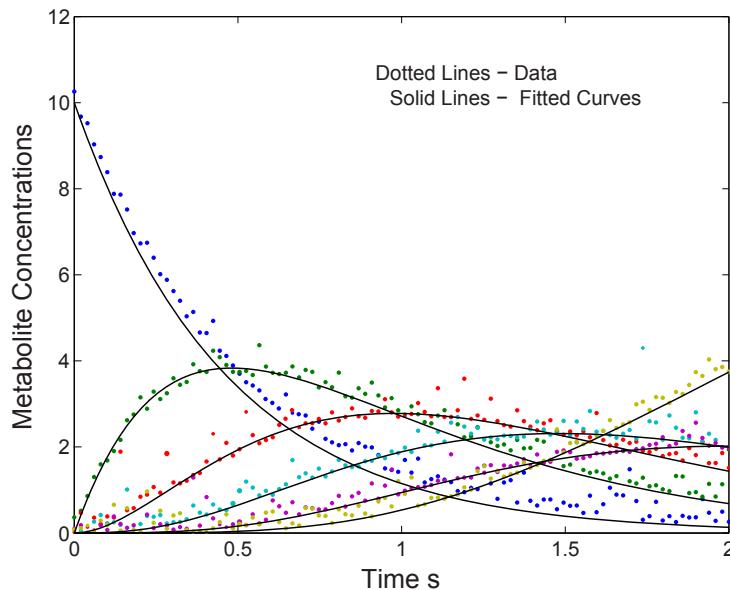
The noisy concentration data for the six metabolites displayed in Figure 10.8 was simulated with all the kinetic rate coefficients set to 2, the initial concentration of the first substrate set to 10, and all others to zero.



**Figure 10.8** Plot of the simulated noisy time series concentration data and the fitted curves for a linear sequence of reactions governed by irreversible mass-action kinetics.

The data in Figure 10.8 has 100 points. The noise was assumed to be exponentially distributed and was added to the simulated curves, and then used to fit the model. The simplex method (the Levenberg-Marquardt gives similar results) was used to fit the model. The five parameters were initialized to 0.1. The fit is shown in Figure 10.9 as solid lines.

A Monte Carlo simulation was run and for each generated synthetic dataset the parameters were optimized. Figure 10.10 shows the distribution of parameter estimates from the Monte Carlo simulations. What is relevant here is that the scatter plots are not skewed in anyway, showing that there are no correlations between the parameters. This implies that there was sufficient data to *identify* all the parameters in the model. We will see a different outcome in the next example.



**Figure 10.9** Plot of the simulated noisy time series concentration data and the fitted curves for a linear sequence of reactions governed by irreversible mass-action kinetics.

Confidence limits were obtained using (10.6) and are shown in Table 10.1.

The confidence limits could also have been evaluated from the Monte Carlo generated data (Figure 10.10) by choosing limits around the mean values of the parameters and making sure that 95% of the points fall between them. The confidence limits constructed in this manner match the ones computed using equation (10.6). In addition, the Hessian was found to be well behaved with no significant correlation between the parameters (10.10). This is consistent with the Monte Carlo study and again suggests there is adequate data to *identify* all the parameters in the model.

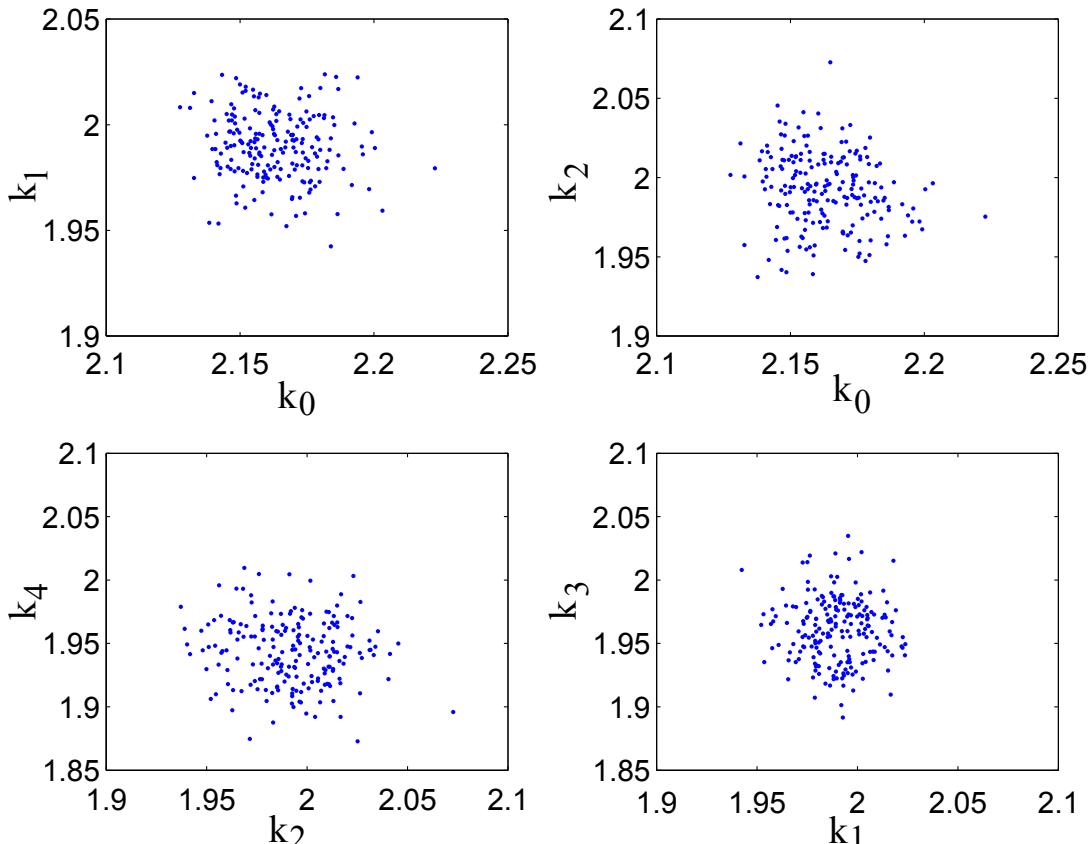
This example was artificially constructed where we were able to adequately measure every state variable in the system. In addition, the level of noise was relatively low and we were able to ‘collect’ many data points for each variable. These ideal conditions will not be available in real systems as we will see in the next example.

### Fitting Data to HIV Proteinase

In our second example we will fit rate constants to data obtained from a model for irreversible inhibition of HIV proteinase [92]. The data was obtained from [www.biokin.com/dynafit/index.html](http://www.biokin.com/dynafit/index.html), and comprises two different time courses at different inhibitor concentrations. There are five parameters to estimate. Initial values for the parameters were

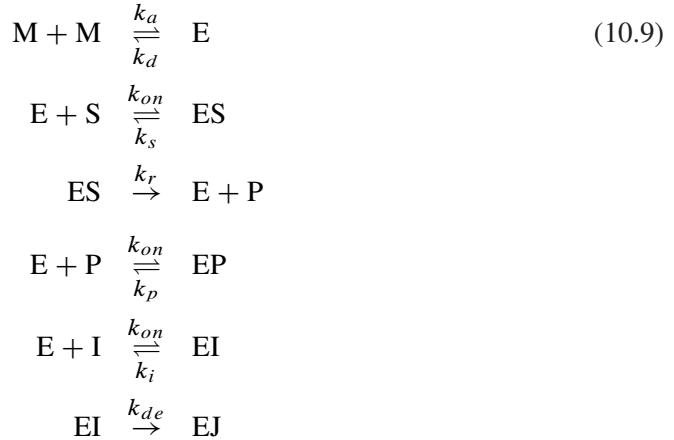
Parameter	Value
$k_0$	$2.16 \pm 0.05$
$k_1$	$1.99 \pm 0.042$
$k_2$	$1.99 \pm 0.083$
$k_3$	$1.96 \pm 0.183$
$k_4$	$1.94 \pm 0.199$

**Table 10.1** The table shows 95% confidence limits for the estimated parameters based on equation (10.6).



**Figure 10.10** Cluster plots for the distribution of fitted parameters for a Monte Carlo simulation. Various parameter combinations are shown.

assigned and substrate concentrations set as described in [92]<sup>3</sup>. The model is described by the following reaction scheme:



In Figure 10.11 we display two data sets along with their respective best fits.

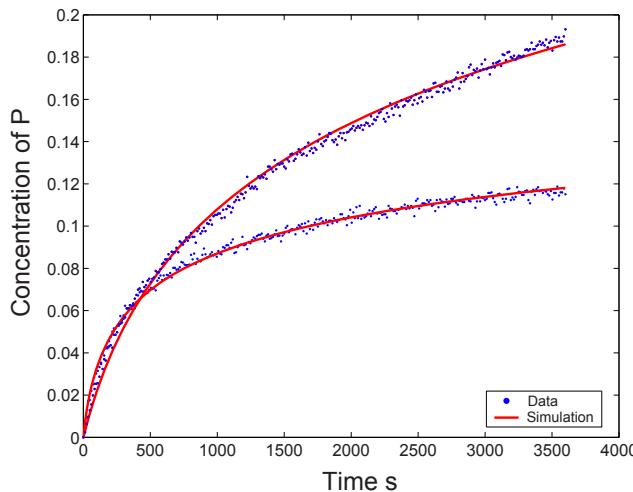
Several combinations of optimizers were run, the simulated annealing and simplex took the maximum time, but of the two, the latter gave better results. Running the GA first followed by running the simplex was also successful. As seen in Figure 10.11, the fits appears to be quite good.

Parameter	Value
$k_s$	$250.9225 \pm 0.69$
$k_r$	$0.199 \pm 0.266$
$k_p$	$35.04 \pm 109.9$
$k_i$	$0.0469 \pm 7.46$
$k_{de}$	$0.2292 \pm 3.76$

**Table 10.2** The table shows 95% confidence limits for the estimated parameters using the Hessian computed at the end of optimization.

Equation (10.6) was used to compute the parameter confidence limits and Table 10.2 shows the confidence limits for the parameters. The confidence limits on  $k_r$ ,  $k_p$ ,  $k_i$  and  $k_{de}$  are substantially greater than their mean values, which is a clear indication that these numbers

<sup>3</sup>In the example described above the 4th and 5th data sets were chosen (Kuzmic 1996, curves D, E of Fig 1, page 264). We used initial values of  $k_{on} = 100$ ,  $k_d = 0.0001$ ,  $k_a = 0.1$ ,  $I = 0.004$ ,  $E = 0.004$ , and  $S = 27$ . In Kuzmic 1996, some of the species levels are also optimized, but here we are more interested in fitting the parameters.



**Figure 10.11** Two time series concentration data sets plotted along with their respective fitted curves.

cannot be trusted (see [122]). To investigate why these parameters have such low confidence, we can carry out a Monte Carlo simulation with a bootstrap using the residuals from the first fit. We generated 500 new synthetic data sets, and refitted the parameters. Cluster plots for various combinations of parameters are shown in Figure 10.10 and reveal some interesting patterns.

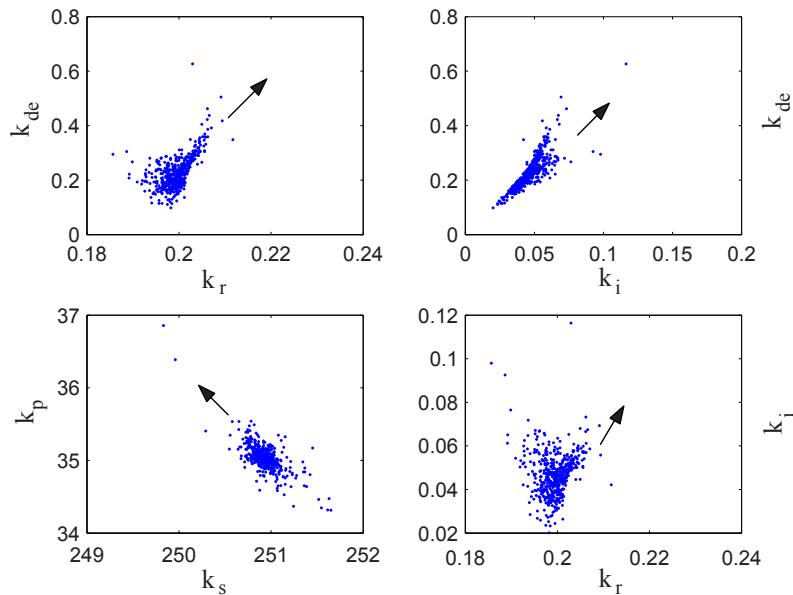
$k_s$	$k_r$	$k_p$	$k_i$	$k_{de}$
$\pm 0.31$	$\pm 0.006$	$\pm 0.37$	$\pm 0.01$	$\pm 0.11$

**Table 10.3** The table shows 95% confidence limits for the estimated parameters for HIV model, using a Monte Carlo simulation.

As indicated by the arrows in Figure 10.12, the Monte Carlo simulations uncovered significant correlation between a number of the parameters. This implies that certain combinations of parameters could change, without having any effect on the concentrations of the species. This is also known as the **observability problem** and shows that with the data at hand, we are unable to identify all the parameters in the system. For example, consider a simple Michaelis-Menten set of reactions:



where the reversible step between the substrate and complex are  $k_1, k_{-1}$ , and the forward reaction rate between complex and product is  $k_2$ . In terms of these basic mass-action reactions, we derive the Michaelis-Menten equation by assuming that after a very brief



**Figure 10.12** Cluster plots for the parameter distributions showing significant correlations for some parameter combinations.

transient time, the complex forms and after it remains constant. Under these assumptions the rate between  $S$  and  $P$  is:

$$\frac{V_{max}S}{K_m + S} \quad (10.11)$$

where  $V_{max} = k_2 e_0$ ,  $e_0$  being the enzyme concentration, and  $K_m = \frac{k_1}{k_{-1} + k_2}$ . Notice that if  $k_2$  is kept constant,  $k_1$  and  $k_{-1}$  can be changed such that  $K_m$  remains the same. This means that the net rate does not change. This is obvious given we are only changing the final amount of complex generated, not the product. When presented with time series data from such a simple model, we will see a correlation between the spread in  $k_1$  and  $k_{-1}$  values. This is generally true for larger models, but it may not be possible to find simple combinations of these parameters that are truly independent. The important point is that the cluster plots show observability of the parameters. It is then a simple step to quantify this, by making the observation that the 2-D cluster plots are sections of the  $m \times m$  (where  $m$  is the number of parameters) probability distribution of the fitted parameters. The eigenvectors of the Hessian (inverse of the covariance matrix) that correspond to the lowest eigenvalues, are directions along which, if the parameters change, no significant change in the sums of squares,  $\epsilon$  is seen. The eigenvalues of the Hessian are,  $\simeq 665, 0.51, 0.0076, 10^{-5}, 10^{-5}$ .

## 10.7 Final Comments

---

Experimental data has been collected, a model has been proposed, data fitting has confirmed that the model is able to reproduce the experimental data, and all the statistical tests pass. One might imagine we are now ready to write the paper and publish. However a critical last step is to ask whether the model can make new, non-trivial predictions. At minimum we should propose experiments to test the proposed model, at best, attempt the tests ourselves. It is worth emphasizing again the section where cross-validation was briefly discussed. This is where a model is tested for its ability to predict data it has never seen before, and is the ultimate test for demonstrating reliability and utility of a model. In biomedical research the chief aim should be the development of trustworthy models, models that could, if necessary, be used in clinical situations.

One of the chief dangers in building models is inadvertently creating a model that is over-fitted, which can give us a false sense of security that we have a good model. It is true that some of the statistical tests we have discussed in this chapter can help spot an over-fitted model, but the best test is cross-validation and ultimately testing new hypotheses generated from the model. I am reminded here of the oft quoted comment made by John von Neumann who it is said to have remarked: “With four parameters I can fit an elephant, and with five I can make him wiggle his trunk”. This quote has worked its way through the biomedical simulation literature mostly as a means to discredit modeling and fails to point out that models must be stress tested to ensure they can be trusted.

When data is fitted to a model, we need to know the range of predictions the model can make beyond the fitted data. Out of range predictions must also be tested. As discussed in Chapter 4, what modelers really seek is a degree of confidence in their model. A model is never truly validated as one can only take measurements to increase confidence in the model. That confidence may increase as the model is subjected to more and more testing. John Tyson who is well known for his pioneering work on cell cycle models uses an interesting approach to model building. Whenever the core model is adjusted, the model is subjected to over 120 phenotype tests, which are predictions that the model is expected to make [25]. This ensures that the authors have a high confidence in their model.

Whether the fit of a model to data is good or not is immaterial unless the model provides new, testable predictions that can eventually be experimentally tested.

## Further Reading and Online Resources

---

1. Berendsen HJ. (2011) A Student’s Guide to Data and Error Analysis. Cambridge University Press. ISBN: 978-0-521-13492-7
2. Draper NR and Smith H (1998) Applied Regression Analysis. 3rd edition. Wiley Series on Probability and Statistics. ISBN-13: 978-047117082

3. Johnson ML, Faunt LM (1992) Parameter estimation by least-squares methods. *Methods in Enzymology*, 210, 1-37.
4. Johnson ML (1994) Use of Least-Squares Techniques in Biochemistry. *Methods in Enzymology*, 240, 1-22.
5. David Liao (2012) Uncertainty propagation d: Sample variance curve fitting. <https://vimeo.com/40379524>.
6. Straume M, Johnson ML (1992) Monte Carlo Method for determining complete confidence probability distributions of estimated model parameters. *Methods in Enzymology*, 210, 117-129.

## Exercises

---

1. Create a simple linear chain model of four steps and three species. Choose nonlinear reversible rate laws for the reactions, assign suitable values to the parameters and run a simulation to obtain time-course data for the three species. Add noise to the simulated data and treat this data as your ‘experimental data’. Fit the experimental data to the model and see how well your parameter estimates agree with the original model. Try different fitting methods such as the Simplex and Levenberg-Marquardt methods to investigate how well each one performs.
2. Implement a Monte-Carlo method to obtain estimates for the parameter confidence limits.
3. Investigate how the degree of noise in your experimental data affects the fitted parameter values.
4. Investigate how omitting one or more of the time series data affects the fitting process. For example, omit data for the first and last species in the pathway. Recompute the parameter confidence limits, what do you observe? Use a Monte Carlo simulation to compute scatter plots for the different parameters.

## Appendix

---

```
import numpy as np
import pylab
import statsmodels.api as sm

# Residual data for S1 and S2
S1 = np.array([1.25646491, 0.947882566, 0.397358056, 0.335372002, 0.245941357, 0.155918422,
              -0.556750428, -0.591033227, -0.659627676, -1.41856797, -1.97506809])
```

```
S2 = np.array([-0.648710057, 1.01931184, -0.521154398, -0.159291555, 0.641344813,
               -0.067860512, -0.663673399, 1.10483282, 0.796177025, -1.78660179,
               1.36881988])

pylab.figure(figsize=(9, 4))
pylab.subplot(1, 2, 1)
probplot1 = sm.ProbPlot(S1) # Compute the probability plot data
probplot2 = sm.ProbPlot(S2)

pylab.title('Q-Q Plot')
pylab.ylim(-2,2); pylab.xlim(-2,2)
pylab.plot (probplot1.theoretical_quantiles, probplot1.sample_quantiles, 's', color='blue', label='S1')
pylab.plot ([−2,2], [−2,2], 'red')
pylab.xlabel('z-score')
pylab.legend(); pylab.grid(True)

pylab.subplot(1, 2, 2)
pylab.title('Q-Q Plot')
pylab.ylim(-2,2); pylab.xlim(-2,2)
pylab.plot (probplot2.theoretical_quantiles, probplot2.sample_quantiles, 'o', color='green', label='S2')
pylab.plot ([−2,2], [−2,2], 'red')
pylab.legend(); pylab.grid(True)
pylab.xlabel('z-score')
pylab.ylabel('Sample Quantiles')
pylab.tight_layout()
pylab.savefig ('probplot.pdf')
pylab.show()
```

**Listing 10.1** Script for probability plots show in Figure 10.4.

# 11

## *The Steady State*

### 11.1 Steady State

---

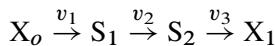
In Chapter 7 we briefly introduced the idea of a steady state. In this chapter we will investigate the steady state in greater detail.

The literature sometimes refers to the steady state as the stationary solution or stationary state, singular point, fixed point, or even equilibrium point. For our purpose we will avoid using the term equilibrium because of possible confusion with thermodynamic equilibrium.

The steady state is one of the most important states to consider in a dynamical model because it is the primary reference point from which to consider a model's behavior. At steady state the concentrations of all molecular species are constant, and there is a net flow of mass through the network. This is in contrast to systems at thermodynamic equilibrium where there is no net flow of mass across the system's boundaries.

The steady state is where the rates of change of all species,  $dS/dt$  are zero, but at the same time the net rates are non-zero, that is  $v_i \neq 0$ . This situation can only occur in an open system, where matter is exchanged with the surroundings.

Equation (7.3) from a previous chapter describes the time evolution for the system:



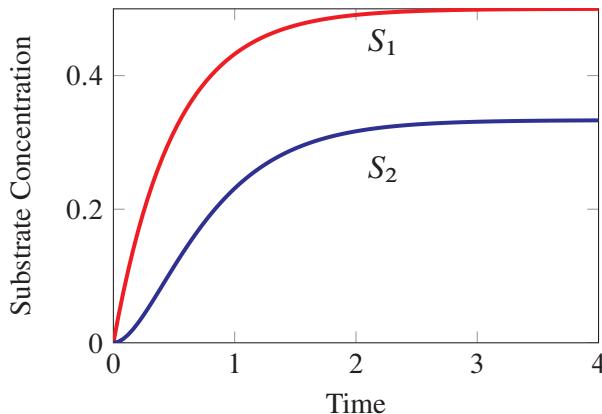
We repeat the equations here for convenience:

$$\begin{aligned} S_1(t) &= v_o \frac{1 - e^{-k_1 t}}{k_1} \\ S_2(t) &= v_o \frac{k_1 (1 - e^{-k_3 t}) + k_3 (e^{-k_1 t} - 1)}{k_3 (k_1 - k_3)} \end{aligned} \quad (11.1)$$

As  $t$  tends to infinity,  $S_1(t)$  and  $S_2(t)$  tend to:

$$S_1(\infty) = \frac{v_o}{k_1} \quad S_2(\infty) = \frac{v_o}{k_3}$$

The reaction rate through each of the three reaction steps is  $v_o$ . This can be confirmed by substituting the solutions for  $S_1$  and  $S_2$  into the reaction rate laws. Given that  $v_o$  is greater than zero and  $S_1$  and  $S_2$  reach constant values given sufficient time, we conclude that this system eventually settles to a steady state rather than thermodynamic equilibrium. The system displays a continuous flow of mass from the source to the sink. This can only continue undisturbed so long as the source material,  $X_o$ , never runs out. Figure 11.1 shows a simulation of this system.



**Figure 11.1** Time course for an open system reaching steady state.  $X_o \xrightarrow{v_o} S_1 \xrightarrow{k_1} S_2 \xrightarrow{k_3}$  where  $v_o = 1, k_1 = 2, k_3 = 3, S_{1o} = 0, S_{2o} = 0$ .  $X_o$  is assumed to be fixed. Tellurium Listing: 11.2.

### Graphical Procedure

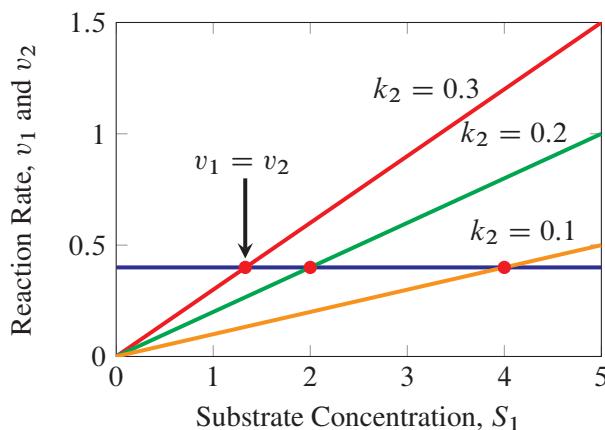
We can also illustrate the steady state using a graphical procedure. Consider the simple model below:



where  $X_o$  and  $X_1$  are fixed boundary species and  $S_1$  is a species that can change (the floating species). For illustration purposes we will assume some simple kinetics for each reaction,  $v_1$  and  $v_2$ . Let us assume that each reaction is governed by first-order mass-action kinetics:

$$v_1 = k_1 X_o \quad v_2 = k_2 S_1$$

where  $k_1$  and  $k_2$  are both first-order reaction rate constants. In Figure 11.2 both reaction rates have been plotted as a function of the floating species concentration,  $S_1$ .



**Figure 11.2** Plot of reaction rates versus concentration of  $S_1$  and different values for  $k_2$  for the system  $X_o \rightarrow S_1 \rightarrow X_1$ . The intersection of the two lines marks the steady state point where  $v_1 = v_2$ .  $X_o = 1, k_1 = 0.4$ . Note that as  $k_2$  decreases the steady state level of  $S_1$  increases.

Note that the reaction rate for  $v_1$  is a horizontal line because it is unaffected by changes in  $S_1$  (no product inhibition). The second reaction,  $v_2$ , is shown as a straight line with slope,  $k_2$ . Notice that the lines intersect. The intersection marks the point when both rates  $v_1$  and  $v_2$  are equal, that is when  $dS_1/dt = 0$  since  $v_1 = v_2$ . This point marks the steady state concentration of  $S_1$ . By varying the value of  $k_2$ , we can observe the effect it has on the steady state. For example, Figure 11.2 shows that as we *decrease*  $k_2$ , the concentration of  $S_1$  *increases*. This should not be difficult to understand; as  $k_2$  decreases, the activity of reaction  $v_2$  also decreases. This causes  $S_1$  to build up in response.

In this simple model it is also straightforward to determine the steady state of  $S_1$  mathematically which amounts to finding a mathematical equation to represent the intersection

point of the two lines. Recall that the model for this system is a single differential equation:

$$\frac{dS_1}{dt} = k_1 X_o - k_2 S_1$$

At steady state, set  $dS_1/dt = 0$ , from which we can solve for the steady state concentration of  $S_1$ :

$$S_1 = \frac{k_1 X_o}{k_2} \quad (11.2)$$

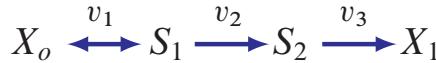
This solution tells us that the steady state concentration of  $S_1$  is a function of *all* the parameters in the system. We can also determine the steady state rate, usually called the pathway flux denoted by  $J$ , by inserting the steady state value of  $S_1$  into one of the rate laws, for example into  $v_2$ :

$$J = k_2 \frac{k_1 X_o}{k_2} = k_1 X_o$$

This answer is identical to  $v_1$  which is not surprising since the pathway flux is completely determined by the first step, and the second step has no influence whatsoever on the steady state flux. This simple example illustrates a classical ‘rate limiting step’ in the pathway; that is one step, and one step only, has complete influence over the pathway flux.

### More Complex Model

A slightly more realistic model is the following:



where the rate law for the first step is now reversible and given by:

$$v_1 = k_1 X_o - k_2 S_1$$

The remaining steps are governed by simple irreversible mass-action rate laws,  $v_2 = k_3 S_1$  and  $v_3 = k_4 S_2$ . The differential equations for this system are:

$$\frac{dS_1}{dt} = (k_1 X_o - k_2 S_1) - k_3 S_1$$

$$\frac{dS_2}{dt} = k_3 S_1 - k_4 S_2$$

The steady state solution for  $S_1$  and  $S_2$  can be obtained by setting both differential equations to zero and solving for  $S_1$  and  $S_2$  to yield:

$$S_1 = \frac{k_1 X_o}{k_2 + k_3}$$

$$S_2 = \frac{k_3 k_1 X_o}{(k_2 + k_3) k_4}$$

The steady state flux,  $J$ , can be determined by inserting one of the solutions into the appropriate rate law. The easiest method is to insert the steady state level of  $S_2$  into  $v_3$  to yield:

$$J = \frac{k_3 k_1 X_o}{k_2 + k_3}$$

Once the first step is reversible, we see that the steady state flux is a function of all the parameters except  $k_4$ , indicating that the first step is no longer the rate limiting step. The equation shows that the ability to influence the flux is shared between the first and second steps. There is no rate limiting step in this pathway. Note that if we set  $k_2 = 0$ , then the solution reverts to the earlier simpler model,  $J = k_1 X_o$ .

We can also make all three steps reversible ( $k_f S_i - k_r S_{i+1}$ ), so that the solution is given by:

$$\begin{aligned} S_1 &= \frac{X_o k_1 (k_4 + k_5) + X_1 k_4 k_6}{k_3 k_5 + k_2 (k_4 + k_5)} \\ S_2 &= \frac{X_1 k_6 (k_2 + k_3) + X_o k_1 k_3}{k_3 k_5 + k_2 (k_4 + k_5)} \end{aligned}$$

The last example illustrates the increase in complexity of deriving a mathematical solution after only a modest increase in model size. In addition, once more complex rate laws are used, such as Hill equations or Michaelis-Menten type rate laws, the solutions become exceedingly difficult to derive. As a result, steady states tend to be computed numerically rather than analytically.

## 11.2 Effect of Different Kinds of Perturbations

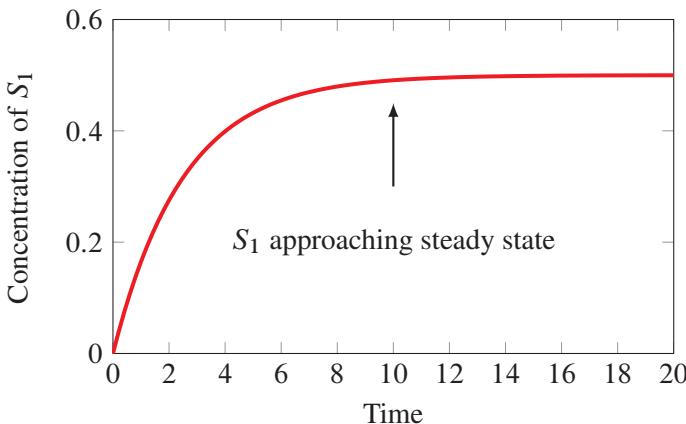
When we talk about model dynamics, we are referring to how species levels and reaction rates change over time as the model evolves. There are a number of ways to elicit a dynamic response in a model. The two we will consider here are perturbations to species and to model parameters around the steady state.

### Effect of Perturbing Floating Species

Consider a two step pathway of the following form:



Assume that  $X_o$  and  $X_1$  are fixed. If the initial concentration of  $S_1$  is zero, we can run a simulation and allow the system to come to steady state. This is illustrated in Figure 11.3.



**Figure 11.3** Species  $S_1$  approaching steady state. Tellurium Listing: 11.3.

Once at steady state, we can consider applying perturbations to see what happens. For example, Figure 11.4 illustrates the effect of injecting 0.35 units of  $S_1$  at  $t = 20$  and watching the system evolve. What we observe is that the concentration of  $S_1$  immediately jumps by the amount 0.35, then relaxes back to the steady state concentration it had before perturbation (Figure 11.4). When we apply perturbations to species concentrations and the change relaxes back to the original state, we call the system **stable**. We will return to this topic in another section.

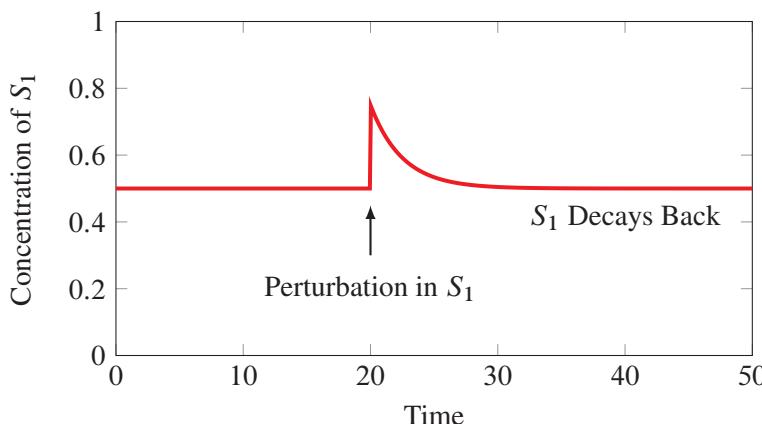
Figure 11.4 illustrates perturbing one of the floating molecular species by physically adding a specific amount of the substance to the pathway. In many cases we will find that the system will fully recover. We are not limited to single perturbations; Figure 11.5 shows multiple perturbations, both positive and negative. Not all systems show recovery like this and those that do not are called **unstable**. That is, when we perturb a species concentration, instead of the perturbation relaxing back, it begins to diverge.

### Effect of Perturbing Species in a Conserved Cycle

Section 3.8 introduced the idea of the conserved cycle, groups of species whose total mass is conserved during the evolution of a network. Figure 11.6 shows the simplest conserved cycle where the total mass,  $S_1 + S_2$ , is constant throughout the system's evolution. Figure 11.7 shows a simulation where  $S_1$  is perturbed by one unit. This causes the total mass in the cycle to increase and results in a net change to the steady state.

### Effect of Perturbing Model Parameters

In addition to perturbing floating species, we can also perturb model parameters. Such parameters include kinetic constants and inputs such as boundary species or addition of



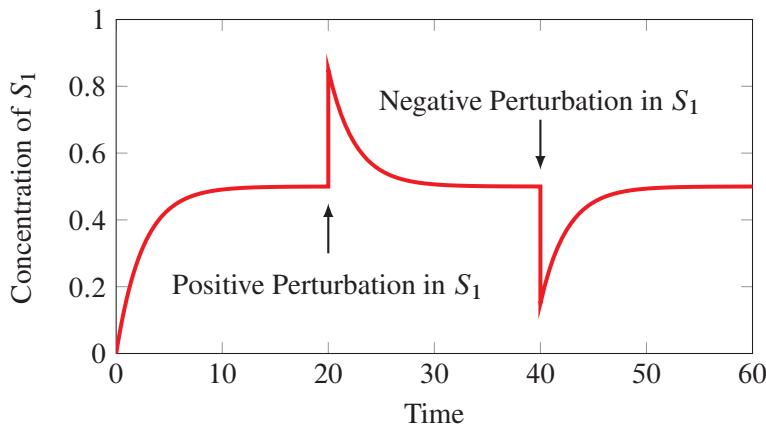
**Figure 11.4** Stability of a simple biochemical pathway at steady state. The steady state concentration of the species  $S_1$  is 0.5. A perturbation is made to  $S_1$  by adding an additional 0.35 units of  $S_1$  at time = 20. The system is considered stable because the perturbation relaxes back to the original steady state. Tellurium model: 11.4.

other effectors such as drugs. We can change a parameter in either two ways: make a permanent change, or make a change and at some later point return the parameter to its original value. If we make a permanent change, the steady state will invariably also show a permanent change. A temporary change will result in the steady state changing, and then recovering to the original state once the parameter is changed back. Figure 11.8 shows the effect of perturbing the rate constant,  $k_1$ , and then restoring the parameter to its original value at some later time point.

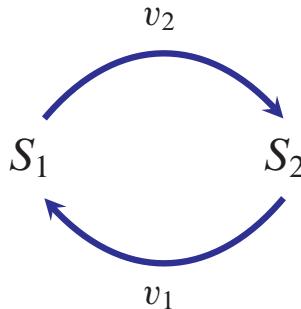
We can also consider other types of perturbations. For example, in studying the infusion of a drug where the drug concentration is a model parameter, one might use a slow linear increase in concentration. Such a perturbation is called a **ramp**. More sophisticated methods might require a sinusoidal change in a parameter, an **impulse**, a **pulse**, or an exponential change. The main point to remember is that parameter changes will usually result in changes to the steady state concentrations and fluxes. For completeness, Figure 11.9 shows what happens when we perturb both a parameter and a species concentration. As expected, the species concentration does not recover to the original steady state.

## 11.3 Computing the Steady State

In those (many) cases where we cannot derive an analytical solution for the steady state, we must revert to numerical methods. There are at least two available methods. The simplest approach is to run a time-course simulation for a sufficiently long period such that the trajectories eventually reach steady state. This method works so long as the steady state is stable; it cannot be used to locate unstable steady states because such trajectories diverge.



**Figure 11.5** Multiple Perturbations. The steady state concentration of the species  $S_1$  is 0.5, and a perturbation is made to  $S_1$  by adding an additional 0.35 units of  $S_1$  at time = 20 and removing 0.35 units at time = 40. In both cases the system relaxes back. Tellurium script: 11.5.

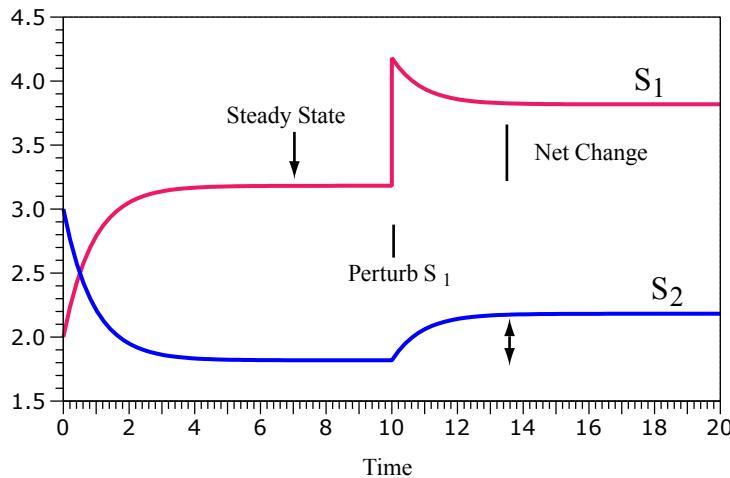


**Figure 11.6** Simple cycle where  $S_1 + S_2$  is constant.

In addition, the method can sometimes be very slow to converge depending on the model kinetics. As a result, many simulation packages will provide an alternative method for computing the steady state where the model differential equations are set to zero, and the resulting equations solved for the concentrations. This type of problem is quite common in many fields and is often represented mathematically in the following form:

$$f(x, p) = 0 \quad (11.3)$$

where  $x$  is the unknown, and  $p$  one or more parameters in the equations. All numerical methods for computing solutions to equation (11.3) start with an initial estimate for the solution, say  $x_1$ . The method is then applied iteratively until the estimate converges on the solution. One of the most well known methods for solving equation (11.3) is called the **Newton-Raphson method**. It can be easily explained using a geometric argument as



**Figure 11.7** Perturbation in  $S_1$  for cycle network 11.6. Because the conserved total  $S_1 + S_2$  changes, the steady state changes after the perturbation.

shown in Figure 11.10. Suppose  $x_1$  is the initial guess for the solution to equation (11.3). The method begins by estimating the slope of equation (11.3) at the value  $x_1$ , that is  $df/dx$ . A line is then drawn from the point  $(x_1, f(x_1))$ , with slope  $df/dx$ , until it intersects the  $x$  axis. The intersection,  $x_2$ , becomes the next guess for the method. This procedure is repeated until  $x_i$  is sufficiently close to the solution. For brevity, the parameter,  $p$ , is omitted from the following equations. From the geometry shown in Figure 11.10 one can express the slope of the line,  $\partial f / \partial x_1$  as:

$$\frac{\partial f}{\partial x_1} = \frac{f(x_1)}{x_1 - x_2}$$

This can be generalized to:

$$\frac{\partial f}{\partial x_k} = \frac{f(x_k)}{x_k - x_{k+1}}$$

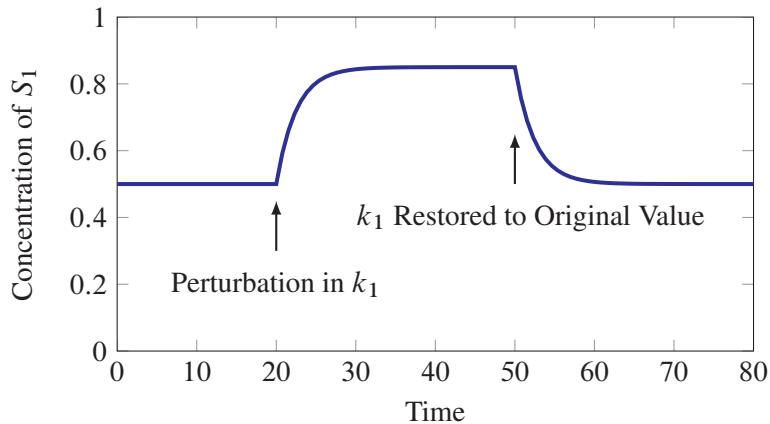
or by rearrangement as:

$$x_{k+1} = x_k - \frac{f(x_k)}{\partial f / \partial x_k} \quad (11.4)$$

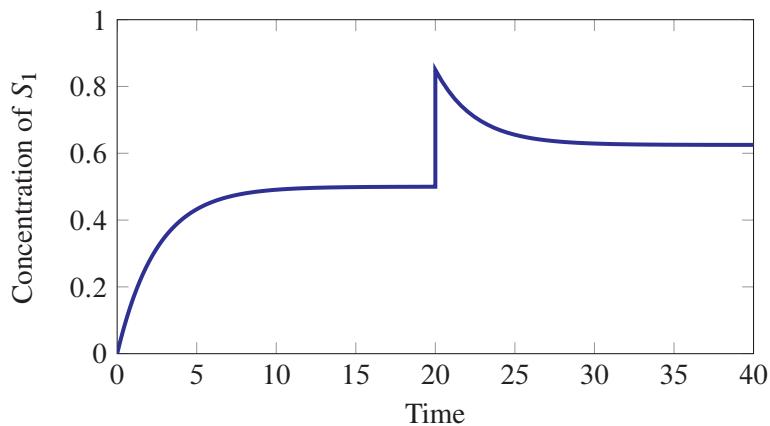
In form (11.4) we see the iterative nature of the algorithm.

Before the advent of electronic calculators with a specific square root button, calculator users would exploit the Newton method to estimate square roots. For example, if the square root of a number,  $a$ , is equal to  $x$ , that is  $\sqrt{a} = x$ , then it is true that:

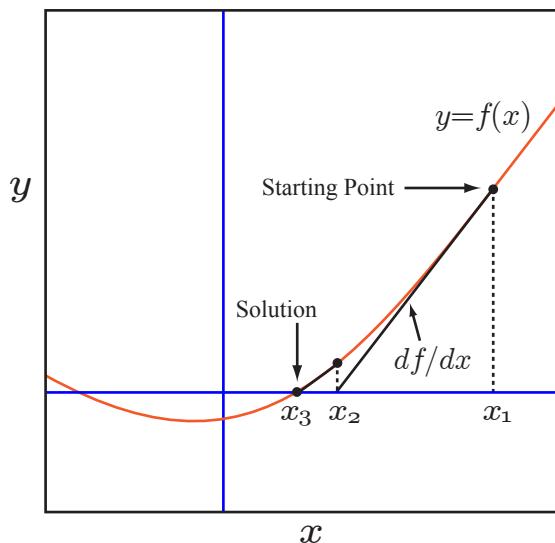
$$x^2 - a = 0$$



**Figure 11.8** Effect of Perturbing Model Parameters. Tellurium script: 11.6.



**Figure 11.9** Effect of Perturbing Model Parameters and Species Concentration.



**Figure 11.10** The geometry of Newton-Raphson's method.

This equation looks like an equation of the form (11.3). We can therefore apply the Newton formula (equation (11.4)) to this equation to obtain:

$$x_{k+1} = \frac{1}{2} \left( x_k + \frac{a}{x_k} \right) \quad (11.5)$$

Table 11.1 shows a sample calculation using this equation to compute the square root of 25. Note that only a few iterations are required to reach convergence.

Iteration	Estimate
0	15
1	8.33333
2	5.666
3	5.0392
4	5.0001525
5	5.0

**Table 11.1** Newton method used to compute the square root of 25, using equation (11.5) with a starting value of 15.

One important point to bear in mind is that the Newton-Raphson method is not guaranteed to converge to the solution. The solution depends heavily on the starting point and the nature of the system. In order to prevent the method from continuing without end in the case

when convergence fails, it is often useful to halt the method after a maximum of iterations (say 100). In a case like this, a new initial start is given and the method is repeated. In biochemical models we can always run a time-course simulation for a short while and use the end point of that as the starting point for the Newton method. This approach is much more reliable because we are staring the Newton-Raphson closer to the solution. If the method does converge to a solution, there are various ways to decide whether convergence has been achieved. Two such tests include:

1. Difference between successive solution estimates. We can test for the difference between solution,  $x_i$ , and the next estimate,  $x_{i+1}$ , if the absolute difference,  $|x_i - x_{i+1}|$ , is below some threshold. At this point we assume convergence has been achieved. Alternatively, we can check whether the relative error is less than a certain threshold (say, 1%). The relative error is given by:

$$\epsilon = \frac{x_{i+1} - x_i}{x_{i+1}} \times 100\%$$

The procedure can be made to stop at the  $i$ -th step if  $|f(x_i)| < \epsilon_f$  for a given  $\epsilon_f$ .

2. Difference between successive  $dS_i/dt$  estimates. Here we estimate the rates of change as the iteration proceeds, and assume convergence has been achieved when the difference between two successive rates of change are below some threshold. The threshold will usually be some small number for example  $10^{-6}$  or less. If we are dealing with a model that has more than one state variable, we can construct the sums of squares of the rates of change:

$$\sum \left( \frac{dS_i}{dt} \right)^2$$

The Newton method can be easily extended to systems of equations so that we express the Newton method in matrix form:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \left[ \frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}} \right]^{-1} \mathbf{f}(\mathbf{x}_k) \quad (11.6)$$

If  $m$  is the number of state variables or floating species in the model, then  $\mathbf{x}_k$  is an  $m$  dimensional vector of species concentrations,  $\mathbf{f}(\mathbf{x})$  is a vector containing the  $m$  rates of change, and  $\partial \mathbf{f}(\mathbf{x})/\partial \mathbf{x}$  the  $m \times m$  matrix called the Jacobian matrix (See 12.4). Note that the Jacobian must be invertible, this is equivalent to the division in the one variable form (11.5).

### Jacobian Matrix

The Jacobian is a common matrix used in many fields especially control theory and dynamical systems theory. We will frequently use it in this book. Given a set of equations:

$$\begin{aligned} y_1 &= f_1(x_1, \dots, x_n) \\ y_2 &= f_2(x_1, \dots, x_n) \\ &\vdots \\ y_m &= f_m(x_1, \dots, x_n) \end{aligned}$$

The Jacobian matrix is defined as the matrix of partial differentials:

$$\mathbf{J} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

### Newton Algorithm

1. Initialize the values of the concentrations,  $\mathbf{x}$ , to some initial guess obtained perhaps from a short time-course simulation.
2. Compute the values for  $\mathbf{f}(\mathbf{x})$ , that is the left-hand side of the differential equation  $(d\mathbf{x}/dt)$ .
3. Calculate the matrix of derivatives,  $\partial\mathbf{f}/\partial\mathbf{x}$ , that is  $d(d\mathbf{x}/dt)/d\mathbf{x}$ , at the current estimate for  $\mathbf{x}$ .
4. Compute the inverse of the matrix,  $\partial\mathbf{f}/\partial\mathbf{x}$ .
5. Using the information calculated so far, compute the next guess,  $\mathbf{x}_{k+1}$ .
6. Compute the sums of squares of the new value of  $\mathbf{f}(\mathbf{x})$  at  $\mathbf{x}_{k+1}$ . If the value is less than some error tolerance, assume the solution has been reached, else return to step 3 using  $\mathbf{x}_{k+1}$  as the new starting point.

Although the Newton method is seductively simple, it requires the initial guess to be sufficiently close to the solution in order for it to converge. In addition, convergence can be slow or not occur at all. A common problem is that the method can overshoot the solution and will then begin to rapidly diverge.

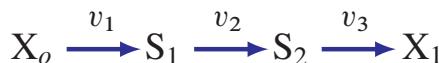
As mentioned previously, another strategy that is frequently used to compute the steady state is to first use a short time-course simulation to bring the initial estimate closer to the steady state. The assumption here is that the steady state is stable. The final point computed in the time-course is used to seed a Newton-like method. If the Newton method fails to converge, a second time-course simulation is carried out. This can be repeated as many times as desired. If there is suspicion that the steady state is unstable, one can also attempt to run a time-course simulation backwards in time. In general however, there is no sure way of computing the steady state automatically, and sometimes human intervention is required to supply good initial estimates.

As a result of these issues, an unmodified Newton method is rarely used in practice for computing the steady state of biochemical models. One common variant, the **Damped Newton method**, is more commonly employed. Both Gepasi and SCAMP use the Damped Newton method for computing the steady state. This method controls the derivative  $df/dx$ , by multiplying it by a factor  $\alpha$ . To prevent overshoot we can limit the range:  $(0 < \alpha < 1)$ . There are many variants on the basic Newton method and good simulation software will usually apply these for estimating the steady state.

In the last ten years more refined Newton-like methods have been devised, and one that is highly recommended is NLEQ2<sup>1</sup>. This is used by both Tellurium [150], PySCeS [125] and roadRunner [10, 156, 166] for computing the steady state. The stiff solver suite sundials<sup>2</sup> also incorporates an equation solver, however in the author's own experience it does not appear to be quite as good as NLEQ2.

## Solving the Steady State for a Simple Pathway

Let's illustrate the use of the Newton-Raphson method to solve the steady state for the following simple pathway. Assume that all three reactions are governed by simple mass-action reversible rate laws. Species  $X_o$  and  $X_1$  are assumed to be fixed, and only  $S_1$  and  $S_2$  are floating species.



The differential equations for the model are as follows:

$$\begin{aligned}\frac{dS_1}{dt} &= (k_1 X_o - k_2 S_1) - (k_3 S_1 - k_4 S_2) \\ \frac{dS_2}{dt} &= (k_3 S_1 - k_4 S_2) - (k_5 S_2 - k_6 X_1)\end{aligned}\tag{11.7}$$

The values for the rate constants and the boundary conditions are given in Table 11.2. This is a problem with more than one variable ( $S_1$  and  $S_2$ ) which means we must use the

<sup>1</sup><http://www.zib.de/en/numerik/software/ant/nleq2.html>

<sup>2</sup><https://computation.llnl.gov/casc/sundials/main.html>

Parameter	Value
$k_1$	3.4
$k_2$	0.2
$k_3$	2.3
$k_4$	0.56
$k_5$	5.6
$k_6$	0.12
$X_o$	10
$X_1$	0

**Table 11.2** Values for example (11.7).

Newton-Raphson matrix form (11.6) to estimate the steady state. To use this we require two vectors,  $\mathbf{x}_k$  and  $\mathbf{f}(\mathbf{x}_k)$  and one matrix,  $\partial \mathbf{f}(\mathbf{x})/\partial \mathbf{x}$ . The  $\mathbf{x}_k$  vector is simply:

$$\mathbf{x}_k = \begin{bmatrix} S_1 \\ S_2 \end{bmatrix}$$

The  $\mathbf{f}(\mathbf{x}_k)$  vector is given by the values of the differential equations:

$$\mathbf{f}(\mathbf{x}_k) = \begin{bmatrix} (k_1 X_o - k_2 S_1) - (k_3 S_1 - k_4 S_2) \\ (k_3 S_1 - k_4 S_2) - (k_5 S_2 - k_6 X_1) \end{bmatrix}$$

The  $\partial \mathbf{f}(\mathbf{x})/\partial \mathbf{x}$  matrix is the two by two **Jacobian matrix**. The elements of the Jacobian require the derivative to be computed. Software can estimate the derivatives numerically when more complex rate laws are applied. In this case however, it is easy to differentiate the equations to obtain the following Jacobian matrix:

$$\frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}} = \begin{bmatrix} \frac{d(dS_1/dt)}{dS_1} & \frac{d(dS_1/dt)}{dS_2} \\ \frac{d(dS_2/dt)}{dS_1} & \frac{d(dS_2/dt)}{dS_2} \end{bmatrix} = \begin{bmatrix} -k_2 - k_3 & -k_4 \\ k_3 & -k_4 - k_5 \end{bmatrix}$$

Notice that the elements of the Jacobian contain only rate constants. This is because the model is linear. This also means we need only evaluate the Jacobian and its inverse once, since the entries are constant. If we used nonlinear rate laws such as the Michaelis-Menten rate law, the Jacobian matrix would also contain terms involving species concentrations. In this case the Jacobian would need to be reevaluated at each iteration because the value for the species concentration will change at each iteration. For the current problem the Jacobian and its inverse is given by:

$$\text{Jacobian} = \begin{bmatrix} -2.86 & 5.6 \\ -0.56 & -11.2 \end{bmatrix}$$

$$\text{Jacobian}^{-1} = \begin{bmatrix} -0.3876 & -0.1938 \\ -0.01938 & -0.09898 \end{bmatrix}$$

Table 11.3 shows the progress of the iteration as we apply equation (11.6). What is interesting is that convergence only takes one iteration. This is because the model is linear. Nonlinear models may require more iterations. We can also see that after the first iteration, the rates of change have very small values. This is usually due to very small numerical errors in the computer arithmetic, but anything as small as  $10^{-14}$  may be considered zero.

Iteration	$S_1$	$S_2$	$dS_1/dt$	$dS_2/dt$
0	1	1	36.74	-10.64
1	13.18	0.6589	$2.8 \times 10^{-14}$	$-1.16 \times 10^{-13}$

**Table 11.3** Newton-Raphson applied to a Three Step Pathway with Linear Kinetics. Starting values for  $S_1$  and  $S_2$  are both set at one. Convergence occurs within one iteration. Note that the values for the rates of change are extremely small at the end of the first iteration, indicating we have converged.

## Computing the Steady State Using Software

The previous section showed how to compute the steady state using the Newton method. In practice we would not write our own solver, but instead use existing software to accomplish the same thing. To illustrate this, the following Tellurium script will define and compute the steady state all at once:

```
import tellurium as te

r = te.loada '''
$Xo -> S1; k1*Xo - k2*S1;
S1 -> S2; k3*S1 - k4*S2;
S2 -> $X1; k4*S2 - k6*X1;

// Initialize value
Xo = 10; X1 = 0;
k1 = 3.4; k2 = 0.2;
k2 = 2.3; k3 = 0.56;
k4 = 5.6; k6 = 0.12;

// Initial starting point
S1 = 1; S2 = 1;
''')

# Compute steady state
```

```
r.getSteadyStateValues()
print r.S1, r.S2
```

Running the above script yields steady state concentrations of 13.1783 and 0.658915 for  $S_1$  and  $S_2$ , respectively. This is the same if we compare these values to those in Table 11.3. Other tools will have other ways to compute the steady state, for example graphical interfaces will generally have a button marked ‘steady state’ that can be selected.

When using Matlab, the function `fsove` can be used to solve systems of nonlinear equations. In Mathematica one would use `FindRoot`.

### Effect of Conserved Cycles

Consider the conserved cycle in Figure 11.6. If we assume simple mass-action kinetics for the two rates,  $v_1$  and  $v_2$ , then we can write the differential equations for the system as:

$$\frac{dS_1}{dt} = k_2 S_2 - k_1 S_1$$

$$\frac{dS_2}{dt} = k_1 S_1 - k_2 S_2$$

From these equations it should be apparent that  $dS_1/dt = dS_2/dt$  due to the conservation law,  $S_1 + S_2 = T$ . To compute the steady state for the system, we must compute the Jacobian:

$$\mathbf{J} = \begin{bmatrix} -k_1 & k_2 \\ k_1 & -k_2 \end{bmatrix}$$

Computing the steady state requires the inverse of the Jacobian. However, in this case the Jacobian is singular, that is the rows of the matrix are linearly dependent and the determinant is zero. This means the inverse cannot be computed and therefore we cannot compute the steady state.

$$\text{Det} = -k_1(-k_2) - k_2(-k_1) = 0$$

Any analysis that requires the inversion of the Jacobian will fail for this system, including the Newton-Raphson method. This is characteristic of networks that include moiety conserved cycles. Modern simulation software avoids this problem by eliminating the dependent rows from the Jacobian, essentially splitting the species into two groups, a dependent and independent group. In the case of the simple conserved cycle (Figure 11.6), one species becomes the independent species, for example  $S_1$ , and the other the dependent species,  $S_2$ . In simulation software it means we only have one differential equation instead of two. The dependent species is computed algebraically from the independent species.

$$\frac{dS_1}{dt} = k_2 S_2 - k_1 S_1$$

$$S_2 = T - S_1$$

A more comprehensive discussion of conservation laws and their effects will be reserved for a separate book.

## 11.4 Introduction to Stability

Biological organisms are continually subjected to perturbations. These perturbations can originate from external influences such as changes in temperature, light, or the availability of nutrients. Perturbations can also arise internally due to the stochastic nature of molecular events or by genetic variation. One of the most remarkable and characteristic properties of living systems is their ability to resist such perturbations and maintain very steady internal conditions. For example the human body can maintain a constant core temperature of  $36.8^{\circ}\text{C} \pm 0.7$  even though external temperatures may vary widely. The ability of a biological system to maintain a steady internal environment is called **homeostasis**, a phrase introduced by Claude Bernard almost 150 years ago. Modern authors may also refer to this behavior as **robustness**.

The concept of homeostasis is related to the idea of stability in a dynamical system. While homeostasis refers to the degree to which a system can resist change, stability is related to whether a system can resist change or not. Thus an unstable system cannot resist any change. We can therefore informally define the stability of a system as follows:

A biochemical pathway is dynamically stable at steady state if small perturbations in the floating species concentrations relax back to the steady state.

We can illustrate a stable system using a simple two step model. Assume that the two step pathway has the following form:

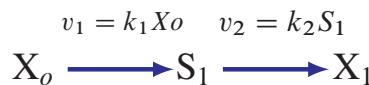
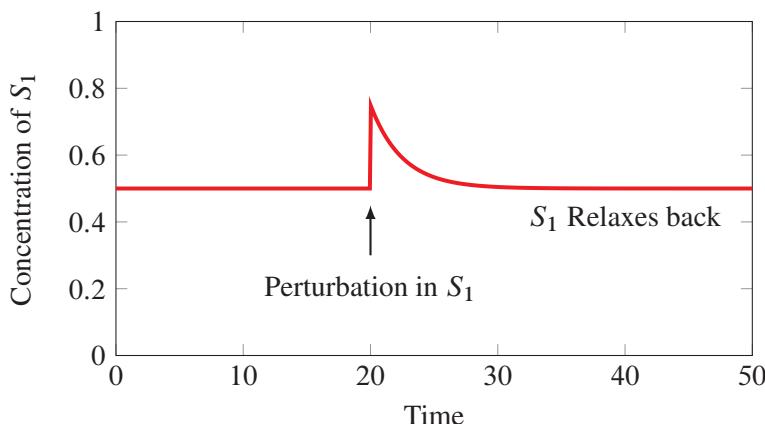


Figure 11.11 illustrates the results from a simulation of a simple two step biochemical pathway with one floating species,  $S_1$ . The initial concentrations of the model are set so that it is at steady state, that is no transients are seen between  $t = 0$  and  $t = 20$ . At  $t = 20$ , a perturbation is made to the concentration of  $S_1$  by injecting 0.25 units of  $S_1$  into the system. The system is now allowed to evolve further. If the system is stable, the perturbation will relax back to the original steady state, as it does in the simulation shown in Figure 11.11. This system is therefore considered stable.

The differential equation for the single floating species,  $S_1$ , is given by:

$$\frac{dS_1}{dt} = k_1 X_o - k_2 S_1 \tag{11.8}$$



**Figure 11.11** Stability of a simple biochemical pathway at steady state. The steady state concentration of the species  $S_1$  is 0.5. A perturbation is made to  $S_1$  by adding an additional 0.25 units of  $S_1$  at time = 20. The system is considered stable because the perturbation relaxes back to the original steady state. See Listing 11.7 for Tellurium script.

with a steady state solution of:

$$S_1 = k_1 X_o / k_2 \quad (11.9)$$

We know from the simulation in Figure 11.11 that the system appears to be stable, but can we show this algebraically? If the system is at steady state, let us make a small perturbation to the steady state concentration of  $S_1$ ,  $\delta S_1$ , and ask what is the new rate of change of  $S_1 + \delta S_1$  as a result of this perturbation? That is, what is  $d(S_1 + \delta S_1)/dt$ ? The new rate of change equation is rewritten as follows:

$$\frac{d(S_1 + \delta S_1)}{dt} = k_1 X_o - k_2(S_1 + \delta S_1)$$

If we insert the solution for  $S_1$ , equation (11.9) into the above equation we get:

$$\frac{d\delta S_1}{dt} = -k_2 \delta S_1 \quad (11.10)$$

This equation shows us that the rate of change of the disturbance,  $\delta S_1$  is *negative*. That is, the system reduces the disturbance so that the system returns back to the original steady state. If the rate of change in  $S_1$  had been positive instead of negative, the perturbation would have continued to diverge away from the original steady state and the system would then be considered unstable. We will return to the question of stability in greater detail in the next chapter.

## 11.5 Sensitivity Analysis

---

Sensitivity analysis at steady state looks at how particular model variables are influenced by model parameters. There are at least two reasons why it is interesting to examine sensitivities. The first is a practical one. Many kinetic parameters used in building biochemical models can have a significant degree of uncertainty about them. By determining how much a parameter has an influence on the model's state, we can decide whether we should try to improve the parameter's accuracy. A parameter that has considerable influence, but at the same time has significant uncertainty, is a parameter that should be determined more carefully by additional experimentation. On the other hand, a parameter that has little influence but has significant uncertainty associated with it, is relatively unimportant.

The second reason for measuring sensitivities is to provide insight. The degree to which a parameter can influence a variable tells us something about how the network responds to perturbations. Such a study can be used to answer questions about robustness and adaptation.

There are two broad approaches to sensitivity analysis, one is termed local and the other global. We will only look at local sensitivity analysis here.

### Local Sensitivity Analysis

Local sensitivities are defined in two ways, absolute and relative. Absolute sensitivities are given by the ratio of the absolute change in the variable to the absolute change in the parameter. That is:

$$S = \frac{\Delta V}{\Delta p}$$

where  $V$  is the variable, and  $p$  the parameter. This equation uses finite changes to the parameter and variable. Unfortunately, because most systems are nonlinear, the value for the sensitivity will be a function of the size of the finite change. To make the sensitivity independent of the size of the change, the sensitivity is usually defined in terms of infinitesimal changes:

$$S = \frac{dV}{dp}$$

Given that the sensitivities only measure perturbations in the immediate vicinity of the reference state, these sensitivities are called local. Although absolute sensitivities are simple, they have one significant drawback. The value can be influenced by the units used to measure the variable and parameter. Often in making experimental measurements, we won't be able to measure the quantity using the most natural units. Instead, we may have measurements in terms of fluorescence, colony counts, staining on a gel, and so on. It is most likely that the variable and parameter units will be quite different, and each laboratory may have its own particular way to express the measurement. Absolute sensitivities are therefore quite difficult to compare, and make reproducibility difficult.

To get around the problem of units, many people use relative sensitivities. These are simple scaled absolute sensitivities:

$$S = \frac{dV}{dp} \frac{p}{V} \quad (11.11)$$

The sensitivity is defined in terms of infinitesimal changes for the same reason cited before. The reader may also recall that elasticities are also measured this way. Relative sensitivities are immune to the units we use, and they correspond more closely to how many measurements are made, often in terms of relative or fold changes. In practice, steady state relative sensitivities should be measured by taking a measurement at the operating steady state, making a perturbation (preferably a small one), waiting for the system to reach a new steady state, and then measuring the system again. It is important to be aware that steady state sensitivities measure how a perturbation in a parameter moves the system from one steady state to another.

Sensitivities also form the basis for metabolic control analysis [82, 42], which is a framework for understanding how perturbations propagate through networks.

## Further Reading

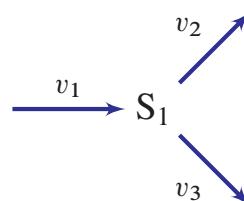
---

1. Tellurium web site <http://tellurium.analogmachine.org>
2. Kipp E, Herwig R, Kowald A, Wierling C and Lehrach H (2005) Systems Biology in Practice, Wiley-VCH Verlag.
3. Sauro HM (2011) Enzyme Kinetics for Systems Biology. ISBN: 978-0982477311.

## Exercises

---

1. Consider the following simple branched network:



where  $v_1 = v_o$ ,  $v_2 = k_1 S_1$  and  $v_3 = k_2 S_1$ .

- (a) Write the differential equation for  $S_1$ .

- (b) Derive the equation that describes the steady state concentration for  $S_1$ .
- (c) Derive the equations for the steady state fluxes through  $v_1$  and  $v_2$ .
- (d) Determine algebraically the scaled sensitivity (See equation 11.11) of the steady state concentration of  $S_1$  with respect to  $v_o$  and  $k_1$ .
- (e) Explain why the signs of the sensitivity with respect to  $v_o$  and  $k_1$  are positive and negative, respectively?
- (f) Assuming values for  $v_o = 1; k_1 = 0.5$  and  $k_2 = 2.5$ , compute the values for the sensitivities with respect to  $k_1$  and  $k_2$ .
- (g) What happens to the sensitivity with respect to  $k_1$  as  $k_1$  increases?
2. Derive equation (11.5).
3. Implement the Newton-Raphson algorithm and use it to find one solution to the quadratic equation:  $4x^2 + 6x - 8 = 0$ .
4. By changing the initial starting point of the Newton-Raphson algorithm, find the second solution to the quadratic equation from the previous question.
5. Using Tellurium, find the steady state for the following model:  
 $X_o \rightarrow S_1; k_1*X_o; S_1 \rightarrow X_1; k_2*S_1; S_1 \rightarrow X_2; k_3*S_1;$   
 Assume that  $X_o$ ,  $X_1$  and  $X_2$  have fixed concentrations. Assign suitable values to the rate constants and compute the steady state concentration of  $S_1$ .
6. Write a Tellurium script to perturb the value of  $X_o$  in the above model. Apply the perturbation as a square pulse; that is, the concentration of  $X_o$  rises, stays constant, then falls back to its original value. Make sure the system is at steady state before you apply the perturbation.
7. Explain what is meant by a stable and unstable steady state.
8. The steady state of a given pathway is stable. Explain the effect in general terms on the steady state if:
- A bolus of floating species is injected into the pathway.
  - A permanent change is applied to a kinetic constant.
9. Why are scaled sensitivities sometimes more advantageous than unscaled sensitivities?
10. Construct a simple linear pathway with four enzymes as shown below:



Assume that the edge metabolites,  $X_o$  and  $X_1$ , are fixed. Assign reversible Michaelis-Menten kinetics to each step and arbitrary values to the kinetics constants. Assign a modest value to the boundary metabolite,  $X_o$ , of 10 mM. Compute the steady state for your pathway. If the software fails to find a steady state, adjust the parameters. Once you have the steady state, use the model to compute the sensitivity of the steady state flux with respect to each of the enzyme maximal activities. You can compute each sensitivity by perturbing each maximal activity and observing what this does to the steady state flux.

How might you use the flux sensitivities in a practical application? Compute the sum of the four sensitivities, what value do you get? Can you make a statement about the sum?

## Appendix

See <http://tellurium.analogmachine.org> for more details of Tellurium.

```
import tellurium as te

# Simulation of a simple closed system
r = te.loada (''''
A -> B; k1 * A;
B -> A; k2 * B;

A = 10; B = 0;
k1 = 1; k2 = 0.5;
'''')

result = r.simulate(0, 3, 100)
r.lplot()
```

**Listing 11.1** Script for Figure 7.1.

```
import tellurium as te

# Simulation of an open system
r = te.loada (''''
$Xo -> S1; vo;
S1 -> S2; k1*S1 - k2*S2;
S2 -> $X1; k3*S2;

vo = 1
k1 = 2; k2 = 0; k3 = 3;
'''')
```

```
result = r.simulate(0, 6, 100)
r.plot()
```

**Listing 11.2** Script for Figure 7.2.

```
import tellurium as te

# Simple steady state system
r = te.loada ('''
    $Xo -> S1; k1*Xo;
    S1 -> $X1; k2*S1;

    k1 = 0.2; k2 = 0.4;
    Xo = 1;   S1 = 0.0;
''')

result = r.simulate(0, 20, 100, ["time", "S1"])
# Plot the results and set the y axis limits
r.plot(ylim=(0,0.6))
```

**Listing 11.3** Script for Figure 11.3.

```
import tellurium as te
import numpy

# Perturbing a species concentration
r = te.loada ('''
    $Xo -> S1; k1*Xo;
    S1 -> $X1; k2*S1;

    Xo = 1;
    S1 = 0.5;
    k1 = 0.2;
    k2 = 0.4;
''')

# Simulate the first part up to 20 time units
m1 = r.simulate(0, 20, 100, ["time", "S1"])

# Perturb the concentration of S1 by 0.35 units
r.S1 = r.S1 + 0.35

# Continue simulating from last end point
m2 = r.simulate(20, 50, 100, ["time", "S1"])
```

```
# Merge and plot the two halves of the simulation
result = numpy.vstack((m1, m2))
te.plotWithLegend(r, result)
```

**Listing 11.4** Script for Figure 11.11.

```
import tellurium as te
import numpy

# Multiple species perturbations
r = te.loada ('''
    $Xo -> S1; k1*Xo;
    S1 -> $X1; k2*S1;

    Xo = 1;
    S1 = 0.0;
    k1 = 0.2;
    k2 = 0.4;
''')

# Simulate the first part up to 20 time units
m1 = r.simulate(0, 20, 100, ["time", "S1"])

# Perturb the concentration of S1 by 0.35 units
r.S1 = r.S1 + 0.35

# Continue simulating from last end point
m2 = r.simulate(20, 40, 50, ["time", "S1"])

# Merge the data sets
m3 = numpy.vstack((m1, m2))

# Do a negative perturbation in S1
r.S1 = r.S1 - 0.35

# Continue simulating from last end point
m4 = r.simulate(40, 60, 50, ["time", "S1"])

# Merge and plot the final two halves of the simulation
result = numpy.vstack((m3, m4))
te.plotWithLegend(r, result)
```

**Listing 11.5** Script for Figure 11.5.

```
import tellurium as te
import numpy
```

```
import pylab

r = te.loada ('''
    $Xo -> S1; k1*Xo;
    S1 -> $X1; k2*S1;

    Xo = 1;
    S1 = 0.5;
    k1 = 0.2;
    k2 = 0.4;
''')

# Simulate the first part up to 20 time units
m1 = r.simulate(0, 20, 5, ["time", "S1"]).copy()

# Perturb the parameter k1
r.k1 = r.k1 * 1.7

# Simulate from the last point
m2 = r.simulate(20, 50, 40, ["time", "S1"]).copy()

# Restore the parameter back to ordinal value
r.k1 = 0.2

# Carry out final run of the simulation
m3 = r.simulate(50, 80, 40, ["time", "S1"])

# Merge all data sets and plot
result = numpy.vstack((m1, m2, m3))
pylab.ylim([0,1])
te.plotWithLegend(r, result)
```

**Listing 11.6** Script for Figure 11.8.

```
import tellurium as te
import numpy

# Stability illustration
r = te.loada ('''
    $Xo -> S1; k1*Xo;
    S1 -> $X1; k2*S1;

    Xo = 1;
    S1 = 0.5;
    k1 = 0.2;
    k2 = 0.4;
''' )
```

```
# Simulate the first part up to 20 time units
m1 = r.simulate(0, 20, 100, ["time", "S1"]).copy()

# Perturb the concentration of S1 by 0.35 units
r.S1 = r.S1 + 0.35

# Continue simulating from last end point
m2 = r.simulate(20, 50, 100, ["time", "S1"]);

# Merge and plot the two halves of the simulation
result = numpy.vstack ((m1, m2))
te.plotWithLegend(r, result)
```

**Listing 11.7** Tellurium script used to generate Figure 11.11.



# 12

## *Stability*

### 12.1 Stability

---

The previous chapter briefly touched on the concept of stability of a biochemical network. This chapter will delve more deeply into this topic. First let's refresh our memory by reviewing the simple model that was used to introduce stability. Figure 11.11 shows a simulation where a species concentration is disturbed, and over time relaxes back to the original steady state. This is an example of a stable steady state.

The differential equation for the single floating species,  $S_1$ , was given by:

$$\frac{dS_1}{dt} = k_1 X_o - k_2 S_1 \quad (12.1)$$

and as shown before, it has the steady state solution:

$$S_1 = k_1 X_o / k_2 \quad (12.2)$$

An important question to ask is whether the steady state is stable or not, that is, whether a perturbation will decay and return to the steady state. The differential equation describing the two step model is given by:

$$\frac{dS_1}{dt} = k_1 X_o - k_2 S_1$$

When the system is at steady state, let us make a small perturbation to the steady state concentration of  $S_1$ ,  $\delta S_1$  and ask how  $\delta S_1$  changes as a result of this perturbation. That is,

what is  $d(\delta S_1)/dt$ ? The new rate of change equation is rewritten as follows:

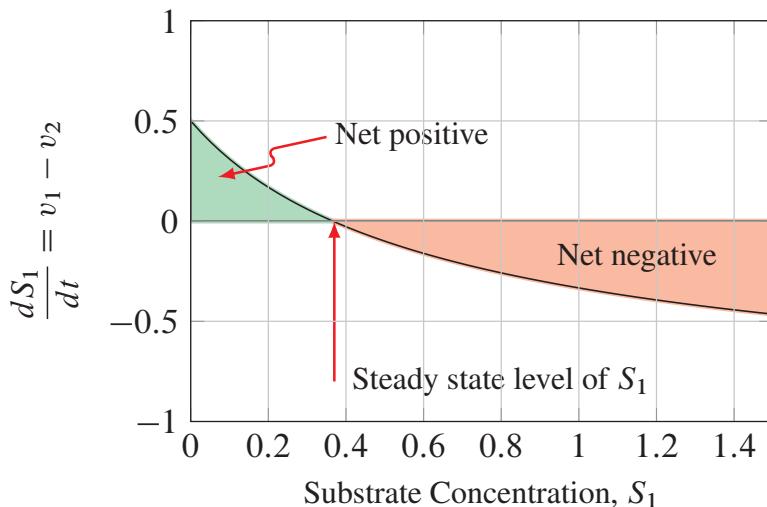
$$\frac{d(S_1 + \delta S_1)}{dt} = k_1 X_o - k_2(S_1 + \delta S_1)$$

If we insert the steady state solution for  $S_1$ , equation (12.2), into the above equation we are left with:

$$\frac{d\delta S_1}{dt} = -k_2\delta S_1 \quad (12.3)$$

In other words the rate of change of the *disturbance itself*,  $\delta S_1$ , is negative. The system attempts to reduce the disturbance so that the system returns back to the original steady state. Systems with this kind of behavior are called **stable**. If the rate of change in  $S_1$  had been positive instead of negative, the perturbation would have continued to diverge away from the original steady state and the system would then be considered **unstable**.

Let's look at this is graphically by plotting the rate of change,  $dS_1/dt$ , as a function of  $S_1$ , as shown in Figure 12.1. The steady state occurs when the net rate of change is zero, marked by the arrow. If the substrate level falls below this value, the net rate goes positive, thereby increasing the level of  $S_1$ . If the substrate rises above the steady state level, the graph shows the net rate of change going negative, so that  $S_1$  decreases. The system is therefore stable.



**Figure 12.1** Rate of change as a function of  $S_1$ . The arrow indicates the steady state for  $S_1$ . When  $S_1$  is below the steady state value, the net change is positive meaning that  $S_1$  will increase. When  $S_1$  is above the steady state value, the net change is negative meaning that  $S_1$  will decrease. The system is therefore stable.

This kind of stability is also called the **internal stability** because it describes the system's stability to perturbations in the internal state. We can informally define a stable system as:

**Internal Stability:** A biochemical pathway is internally stable if at steady state, small perturbations to the floating species relax back to the steady state.

**Caveat:** If the perturbed species is part of a conserved cycle (See section 3.8), then the total mass in the cycle must remain constant during the perturbation. This may require perturbing one species in a positive direction and another in a negative direction.

Let us divide both sides of equation (12.3) by  $\delta S_1$  and taking the limit, we find that  $\partial(dS_1/dt)/\partial S_1$  is equal to  $-k_2$ . The stability of this simple system can therefore be determined by inspecting the sign of  $\partial(dS_1/dt)/\partial S_1$ . In this case  $\partial(dS_1/dt)/\partial S_1 = -k_2$  which is negative, meaning the system is *stable*. It is worth noting that the larger the rate constant,  $k_2$ , the quicker the system relaxes back to steady state.

For systems with more than one species, a system's stability can be determined by looking at all the terms  $\partial(dS_i/dt)/\partial S_i$  which are given collectively by the expression:

$$\frac{d(\delta \mathbf{s}/dt)}{d\mathbf{s}} = \mathbf{J} \quad (12.4)$$

where  $\mathbf{J}$  is called the **Jacobian matrix** containing elements of the form  $\partial(dS_i/dt)/\partial S_i$ . Using this result we can generalize equation (12.3) to:

$$\frac{d(\delta \mathbf{s})}{dt} = \mathbf{J} \delta \mathbf{s} \quad (12.5)$$

where  $\mathbf{J}$  is given by

$$\begin{bmatrix} \frac{\partial(dS_1/dt)}{\partial S_1} & \dots & \frac{\partial(dS_1/dt)}{\partial S_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial(dS_m/dt)}{\partial S_1} & \dots & \frac{\partial(dS_m/dt)}{\partial S_m} \end{bmatrix}$$

Equation (12.5) is an example of an *unforced* linear differential equation and has the general form:

$$\frac{d\mathbf{x}}{dt} = \mathbf{A}\mathbf{x}$$

Solutions to unforced linear differential equations are well known and take the form:

$$x_j(t) = c_1 \mathbf{K}_1 e^{\lambda_1 t} + c_2 \mathbf{K}_2 e^{\lambda_2 t} + \dots + c_n \mathbf{K}_n e^{\lambda_n t}$$

The solution involves a sum of exponentials,  $e^{\lambda_i t}$ , constants  $c_i$ , and vectors,  $\mathbf{K}_i$ . The exponents of the exponentials are given by the eigenvalues (See Appendix E) of the matrix,  $\mathbf{A}$ , and  $\mathbf{K}_i$ , the corresponding eigenvectors. The  $c_i$  terms are related to the initial conditions assigned to the problem. It is possible for the eigenvalues to be complex, but in general if

the real parts of the eigenvalues are negative, the exponents will decay. If they are positive, the exponents will grow. We can therefore determine the stability properties of a given model by computing the eigenvalues of the Jacobian matrix and looking for any positive eigenvalues. Note that the elements of the Jacobian matrix will often be a function of the species levels; it is therefore important that the Jacobian be evaluated at the steady state of interest.

We can formally define the internal stability of a biochemical system as follows:

The steady state for the biochemical system:

$$\frac{ds}{dt} = \mathbf{Nv} \quad (12.6)$$

is stable if all the eigenvalues of the system's Jacobian matrix have negative real parts. The system is unstable if at least one of the eigenvalues has a positive real part.

There are many software packages that compute the eigenvalues of a matrix, and there are a small number of packages that can compute the Jacobian directly from a biochemical model. For example, the script below is taken from Tellurium. It defines a simple model, initializes the model values, computes the steady state, and then prints out the eigenvalues of the Jacobian matrix (Listing 12.1). For a simple one variable model, the Jacobian matrix only has a single entry and the eigenvalue corresponds to that entry. The output from running the script is given below, showing that the eigenvalue is  $-0.3$ . Since we have a negative eigenvalue, the pathway must be stable to perturbations in  $S_1$ .

```
import tellurium as te

rr = te.loada ('''
$Xo -> S1; k1*Xo;
S1 -> $X1; k2*S1;

// Set up the model initial conditions
Xo = 1; X1 = 0;
k1 = 0.2; k2 = 0.3;
''')

# Evaluation of the steady state
rr.getSteadyStateValues()

# print the eigenvalues of the full Jacobian matrix
print print rr.getFullEigenValues()

# Output follows:
[[[-0.3  0. ]]]
```

---

**Listing 12.1** Computing eigenvalues to determine stability.

---

**Example 12.1**

The following system:

$$\rightarrow S_1 \rightarrow S_2 \rightarrow$$

is governed by the set of differential equations:

$$\frac{dS_1}{dt} = 3 - 2S_1$$

$$\frac{dS_2}{dt} = 2S_1 - 4S_2$$

The Jacobian matrix is computed by differentiating the equations with respect to  $S_1$  and  $S_2$ :

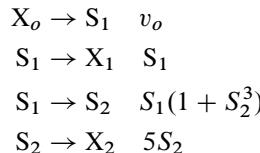
$$\mathbf{J} = \begin{bmatrix} -2 & 0 \\ 2 & -4 \end{bmatrix}$$

The eigenvalues for this matrix are:  $-2$  and  $-4$ , respectively. Since both eigenvalues are negative, the system is stable to small perturbations in  $S_1$  and  $S_2$ .

---

**Example 12.2**

Consider the system:



where  $X_o$ ,  $X_1$  and  $X_2$  are fixed species. At steady state  $S_1 = 2.295$  and  $S_2 = 1.14$  with parameter values  $v_o = 8$ . Determine whether this steady state is stable or not. The differential equations for the system are given by:

$$\frac{dS_1}{dt} = v_o - S_1 - S_1(1 + S_2^3)$$

$$\frac{dS_2}{dt} = S_1(1 + S_2^3) - 5S_2$$

The Jacobian matrix is computed by differentiating the equations with respect to the steady state values of  $S_1$  and  $S_2$ :

$$\mathbf{J} = \begin{bmatrix} -2 - S_2^3 & -3S_1S_2^2 \\ 1 + S^3 & -5 + 3S_1S_2^2 \end{bmatrix} = \begin{bmatrix} -3.4815 & -8.948 \\ 2.482 & 3.948 \end{bmatrix}$$

The eigenvalues for this matrix are:  $0.2333 + 2.9i$  and  $0.2332 - 2.9i$ , respectively. Since the real parts of the eigenvalues are positive, the system is unstable to small perturbations in  $S_1$  and  $S_2$ .

The pattern of eigenvalues tells us a lot about stability, but also about the kind of the transients that occur after a perturbation. The following sections will investigate this subject further.

## 12.2 Jacobian for Biochemical Systems

---

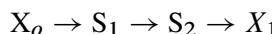
For a given set of differential equations, we can compute the Jacobian by differentiating the equations with respect to the model variables. However for biochemical networks, the Jacobian can be written in a special way that highlights the importance of the network structure and kinetics of the biochemical reaction steps. To do this, let us first define the unscaled elasticity (D.18) as:

$$\mathcal{E}_S^v = \frac{\partial v}{\partial S}$$

where  $v$  is a reaction rate and  $S$  an effector of the reaction. For example, if  $v = k_1 S$ , the unscaled elasticity,  $\mathcal{E}_S^v = k_1$ . The matrix of unscaled elasticities can be defined as:

$$\frac{\partial \mathbf{v}}{\partial \mathbf{s}} = \begin{bmatrix} \mathcal{E}_{S_1}^{v_1} & \mathcal{E}_{S_2}^{v_1} & \mathcal{E}_{S_3}^{v_1} \\ \mathcal{E}_{S_1}^{v_2} & \mathcal{E}_{S_2}^{v_2} & \mathcal{E}_{S_3}^{v_2} \\ \mathcal{E}_{S_1}^{v_3} & \mathcal{E}_{S_2}^{v_3} & \mathcal{E}_{S_3}^{v_3} \end{bmatrix}$$

The example shows a three by three elasticity matrix. An elasticity matrix has  $n$  rows representing  $n$  reactions and  $m$  columns represent  $m$  species. Many entries in the elasticity matrix will often be zero. For example, consider the pathway:



where  $X_o$  and  $X_1$  are fixed species. The pathway has three reactions, which we will designate  $v_1$ ,  $v_2$ , and  $v_3$  and two floating species,  $S_1$  and  $S_2$ . The unscaled elasticity matrix will therefore be a three by two matrix. If we assume reversibility or product inhibition in all three reactions, the entries in the matrix will be:

$$\frac{\partial \mathbf{v}}{\partial \mathbf{s}} = \begin{bmatrix} \mathcal{E}_{S_1}^{v_1} & \mathcal{E}_{S_2}^{v_1} \\ \mathcal{E}_{S_1}^{v_2} & \mathcal{E}_{S_2}^{v_2} \\ \mathcal{E}_{S_1}^{v_3} & \mathcal{E}_{S_2}^{v_3} \end{bmatrix} = \begin{bmatrix} \mathcal{E}_{S_1}^{v_1} & 0 \\ \mathcal{E}_{S_1}^{v_2} & \mathcal{E}_{S_2}^{v_2} \\ 0 & \mathcal{E}_{S_2}^{v_3} \end{bmatrix}$$

Note that the entries  $\mathcal{E}_{S_2}^{v_1}$  and  $\mathcal{E}_{S_1}^{v_3}$  are zero because  $S_2$  has no direct effect on  $v_1$ , and  $S_1$  has no direct effect on  $v_3$ . Some of the unscaled elasticities will also be negative. For example,  $\mathcal{E}_{S_2}^{v_2}$  will be negative because increases in  $S_2$  will slow down the  $v_2$  reaction rate due to product inhibition.

Recall that an element of the Jacobian is defined as:

$$\frac{\partial(dS_j/dt)}{\partial S_j}$$

that is the differential equation differentiated with respect to a species. However, we also know that the vector of rates of change is given by the system equation:

$$\frac{ds}{dt} = \mathbf{N}\mathbf{v}$$

Differentiating this with respect to  $s$  yields:

$$\frac{\partial}{\partial s} \left( \frac{ds}{dt} \right) = \mathbf{N} \frac{\partial \mathbf{v}}{\partial s}$$

Hence, the Jacobian is the product of the stoichiometry and the unscaled elasticity matrix:

$$\mathbf{J} = \mathbf{N} \frac{\partial \mathbf{v}}{\partial s}$$

Given that stability is determined from the Jacobian, this result indicates that stability is a function of network topology and the kinetics of the individual reactions. The result indicates that it is not always possible to discern the functional dynamics of a motif (Chapter 3) just from the topological pattern. The dynamics also depend on the kinetics of the constituent parts.

The dynamics of a network is a function of the network topology *and* the kinetics of its constituent parts.

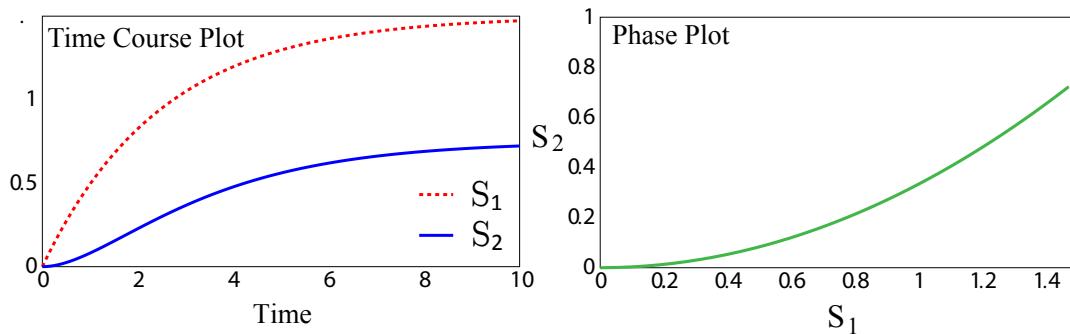
## 12.3 External Stability

There is one other type of stability that is useful with respect to biochemical systems, namely **external stability**. This refers to the idea that if a system is externally stable, then a finite change to an input of the system should elicit a finite change to the internal state of the system. In control theory this is called BIBO, or Bounded Input Bounded Output stability. It is very important to bear in mind that the finite change in the internal state refers to a linearized system. When the system is nonlinear, the output may be bounded by physical constraints.

A system that is internally unstable will also be unstable to changes in the systems inputs. In biochemical systems such inputs could be the boundary species that feed a pathway, a drug intervention, or the total mass of a conserved cycle. External stability can be determined using the same criteria used for internal stability, that is the real parts of eigenvalues of the Jacobian matrix should all be negative.

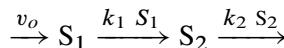
## 12.4 Phase Portraits

The word **phase space** refers to a space where all possible states are shown. For example, in a biochemical pathway with two species,  $S_1$  and  $S_2$ , the phase space consists of all possible trajectories of  $S_1$  and  $S_2$  in time. For a two dimensional system with species  $S_1$  and  $S_2$ , the phase space can be conveniently displayed with  $S_1$  on one axis and  $S_2$  on the other. A line on a two dimensional plane will represent how  $S_1$  and  $S_2$  move with respect to each other in time. Figure 12.2 shows a time-course plot for a simple three step pathway with two species and the corresponding trajectory in the phase plot. In a real phase plot we would have all possible trajectories shown rather than just one. Figure 12.3 shows the same phase plot but this time with forty-four trajectories. Note they all converge on a single point that represents the system's steady state.



**Figure 12.2** Time course simulation plot and corresponding phase plot.

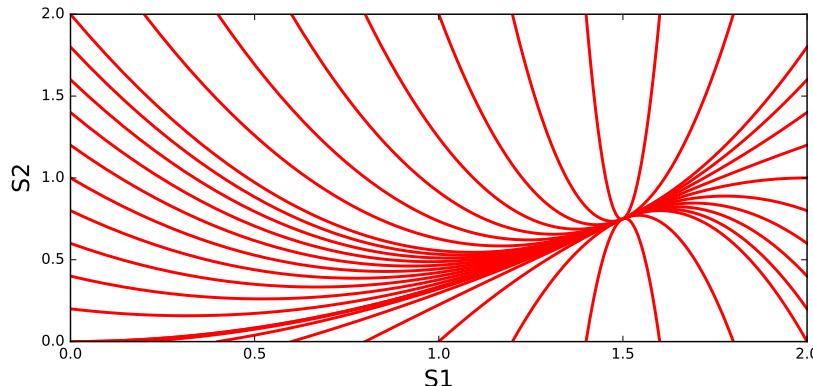
A visual representation of the phase space is often called a **phase portrait** or **phase plane**. To illustrate a phase portrait consider the following simple reaction network:



with two linear differential equations:

$$\frac{dS_1}{dt} = v_o - k_1 S_1$$

$$\frac{dS_2}{dt} = k_1 S_1 - k_2 S_2$$



**Figure 12.3** Multiple trajectories plotted on the phase plot Tellurium Listing: 12.5.

We can assign particular values to the parameters, set up some initial conditions, and plot the evolution of  $S_1$  and  $S_2$  in phase space. If we replot the solution using many different initial conditions, we get something that looks like the plots shown in Figures 12.3 to 12.9.

The plots illustrate a variety of transient behaviors around the steady state. These particular transient behaviors apply specifically to linear differential equations. If we have a nonlinear system and we linearize the system around the steady state, the linearized system will also behave in a way suggested by these plots.

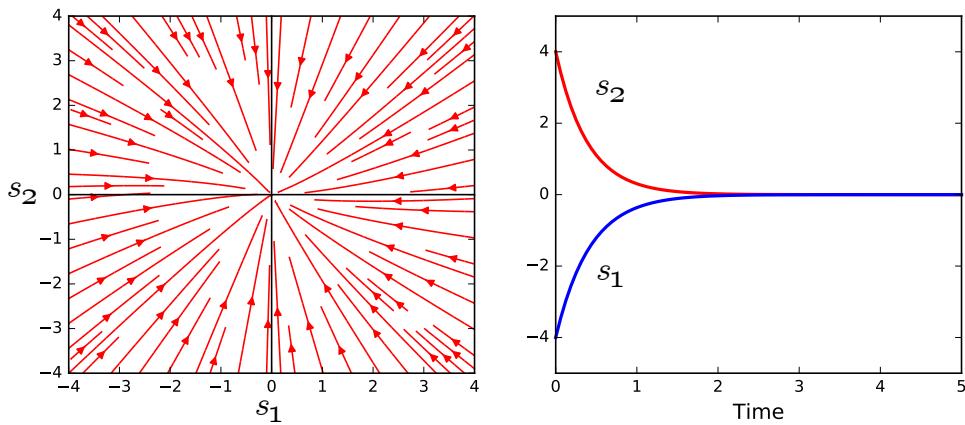
Consider the general two dimensional linear set of differential equations:

$$\begin{aligned}\frac{dS_1}{dt} &= a_{11}S_1 + a_{12}S_2 \\ \frac{dS_2}{dt} &= a_{21}S_1 + a_{22}S_2\end{aligned}$$

As we've seen already, such a two dimensional linear system of differential equations has solutions of the form:

$$\begin{aligned}S_1 &= c_1 k_1 e^{\lambda_1 t} + c_2 k_2 e^{\lambda_2 t} \\ S_2 &= c_3 k_3 e^{\lambda_3 t} + c_4 k_4 e^{\lambda_4 t}\end{aligned}$$

That is, a sum of exponential terms. The  $c_i$  and  $k_i$  terms are constants related to the initial conditions and eigenvectors, respectively, but the  $\lambda_i$  terms or eigenvalues determine the qualitative pattern that a given behavior might have. It should be noted that the eigenvalues can be complex or real numbers. In applied mathematics,  $e$  raised to a complex number immediately suggests some kind of periodic behavior. Let us consider different possibilities for the various eigenvalues.



**Figure 12.4** Trajectories for a two species reaction network. On the left is the phase plot and on the right, a single transient as a function of time. This system illustrates a stable node corresponding to *Negative Eigenvalues* in the Jacobian. Matrix  $A$ :  $a_{11} = -2, a_{12} = 0, a_{21} = -0.15, a_{22} = -2$ . Corresponding eigenvalues:  $\lambda_1 = -2, \lambda_2 = -2$ . The symmetry in the trajectories is due to eigenvalues of the same magnitude.

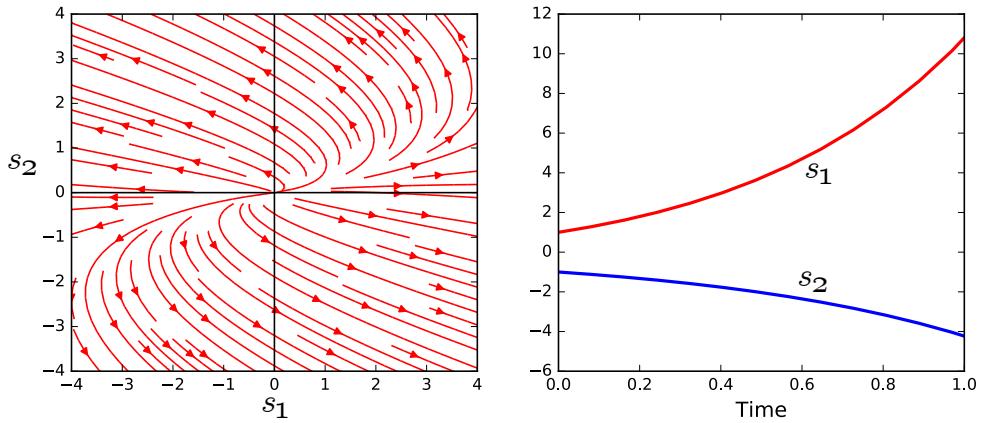
- **Both Eigenvalues have the same sign, different magnitude but are real.** If both eigenvalues are negative, the equations describe a system known as a **stable node**. All trajectories move towards the steady state point. If the eigenvalues have the same magnitude and the  $c_i$  terms have the same magnitude, the trajectories move to the steady state in a symmetric manner as shown in Figure 12.4.

If the two eigenvalues are both positive, the trajectories move out from the steady state reflecting the fact that the system is unstable. Such a point is called an **unstable node**. If the two eigenvalues have different magnitudes but are still positive, the trajectories twist as shown in Figure 12.5.

- **Real Eigenvalues but of opposite sign.** If the two eigenvalues are real but of opposite sign, we see behavior called a **saddle-node** shown in Figure 12.6. This is where the trajectories move towards the steady state in one direction, called the stable manifold, and form a stable ridge. In all other directions trajectories move away, resulting in an unstable manifold. Since trajectories can only move towards the steady state if they are exactly on the stable ridge, the saddle nodes are generally considered unstable.

- **Complex Eigenvalues.** Sometimes the eigenvalues can be complex, that is of the form  $a + ib$  where  $i$  is the imaginary number. It may seem strange that the solution to a differential equation that describes a physical system can admit complex eigenvalues. To understand what this means we must recall Euler's formula:

$$e^{i\theta} = \cos(\theta) + i \sin(\theta)$$



**Figure 12.5** Trajectories for a two species reaction network. On the left is the phase plot and on the right a single transient as a function of time. This system illustrates an unstable node, also called an improper node corresponding to *Positive Eigenvalues*. Matrix  $A$ :  $a_{11} = 1.2, a_{12} = -2, a_{21} = -0.05, a_{22} = 1.35$ . Corresponding eigenvalues:  $\lambda_1 = 1.6, \lambda_2 = 0.95$ .

Description	Eigenvalues	Behavior
Both Positive	$r_1 > r_2 > 0$	Unstable
Both Negative	$r_1 < r_2 < 0$	Stable
Positive and Negative	$r_1 < 0 < r_2$	Saddle point
Complex Conjugate	$r_1 > r_2 > 0$	Unstable spiral
Complex Conjugate	$r_1 < r_2 < 0$	Stable spiral
Pure Imaginary	$r_1 = r_2 = 0$	Center

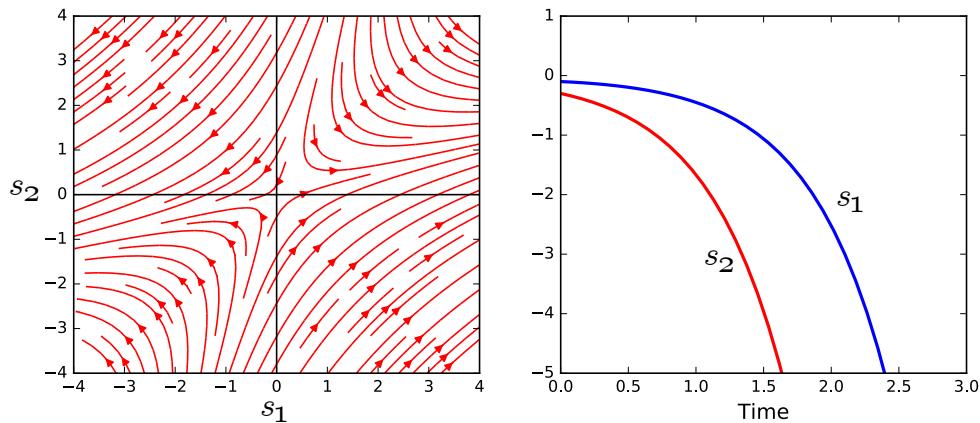
**Table 12.1** Summary of Node Behaviors.

Extended to:

$$e^{(a+bi)t} = e^{at} \cos(bt) + i e^{bt} \sin(bt) \quad (12.7)$$

When the solutions are expressed in sums of sine and cosine terms, the imaginary parts cancel out, leaving just trigonometric terms with real parts (The proof is provided at the end of the chapter in an appendix). This means that systems with complex eigenvalues show periodic behavior.

Figures 12.7, 12.8, and 12.9 show typical trajectories when the system admits complex eigenvalues. If the real parts are positive, the spiral trajectories move outwards away from the steady state. Such systems are unstable. In a pure linear system, the trajectories will expand out forever. They will only stop and converge to a stable oscillation if the system



**Figure 12.6** Trajectories for a two species reaction network. On the left is the phase plot and on the right a single transient as a function of time. This system illustrates a saddle node corresponding to *One Positive and One Negative Eigenvalue*. Matrix  $A$ :  $a_{11} = 2, a_{12} = -1, a_{21} = 1, a_{22} = -2$ . Corresponding eigenvalues:  $\lambda_1 = -1.73, \lambda_2 = 1.73$ .

has nonlinear elements which limits the expansion. In these cases we observe limit cycle behavior.

If the real parts of the eigenvalues are negative, the spiral trajectory moves into the steady state and is therefore considered stable.

### Conjugate Pair

A complex conjugate pair is a complex number of the form:  $a \pm bi$ . The eigenvalues for a two variable linear system with matrix  $A$  can be computed directly using the relation:

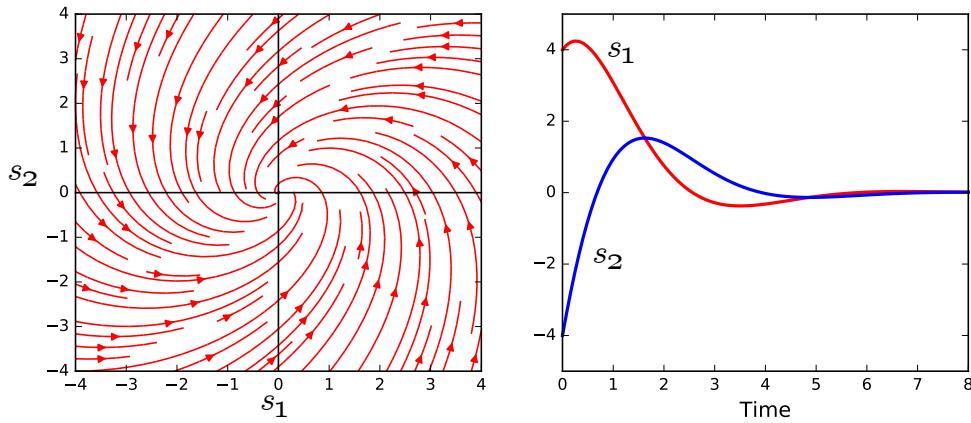
$$\lambda = \frac{\text{tr}(A) \pm \sqrt{\text{tr}^2(A) - 4 \det(A)}}{2}$$

where  $\text{tr}(A) = a + d$ , and  $\det(A) = ad - bc$ . If the term in the square root is negative, the eigenvalues will always come out as a conjugate pair owing to the  $\pm$  term. If  $\text{tr}^2(A) - 4 \det(A) < 0$ , then the solution will be the conjugate pair:

$$\lambda = \frac{\text{tr}(A)}{2} \pm \frac{\sqrt{\text{tr}^2(A) - 4 \det(A)}}{2}$$

Therefore a complex eigenvalue will always be accompanied by its conjugate partner.

- **Imaginary Eigenvalues with Zero Real Parts.** It is possible for the pair of eigenvalues to have no real component but retain an imaginary part. In this situation the behavior



**Figure 12.7** Trajectories for a two species reaction network. On the left is the phase plot and on the right, a single transient as a function of time. This system illustrates a stable spiral node corresponding to *Negative Complex Eigenvalues*. Matrix  $A$ :  $a_{11} = -0.5, a_{12} = -1, a_{21} = 1, a_{22} = -1$ . Corresponding eigenvalues:  $\lambda_1 = -0.75 + 0.97i, \lambda_2 = -0.75 - 0.97i$ .

is called a center. This is where the trajectory orbits the steady state. The oscillation is an unusual one in the sense that it implies zero dampening in the system, in other words zero energy loss. Such a situation would be very rare in biology and even in non-living systems, such behavior tends to be idealized. For example a pendulum in a vacuum with no friction at the fulcrum. The other unusual aspect of a center is that the oscillation is depending on the starting condition. Again we can relate this to a pendulum where the swing depending on how much force we initially apply to the pendulum bob. Biological oscillators are invariably energy dependent and the frequency is independent of the initial conditions. Biological oscillators therefore tend not to be center types but are a behavior that results from nonlinearities in the system.

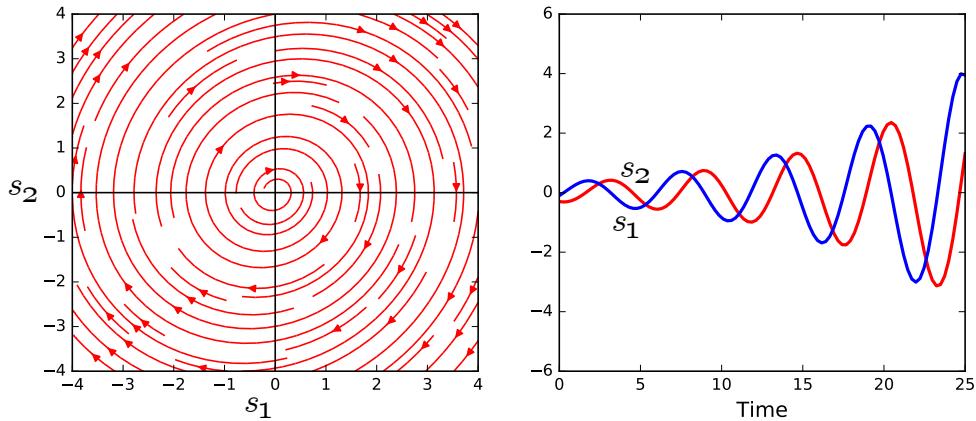
## 12.5 Bifurcation Plots

In its simplest form, a bifurcation plot is just a plot of the steady state value of a system variable, such as a concentration or flux versus a parameter of the system. For example, we know that the steady state solution for the simple system:

$$\frac{dS_1}{dt} = k_1 X_o - k_2 S_1 \quad (12.8)$$

was given by:

$$S_1 = k_1 X_o / k_2 \quad (12.9)$$



**Figure 12.8** Phase portrait for a two species reaction network. Unstable spiral node. **Positive Complex Eigenvalues.** Matrix  $A$ :  $a_{11} = 0, a_{12} = 1.0, a_{21} = -1.2, a_{22} = 0.2$ . Corresponding eigenvalues:  $\lambda_1 = 0.1 + 1.09i, \lambda_2 = 0.1 - 1.09i$ .

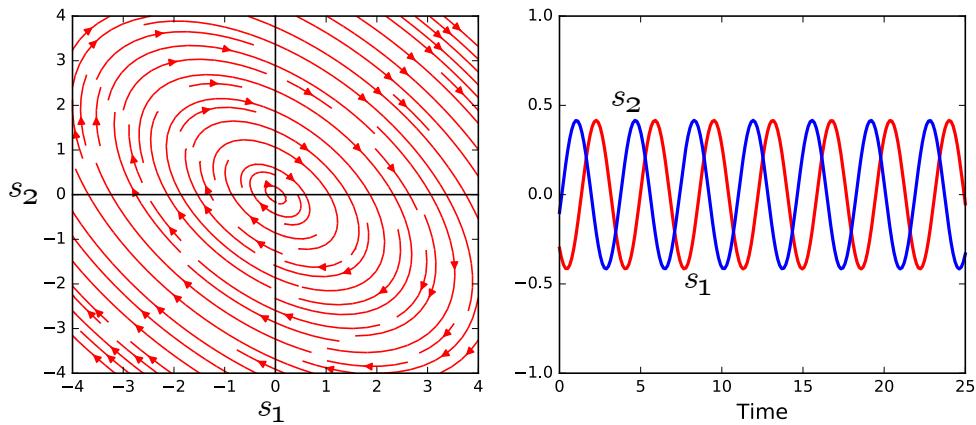
We can plot the steady state value of  $S_1$  as a function of  $k_2$  as shown in Figure 12.11. This isn't a particularly interesting bifurcation plot however and misses one of the most important characteristics.

Equation (12.9) shows that the simple system (12.8) only has one steady state for a given set of parameters. That is, if we set values to  $X_o, k_1$ , and  $k_2$ , we find there is only *one value* of  $S_1$  that satisfies these parameter settings. This is what Figure 12.11 also demonstrates. What is more interesting is when a system admits multiple possible steady state values for a given set of parameter values. To illustrate this behavior, let us look at a common system that can admit three possible steady states. It is in these cases that bifurcation plots become particularly useful and more interesting.

## Bistable Systems

Bifurcation plots can be useful for identifying changes in qualitative behavior, particularly systems that have multiple steady states. Consider the system shown Figure (12.12). This shows a gene circuit with a positive feedback loop. As the transcription factor  $x$  accumulates, it binds to an operator site upstream of the gene which increases its synthesis. The more transcription factor made, the higher the rate of expression.

At first glance this would seem to be a very unstable situation. One might imagine that the transcription factor concentration would continue to increase without limit. However, the physical limits, in this case the saturation of the translation, transcription, and degradation machinery, ultimately limits the upper value for the concentration of transcription factor. To investigate the properties of this networks we will construct a simple model. This model



**Figure 12.9** Phase portrait for a two species reaction network. Center node. **Complex Eigenvalues, Zero Real Part.** Matrix  $A$ :  $a_{11} = 1, a_{12} = 2.0, a_{21} = -2, a_{22} = -1$ . Corresponding eigenvalues:  $\lambda_1 = 0 + 1.76i, \lambda_2 = 0 - 1.76i$ . The script for generating these phase plots can be found in the chapter appendix: 12.6

uses the following kinetic laws for the synthesis and degradation steps:

$$v_1 = b + k_1 \frac{x^4}{k_2 + x^4}$$

$$v_2 = k_3 x$$

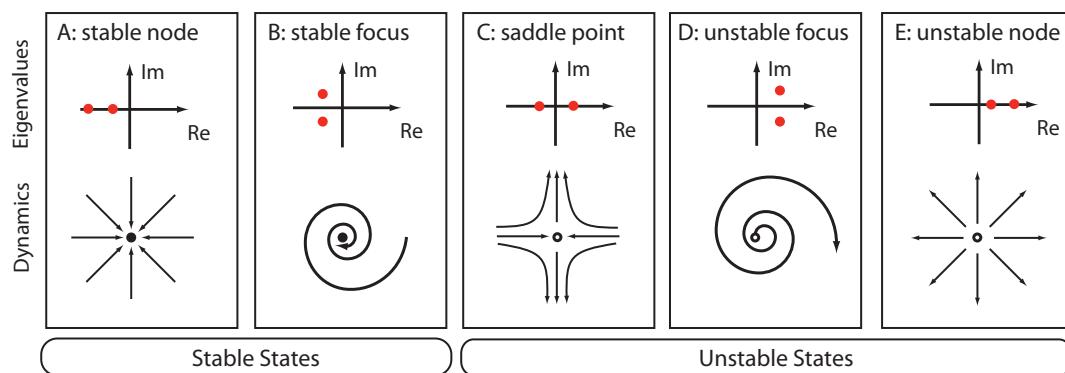
$v_1$  is a Hill like equation with a Hill coefficient of four and a basal rate of  $b$ .  $v_2$  is a simple irreversible mass-action rate law. The differential equation for the model is:

$$\frac{dx}{dt} = v_1 - v_2$$

To find the steady state, set the differential equation to zero and attempt to analytically solve for  $x$ . If we try this we get a solution that is complex to interpret.

A better way to understand what is going on, is to plot both rate laws as a function of transcription factor,  $x$ . When we do this we obtain Figure 12.13. The intersection points marked with filled circles indicate the steady state solutions because at these points,  $v_1 = v_2$ . If we vary the slope of  $v_2$  by changing  $k_3$ , the intersection points will change (Figure 12.14). At a high  $k_3$  value, only one intersection point remains (Panel c), the low intersection point. If the value of  $k_3$  is low, only the high intersection point remains (Panel a). However with the right set of parameter values, we can make a system with three steady state values (Panel b).

We can determine the three different steady state stabilities by doing a simple graphical analysis on Figure 12.13. Figure 12.15 shows the same plot but with perturbations.



**Figure 12.10** Summary of behaviors including dynamics and associated eigenvalues for a two dimensional linear system. Adapted from “Computational Models of Metabolism: Stability and Regulation in Metabolic Networks”, Adv in Chem Phys, Vol 142, Steuer and Junker.

Starting with the first steady state in the low left corner of Figure 12.15, consider a perturbation made in  $x$ ,  $\delta x$ . This means that both  $v_1$  and  $v_2$  increase, however  $v_2 > v_1$  meaning that after the perturbation, the rate of change in  $x$  is *negative*. Since it is negative, this restores the perturbation back to the steady state. The same logic applies to the upper steady state. This tells us that the lower and upper steady states are both stable.

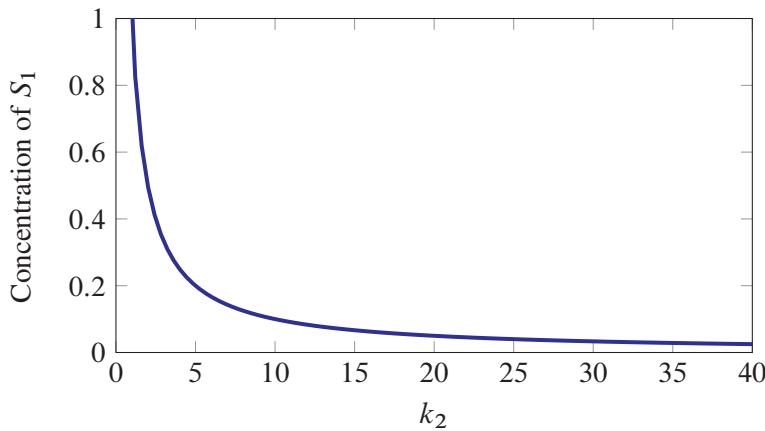
What about the middle steady state? Consider again a perturbation,  $\delta x$ . This time  $v_1 > v_2$  which means that the rate of change of  $x$  is *positive*. Since it is positive, the perturbation, instead of falling back, continues to grow until  $x$  reaches the upper steady state. We conclude that the middle steady state is unstable. This system possess three steady states, one unstable and two stable. Such a system is known as a **bistable system** because it can rest in one of two stable states but not the third.

Another way to observe the different steady states is to run a time-course simulation at many different starting points. Figure 12.16 shows the plots generated using the script in Listing 12.2. The plots show two steady states, a high state at around 40, and a low state at around 3. Notice that there is no third state observed. As we have already discussed, the middle steady state is unstable, and all trajectories diverge from this point. It is therefore not possible when doing a time-course simulation to observe an unstable steady state since there is no way to reach it.<sup>1</sup>

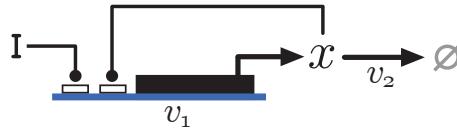
```
import tellurium as te
import numpy

rr = te.loada ('''
```

<sup>1</sup>The author has been reliably informed that running time backwards in a time-course simulation will cause the simulation to converge on the unstable steady state. The author has not tried this himself, however.



**Figure 12.11** Steady state concentration of  $S_1$  as a function of  $k_2$  for the system,  $dS_1/dt = k_1X_o - k_2S_1$ .



**Figure 12.12** System with Positive Feedback.

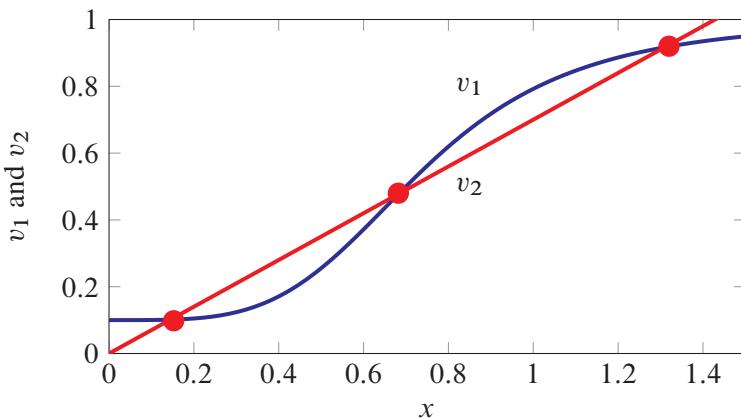
```
J1: $Xo -> x; 0.1 + k1*x^4/(k2+x^4);
x -> $w; k3*x;

k1 = 0.9;
k2 = 0.3;
k3 = 0.7;
x = 0.05;
''')

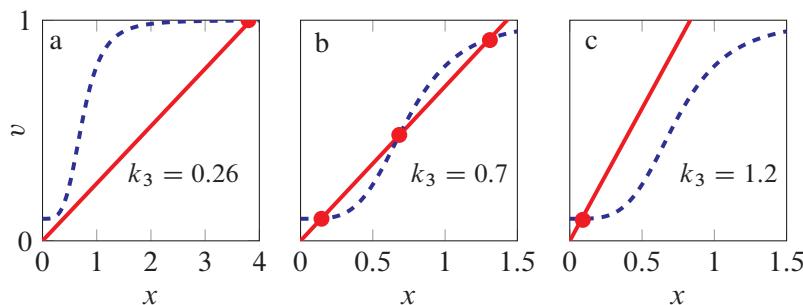
m = rr.simulate(0, 15, 100)
for i in range(1, 10):
    rr.x = i*0.2
    mm = rr.simulate(0, 15, 100, ["x"])
    m = numpy.hstack((m, mm))
te.plotArray(m)
```

**Listing 12.2** Tellurium script used to generate Figure 12.16.

We can get an estimate for the values of all three steady states from Figure 12.13. Reading



**Figure 12.13** Reaction velocities,  $v_1$  and  $v_2$ , as a function of  $x$  for the system in Figure 12.11. The intersection points marked by full circles indicate possible steady states. Computed using the SBW rate law plotter.  $k_1 = 0.9$ ;  $k_2 = 0.3$ ;  $k_3 = 0.7$ ;  $b = 0.1$ .

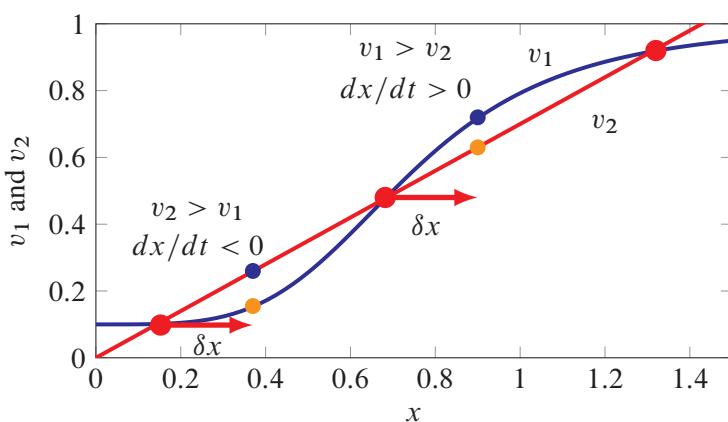


**Figure 12.14**  $v_1$  and  $v_2$  plotted against  $x$  concentration. Intersection points on the curves mark the steady state points. Panel a) One intersection point at a high steady state; b) Three steady states; c) One low steady state.

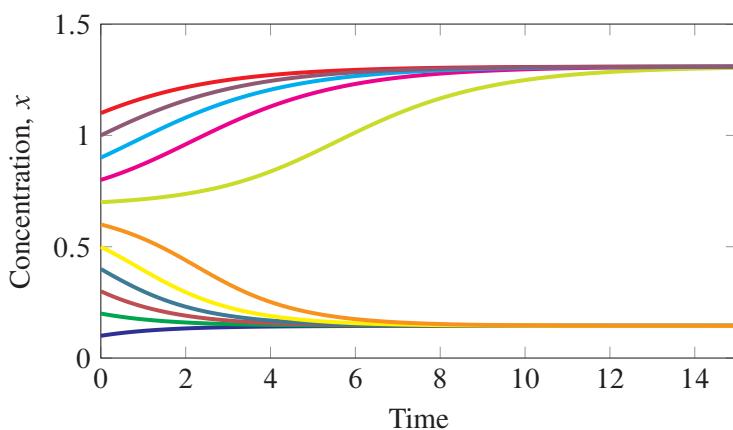
from the graph we find  $x$  values at 0.145, 0.683, and 1.309. It is also possible to use the steady state solver from Tellurium to locate the steady states. Listing 12.3 shows a simple script to compute them. By setting an appropriate initial condition, we can use Tellurium to pin point all three steady states. For example, if we use an initial value of  $x$  at 0.43, the steady state solver will locate the third steady state at 0.683. Steady state solvers such as the one included with Tellurium can be used to find unstable states, providing the initial starting point is close enough.

```
import tellurium as te

r = te.loada '''
    $Xo -> x; 0.1 + k1*x^4/(k2+x^4);
```



**Figure 12.15** A graphical understanding of the stability of the steady states. See text for details. Computed using the SBW rate law plotter.  $k_1 = 0.9$ ;  $k_2 = 0.3$ ;  $k_3 = 0.7$ ;  $b = 0.1$ .



**Figure 12.16** Time course data generated from Tellurium model 12.2. Each line represents a different initial concentration for  $x$ . Some trajectories transition to the low state while others to the upper state.

Kinetic Order	Elasticity
First-Order	1.0
Zero-Order	0.0
Sigmoidal	> 1.0

**Table 12.2**

```

x -> $w; k3*x;

// Initialization here
k1 = 0.9; k2 = 0.3;
k3 = 0.7;
'')

# Compute steady state
print r.getSteadyStateValues()

```

**Listing 12.3** Basic bistable model.

## Stability of Positive Feedback

What determines the stability of a positive feedback system? Let us consider the same genetic network with positive feedback as before (Figure 12.12). The differential equation for this system is:

$$\frac{dx}{dt} = v_1(x) - v_2(x)$$

where we have explicitly shown that each reaction rate is a function of  $x$ . To determine whether the system is stable to small perturbations we can differentiate the equation with respect to  $x$  to form the Jacobian. Notice there is only one element in the Jacobian because we only have one state variable:

$$\frac{dx/dt}{dx} = \frac{\partial v_1}{\partial x} - \frac{\partial v_2}{\partial x}$$

The terms on the right are unscaled elasticities (D.18). If the expression is positive, the system is unstable because it means that  $dx/dt$  is increasing if we increase  $x$ . We can scale both sides to yield:

$$J_s = \varepsilon_x^1 - \varepsilon_x^2$$

where the right-hand term now includes the scaled elasticities (D.17). The criteria for stability is again that  $\varepsilon_x^1 > \varepsilon_x^2$ . Therefore, if the positive feedback is stronger than the effect of  $x$  on the degradation step  $v_2$ , the system will be unstable.

Recall that the elasticities are a measure of the kinetic order of the reaction. Thus an elasticity of one means the reaction is first-order. A saturable irreversible Michaelis-Menten

reaction will have a variable kinetic order between one and zero (near saturation). A Hill equation can, depending on the Hill coefficient, have kinetic orders greater than one (Table 12.2). Knowing this information, there are at least two ways to make sure that the elasticity for the feedback elasticity,  $v_1$ , is greater than the elasticity for the degradation step,  $v_2$ :

1.  $v_1$  is modeled using a Hill equation with a Hill coefficient  $> 1$  and  $v_2$  is first-order or less.
2. A Hill coefficient = 1 on  $v_1$ , with Michaelis-Menten saturable kinetics on  $v_2$  to ensure less than first-order kinetics on  $v_2$ .

By substituting the three possible steady state values for  $x$  into the equation for  $dx/dt$ , we can compute the value for the Jacobian element in each case (Table 12.3).

Steady State $x$	Jacobian: $(dx/dt)/dx$	Elasticity, $\varepsilon_x^{v_1}$
0.145	-0.664	0.052
0.683	0.585	1.835
1.309	-0.47	0.33

**Table 12.3** Table of steady state values of  $x$  and corresponding values for the Jacobian element. Negative Jacobian values indicate a stable steady state, positive elements indicate an unstable steady state. The table shows one stable and two unstable steady states.

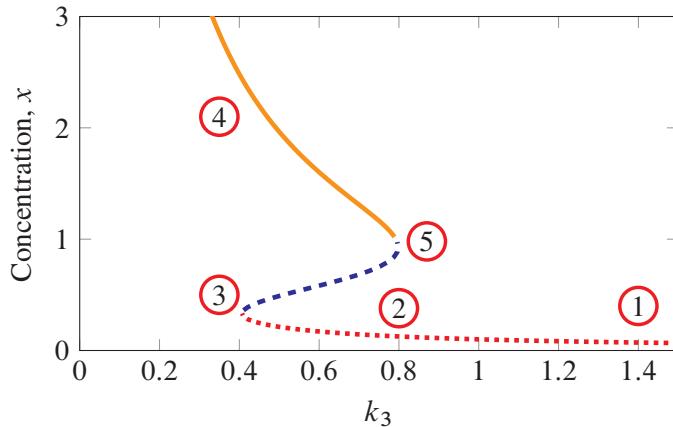
The unstable steady state at  $x = 0.683$  has an elasticity for  $v_1$  of 1.835. Note this value is greater than the elasticity of the first-order degradation reaction,  $v_2$ , which equals one. Therefore this state is unstable.

## Bifurcation Plot

Let's now return to the question of plotting a bifurcation graph for the bistable system in Figure 12.12. Figure 12.14 shows both reaction rates,  $v_1$  and  $v_2$  plotted as a function of the intermediate species  $x$ . In this figure we see three intersection points, marking the three possible steady states. By varying the degradation constant  $k_3$ , we can change the behavior of the system so that it exhibits a single high steady state, three separate steady states, or a single low steady state (See Figure 12.13).

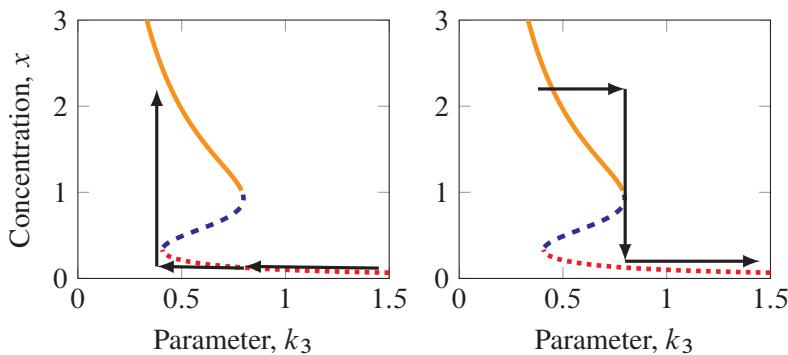
If we track the intersection points as we vary the value of the rate constant  $k_3$ , we obtain the bifurcation plot shown in Figure 12.17.

Figure 12.17 shows that at some value of the parameter  $k_3$ , the system has three possible steady states, outside this range only a single steady state persists. Bifurcation diagrams



**Figure 12.17** Plotting intersection points from Figure 12.13 as a function of  $k_3$ . Dotted line marks the lower intersection point, dashed line the middle intersection points, and solid line the upper intersection point. Computed using the SBW AUTO C# Tool.

are extremely useful for uncovering and displaying such information. Drawing bifurcation diagrams is not easy, however, and there are some software tools that can help. Figure 12.17 for example was generated using the SBW Auto C# tool<sup>2</sup>. Another useful tool for drawing bifurcation diagrams is Oscill8<sup>3</sup>. Both tools can read SBML. Figure 12.17 was generated first by entering the model into Tellurium (Shown in Listing 12.3) to generate the SBML. The model was then passed to Auto C# to produce the bifurcation diagram.



**Figure 12.18** Depending on whether we increase or decrease  $k_3$ , the steady state path we traverse will be different. This is a characteristic of hysteresis.

<sup>2</sup> [http://jdesigner.sourceforge.net/Site/Auto\\_C.html](http://jdesigner.sourceforge.net/Site/Auto_C.html)

<sup>3</sup> <http://oscill8.sourceforge.net/>

The bifurcation plot shows how the steady state changes as a function of a parameter, in this case  $k_3$ . Of interest is the following observation. If we start  $k_3$  at a high value of 1.4 (Marker 1), we see that there is only one low steady state. As  $k_3$  is lowered, we pass the point at approximately  $k_3 = 0.8$  (Marker 2) where three steady states emerge. We continue lowering  $k_3$ , and see that the concentration of  $x$  rises very slowly until about 0.4 (marker 3). At this point the system jumps to a single steady state, but now at a high level (Marker 4). The interesting observation is that if we now increase the value of  $k_3$ , we do not traverse the same path. As we increase  $k_3$  beyond 0.4, we do not drop back to the low state, but continue along the high state until we reach  $k_3 = 0.8$  (Marker 5), at which point we jump down to the low state (Marker 2). The direction in which we traverse the parameter  $k_3$  affects the type of behavior we observe. This special phenomena is called **hysteresis**, Figure 12.18.

Hysteresis is where the behavior of a system depends on its past history.

## Irreversible Bistability

It is possible to design an irreversible bistable switch. Figure 12.20 shows the bifurcation plot for such a system. This is modified from the ‘Mutual activation’ model in the review by Tyson [177], Figure 1e. In this example increasing the signal results in the system switching to the high state at around 2.0. If we reduce the signal from a high level, we traverse the top arc. If we assume the signal can never be negative, we will remain at the high steady state even if the signal is reduced to zero. The bifurcation plot in the negative quadrant of the graph is physically inaccessible. This means it is not possible to transition to the low steady state by decreasing signal. As a result, the bistable system is **irreversible**, that is, once it is switched on, it will always remain on.

```
import tellurium as te

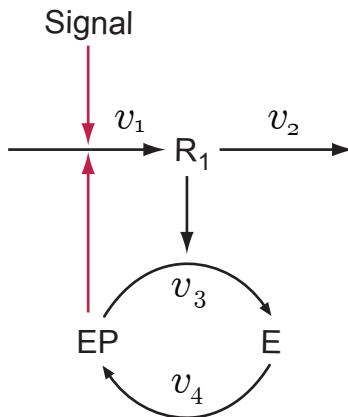
r = te.loada ('''
$X -> R1; k1*EP + k2*Signal;
R1 -> $w; k3*R1;
EP -> E; Vm1*EP/(Km + EP);
E -> EP; ((Vm2+R1)*E)/(Km + E);

Vm1 = 12; Vm2 = 6;
Km = 0.6;
k1 = 1.6; k2 = 4;
E = 5; EP = 15;
k3 = 3; Signal = 0.1;
''')

result = r.simulate(0, 40, 500)
```

```
r.plot()
```

**Listing 12.4** Script for Figure 12.20.



**Figure 12.19** System with Positive Feedback using a covalent modification cycle,  $E$ ,  $EP$ .

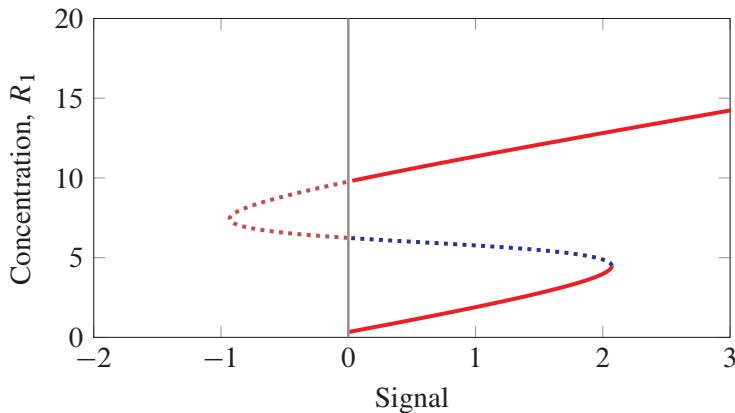
The Tellurium script for the model is shown in Listing 12.4. To create Figure 12.20, first install Oscill8<sup>4</sup>, then launch Tellurium. Load the script into Tellurium. Run the script (press green button in toolbar) to put the model into memory. Go to the SBW menu and select Oscill8. Once in Oscill8, select Run; 1 Parameter. In the new dialog box select continuation and the parameter “Signal”. Then select run to view the bifurcation plot.

## Toggle Switch

The final example we will consider is the toggle switch. This example illustrates the idea of an attractor basin in a phase plot. The system in question is shown in Figure 12.21 and is comprised of just two nodes,  $S_1$  and  $S_2$ . Each node inhibits the other. This is a high level diagram, but mechanistic realizations can be made using protein or gene regulatory networks. In fact, one of the first synthetic biology constructs was the toggle switch made from an engineered gene regulatory network [47]. Intuitively, one can imagine the type of behavior this system might exhibit. For example, if  $S_1$  has a high concentration then this will repress  $S_2$ . Since  $S_2$  is now low, repression of  $S_1$  is negligible. This state appears stable. Alternatively we could imagine that  $S_2$  is at a high concentration. This will repress  $S_1$  and thus we have another state that appears stable.

We can better study the behavior of the toggle switch by building a computer simulation. Using the set of differential equations shown in equations (12.10), we describe the model

<sup>4</sup><http://oscill8.sourceforge.net/>



**Figure 12.20** Bifurcation diagram for species  $R_1$  with respect to the signal. Signal from the model shown in Tellurium script 12.4. The continuous line represents stable steady state points, the dotted line the unstable steady states. Plotted using Oscil8 <http://oscil8.sourceforge.net/>.

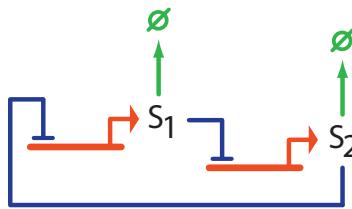


**Figure 12.21** The toggle switch. Two mutually inhibited nodes,  $S_1$  and  $S_2$ .

as:

$$\begin{aligned}\frac{dS_1}{dt} &= \frac{k_1}{1 + S_2^{n_1}} - k_2 S_1 \\ \frac{dS_2}{dt} &= \frac{k_3}{1 + S_1^{n_2}} - k_4 S_2\end{aligned}\tag{12.10}$$

A suitable set of parameter values are given by  $k_1 = 6; k_3 = 6; k_2 = 2; k_4 = 2$ . Figure 12.23 shows the phase plot for this system. The three points marked by round circles represent the steady state locations. Note that there is a high  $S_1$ /low  $S_2$  state, and a low  $S_1$ /high  $S_2$  state. These correspond to the states we intuitively described before. There is also a third point which represents a medium level of  $S_1$  and  $S_2$ . The arrows in the diagram represent the direction of change of  $S_1$  and  $S_2$  in time. The diagram has been subdivided into four **basins of attraction**. For example, the phase plot highlights one of the direction arrows in the basin labelled two. The arrows initially point right to left and down. This tells us that if we started a time-course simulation at the first arrow, the concentration of  $S_1$  and  $S_2$  would both decline. Initially, the arrow points roughly in the direction of the unstable



**Figure 12.22** The toggle switch constructed from a gene regulatory network.

state but as it moves, its direction changes and eventually moves towards the upper stable state to the left.

The phase plot gives us a condensed view of how any initial condition will evolve. Points that start in basin one will converge to the third state, basin two points converge on the first state, basin three to the third state, and finally basin four to the first state.

## Nullclines

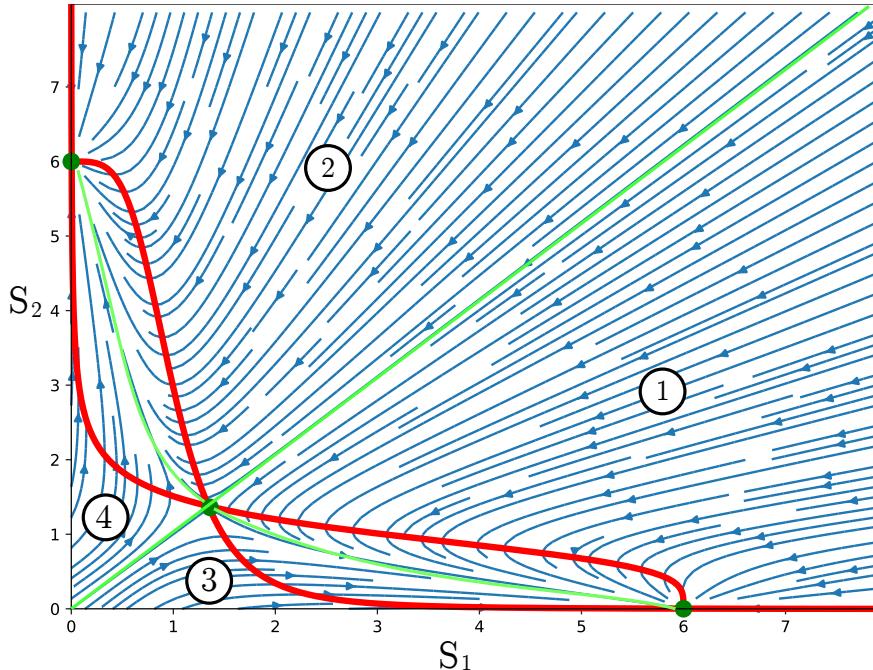
The toggle switch model from the previous section also allows us to introduce the concept of nullclines. We've seen various mechanisms for visualizing a system, especially related to its steady state stability. For example, in Figure 12.1 we show how the stability of a one dimensional system can be visualized, likewise for Figure 12.13, the bifurcation plot in Figure 12.17, and the phase plot in Figure 12.23. There is yet another plot that can help us understand the instability of a two dimensional system and that has to do with plotting the nullclines. The nullcline is the solution to a differential equation when set to zero as a function of the two system variables. For example, the toggle switch differential equation for  $S_1$  is:

$$\frac{dS_1}{dt} = \frac{k_1}{1 + S_2^{n_1}} - k_2 S_1$$

We can set this to zero and find all combinations of  $S_1$  and  $S_2$  that satisfy this equation. These points form a line on a two dimensional plane. Such a line is called the **nullcline**. Figure 12.23 shows two nullclines corresponding to the two differential equations for the toggle model. Note that where the nullclines intersect, we find the steady state because at these points, both equations yield the same values for  $S_1$  and  $S_2$ .

Although useful, many of the plots we have reviewed are limited to systems of two or three variables. The nullclines in an  $n$ -dimensional system cannot easily be visualized. Likewise, a two dimensional phase plot can only take a slice through the dynamics of a system with many variables. Nevertheless, these techniques, in particular bifurcation plots, are extremely useful in delineating the potential behavioral modes of networks. See [26, 27] for examples of bifurcation plots in studying protein signaling pathways.

There are more elaborate designs for toggle switches, but a discussion of these are beyond



**Figure 12.23** Phase portrait for a toggle switch, equation (12.10). The four numbered areas mark the four basins of attraction. The two thick lines traversing and crossing over each other near  $(1.3, 1.3)$  are the two nullclines and their intersection points make the steady state. Note that the center intersection point shows trajectories moving away from the point. Diagram generated using the Python script:  $12.6, k_1 = 6; k_3 = 6; k_2 = 2; k_4 = 2$ .

the scope of this book.

## Further Reading

1. Edelstein-Keshet L (2005) Mathematical Models in Biology. SIAM Classical In Applied Mathematics. ISBN-10: 0-89871-554-7
2. Fall CP, Marland ES, Wagner JM, Tyson JJ (2000) Computational Cell Biology. Springer: Interdisciplinary Applied Mathematics. ISBN 0-387-95369-8
3. Steuer R and Junker BH (2009). Computational models of metabolism: stability and regulation in metabolic networks. *Advances in chemical physics*, 142, 105.

## Exercises

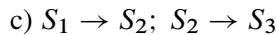
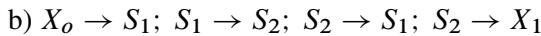
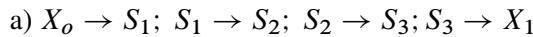
---

1. Determine the Jacobian matrix for the following systems:

$$\text{a) } \frac{dx}{dt} = x^2 - y^2 \quad \frac{dy}{dt} = x(1-y)$$

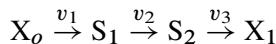
$$\text{b) } \frac{dx}{dt} = y - xy \quad \frac{dy}{dt} = xy$$

2. Compute the steady state solutions to the two systems in the previous question.
3. Determine the stability of the solutions from the previous question.
4. Determine the Jacobian in terms of the elasticities and stoichiometry matrix for the following systems:



Assume all reactions are product insensitive,  $X_i$  species are fixed, and in c)  $S_3$  regulates the first step.

5. Show that the following system is stable to perturbations in  $S_1$  and  $S_2$  by computing the eigenvalues at steady state (See Listing 12.1):



The three rate laws are given by:

$$v_1 = \frac{V_{m_1} X_o}{Km_1 + X_o + S_1/K_1}$$

$$v_2 = \frac{V_{m_2} S_1}{Km_2 + S_1 + S_2/K_2}$$

$$v_3 = \frac{V_{m_3} S_2}{Km_3 + S_2}$$

Assign the following values to the parameters:  $X_o = 1; X_1 = 0; V_{m_1} = 1.5; V_{m_2} = 2.3; V_{m_3} = 1.9; Km_1 = 0.5; Km_2 = 0.6; Km_3 = 0.45; K_1 = 0.1; K_2 = 0.2$ .

6. Show that the following system is unstable. What kind of unstable dynamics does it have?

```
import tellurium as te

r = te.loada ('''
J0: $X0 -> S1; VM1*(X0-S1/Keq1)/(1+X0+S1+pow(S4,h));
J1: S1 -> S2; (10*S1-2*S2)/(1+S1+S2);
J2: S2 -> S3; (10*S2-2*S3)/(1+S2+S3);
J3: S3 -> S4; (10*S3-2*S4)/(1+S3+S4);
J4: S4 -> $X1; Vm4*S4/(KS4+S4);

X0 = 10;           X1 = 0;
S1 = 0.973182;   S2 = 1.15274;
S3 = 1.22721;    S4 = 1.5635;
VM1 = 10;          Keq1 = 10;
h = 10;            Vm4 = 2.5;
KS4 = 0.5;
'''')
```

7. Show that the following system is unstable. What kind of unstable dynamics does it have?

```
import tellurium as te

r = te.loada ('''
J0: $src -> X;      k1*S;
J1: X -> R;         (kop + ko*EP)*X;
J2: R -> $waste;   k2*R;
J3: E -> EP;        Vmax_1*R*E/(Km_1 + E);
J4: EP -> E;        Vmax_2*EP/(Km_2 + EP);

src = 0;             kop = 0.01;
ko = 0.4;            k1 = 1;
k2 = 1;              R = 1;
EP = 1;              S = 0.2;
Km_1 = 0.05;         Km_2 = 0.05;
Vmax_2 = 0.3;        Vmax_1 = 1;
KS4 = 0.5;
'''')

result = r.simulate(0, 500, 1000)
r.plot()
```

8. Using Figure 12.23, show graphically that the toggle switch model has two stable and one unstable steady states.

## 12.6 Appendix

---

Proof that the presence of imaginary numbers in the solution to a set of differential equations means that the solution is periodic (Equation (12.7)). Consider the system:

$$x(t) = c_1 z_1 e^{(\lambda+i\mu)t} + c_2 z_2 e^{(\lambda-i\mu)t}$$

where  $z_1$  and  $z_2$  are corresponding conjugate eigenvectors. Using Euler's formula,  $e^{i\mu} = \cos(\mu) + i \sin(\mu)$  and that  $e^{(\lambda+i\mu)t} = e^{\lambda t} e^{i\mu t}$  we obtain:

$$\begin{aligned} x(t) &= c_1 z_1 e^{\lambda t} (\cos(\mu t) + i \sin(\mu t)) \\ &\quad + c_2 z_2 e^{\lambda t} (\cos(\mu t) - i \sin(\mu t)) \end{aligned}$$

Writing the conjugate eigenvectors as  $z_1 = a + bi$  and  $z_2 = a - bi$ , we get:

$$\begin{aligned} x(t) &= c_1(a + bi)e^{\lambda t}(\cos(\mu t) + i \sin(\mu t)) \\ &\quad + (a - bi)e^{\lambda t}(\cos(\mu t) - i \sin(\mu t)) \end{aligned}$$

Multiply out and separate the real and imaginary parts:

$$\begin{aligned} x(t) &= e^{\lambda t} [c_1(a \cos(\mu t) - b \sin(\mu t) + i(a \sin(\mu t) + b \cos(\mu t))) \\ &\quad + c_2(a \cos(\mu t) - b \sin(\mu t) - i(a \sin(\mu t) + b \cos(\mu t)))] \end{aligned}$$

The complex terms cancel leaving only the real parts. If we set  $c_1 + c_2 = k_1$  and  $(c_1 - c_2)i = k_2$  then:

$$\begin{aligned} x(t) &= e^{\lambda t} [k_1(a \cos(\mu t) - b \sin(\mu t)) \\ &\quad + k_2(a \sin(\mu t) + b \cos(\mu t))] \end{aligned}$$

The solution is real when the constants  $c_1$  and  $c_2$  are real. This will only be the case when the eigenvalues are a conjugate pair,  $(a \pm ib)$ , which is the case we are considering. Therefore, systems that admit a complex pair of conjugate eigenvalues result in periodic real solutions.

```
# Plot a phase portrait for a simple species pathway
import tellurium as te
import matplotlib.pyplot as plt

rr = te.loada ('''$Xo -> S1; k1*Xo;
S1 -> S2; k2*S1;
S2 -> $X1; k3*S2;
k1 = 0.6; Xo = 1;'''')
```

```
k2 = 0.4; k3 = 0.8;
''')

plt.figure(figsize=(9,4))
S1Start = 0
S2Start = 0
for i in range(1, 11):
    rr.S1 = S1Start
    rr.S2 = S2Start
    m = rr.simulate(0, 10, 120, ["S1", "S2"])
    p = te.plotArray(m, show=False)
    plt.setp (p, color='r')
    S1Start = S1Start + 0.2
S1Start = 2
S2Start = 0
for i in range(1, 11):
    rr.S1 = S1Start
    rr.S2 = S2Start
    m = rr.simulate(0, 10, 120, ["S1", "S2"])
    p = te.plotArray(m, show=False)
    plt.setp (p, color='r')
    S2Start = S2Start + 0.2
S2Start = 0
S1Start = 0
for i in range(1, 11):
    rr.S1 = S1Start
    rr.S2 = S2Start
    m = rr.simulate(0, 10, 120, ["S1", "S2"])
    p = te.plotArray(m, show=False)
    plt.setp (p, color='r')
    S2Start = S2Start + 0.2
S1Start = 0
S2Start = 2
for i in range(1, 11):
    rr.S1 = S1Start
    rr.S2 = S2Start
    m = rr.simulate(0, 10, 120, ["S1", "S2"])
    p = te.plotArray(m, show=False)
    plt.setp (p, color='r')
    S1Start = S1Start + 0.2
plt.xlim ([0, 2])
plt.ylim ([0, 2])
plt.xlabel ('S1', fontsize=16)
plt.ylabel ('S2', fontsize=16)
plt.savefig ("plot.pdf")
plt.show()
```

---

**Listing 12.5** Script for Figure 12.3.

```
# Toggle switch, nullcline and phase portrait
import numpy as np, matplotlib.pyplot as plt
import tellurium as te

Y, X = np.mgrid[0:8:200j, 0:8:200j]
U, V = np.mgrid[0:8:200j, 0:8:200j]

r = te.loada('''
J1: -> x; k1/(1+y^n1) - k2*x;
J2: -> y; k3/(1+x^n2) - k4*y;

x = 4; y = -4;
k1 = 12; k3 = 12; k2 = 2; k4 = 2
n1 = 4; n2 = 4
''')

for idx in range (200):
    for idy in range (200):
        r.x = X[idx,idy]; r.y = Y[idx,idy]
        U[idx,idy] = r["x'"]
        V[idx,idy] = r["y'"]

plt.subplots(1,2, figsize=(8,6))
plt.subplot(111)
plt.xlabel('x', fontsize='16')
plt.ylabel('y', fontsize='16')
plt.streamplot(X, Y, U, V, density=[2, 2])

# Plot the nullclines
nullcline_x = np.linspace(0, 8, 200)
nullcline_y = (12 / (1 + nullcline_x**4))/2
plt.plot(nullcline_x, nullcline_y, lw=4, color='red')

nullcline_y = (12 / (1 + nullcline_x**4))/2
plt.plot(nullcline_y, nullcline_x, lw=4, color='red')

plt.plot(0, 6, '.', color='green', markersize=20)
plt.plot(6, 0, '.', color='green', markersize=20)
plt.plot(1.36, 1.36, '.', color='green', markersize=20)

plt.ylim((0,7))
plt.xlim((0,7))
```

```
plt.savefig ('c:\\tmp\\phase.pdf')
plt.show()
```

**Listing 12.6** Script for Figure 12.9.

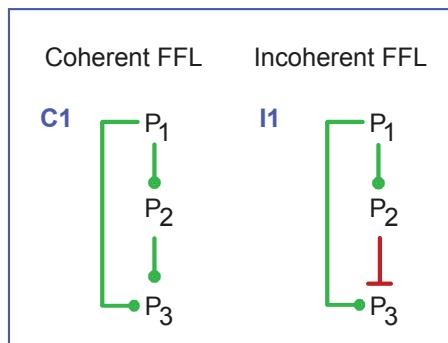


# 13

## *Modeling FeedForward Networks*

In this penultimate chapter we return to the feedforward loops (FFL) that we introduced in Chapter 1. FFLs offer an interesting example where simulation can yield useful insight into the properties of a particular network configuration.

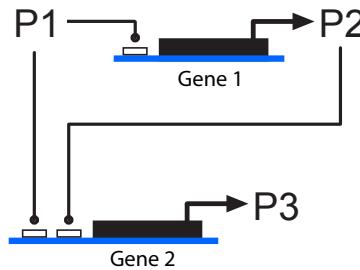
Recall that an enrichment study of gene regulatory motifs in *E. coli* and yeast revealed two types of FFLs that appear more common than others (Figure 1.21), the coherent type I and incoherent type I motifs. Schematically these are shown in Figure 13.1. In order to study the potential dynamics of this motif, we must first convert the schematic diagrams into kinetic models.



**Figure 13.1** Schematics of Type I coherent and incoherent feedforward loops.

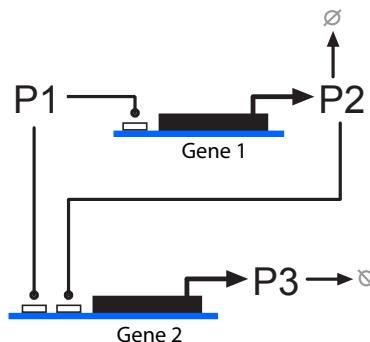
## 13.1 Coherent Type I Motif

Let us first address the coherent feedforward loop (Figure 13.1). In order to convert this into a kinetic model of a gene regulatory network, we have to consider the nature of the interactions between  $P_1$  and  $P_2$ , and the convergence of  $P_1$  and  $P_2$  onto  $P_3$ . Biologically we will assume that  $P_1$  and  $P_2$  are transcription factors such that  $P_1$  activates a gene (hidden in the diagram) that expresses  $P_2$ . Likewise,  $P_1$  and  $P_2$  in turn stimulate the expression of  $P_3$ , again via a hidden gene. This gives us the diagram shown in Figure 13.2.



**Figure 13.2** Conversion of the coherent schematic into a biological model - Step 1.

However, there is a problem with the model shown in Figure 13.2. If we were to simulate this model (assuming we've assigned suitable rate laws and values to the kinetic parameters), we will find that the concentration of  $P_2$  and  $P_3$  increases indefinitely, and the network fails to come to a steady state. Degradation steps for the transcription factors are missing. Adding these steps to the model yields the network shown in Figure 13.3.



**Figure 13.3** Conversion of the coherent schematic into a biological model - Step 2.

We can continue to add detail to this model. In particular the expression of  $P_2$  and  $P_3$  appears very simplistic; there doesn't seem to be separate transcription and translation steps. That is, we are not modeling the production of mRNA and protein expression. Sometimes the presence of mRNA can be an important part of the model. Adding mRNA as an intermediate between the gene and the expressed protein will add delays to the system which

can, in some cases, markedly affect behavior. The decision whether to add an extra mRNA step is entirely dependent on the purpose of the model. Recall that models are not replicas of reality, but rather useful tools to help explain a given set of observations and at the same time, make useful predictions. If a model without the mRNA step can still explain the observations and make useful predictions, then there is no need to include it. It is up to the systems or synthetic biologist to make a judgement call on what detail a model should include or not. Sometimes one might be told that a model  $X$  is wrong because it doesn't include feature  $Y$ . The truth is, even if we added feature  $Y$ , the model is still wrong. No model is right, just perhaps useful.

In summary, the model includes two genes to express  $P_2$  and  $P_3$ , respectively. We've also added two degradation steps, one for  $P_2$  and another for  $P_3$ .  $P_1$  will be our input to the model which means there is no degradation step for  $P_1$ .

We now must decide on the nature of the interactions between the various inputs and gene expression. A common rate law used to model gene expression is the Hill equation. This choice allows us to include sigmoidicity in the response if needed. For example, a Hill equation for activation is:

$$v = \frac{V_m P^n}{(K_H + P)^n}$$

where  $V_m$  is the maximal rate of gene expression,  $K_H$  is the concentration of  $P$  at half-maximal activity,  $n$  is the Hill coefficient that determines the degree of sigmoidicity, and  $P$  is the concentration of transcription factor. We can use the Hill equation to describe the expression  $P_2$ . The more interesting problem is when we need to assign a rate law for the expression of  $P_3$  which depends on  $P_1$  and  $P_2$ . The expression of  $P_3$  could depend on  $P_1$  and  $P_2$  in at least two ways:

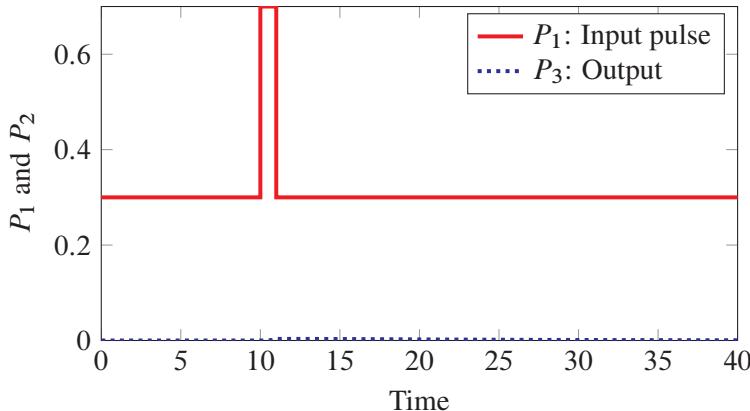
1.  $P_1$  and  $P_2$  must both be bound in order for  $P_3$  expression.
2.  $P_1$  or  $P_2$  bound will result  $P_3$  in expression.

In the first case, the action of  $P_1$  and  $P_2$  is like an AND gate. If either is not present, no expression occurs. Only if both transcription factors bind do we get expression. The second case is like an OR gate; if either of the transcription factors are present, we see expression. We can model these two situations using a modified Hill equation. The reader is referred to the companion book 'Enzyme Kinetics for Systems Biology' for more details, but the two relevant rate laws are:

$$\text{AND Gate: } v = V_m \frac{K_1 K_2 P_1^{n_1} P_2^{n_2}}{1 + K_1 P_1^{n_1} + K_2 P_2^{n_2} + K_1 K_2 P_1^{n_1} P_2^{n_2}} \quad (13.1)$$

$$\text{OR Gate: } v = V_m \frac{K_1 P_1^{n_1} + K_2 P_2^{n_2}}{1 + K_1 P_1^{n_1} + K_2 P_2^{n_2}}$$

The  $K$  terms are thermodynamic constants related to the binding and unbinding of the



**Figure 13.4** Effect of a short pulse ( $P_1$ ) on the concentration of  $P_3$  for a Coherent Type I motif using an AND gate like rate law.

transcription factors to operator sites. The  $n$  terms are Hill coefficient type constants and  $V_m$ , the maximal activity.

The final consideration are the degradation rates for the three transcription factors. Here we assume that the reactions are governed by simple first-order mass-action rate laws. With the network and kinetic laws in place, we can now consider carrying out simulations.

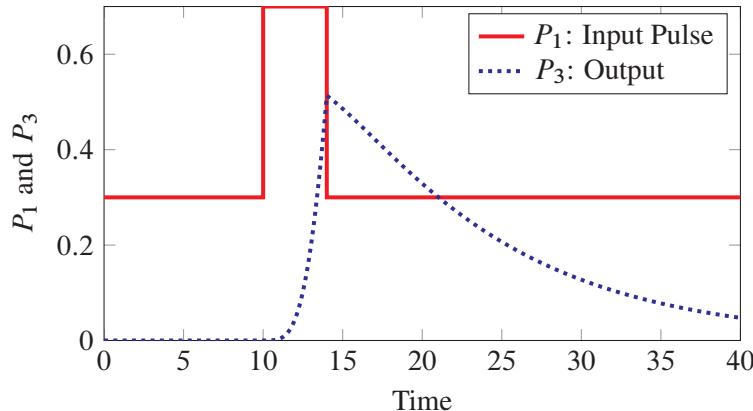
### Simulating the Coherent Type I Motif: AND Gate

Let us first consider using the AND gate based rate law in the coherent type I motif (13.1). Listing 13.1 in the chapter appendix shows the model we will investigate. It is worth mentioning how gene expression is modeled using Tellurium. Consider the step  $P_1$  to  $P_2$ . To model this, we will use the following lines:

```
$G -> P2; Vm*P1^4/(Km1 + P1^4);
P2 -> $w; k1*P2;
```

The first line describes the expression of  $P_2$  as a function of  $P_1$ . The reaction  $\$G \rightarrow P2$  represents the production of  $P_2$  from a *fixed* species called  $G$ , short for gene. The expression rate is given by a Hill equation which is a function of  $P_1$ . The second line represents the degradation step for  $P_2$ . The reaction  $P2 \rightarrow \$w$  represents the disappearance of  $P_2$  into a *fixed* species call  $w$ , short for waste. A similar syntax is used for the expression of  $P_3$  except an AND gate like rate law is used to combine the activities of  $P_1$  and  $P_2$ . We have set the Hill coefficients to a value of 4, set the maximal rates of unity and the half-maximal constants to 0.5.

Figures 13.4 and 13.5 illustrate one of the key properties of the coherent type I feedforward



**Figure 13.5** Effect of a wide pulse ( $P_1$ ) on the concentration of  $P_3$  for a Coherent Type I network using an AND gate like rate law. Note how the response falls off immediately once the pulse turns off.

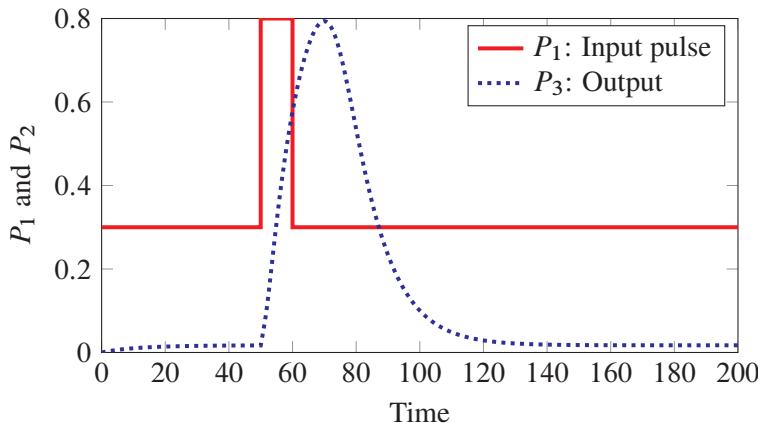
network using an AND gate, **noise rejection**. That is, the output is active if the input signal persists long enough. If the signal is only momentary, then the output of the FFL does not respond. This enables the network to be used as a noise filter where short fluctuations in the input do not trigger a response, but larger more lasting signals do. Another characteristic is that the response, when it occurs, is delayed, but once the input signal subsides, the response falls off immediately.

If a pulse signal is applied to  $P_1$ , the signal travels two routes to get to  $P_3$ . If the pulse is too short, it takes too long for the signal to travel via  $P_2$ . By the time the signal reaches  $P_3$  via  $P_2$ , the direct signal from  $P_1$  has subsided, meaning that a short time signal will not activate  $P_3$ . This effect is shown in Figure 13.4.

In sharp contrast, if the pulse width of  $P_1$  is wide enough, activation via the direct route persists long enough such that the indirect route via  $P_2$  has time to also reach  $P_3$ . This means that signals from both routes will be present at  $P_3$ , and since we assumed that activation of  $P_3$  was via an AND gate like response,  $P_3$  will now respond. The network therefore acts as a transient signal filter (Figure 13.5).

### Simulating the Coherent Type I Motif: OR Gate

In the last section we saw the effect of using an AND gate like response on the output. Here we will consider the alternative, an OR gate. Listing 13.2 in the chapter appendix shows the script used to model the OR gate network. Figure 13.6 shows the results of applying a pulse to the model. We can see that the network acts as a pulse shifter; the input pulse appears on the output, but shifted forward in time. One characteristic is that the output pulse rise is delayed from the initial pulse edge. Once the pulse ends, the output is also delayed.



**Figure 13.6** Effect of pulse ( $P_1$ ) on the concentration of  $P_3$  for an Coherent Type I network using an OR gate like rate law. Note how the response rises immediately and is delayed once the pulse turns off.

## 13.2 Incoherent Type I Motif

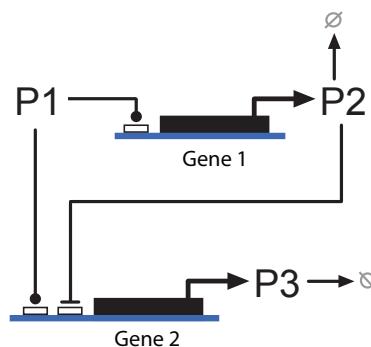
The second common type of motif found in *E. coli* and yeast is the incoherent type I motif. As with the coherent type I motif in the last section, we have to consider how we are going to implement the motif as a simulation model. Figure 13.7 shows a genetic circuit that is one possible realization of the incoherent type I motif. As before, we have added degradation steps to  $P_2$  and  $P_3$  which we assume will follow simple first-order reaction kinetics. We assign a Hill like activation rate law between  $P_1$  and  $P_2$ .

The more interesting question concerns the dual control of  $P_3$  expression. For the incoherent network,  $P_3$  is both activated and inhibited at the same time. One possible kinetic model to use is a non-competitive one. That is,  $P_2$  acts as a repressor but always dominates because even if the activator  $P_1$  binds, it cannot easily dislodge the repressor. We can also consider a competitive model where the activator competes with the repressor on the same binding site. Here we will only review the non-competitive model, the equation for which is given below:

$$V = \frac{K_1 P_1}{(1 + K_2 P_1 + K_3 P_2 + K_3 P_1 P_2^8)}$$

### Properties of the Incoherent Type I Motif

The incoherent type I feedforward network (Figure 13.7) has a completely different response compared to the coherent type I network.



**Figure 13.7** Conversion of the Incoherent schematic into a biological model. Note that the regulation on  $P_3$  is antagonistic compared to the coherent type I network.

Using the non-competitive model, the incoherent type I network has a number of interesting behaviors, including pulse generator, concentration band detector, and frequency band pass filter. The pulse generator simulation is shown in Figure 13.8 where an input step function is applied to the input,  $P_1$ , and the effect on  $P_3$  observed. In this case,  $P_3$  rises rapidly then falls off in a pulse like manner even though the input signal remains on. This is due to the delay in the inhibition route. Initially the inhibition is weak and the output rises. Eventually  $P_2$  increases to the extent that it begins to repress the production of  $P_3$ .

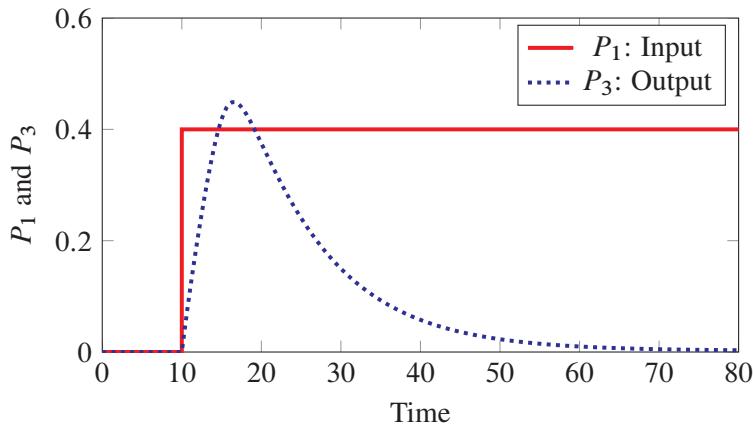
The second type of behavior that the network can display is to act as a concentration band detector. The simulation shown in Figure 13.9 shows the network turning on for a specific range of input signal. The position of the peak in the response can be adjusted by changing the strength of inhibition and the input threshold from  $P_1$  to  $P_3$ .

An incoherent type I network can also act as a band pass filter. That is, the network will respond more strongly if the input signal varies at a specific range of frequencies. At high and low frequencies the network will not respond, but at mid range frequencies it will. In addition, the incoherent type I loop can be used to build a response accelerator. The activator loop is made stronger so that the initial response rises faster, but then the repression brings the response back down to the desired steady state.

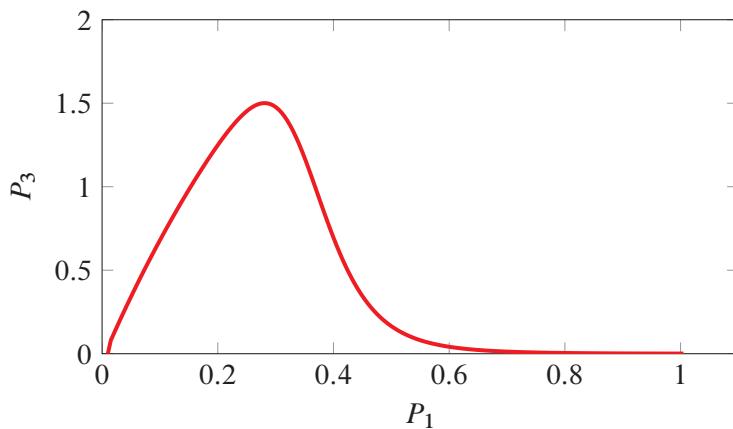
A series of synthetic incoherent type I networks have been built in *E. coli* illustrating the band pass behavior [39]. Further details on the properties of feedforward networks can also be found in the book by Alon [3].

## Further Reading

1. Alon U (2006) An Introduction to Systems Biology: Design Principles of Biological Circuits, Chapman & Hall/Crc Mathematical and Computational Biology Series.
2. Alon U (2007) Network motifs: theory and experimental approaches. Nature Re-



**Figure 13.8** Simulation of the pulse generation characteristics of an incoherent type I feedforward network. The  $x$ -axis shows the value of the input signal over time. At 10 time units, a step signal is applied, this causes the output to rise then fall in a pulse like manner. The width of the pulse can be adjusted by changing the degree of cooperativity on  $P_3$ . See Tellurium script 13.3.



**Figure 13.9** Simulation of the concentration band detector using an incoherent type I feedforward network. The  $x$ -axis shows the value of the input signal,  $P_1$ , and the  $y$ -axis the steady state concentration of the output signal,  $P_3$ . The network only responds in a particular range of input concentration. See Tellurium script 13.4.

views Genetics, 8, 450-461.

3. Yosef N and Regev A (2011), Impulse Control: Temporal Dynamics in Gene Expression. Cell 144, 886–896.

## Exercises

---

1. Study the behavior of an Incoherent Type I FFL by using a competitive model for the activator/repression step. In a competitive model both the repressor and activator bind to the same operator site.

## Appendix

---

See <http://tellurium.analogmachine.org> for more details of Tellurium.

```
import tellurium as te
import numpy

# Coherent Type I Genetic Network, noise filter
rr = te.loada ('''$G2 -> P2; Vmax2*P1^4/(Km1 + P1^4);
P2 -> $w; k1*P2;
$G3 -> P3; Vmax3*P1^4*P2^4/(Km1 + P1^4*P2^4);
P3 -> $w; k1*P3;

Vmax2 = 1; Vmax3 = 1;
Km1 = 0.5; k1 = 0.1;
P1 = 0; P2 = 0; P3 = 0;
''')

rr.getSteadyStateValues()
print rr.getFloatingSpeciesConcentrations()

# Pulse width
# Set to 1 for no effect
# Set to 4 for full effect
width = 1
rr.P1 = 0.3
m1 = rr.simulate(0, 10, 100, ["time", "P1", "P3"])
rr.P1 = 0.7 # input stimulus
m2 = rr.simulate(10, 10 + width, 100, ["time", "P1", "P3"])
rr.P1 = 0.3
m3 = rr.simulate(10 + width, 40, 100, ["time", "P1", "P3"])
m = numpy.vstack((m1, m2))
```

```
result = numpy.vstack((m, m3))
te.plotWithLegend(rr, result)
```

**Listing 13.1** Script for Figure 13.4.

```
import tellurium as te
import numpy

# Coherent Type I Genetic Network, delay circuit using OR gate
rr = te.loada ('''
$G2 -> P2; Vmax2*P1^4/(Km1 + P1^4);
P2 -> $w; k1*P2;
$G3 -> P3; Vmax3*(P1^4 + P2^4)/(Km1 + P1^4 + P2^4);
P3 -> $w; k1*P3;

Vmax2 = 1; Vmax3 = 0.1;
Km1 = 0.5; k1 = 0.1;
P1 = 0; P2 = 0; P3 = 0;
''')

rr.getSteadyStateValues()
print rr.getFloatingSpeciesConcentrations()

# Pulse width
# Set to 1 for no effect
# Set to 4 for full effect
width = 10
rr.P1 = 0.3
m1 = rr.simulate(0, 50, 200, ["Time", "P1", "P3"]).copy()
rr.P1 = 0.8 # Input stimulus
m2 = rr.simulate(50, 50 + width, 200, ["Time", "P1", "P3"]).copy()
rr.P1 = 0.3
m3 = rr.simulate(50 + width, 200, 200, ["Time", "P1", "P3"]).copy()
m = numpy.vstack((m1, m2))
result = numpy.vstack((m, m3))
te.plotWithLegend(rr, result)
```

**Listing 13.2** Script for Figure 13.6.

```
import tellurium as te
import numpy

# Incoherent Type I Genetic Network, Pulse generator
rr = te.loada ('''
$G1 -> P2; t1*a1*P1/(1 + a1*P1);
P2 -> $w; gamma_1*P2;
```

```

$G3 -> P3; t2*b1*P1/(1 + b1*P1 + b2*P2 + b3*P1*P2^8);
P3 -> $w; gamma_2*P3;

P2 = 0;
P3 = 0;
P1 = 0.01;
G3 = 0;
G1 = 0;
t1 = 5;
a1 = 0.1;
t2 = 1;
b1 = 1;
b2 = 0.1;
b3 = 10;
gamma_1 = 0.1;
gamma_2 = 0.1;
''')

# Time course response for a step pulse
rr.P1 = 0.0;
m1 = rr.simulate(0, 10, 100, ["Time", "P1", "P3"])
rr.P1 = 0.4 # Input stimulus
m2 = rr.simulate(10, 50, 200, ["Time", "P1", "P3"])
m = numpy.vstack((m1, m2))
te.plotWithLegend(rr, m)

```

**Listing 13.3** Script for Figure 13.8.

```

import tellurium as te
import numpy as np

# Steady state band detector
rr = te.loada ('''
$G1 -> P2; t1*a1*P1/(1 + a1*P1);
P2 -> $w; gamma_1*P2;
$G3 -> P3; t2*b1*P1/(1 + b1*P1 + b2*P2 + b3*P1*P2^8);
P3 -> $w; gamma_2*P3;

P2 = 0; P3 = 0;
P1 = 0.01; G3 = 0;
G1 = 0;
t1 = 5; a1 = 0.05;
t2 = 0.8; b1 = 1;
b2 = 0.1; b3 = 10;
gamma_1 = 0.1;
gamma_2 = 0.1;
'''')

```

```
# Steady state response
n = 200
m = np.empty([n, 2])
for i in range (0, n):
    m[i, 0] = rr.P1
    m[i, 1] = rr.P3
    rr.getSteadyStateValues()
    rr.P1 = rr.P1 + 0.005
te.plotArray(m)
```

**Listing 13.4** Script for Figure 13.9.

# 14

## *Behavior of Stochastic Models*

### 14.1 Introduction

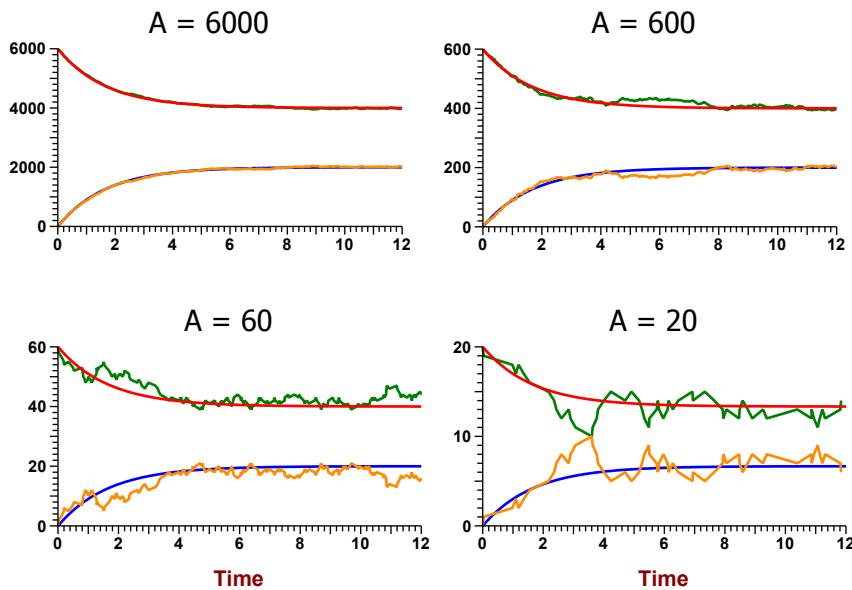
---

Chapter 6 described some basic concepts in stochastic kinetics, in particular how to simulate stochastic models. What was not discussed is the kind of behavior that can emerge from a stochastic system. At first glance it may seem that a stochastic model would just be a noisy version of the equivalent deterministic model. In some cases this is true. Take for example a simple equilibration model such as:



where the forward rate is given by  $k_1 A$ , and the reverse rate by  $k_2 B$ . Given starting concentrations for  $A$  and  $B$ , we can easily determine by simulation the time trajectories for  $A$  and  $B$ . We can compute the trajectories for both the stochastic and equivalent deterministic model. The two stochastic rate constants  $k_1$  and  $k_2$  are numerically equal to the deterministic equivalents because both reactions are first-order. Figure 14.1 shows plots generated from the model (Listing 14.2) and compares four simulations that were carried out using different initial conditions. The first plot at the top left corner starts with 6000 molecules. Under these conditions, the stochastic and deterministic simulations seem almost indistinguishable. At 600 molecules we begin to see a difference, and by 20 molecules, the stochastic trajectories are very noisy. However even at 20 molecules the stochastic data still appears to roughly follow the deterministic trajectories. In fact the mean stochastic levels at equilibrium are identical to the deterministic concentrations. In this case the stochastic simulation is the same as the deterministic model except noisy.

In general this will not always be the case. If stochastic simulations simply represented



**Figure 14.1** Comparison of deterministic and stochastic simulation for an isomerization reaction using different initial conditions. Generated using Tellurium script 14.2.

noisy versions of the equivalent deterministic simulations, then stochastic models would probably be of little interest. However it turns out there are a number of common situations where a stochastic model can yield very different behavior compared to the equivalent deterministic model. In this book we will look at three interesting situations where stochasticity makes a significant difference:

- Bursting and extinction events
- Stochastic focusing
- Chatter

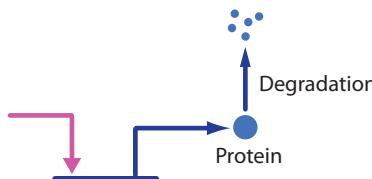
## 14.2 Stochastic Bursting

It is hard to imagine that a single reaction event could make any real difference to the future evolution of a larger system, but in some special cases this holds true. Bursting is where a system at one point is relatively quiescent, then suddenly shows marked activity until it returns to the quiescent state again. The time interval between quiescent and active states is generally irregular. A clear example of bursting occurs during transcription [58, 21].

## Transcriptional Bursting

When the number of transcription factors (or RNA polymerase) is very small, the binding and unbinding to the operator and promoter sites leads to random transcription events such that transcription acts in an on/off manner with **bursts** of mRNA production followed by periods of silence. This is called the random telegraph model [93]. We can construct a bursting model as follows.

The model has three parts, the first is the binding of unbound transcription factor to an operator site, the second computes the level of gene expression as a function of bound transcription factor. Finally, the third part involves the degradation of the expressed protein. A schematic of the network is shown in Figure 14.2. Listing 14.3 in the Appendix is the Tellurium script that generated the data in Figure 14.3. The simulation uses the Gillespie method to generate the results. The lower curve shows the on/off behavior of the transcription factor. Given that there is only one expression cassette, there is only one bound complex at any one time, hence the bound state varies between zero and one. When the transcription factor is bound, there is a burst of protein synthesis. When the bound transcription factor is released, the protein level decays as a result of protein degradation. We see therefore, a rapid rise in protein followed by a slow decay. For comparison, the same model is also simulated using a deterministic simulation as shown in Figure 14.4. The deterministic and stochastic simulations are obviously very different.

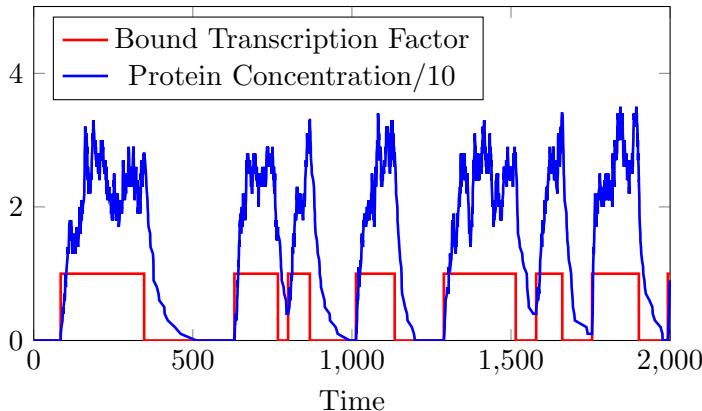


**Figure 14.2** Simple bursting model.

## Ion Channel Bursting

Ion channels are common membrane bound proteins found in many cells, particularly nerve tissue. There are a great variety of ion channels, but what they have in common is the ability to transport ions across membranes. Many channels act as gates, opening and closing in response to specific stimuli. Some channels open or close depending on the local potential difference across the membrane, while others open and close depending on whether a specific ligand is bound or not. Examples of ligands that can bind to ion channels include acetylcholine, glutamate, or ATP.

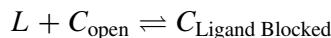
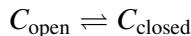
Ion channels tend to be specific about what ions they transport, for example there are sodium, potassium, calcium, and proton channels to name a few. Many of the voltage control ion channels are involved in nerve conduction and have been studied for many years.



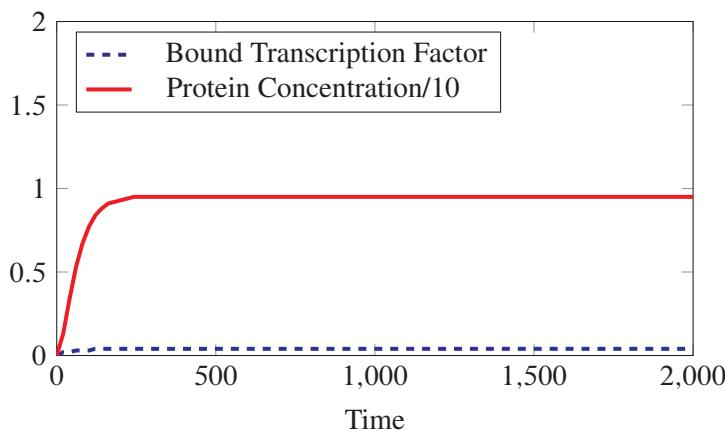
**Figure 14.3** Bursting from a simple gene expression model. Note that the level of protein has been reduced ten fold in order to make a clearer comparison. Upper curve represents protein. Generated from Listing 14.3.

In the following section we will look at a very simple model of a ligand gated channel.

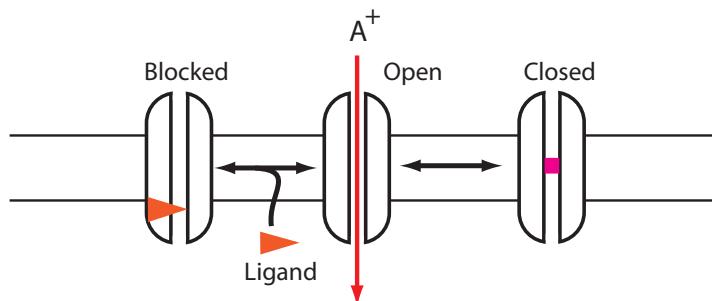
Consider a channel,  $C$ , that exists in two states, closed and open with an equilibrium distribution between the two. Furthermore, consider a ligand,  $L$ , that can bind to the open channel forming a blocked channel (Figure 14.5). We can represent the model using the following reactions:



Listing 14.5 in the Appendix shows the Tellurium code to run the model. We assume that the ligand concentration is much higher than the concentration of ion channels such that when ligand binds, the concentration of ligand hardly changes. As a result, we can fix the level of ligand. Each time an ion channel opens, a flood of ions move across the membrane resulting in a burst of electrical activity. The ligand concentration controls the duration between openings. The kinetics in the model has been arranged so that the transition between closed and open is slow compared to the transition between open and blocked (bound to ligand). At low ligand concentration the behavior of the system is dominated by the slow transitions between the open and closed states leading to fewer bursts but longer lasting ones. As the ligand concentration is increased, the equilibrium shifts away from the slow open/closed transitions to the much faster open/blocked states. This means the bursts become much shorter and more frequent in duration. This in turns means that the bursts in ion current tend to average out more so the bursting is less noticeable.



**Figure 14.4** Simple gene expression model based on a deterministic description. Note that the deterministic behavior is completely different from the stochastic bursting seen in Figure 14.3. Upper curve represents protein.



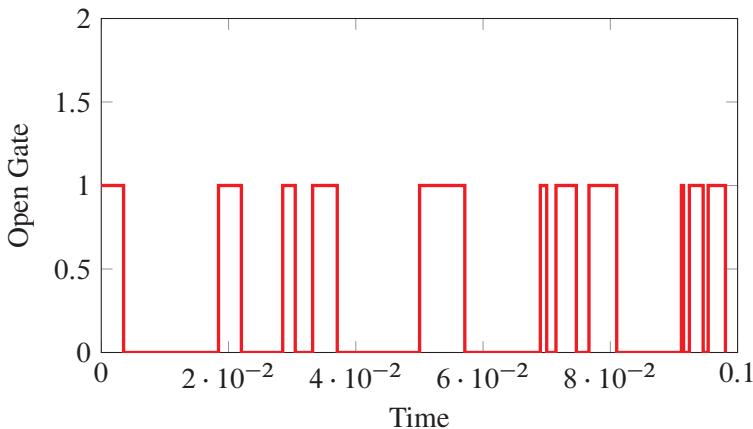
**Figure 14.5** Simple model of a ligand gated channel.

## 14.3 Stochastic Focusing

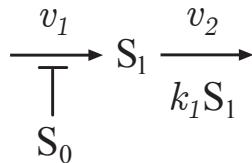
There are important and common situations where a stochastic model can yield completely different results from the equivalent deterministic model. Consider the enzymatic model shown in Figure 14.7. In this model the first step is competitively inhibited by a signal molecule called  $S_0$ . The rate law for the first step is therefore given by:

$$v_1 = \frac{V_m}{K_m + S_0}$$

Assume the substrate for the first reaction is fixed and the second step governed by a simple first-order reaction. We can run a simulation of this model either as a deterministic or as a stochastic model. If we assign values shown in the Tellurium Listing 14.4 in the



**Figure 14.6** Bursting in an ion channel model. Each signal represents an open channel and the width of the signal indicates how long the channel remains open. Once a channel is open, ions pour across the membrane resulting in a burst of current. Tellurium script: 14.5.



**Figure 14.7** Two step model where the first step is inhibited by a signal  $S_0$ .

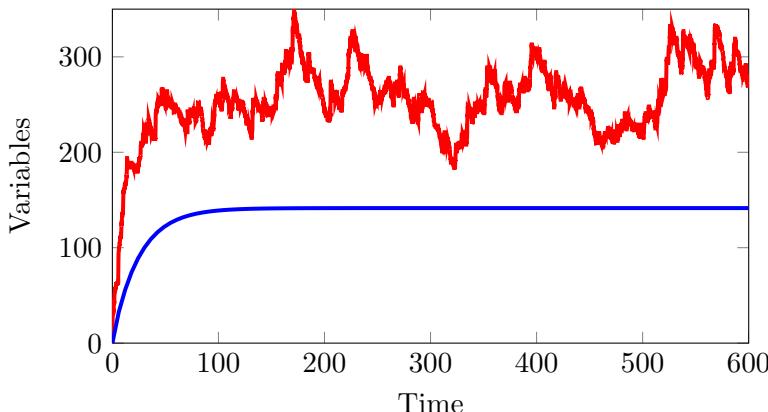
Appendix, we can compute the deterministic steady state for the intermediate,  $S_1$ , to be 141.5 concentration units. In order to simulate the same model stochastically, we need to consider possible adjustments to the values for the constants. For example, the first-order constant  $k_3$  remains unchanged. What about the competitive inhibition rate law, in particular the  $K_m$  and  $V_m$  parameters? Since we know the model is a competitive one, we can unwrap the rate law into individual elementary steps, however this may obscure the origins of the unusual behavior we observe in the stochastic model. In addition, although research is still ongoing in this important area, recent evidence suggests that the classic methods, such as Gillespie SSA can be applied to non-elementary systems without significant loss of accuracy [142, 22, 104, 147]. We therefore assume that the competitive rate law can be legitimately used in the stochastic models and that the  $K_m$  and  $V_m$  constants will have the same values compared to the deterministic model.

With the model in place, we must now consider the level of signal,  $S_0$ . Specifically, we will assume the signal has a stochastic profile with a mean and variance of concentrations. We can generate such a profile for  $S_0$  by adding two new reactions, one that makes  $S_0$ , and

another that degrades it. A simple equilibration model will do, that is:

$$X_o \rightleftharpoons S_o$$

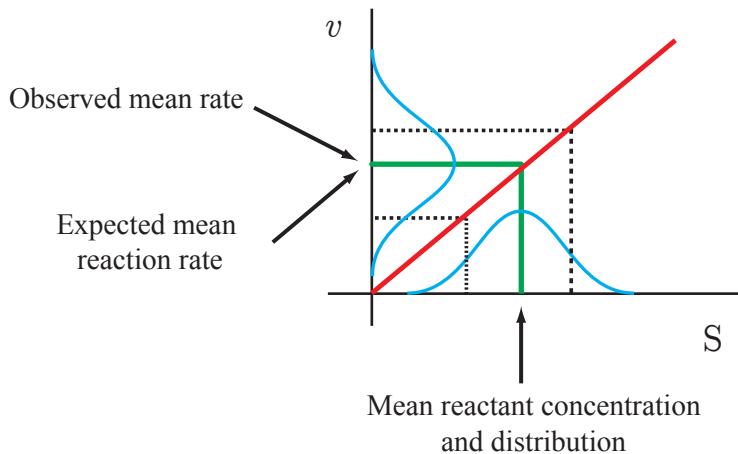
By adding these two steps to the model, we ensure  $S_o$  is noisy. Because the noise enters the system externally, it is called **extrinsic noise**. We now wish to compute the steady state concentration of  $S_1$ . We know that  $S_1$  will show a noisy profile due to contributions from the extrinsic noise,  $S_o$ , and noise generated by the model itself, called **intrinsic noise**. To compute the steady state level we must run the simulation for a long time (ignoring any initial transient) and compute the mean concentration by averaging over the  $S_1$  trajectory. It is important that we use as much data as possible for this to obtain a reasonable estimate. From the simulation we estimate the mean concentration of  $S_1$  in the stochastic model to be approximately 245.0 concentration units (Figure 14.8). This is significantly larger than the deterministic steady state value of 141.5 concentration units. This demonstrates that the deterministic and stochastic models are not identical. The stochastic model predicts a different mean steady state concentration. The question is, why?



**Figure 14.8** Stochastic Focusing. Lower line represents the deterministic simulation and the upper line the equivalent stochastic model. In this case the stochastic solution does not follow the deterministic model. Generated from Tellurium script: 14.4.

To explain why the concentration levels are different, we must understand what happens to noise as it propagates through a network. Imagine a simple first-order step, shown in Figure 14.9. Also imagine that the concentration of substrate,  $S$ , has a mean and distribution of values as indicated by a bell shaped curve. This distribution is an input to the reaction and we can imagine that a distribution of reaction rates will inevitably occur. Because the rate law is linear however, the shape of the distribution of reaction rates will actually remain unchanged. If we measure the mean reaction rate, we find that it corresponds to the expected reaction rate for a deterministic model system. Noise is unaffected by the first-order reaction.

Now consider a different case where the reaction rate is modeled by a competitive inhibitor.

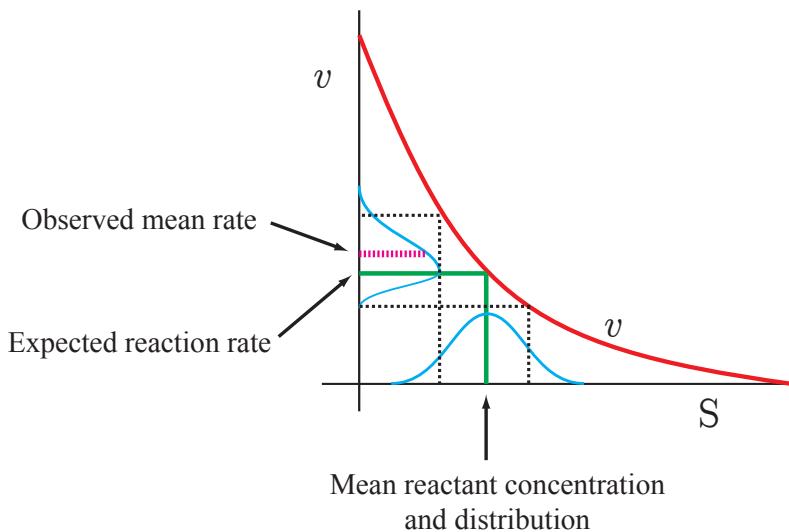


**Figure 14.9** Propagation of stochastic noise through a first-order reaction.

In this case the response is no longer linear and the rate curve has significant curvature. Let us again apply an input concentration with a given mean and distribution of values (See Figure 14.10) to observe what happens. Because the curvature is negative, values above the mean concentration of  $S$  will get compressed, while values below the mean concentration will get stretched. This results in a distortion to the input bell shape curve as it emerges as a reaction rate. The distortion means that the mean reaction rate is shifted higher compared to the expected deterministic reaction rate because the output distribution is stretched upwards. The change in mean rate leads to changes in the steady state levels as compared to the deterministic case. This effect is called **stochastic focusing** [133, 87] and can be negative (stochastic defocusing) as demonstrated in this example, or positive depending on the sign of the curvature. If severe enough, stochastic focusing can result in major changes to the qualitative behavior of the network such that the stochastic simulation bears no resemblance to the deterministic one.

## 14.4 Chatter

In Chapter 12 we briefly talked about bistable systems, systems that can exist in one of two stable steady states for a given set of parameters. If we model bistable systems using a stochastic based model, it is possible to generate behavior where the system jumps periodically from one steady state to the other. The reason for this is that if stochastic fluctuations are large enough compared to the gap between the two steady states, it is possible for the system to jump from one basin of attraction to the other. Figures 14.11 and 14.12 are two runs from the same model illustrating random jumps between the high and low states of a bistable system. In Figure 14.11 we see the system starting at the high state of around 20



**Figure 14.10** Stochastic focusing due to a non-linear rate law. Note that the curvature of the rate function causes the mean reaction velocity to increase.

molecules. This state lasts until  $t = 20$  when the system jumps to its low state. At  $t = 50$  the system jumps back to the high state. The frequency of jumps between the two states is a function of the system's parameters. The jumps are not regular and Figure 14.11 shows an example where there is one jump to the low state at  $t = 60$ , and some short lived jumps to the low state at  $t = 135$  and  $140$ . The effect where a system spontaneously switches between two states as a result of noise is called **chatter** [175, 43] or chattering.

Chattering in bistable systems has been observed in both natural and synthetic systems. Work by Ozbudak et. al [127] shows that a population of *E. coli* cells are distributed bimodally between the on (lactose utilization) and off state. A more detailed analysis by Egbert and Klavins [37] using a synthetic circuit shows very clearly that a population of cells can be distributed between the two bistable states. The degree of distribution can be controlled by changing the system's parameters. Many examples now exist in the literature including work in the field of neuroscience.

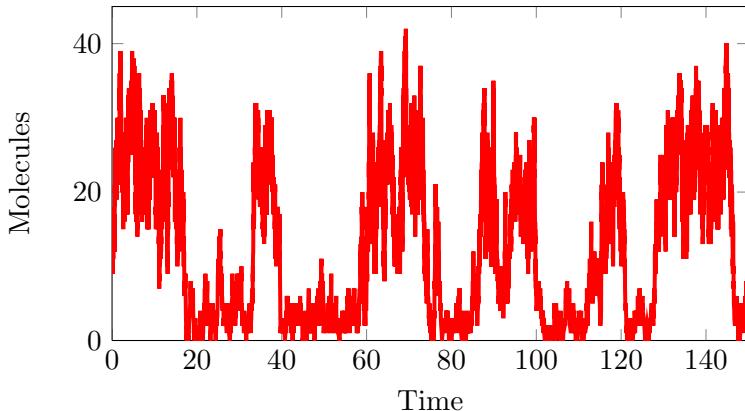
```

import tellurium as te
import pylab

# Chattering in a bistable system
r = te.loada ('''
$X0 -> x; b1 + Vm*(x/Km)*(1+(x/Km))^(n-1)/((1+(x/Km))^(n+k2));
x -> $w; k3*x;

k2 = 200; k3 = 4.2;

```



**Figure 14.11** Example of chatter in a bistable system that is stochastically modeled. The system has two steady states at roughly 5 and 22. Noise randomly flips the system from one state to the other.

```

Vm = 110; Km = 3.6;
n = 3.7; b1 = 10;

x = 15; # Initialize number of molecules
''')

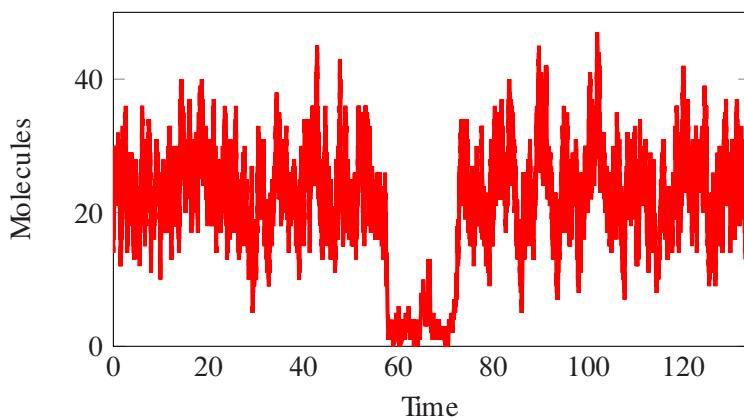
m = r.gillespie(0, 140, ["Time", "x"])
# Plot and set the x axis limits
r.plot (xlim=(0,140))

```

**Listing 14.1** Script for Figure 14.6.

## Further Reading

1. Rao, CV, Wolf, DM and Arkin, AP (2002), Control, exploitation and tolerance of intracellular noise, 420:6912, 231-237.
2. Kærn M, Elston TC, Blake WJ and Collins JJ (2005) Stochasticity in gene expression: from theories to phenotypes, Nature Reviews Genetics, 6, 451-464.
3. Raj A, van Oudenaarden A (2008) Nature, Nurture, or Chance: Stochastic Gene Expression and Its Consequences. Cell, 135:2, 216-226.
4. Eldar, A and Elowitz, MB (2010) Functional roles for noise in genetic circuits, Nature, 467:7312, 167–173.



**Figure 14.12** Example where there is a single transition to the low state at about  $t = 60$ .

5. Ingalls B (2013) Mathematical Modeling in Systems Biology: An Introduction, MIT Press. ISBN: 978-0262018883

## Exercises

1. The following modified model is taken from the work of Ribeiro and Lloyd-Price [146]. Run a simulation of the model using the given parameters. Explain why this model shows bimodal behavior.

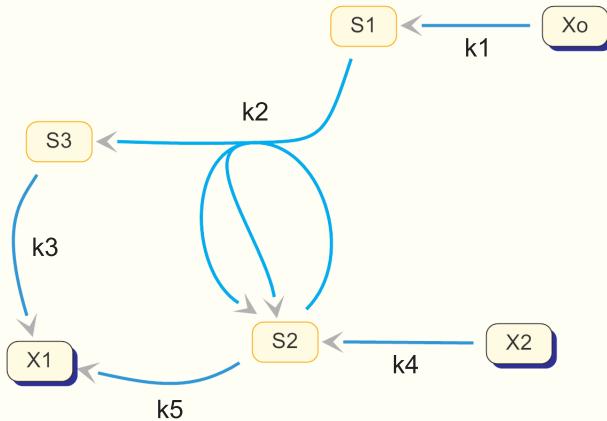
```
import tellurium as te

r = te.loada """
    ProA -> A + ProA; g*ProA;
    ProB -> B + ProB; g*ProB;
    A + ProB -> ProBA; a0*A*ProB;
    B + ProA -> ProAB; a0*B*ProA;
    ProBA -> ProB + A; a1*ProBA;
    ProAB -> ProA + B; a1*ProAB;
    A -> $w; d*A;
    B -> $w; d*B;
    ProAB -> ProA; d*ProAB;
    ProBA -> ProB; d*ProBA;

    g = 0.2;      d = 0.005;
    a0= 0.3;     a1 = 0.01;
    A = 0;        B = 0;
    ProA = 1;    ProB = 1;
"""
```

```
result = r.gillespie(0, 2000000, ["Time", "A"]);
r.plot()
```

2. The following model should be simulated as a deterministic model (i.e. using ODEs) and as a stochastic model.



**Figure 14.13** Reaction Scheme:  $X_0$ ,  $X_1$ , and  $X_2$  are boundary species. Be very careful that you replicate this model exactly as given. Assume all reactions are simple irreversible mass-action. Parameter values are as follows:  $k_1 = 0.1$ ;  $k_2 = 0.1$ ;  $k_3 = 0.01$ ;  $k_4 = 0.05$ ;  $k_5 = 10.1$ ;  $X_0 = 10$ ;  $X_2 = 1$ .

- Enter the model into Tellurium and run a deterministic simulation. Show the graphs for  $S_1$ ,  $S_2$ , and  $S_3$  over a time period of 800 time units.
  - Adjust the model so that it is ready for a stochastic simulation. Use the same values for the rate constants and initial conditions. Plot  $S_1$  and  $S_3$  on one graph and  $S_2$  on another graph.
  - Observe the significant difference between the deterministic and stochastic simulations. Why is this the case? Explain why the dynamics of the stochastic simulation are so different considering the number of molecules involved and the kind of reactions in the models.
  - Given your answer in (c), provide one situation where you think it is important to use a stochastic model rather than a deterministic one.
3. Expand the simple gene expression model in Figure 14.2 to include transcription and translation. Develop a stochastic model of the expanded system. Investigate whether the translation machinery, which is present in higher concentration, can act as a buffer to the mRNA bursting.

## Appendix

See Appendix H for more details of Tellurium.

```
import tellurium as te
import matplotlib.pyplot as plt
import roadrunner

rr = te.loada ('''
A -> B; k1*A;
B -> A; k2*B;
k1 = 0.2; k2 = 0.4;
''')

starting = 6000 # 10 zepto molar 10^(-21) = 6000 molecules

rr.model["init(A)"] = starting
rr.model["init(B)"] = 0

plt.subplot(221)
plt.title("A = 6000")
m1 = rr.gillespie(0, 12, ["time", "A", "B"])
te.plotArray(m1)
rr.model["init(A)"] = starting
rr.model["init(B)"] = 0

m2 = rr.simulate(0, 12, 100)
te.plotArray(m2)

starting = 600
rr.model["init(A)"] = starting
rr.model["init(B)"] = 0

plt.subplot(222)
plt.title("A = 600")
m1 = rr.gillespie(0, 12, ["time", "A", "B"])
te.plotArray(m1)
rr.model["init(A)"] = starting

rr.model["init(B)"] = 0
m2 = rr.simulate(0, 12, 100)
te.plotArray(m2)

starting = 60
rr.model["init(A)"] = starting
rr.model["init(B)"] = 0
```

```

plt.subplot(223)
plt.title("A = 60")
m1 = rr.gillespie(0, 12, ["time", "A", "B"])
te.plotArray(m1)
rr.model["init(A)"] = starting
rr.model["init(B)"] = 0

m2 = rr.simulate(0, 12, 100)
te.plotArray(m2)
plt.xlabel("Time")
starting = 20
rr.model["init(A)"] = starting
rr.model["init(B)"] = 0

plt.subplot(224)
plt.title("A = 20")
m1 = rr.gillespie(0, 12, ["time", "A", "B"])
te.plotArray(m1)
rr.model["init(A)"] = starting
rr.model["init(B)"] = 0

m2 = rr.simulate(0, 12, 100)
plt.xlabel("Time")
te.plotArray(m2)

```

**Listing 14.2** Script for Figure 14.1.

```

import tellurium as te

r = te.loada (''
    // Transcription binding/unbinding step
    TR + Gene1 -> TR_B; k1*TR*Gene1;
    TR_B -> TR + Gene1; k2*TR_B;
    // Protein synthesis
    $g -> product; Vm*TR_B;
    // Protein degradation
    product -> $w; k3*product;

    // TR = free transcription factor;
    // TR_B = bound transcription factor
    Gene1 = 1; TR = 1; Vm = 1;
    k1 = 0.01; k2 = 0.01; k3 = 0.04;
    TR_B = 0; product = 0;
    '')
seed = 1.22012

```

```
m = r.gillespie(0, 2000, ["time", "TR_B", "product"], seed)
r.plot()
```

**Listing 14.3** Script for Figure 14.2.

```
import tellurium as te
import matplotlib.pyplot as plt

# Stochastic Focusing Model
# Modified from: Paulsson J, Berg OG, Ehrenberg M.
# Proc. Natl. Acad. Sci. USA 97(13), 7148-53 (2000)
# Stochastic focusing: fluctuation-enhanced sensitivity
# of intracellular regulation.
r = te.loada ('''
$src -> So;      k1*src;
So -> $srr;      k2*So;
J1: $Xo -> S1; Vm/(Km + So);
J2: S1 -> $w;   S1*k3;
k1 = 20;          k2 = 6;
Vm = 20;          Km = 0.2;
k3 = 0.04;        src = 1;
S1 = 0;           So = 0;
''')

m1 = r.simulate(0, 600, 100, ["Time", "S1"])
r.plot()
r.S1 = 0
r.So = 0
m2 = r.gillespie(0, 600, ["Time", "S1"])
r.plot(xtitle="Time", ytitle="Variable")
```

**Listing 14.4** Script for Figure 14.8.

```
import tellurium as te
import pylab

# Bursting in a simple ion channel model
r = te.loada (''
open -> closed; k1*open;
closed -> open; k2*closed;

$ligand + open -> closedLigand; k3*ligand*open;
LigandBlocked -> $ligand + open; k4*LigandBlocked;

open = 1;
closed = 0;
```

```
LigandBlocked = 0;
ligand = 1E-7;
k1 = 400; k2 = 75;
k3 = 8E8;
k4 = 3000;
''')

result = r.gillespie(0, .2, ["time", "open"])
# Plot and set the x and y axes limits
r.plot (xlim=(0,0.1),ylim=(0,2))
```

**Listing 14.5** Script for Figure 14.6.

# **Appendices**



# A

## *List of Symbols and Abbreviations*

### Symbols

---

$a, b, c, \dots$	Used to indicate stoichiometric amounts
$c$	Equilibrium ratio of relaxed and tense form in MWC model
$c_i$	Stoichiometric coefficient
$\Delta$	Change
$D_A$	Diffusion coefficient
$\varepsilon_S^v$	Elasticity coefficient
$h$	Hill coefficient
$k, k_i$	Rate constant
$\Gamma$	Mass-action ratio
$\alpha_i$	Normalized substrate concentration, $S_i / K_m$
$\beta$	Normalized inhibitor concentration, $I / K_i$
$\gamma$	Normalized activator concentration, $A / K_A$
$\Delta_r G$	Reaction free energy change
$J$	Flux
$K_a$	Association constant
$K_d$	Dissociation constant
$K_{eq}$	Equilibrium constant
$K_H$	Half-maximal activity
$K_i$	Inhibition constant
$K_m$	Michaelis constant
$K_s$	Michaelis constant with respect to substrate
$K_p$	Michaelis constant with respect to product

---

$L$	Allosteric constant
$L$	Liter (US spelling)
$M$	Modifier
mol	Mole
$n_i$	Amount of substance $i$
$n$	Number of subunits
$P$	Product concentration
$P_A$	Permeability coefficient
$\pi, \pi_i$	Normalized product concentration, $P_i / K_m$
$\rho$	Disequilibrium constant
$R$	Gas constant
$\sigma$	Modifier factor
$S$	Substrate concentration or entropy depending on context
$T$	Temperature
$t$	Time
$t_{1/2}$	Half-life
$v_f$	Forward reaction rate
$v_i$	$i^{\text{th}}$ reaction rate
$v_r$	Reverse reaction rate
$V$	Volume
$V_m$	Maximal velocity
$X_o$	Reference state for variable $X$

---

## Non-Mathematical Abbreviations

---

AMP, ADP, ATP	Adenine nucleotides
cAMP	Cyclic AMP
CTP	Cytidine triphosphate
DHAP	Dihydroxyacetone phosphate
DNA	Deoxyribonucleic acid
F6P	Fructose-6-Phosphate
FFL	Feedforward Loop
KNF	Koshland, Nemethy and Filmer model
IPTG	Isopropyl $\beta$ -D-1-thiogalactopyranoside
LacI	Lactose Operon Repressor
mRNA	Messenger RNA
MWC	Monod, Wyman and Changeux model
NAD/NADH	Nicotinamide adenine dinucleotide
PEP	Phosphoenolpyruvate
PFK	Phosphofructokinase
RBS	Ribosome binding site
RNA	Ribonucleic acid
SBGN	Systems Biology Graphical Notation
SBML	Systems Biology Markup Language
TF	Transcription factor
TPase	Triose phosphate isomerase

---



# B

## *Useful Numbers*

### B.1 Useful Numbers

---

Numerical data are essential when building models. Sadly such data are scarce, or at least the right kind of data are scarce. In section 4.14 we briefly reviewed some of the potential sources of data for model building [135]. Here we list some basic numbers that a modeler might find useful when building a model. There has been a slow realization that reporting quantitative data is important in systems biology (*nil admirari*). It is therefore worth pointing out a number of efforts to collect and categorize potentially useful data. Two efforts stand out, the cybercell effort<sup>1</sup> started by David Wishart [171], and the more active and larger effort called bionumbers [119] started in 2007 by Ron Milo, Paul Jorgensen and Mike Springer. Other useful sources include the excellent text book ‘Physical Biology of the Cell’ [136], Bernhard Palsson’s book ‘Systems Biology: Simulation of Dynamic Network States’ [131], and of course the book that probably inspired the numbers trend, Uri Alon’s ‘An Introduction to Systems Biology: Design Principles of Biological Circuits’ [3]. The following tables list data gleaned from bionumbers<sup>2</sup>, cybercell and the Phillips book [136]. Some of the yeast data came from an interesting review by Warner [181]. Any mention of data for modeling would be amiss if we didn’t mention Robert Alberty’s book ‘Thermodynamics of Biochemical Reactions’ [1]. This book is a rich source of information on many aspects related to thermodynamics.

---

<sup>1</sup><http://ccdb.wishartlab.com/CCDB/>

<sup>2</sup>[www.biоНumbers.org](http://www.biоНumbers.org)

**Cell Sizes:**

1. Bacteria (*E. coli*) Diameter:  $0.7\text{-}1.4 \mu\text{m}$   
Length:  $2\text{-}4 \mu\text{m}$ ; Volume:  $0.5\text{-}5 \mu\text{m}^3$
2. Yeast (*S. cerevisiae*):  
Diameter:  $3\text{-}6 \mu\text{m}$   
Volume:  $20\text{-}160 \mu\text{m}^3$
3. Mammalian HeLa Cell:  
Diameter:  $15\text{-}30 \mu\text{m}$   
Volume:  $500\text{-}5000 \mu\text{m}^3$

**Length Scales:**

4. Nucleus volume:  $10\%$  of cell volume
5. Cell membrane thickness:  $4\text{-}10 \text{ nm}$
6. Average protein diameter:  $3\text{-}6 \text{ nm}$
7. DNA diameter:  $2 \text{ nm}$
8. Water molecule diameter:  $0.3 \text{ nm}$

**Concentration Equivalents in *E. coli*:**

9. Concentration of 1 nM in:  
*E. coli*:  $1 \text{ molecule/cell}$   
HeLa:  $1000 \text{ molecules/cell}$
10. Concentration in *E. coli* of  
ATP:  $2 \text{ mM}$   
Pyruvate:  $0.37 \text{ mM}$   
LacI:  $1\text{-}50 \text{ nM}$   
Pyruvate Kinase:  $11 \mu\text{M}$
11. Number of receptor proteins:  $1,000,000$
12. Number of soluble proteins:  $2\text{-}4 \text{ million}$
13. Number of ribosomes:  $18,000$

**Diffusion Rates:**

14. Diffusion coefficient for average protein:  
 $D = 5\text{-}15 \mu\text{m}^2 \text{ s}^{-1}$  which equals  
 $10 \text{ millisec}$  to traverse *E. coli  
 $10 \text{ secs}$  to traverse HeLa cell*
15. Yeast (*S. cerevisiae*):  
Diameter:  $3\text{-}6 \mu\text{m}$   
Volume:  $20\text{-}160 \mu\text{m}^3$
16. Mammalian HeLa Cell:  
Diameter:  $15\text{-}30 \mu\text{m}$   
Volume:  $500\text{-}5,000 \mu\text{m}^3$

**Reaction Rates for *E. coli*:**

- 17. Cell Division Time: 30-60 mins
- 18. Rate of replication: 2,000 bp/s
- 19. Protein synthesis: 1,000 proteins/s
- 20. Lipid synthesis: 20,000 lipids/s
- 21. Ribosome rate: 25 amino acids per sec
- 22. Transcription rate: 45 mins
- 23. ATPs to make a cell: 55 billion
- 24. Reaction rate Pyruvate kinase:  
500,000 molecules per second

**Reaction Rates for Yeast:**

- 25. Ribosomes made per second: 2,000
- 26. Ribosomes in Yeast: 200,000

**Energetics:**

- 27.  $\Delta G$  that represents an order of magnitude ratio between products and reactants: 6 kJ/mol
- 28. Membrane potential: 70-200 mV

**Fundamental Constants:**

- 29. Avogadro's Number:  $6.02214129 \times 10^{23} \text{ mol}^{-1}$
- 30. Gas Constant:  $8.314472 \text{ J mol}^{-1} \text{ K}^{-1}$
- 31. Faraday Constant:  $96485.3383 \text{ C mol}^{-1}$



# C

## *Answers to Questions*

### **Chapter 1**

1. Assume length of cell is  $2\mu m$ : 250
2. Assume *E. coli* is a cylinder: volume =  $1.57\mu m^3$  or  $1.57 \times 10^{-15} L$
- 3a. Assume *E. coli* is a cylinder: Area =  $7.85 \mu m^2$
- 3b. Approximately 218,000 proteins
4. NA
5. 25 minutes (1500 seconds)
6. 0.67 seconds
- 7a.  $5 \times 10^{-16}$  mmoles per second per volume of *E. coli*
- 7b.  $3 \times 10^5$  per second per *E. coli*



# D

## *Enzyme Kinetics in a Nutshell*

This appendix gives a very brief summary of some of the main points in enzyme kinetics. It is not meant to be a thorough treatment. For a much fuller account, the reader is directed to one of a number of enzyme kinetics textbooks, including the companion textbook by the same author: ‘Enzyme Kinetics for Systems Biology’.

### **Enzymes**

Enzymes are protein molecules that can accelerate a chemical reaction without changing the reaction equilibrium constant.

### **Enzyme Kinetics**

Enzyme kinetics is a branch of science that deals with the many factors that can affect the rate of an enzyme-catalysed reaction. The most important factors include the concentration of enzyme, reactants, products, and the concentration of any modifiers such as specific activators, inhibitors, pH, ionic strength, and temperature. When the action of these factors is studied, we can deduce the kinetic mechanism of the reaction. That is, the order in which substrates and products bind and unbind, and the mechanism by which modifiers alter the reaction rate.

## D.1 Michaelis-Menten Kinetics

The standard model for enzyme action describes the binding of free enzyme to the reactant forming an **enzyme-reactant complex**. This complex undergoes a transformation, releasing product and free enzyme. The free enzyme is then available for another round of binding to new reactant.

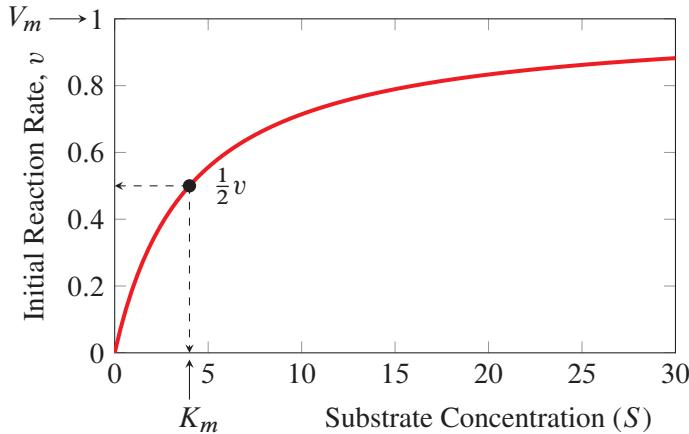


where  $k_1$ ,  $k_{-1}$  and  $k_2$  are rate constants,  $S$  is substrate,  $P$  is product,  $E$  is the free enzyme, and  $ES$  the enzyme-substrate complex.

By assuming a steady state condition on the enzyme substrate complex, we can derive the Briggs-Haldane equation relation (sometimes mistakenly called the Michaelis-Menten equation):

$$v = \frac{V_m S}{K_m + S} \quad (\text{D.2})$$

where  $V_m$  is the maximal velocity, and  $K_m$  the substrate concentration that yields half the maximum velocity.



**Figure D.1** Relationship between the initial rate of reaction and substrate concentration for a simple Michaelis-Menten rate law. The reaction rate reaches a limiting value called the  $V_m$ .  $K_m$  is set to 4.0 and  $V_m$  to 1.0. The  $K_m$  value is the substrate concentration that gives half the maximal rate.

## D.2 Reversibility and Product Inhibition

*In vivo* it is unlikely that an enzyme reaction is completely irreversible. Even if an enzyme

shows no reverse reaction rate from product to substrate, there is still likely to be some degree of product inhibition because the product can bind to the active site and compete with the substrate.

## D.3 Reversible Rate laws

---

An alternative and more realistic model is the reversible form:



The aggregate rate law for the reversible form of the mechanism can also be derived and is given by:

$$v = \frac{V_f S/K_S - V_r P/K_P}{1 + S/K_S + P/K_P} \quad (D.4)$$

## D.4 Haldane Relationship

---

For the reversible enzyme kinetic law there is an important relationship:

$$K_{eq} = \frac{P_{eq}}{S_{eq}} = \frac{V_f K_P}{V_r K_S} \quad (D.5)$$

This equation shows that the four kinetic constants,  $V_f$ ,  $V_r$ ,  $K_P$  and  $K_S$  are not independent. Haldane relationships can be used to eliminate one of the kinetic constants by substituting the equilibrium constant in its place. This is useful because equilibrium constants tend to be known compared to kinetic constants which may be unknown. By incorporating the Haldane relationship, we can eliminate the reverse maximal velocity ( $V_r$ ) from D.4 to yield the equation:

$$v = \frac{V_f / K_S (S - P / K_{eq})}{1 + S / K_S + P / K_P} \quad (D.6)$$

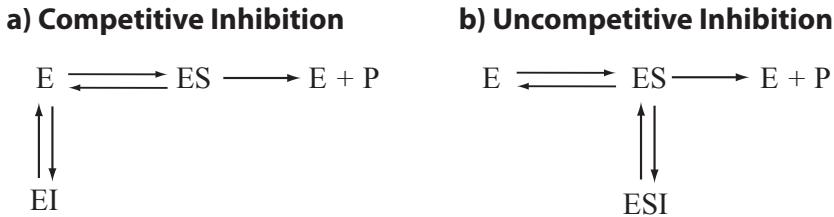
Separating out the terms makes it easier to see that the above equation can be partitioned into a number of distinct parts:

$$v = V_f \cdot (1 - \Gamma / K_{eq}) \cdot \frac{S / K_S}{1 + S / K_S + P / K_P} \quad (D.7)$$

where  $\Gamma = P/S$ . The first term,  $V_f$ , is the maximal velocity; the second term,  $(1 - \Gamma / K_{eq})$ , indicates the direction of the reaction according to thermodynamic considerations. The last term refers to the fractional saturation with respect to substrate. Thus we have a maximal velocity, a thermodynamic and a saturation term.

## D.5 Competitive Inhibition

There are many molecules capable of slowing down or speeding up the rate of enzyme catalyzed reactions. Such molecules are called enzyme inhibitors and activators. One common type of inhibition, called **competitive inhibition**, occurs when the inhibitor is structurally similar to the substrate so that it competes for the active site by forming a dead-end complex.



**Figure D.2** Competitive and Uncompetitive Inhibition.  $P$  is the concentration of product,  $E$  is the free enzyme,  $ES$  the enzyme-substrate complex, and  $ESI$  the enzyme-substrate-inhibitor complex.

The kinetic mechanism for a pure competitive inhibitor is shown in Figure D.2(a), where  $I$  is the inhibitor and  $EI$  the enzyme inhibitor complex. If the substrate concentration is increased, it is possible for the substrate to eventually out compete the inhibitor. For this reason the inhibitor alters the enzyme's apparent  $K_m$ , but not the  $V_m$ .

$$\begin{aligned} v &= \frac{V_m S}{S + K_m \left(1 + \frac{I}{K_i}\right)} \\ &= \frac{V_m S / K_m}{1 + S / K_m + I / K_i} \end{aligned} \quad (\text{D.8})$$

At  $I = 0$ , the competitive inhibition equation reduces to the normal irreversible Michaelis-Menten equation. Note that the term  $K_m(1 + I/K_i)$  in the first equation more clearly shows the impact of the inhibitor,  $I$ , on the  $K_m$ . The inhibitor has no effect on the  $V_m$ .

A reversible form of the competitive rate law can also be derived:

$$v = \frac{\frac{V_m}{K_s} \left(S - \frac{P}{K_{eq}}\right)}{1 + \frac{S}{K_s} + \frac{P}{K_p} + \frac{I}{K_i}} \quad (\text{D.9})$$

where  $V_m$  is the forward maximal velocity, and  $K_s$  and  $K_p$  are the substrate and product half saturation constants.

Sometimes reactions appear irreversible, where no discernable reverse rate is detected, and yet the forward reaction is influenced by the accumulation of product. This effect is caused by the product competing with substrate for binding to the active site and is often called **product inhibition**. Given that product inhibition is a type of competitive inhibition, we will briefly discuss it. An important industrial example of this is the conversion of lactose to galactose by the enzyme  $\beta$ -galactosidase where galactose competes with lactose, slowing the forward rate [53].

To describe simple product inhibition with an irreversible reaction, we can set the  $P/K_{eq}$  term in the reversible Michaelis-Menten rate law (D.4) to zero. This yields:

$$v = \frac{V_m S}{S + K_m \left(1 + \frac{P}{K_p}\right)} \quad (\text{D.10})$$

It is not surprising to discover that equation (D.10) has exactly the same form as the equation for competitive inhibition (D.8). As the product increases, it out competes the substrate and therefore slows down the reaction rate.

We can also derive the equation by using the following mechanism and the rapid-equilibrium assumption:



where the reaction rate  $v$  is assumed to be proportional to  $ES$ .

## D.6 Cooperativity

---

Many proteins are known to be oligomeric, meaning they are composed of more than one identical protein subunit where each subunit has one or more binding sites. Often the individual subunits are identical.

If the binding of a ligand (a small molecule that binds to a larger macromolecule) to one site alters the affinity at other sites on the same oligomer, it is called **cooperativity**. If ligand binding increases the affinity of subsequent binding events, it is termed **positive cooperativity** whereas if the affinity decreases, it is termed **negative cooperativity**. One characteristic of positive cooperativity is that it results in a sigmoidal response instead of the usual hyperbolic response.

The simplest equation that displays sigmoid like behavior is the Hill equation:

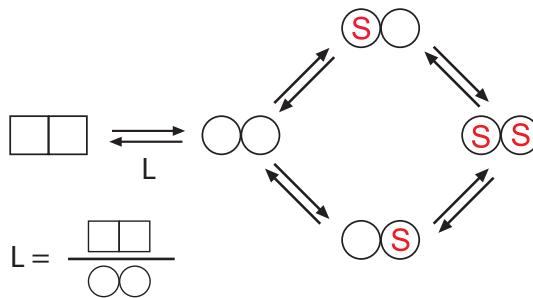
$$v = \frac{V_m S^n}{K_d + S^n} \quad (\text{D.12})$$

One striking feature of many oligomeric proteins is the way individual monomers are physically arranged. Often one will find at least one axis of symmetry. The individual protein monomers are not arranged in a haphazard fashion. This level of symmetry may imply that

the gradual change in the binding constants as ligands bind, might be physically implausible. Instead, one might envision transitions to an alternative binding state that occurs within the entire oligomer complex. This model was originally suggested by Monod, Wyman and Changeux [120], abbreviated as the MWC model. The original authors laid out the following criteria for the MWC model:

1. The protein is an oligomer.
2. Oligomers can exist in two states: R (relaxed) and T (tense). In each state, symmetry is preserved and all subunits must be in the same state for a given R or T state.
3. The R state has a higher ligand affinity than the T state.
4. The T state predominates in the absence of ligand S.
5. The ligand binding microscopic association constants are all identical.

Given these criteria, the MWC model assumes that an oligomeric enzyme may exist in two conformations, designated T (tensed, square) and R (relaxed, circle). The equilibrium between the two states has an equilibrium constant  $L = T/R$ , which is also called the **allosteric constant**. If the binding constants of ligand to the two states are different, the distribution of the R and T forms can be displaced towards either one form or the other. By this mechanism, the enzyme displays sigmoid behavior. A minimal example of this model is shown in Figure D.3.



**Figure D.3** A minimal MWC model, also known as the exclusive model, showing alternative microscopic states in the circle (relaxed) form.  $L$  is called the allosteric constant. The square form is called the tense state.

In the **exclusive model** (Figure D.3) the ligand can only bind to the relaxed form (circle). The mechanism that generates sigmoidicity in this model works as follows. When ligand binds to the relaxed form, it displaces the equilibrium from the tense form to the relaxed form. In doing so, additional ligand binding sites become available. Thus, one ligand binding may generate four or more new binding sites. Eventually there are no more tense states remaining, at which point the system is saturated with ligand. The overall binding

curve will therefore be sigmoidal and will show positive cooperativity. Given the nature of this model, it is not possible to generate negative cooperativity. By assuming equilibrium between the various states, it is possible to derive an aggregate equation for the dimer case of the exclusive MWC model:

$$v = V_m \frac{\frac{S}{k_R} \left(1 + \frac{S}{k_R}\right)}{\left(1 + \frac{S}{k_R}\right)^2 + L}$$

This also generalizes to  $n$  subunits as follows:

$$Y = \frac{\frac{S}{k_R} \left(1 + \frac{S}{k_R}\right)^{n-1}}{\left(1 + \frac{S}{k_R}\right)^n + L} \quad (\text{D.13})$$

For more generalized reversible rate laws that exhibit sigmoid behavior, the reversible Hill equation is a good option. Invoking the rapid-equilibrium assumption, we can form a reversible rate law that shows cooperativity:

$$v = \frac{V_f \alpha (1 - \rho) (\alpha + \pi)}{1 + (\alpha + \pi)^2}$$

where  $\rho = \Gamma/K_{eq}$  and  $\alpha$  and  $\pi$  are the ratio of reactant and product to their respective equilibrium constant,  $\alpha/K_S$  and  $\pi/K_P$ . For an enzyme with  $h$  (using the author's original notation) binding sites, the general form of the reversible Hill equation is given by:

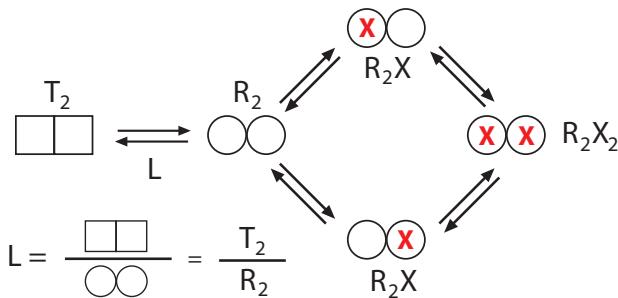
$$v = \frac{V_f \alpha (1 - \rho) (\alpha + \pi)^{h-1}}{1 + (\alpha + \pi)^h} \quad (\text{D.14})$$

## D.7 Allostery

---

An allosteric effect is where the activity of an enzyme or other protein is affected by the binding of an effector molecule at a site on the protein's surface, other than the active site. The MWC model described previously can be easily modified to accommodate allosteric action.

The key to including allosteric effectors is to influence the equilibrium between the tense (T) and relaxed (R) states (See Figure D.4). To influence the sigmoid curve, an allosteric effector need only displace the equilibrium between the tense and relaxed forms. For example, to behave as an activator, an allosteric effector needs to preferentially bind to the



**Figure D.4** Exclusive MWC model based on a dimer showing alternative microscopic states in the form of  $T$  and  $R$  states. The model is exclusive because the ligand,  $X$ , only binds to the  $R$  form.

$R$  form and shift the equilibrium away from the less active  $T$  form. An allosteric inhibitor would do the opposite, that is bind preferentially to the  $T$  form so that the equilibrium shifts towards the less active  $T$  form. In both cases the  $V_m$  of the enzyme is unaffected.

The net result of this is to modify the normal MWC aggregate rate law to the following if the effector is an inhibitor:

$$v = V_m \frac{\alpha (1 + \alpha)^{n-1}}{(1 + \alpha)^n + L(1 + \beta)^n} \quad (\text{D.15})$$

where  $\alpha = S/K_s$ ,  $\beta = I/K_I$ , and  $K_s$  and  $K_I$  are kinetic constants related to each ligand. A MWC model that is regulated by an inhibitor or an activator is described by the equation:

$$v = V_m \frac{\alpha (1 + \alpha)^{n-1}}{(1 + \alpha)^n + L \frac{(1 + \beta)^n}{(1 + \gamma)^n}}$$

There are also reversible forms of the allosteric MWC model but they are fairly complex. Instead, it is possible to modify the reversible Hill rate law to include allosteric ligands.

$$v = \frac{V_f \alpha \left(1 - \frac{\Gamma}{K_{eq}}\right) (\alpha + \pi)^{h-1}}{\frac{1 + \mu^h}{1 + \sigma \mu^h} + (\alpha + \pi)^h} \quad (\text{D.16})$$

where:

$$\begin{aligned} \sigma < 1 && \text{inhibitor} \\ \sigma > 1 && \text{activator} \end{aligned}$$

## Simple Hill Equations

When modeling gene regulatory networks, we often need simple activation and repression rate laws. It is common to use the following Hill like equations to model activation and repression, respectively. The third equation shows one example of how we can model dual repression and activation, where  $S_1$  acts as the activator and  $S_2$  the inhibitor.  $n_1$  and  $n_2$  are Hill like coefficients which may be used to alter the responsiveness of each factor.

$$\text{Activation: } v = \frac{V_m S^n}{K + S^n}$$

$$\text{Repression: } v = \frac{V_m}{K + S^n}$$

$$\text{Dual: } v = \frac{V_m S_1^{n_1}}{1 + K_1 S_1^{n_1} + K_2 S_2^{n_2} + K_3 S_1^{n_1} S_2^{n_2}}$$

## D.8 Elasticities

Elasticities measure the response of a chemical reaction rate to changes in the immediate environment. For example, given a simple reaction such as:



we can measure two elasticities, one with respect to  $S$  and the other with respect to  $P$ . Each elasticity gives us the response of the reaction rate when either  $S$  or  $P$  are changed, respectively. Mathematically, the elasticity is defined in terms of a scaled derivative:

$$\varepsilon_S^v = \frac{\partial v}{\partial S} \frac{S}{v} \simeq \frac{v\%}{S\%} \quad (\text{D.17})$$

According to the definition, one can interpret an elasticity as a ratio of relative changes. Even though the elasticity is only defined for infinitesimal changes, we can approximate the elasticity in terms of small finite changes and conveniently interpret it as the ratio of percentage changes. For example, if we were to make a 2% change in  $S$ , and in turn observed a 0.5% change in the reaction velocity, then the value of the elasticity is given approximately by the ratio  $0.5/2 = 0.25$ . Full details of the elasticity and its properties can be found in the companion book Enzyme Kinetics for Systems Biology.

### Unscaled Elasticity

We can also define the unscaled elasticity as:

$$\mathcal{E}_S^v = \frac{\partial v}{\partial S} \quad (\text{D.18})$$

---

**Further Reading**

---

1. Sauro HM (2012) Enzyme Kinetics for Systems Biology. 2nd Edition, Ambrosius Publishing ISBN: 978-0982477335

# E

## *Math Fundamentals*

This Appendix highlights some of the notation used in this book and summarizes the most important mathematical concepts that students should be familiar with.

### **E.1 Notation**

---

#### **Sum and Product:**

$$a_1 + a_2 + a_3 + \dots + a_n = \sum_{i=1}^n a_i$$

$$a_1 \times a_2 \times a_3 \times \dots \times a_n = \prod_{i=1}^n a_i$$

#### **Vectors and Matrices:**

Bold lower case letters indicate vectors, for example:  $\mathbf{v}, \mathbf{s}$

Bold upper case letters indicate matrices, for example:  $\mathbf{N}, \mathbf{X}$

**Derivatives:**

On the left is Leibniz's notation and on the right Lagrange's notation:

$$\frac{df}{dx} \equiv f'(x)$$

$$\frac{d^2f}{dx^2} \equiv f''(x)$$

$$\frac{d^n f}{dx^n} \equiv f^{(n)}(x)$$

**E.2 Short Table of Derivatives**

---

$$\frac{d}{dx}[c] = 0$$

$$\frac{d}{dx}[x] = 1$$

$$\frac{d}{dx}[cu] = c \frac{du}{dx}$$

$$\frac{d}{dx}[u + v] = \frac{du}{dx} + \frac{dv}{dx}$$

$$\frac{d}{dx}[uv] = u \frac{dv}{dx} + v \frac{du}{dx}$$

$$\frac{d}{dx}[u/v] = \frac{v \frac{du}{dx} - u \frac{dv}{dx}}{v^2}$$

$$\frac{d}{dx}[u^n] = nu^{n-1} \frac{du}{dx}$$

$$\frac{d}{dx}[f(u)] = \frac{df}{du} f(u) \frac{du}{dx}$$

$$\frac{d}{dx}[\ln u] = \frac{1}{u} \frac{du}{dx}$$

$$\frac{de^u}{dx} = e^u \frac{du}{dx}$$

$$\frac{d}{dx}[\sin(u)] = \cos(u) \frac{du}{dx}$$

$$\frac{d}{dx}[\cos(u)] = -\sin(u) \frac{du}{dx}$$

## E.3 Logarithms

---

$$\log(AB) = \log(A) + \log(B)$$

$$\log(A/B) = \log(A) - \log(B)$$

$$\log(A^n) = n \log(A)$$

$$x^n \times x^m = x^{n+m}$$

$$\frac{x^n}{n^m} = x^{n-m}$$

$$(x^n)^m = x^{n \times m}$$

## E.4 Partial Derivatives

---

If the value of a given function depends on two variables, then we write this function in the form:

$$u = f(x, y)$$

If it is possible to change  $x$  without affecting  $y$ , then  $x$  and  $y$  are called independent variables. The rate of change of  $u$  with respect to  $x$  when  $x$  varies, but  $y$  remains constant, is called the **partial derivative** of  $u$  with respect to  $x$ . Partial derivatives are denoted using the partial symbol,  $\partial$ . For example, the partial derivative of  $u$  with respect to  $x$ :

$$\frac{\partial u}{\partial x}$$

Likewise, the partial derivative of  $u$  with respect to  $y$  is:

$$\frac{\partial u}{\partial y}$$

To find a partial derivative we simply differentiate with respect to the variable of interest while treating the remaining variables as constants. For example, if the reaction rate for a given reaction is  $v = k_1 S - k_2 P$ , where  $S$  is the reactant,  $P$  the product, and  $k_1$  and  $k_2$  the rate constants. In a controlled environment we should in principle be able to independently change  $S$  and  $P$ . Therefore, we can write down the partial derivatives of the reaction rate with respect to  $S$  and  $P$  as follows:

$$\frac{\partial v}{\partial S} = k_1$$

$$\frac{\partial v}{\partial P} = -k_2$$

In order to indicate what variables are kept constant in the partial derivative, the following notation is sometimes used, particularly in thermodynamics:

$$\left(\frac{\partial v}{\partial S}\right)_P = k_1$$

$$\left(\frac{\partial v}{\partial P}\right)_S = -k_2$$

For functions with many variables,  $x, y, z, \dots$ , the notation extends to:

$$\left(\frac{\partial u}{\partial x}\right)_{y,z,\dots}$$

Like derivatives, partial derivatives are defined in terms of limits. For example, the partial derivatives for the function,  $f(x, y)$  are defined as:

$$\begin{aligned}\frac{\partial f(x, y)}{\partial x} &= \lim_{h \rightarrow 0} \frac{f(x + h, y) - f(x, y)}{h} \\ \frac{\partial f(x, y)}{\partial y} &= \lim_{h \rightarrow 0} \frac{f(x, y + h) - f(x, y)}{h}\end{aligned}$$

The graphical interpretation of a partial derivative,  $\partial f(x, y)/\partial x$ , is that it represents the slope of the function,  $f(x, y)$  in the  $x$  direction.

## E.5 Differential Equations

---

Differential equations are equations that contain derivatives. For example, the following is a differential equation:

$$\frac{dy}{dx} + y^2 = 0$$

An **ordinary differential equation** is where the derivative is a function of single independent variable. In science and engineering this independent variable is often time. For example, the following equations are ordinary differential equations:

$$\frac{dy}{dx} = ay$$

$$\frac{dy}{dx} = 2x + 3y - 8$$

$$\frac{d^2y}{dx^2} - x \frac{du}{dx} = 0$$

A differential equation expressed in terms of the first derivative ( $dy/dx$ ) is called a first-order differential equation. A differential equation that is expressed in terms of second-order derivatives ( $d^2y/dx^2$ ) is called a second-order differential equation. When solving

differential equations the objective is to find the function  $y(x)$  such that when differentiated, gives the original differential equation. For example, the solution to:

$$\frac{dy}{dx} = ay$$

is

$$y = y_0 e^{ax} \quad (\text{E.1})$$

If we differentiate solution (E.1), we get back the original differential equation.

Differential equations are used frequently to model physical systems, describing the rate of change of some variable with respect to time,  $t$ . They are useful because we may not explicitly know the solution  $y(t)$ , but we will often know the rate of change of the variable at any given moment in time,  $dy/dt$ . This means we can at least obtain a numerical solution to  $y(t)$  even if the analytical solution is unobtainable.,

Differential equations can be further classified as autonomous or non-autonomous. Autonomous differential equations are the most common in biochemical models. These equations do not depend on time, that is the right-hand side of the differential equation has no terms relating explicitly to time. For example, equation E.2 is autonomous, while equation E.3 is non-autonomous:

$$\frac{dx}{dt} = x^2 + 10 \quad (\text{E.2})$$

$$\frac{dx}{dt} = x^2 + t - 5 \quad (\text{E.3})$$

A **partial differential equation** is one where the derivatives are functions of more than one independent variables. Often in science and engineering, partial differential equations are a function of independent variables, time and space. For example, the following equation is a partial differential equation:

$$\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} = \frac{\partial p}{\partial x}$$

Note the use of the partial  $d$  ( $\partial$ ) in the partial differential equation to indicate that the function  $u$  is differentiated with respect to more than one variable.

## E.6 Taylor Series

---

Expressions like  $1 + 2x + 6x^2$  and  $2 + 4x + x^2 - 3x^3$  that contain the sum of a number of terms raised to a positive power are called polynomials. The only operations allowed in a polynomial are addition, subtraction, multiplication and non-negative integer powers. One of the simplest polynomials is the straight line,  $y = a + bx$ , termed a polynomial of first degree. The coefficients,  $a$  and  $b$ , can be chosen so that the line will pass through any two points. As such, we can express any straight line using  $y = a + bx$ . Similarly for a

polynomial of second degree,  $y = a + bx + cx^2$ , a parabola, we can choose the constants,  $a$ ,  $b$ , and  $c$  so that the curve passes through any three points.

It follows that we can find a polynomial equation of  $n^{\text{th}}$  degree that will pass through any  $n + 1$  points. If the polynomial has an infinite number of terms, we imagine it could be made to follow any function,  $f(x)$ , by suitable adjustment of the polynomial coefficients. Although this statement may not always be true, in many cases it is, which makes the polynomial series very useful.

A polynomial of infinite degree is called a polynomial series:

$$f(x) = c_0 + c_1x + c_2x^2 + c_3x^3 + \dots$$

The question is, how can we find the polynomial series that will represent a given function, for example  $\sin(x)$ ? To answer this we have to determine the constants,  $c_0$ ,  $c_1$ , etc. in the polynomial equation. Let us assume that we wish to know the value of  $\sin(x)$  at  $x = 0$  using a polynomial series. At  $x = 0$ , all terms vanish except for  $c_0$ , therefore at  $x = 0$ :

$$f(0) = c_0$$

We can therefore interpret the first constant,  $c_0$ , as the value of the function at  $x = 0$ . What about  $c_1$ ? Let us take the derivative of the series, that is:

$$f'(x) = c_1 + 2c_2x + 3c_3x^2 + \dots$$

If we set  $x = 0$ , we find that:

$$f'(0) = c_1$$

The second constant,  $c_1$ , in the polynomial series is the first derivative of the function. If we take the second derivative we can also show at  $x = 0$ ,  $f''(0) = 2c_2$ , that is  $c_2 = f''(0)/2$ . For the third derivative we can show that  $f'''(0) = 3 \times 2 \times c_3$ , that is  $c_3 = f'''(0)/(3!)$ . This pattern continues for the remaining terms in the polynomial so we can write:

$$f(x) = f(0) + f'(0)x + \frac{f''(0)}{2!}x^2 + \frac{f'''(0)}{3!}x^3 + \dots$$

This series is called the **Maclaurin series** for the function,  $f(x)$ . It approximates the function around the specific value of  $x = 0$ . To illustrate the use of the Maclaurin series, consider expanding  $\sin(x)$  around  $x = 0$ .  $f(0)$  equals  $\sin(0) = 0$ .  $f'(0) = \cos(0) = 1$ , and so on. We can therefore write the series as:

$$\sin(x) = 0 + 1x + 0 - \frac{1}{3!}x^3 + 0 + \frac{1}{5!}x^5 - \dots$$

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \dots$$

Function	Second-order approximation
$\frac{1}{1+x}$	$1 + x + x^2$
$\sqrt{1+x}$	$1 + \frac{x}{2} + \frac{x^2}{8}$
$\sin(x)$	$x - \frac{x^3}{3!}$

**Table E.1** Examples of common approximations.

What if we wanted to approximate a function about an arbitrary value  $x_o$ ? To do this we use the Taylor series, which is a generalization of the Maclaurin series. The **Taylor series** is defined by:

$$f(x) = f(x_o) + \frac{\partial f}{\partial x} \Big|_{x=x_o} (x - x_o) + \frac{1}{2!} \frac{\partial^2 f}{\partial x^2} \Big|_{x=x_o} (x - x_o)^2 + \dots + \frac{1}{n!} \frac{\partial^n f}{\partial x^n} \Big|_{x=x_o} (x - x_o)^n + \dots \quad (\text{E.4})$$

where the approximation is now centered on  $x_o$ . If we set  $x_o$  equal to zero, we obtain the Maclaurin series.

If we keep three terms in the Taylor series, we obtain another very important approximation called the **quadratic approximation**:

$$f(x) = f(x_0) + \delta x \frac{df}{dx} \Big|_{x=x_0} + \frac{1}{2} (\delta x)^2 \frac{d^2 f}{dx^2} \Big|_{x=x_0} \quad (\text{E.5})$$

In optimization strategies, we can often approximate the fitness surface using a quadratic function near the optimum.

## Taylor Series in Two Dimensions

It is possible to derive a Taylor series for equations with multiple variables, for example  $f(x, y)$ . In this case the expansion is a little bit more complicated. A Taylor series expansion around  $x_o$  and  $y_o$  for the function  $f(x, y)$  is given by:

$$\begin{aligned} f(x, y) \approx & f(x_o, y_o) + \frac{\partial f}{\partial x} \Big|_{x=x_o} (x - x_o) + \frac{\partial f}{\partial y} \Big|_{y=y_o} (y - y_o) + \\ & \frac{1}{2!} \left[ (x - x_o)^2 \frac{\partial^2 f}{\partial x^2} \Big|_{x=x_o} + 2(x - x_o)(y - y_o) \frac{\partial^2 f}{\partial x \partial y} \Big|_{x=x_o, y=y_o} + (y - y_o)^2 \frac{\partial^2 f}{\partial y^2} \Big|_{y=y_o} \right] \\ & + \dots \end{aligned}$$

where all derivatives are evaluated at the operating point,  $x_o, y_o$ . There is a very nice compact form for this equation, which can be written in terms of vectors and a matrix:

$$f(\mathbf{x}_o) = f(\mathbf{x}_o) + (\mathbf{x} - \mathbf{x}_o)^T \nabla f + \frac{1}{2!}(\mathbf{x} - \mathbf{x}_o)^T \mathbf{H} (\mathbf{x} - \mathbf{x}_o) + \dots$$

where  $\nabla f$  is called the gradient (or grad f), and  $\mathbf{H}$  the Hessian. Note that in vector notation,  $\mathbf{v}^T$  means that the vector is in row form, because by convention a vector is often depicted as a column. Given a function,  $f(x, y, \dots)$ ,  $\nabla f$  is just the vector of partial derivatives:

$$\nabla f = \left[ \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \dots \right]^T$$

The following equivalent notation in terms of the unit vectors is also frequently found in the literature for  $\nabla f$ :

$$\nabla f = \frac{\partial f}{\partial x} \mathbf{i} + \frac{\partial f}{\partial y} \mathbf{j} + \dots$$

where  $\mathbf{i}$  and  $\mathbf{j}$  are the standard basis vectors, i.e  $\mathbf{i} = [1, 0, 0, \dots]$ ,  $\mathbf{j} = [0, 1, 0, \dots]$ , etc. The Hessian matrix,  $\mathbf{H}$ , for the function  $f(x, y)$  is given by:

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial x \partial x} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y \partial y} \end{bmatrix} = \left[ \frac{\partial^2 f}{\partial x_i \partial x_j} \right]$$

which can be naturally extended to functions with any number of variables.

## E.7 Total Derivative

---

Consider the function:

$$f(t) = f(x(t), y(t), t)$$

The derivative of  $f(t)$  with respect to  $t$ , is given by the chain rule:

$$\frac{df}{dt} = \frac{\partial f}{\partial x} \frac{dx}{dt} + \frac{\partial f}{\partial y} \frac{dy}{dt} + \frac{\partial f}{\partial t}$$

Note the use of partial derivatives. This equation is often abbreviated to:

$$df = \frac{\partial f}{\partial x} dx + \frac{\partial f}{\partial y} dy + \frac{\partial f}{\partial t} dt$$

where it is called the **total derivative**. Operationally, the total derivative computes the change in  $f$ , given small changes in  $x$  and  $y$ .

## E.8 Eigenvalues and Eigenvectors

---

A square matrix such as  $\mathbf{A}$  can be used to transform a vector,  $\mathbf{v}$  in specific ways. For example, if the matrix  $\mathbf{A}$  is:

$$\begin{bmatrix} 2 & 0 \\ 0 & 4 \end{bmatrix}$$

then the result of multiplying  $\mathbf{A}$  into  $\mathbf{v}$  will yield a vector that is similar to  $\mathbf{v}$  but where the first element is scaled by 2 and the second element by 4.

For an arbitrary square matrix, if it is possible to find a vector  $\mathbf{v}$  such that when we multiply the vector by  $\mathbf{A}$  we get a scaled version of  $\mathbf{v}$ , then we call the vector  $\mathbf{v}$  an **eigenvector** of  $\mathbf{A}$  and the scaling value, the **eigenvalue** of  $\mathbf{A}$ . For a matrix of dimension  $n$ , there will be at most  $n$  eigenvalues and  $n$  eigenvectors. In the case of the simple example above, the eigenvalues are 2 and 4, respectively, while the two eigenvector are:

$$\begin{bmatrix} \alpha \\ 0 \end{bmatrix} \quad \begin{bmatrix} 0 \\ \alpha \end{bmatrix}$$

The definition of an eigenvector and eigenvalue is often given in the form:

$$\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$$

We can rearrange this equation as follows:

$$\begin{aligned} \mathbf{A}\mathbf{v} &= \lambda\mathbf{I}\mathbf{v} \\ \mathbf{A}\mathbf{v} - \lambda\mathbf{I}\mathbf{v} &= 0 \\ (\mathbf{A} - \lambda\mathbf{I})\mathbf{v} &= 0 \end{aligned}$$

From linear algebra, we know there will be non-zero solutions to  $(\mathbf{A} - \lambda\mathbf{I})\mathbf{v} = 0$  if  $\det(\mathbf{A} - \lambda\mathbf{I}) = 0$ . We can use this observation to compute the eigenvalues and eigenvectors of a matrix. For example, consider the matrix:

$$\begin{bmatrix} 3 & 6 \\ 1 & 4 \end{bmatrix}$$

Computing  $\mathbf{A} - \lambda\mathbf{I}$  yields:

$$\begin{aligned} \mathbf{A} - \lambda\mathbf{I} &= \begin{bmatrix} 3 - \lambda & 6 \\ 1 & 4 - \lambda \end{bmatrix} \\ \det(\mathbf{A} - \lambda\mathbf{I}) &= (3 - \lambda)(4 - \lambda) - 6 \\ &= \lambda^2 - 7\lambda + 6 \\ &= (\lambda - 6)(\lambda - 1) \end{aligned}$$

The eigenvalues are therefore 6 and 1. With two distinct eigenvalues there will be two eigenvectors. First we consider  $\lambda = 6$ .

$$(A - \lambda I)v = 0$$
$$\left( \begin{bmatrix} 3 & 6 \\ 1 & 4 \end{bmatrix} - \begin{bmatrix} 6 & 0 \\ 0 & 6 \end{bmatrix} \right) v = 0$$
$$\begin{bmatrix} -3 & 6 \\ 1 & -2 \end{bmatrix} v = 0$$

By inspection we can see that the eigenvector is:

$$\begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

Likewise we can do the same for the other eigenvalue,  $\lambda = 1$  where the corresponding eigenvector is:

$$\begin{bmatrix} -3 \\ 1 \end{bmatrix}$$

---

## Further Reading

1. Smail LL (1953) Analytical Geometry and Calculus. Appleton-Century-Crofts ISBN: 978-0982477311

# F

## Statistics Reminder

### F.1 Mean

---

The **mean** is the sum of values divided by the number of values:

$$\bar{x} = \frac{1}{N} \sum_{i=1}^N x_i$$

The mean is not necessarily the middle value but depends on the skewness of the values. The central value is called the **median**.

### F.2 Deviation

---

A measure of deviation of a variable  $y$  from its mean is called the **standard deviation**, denoted by  $\sigma$ . A related measure,  $\sigma^2$ , is called the **variance**. Consider the mean of a set of numbers,  $x_i$ , denoted by  $\bar{x}$ . We can compute the deviation of each  $x_i$  from the mean by:

$$d_i = x_i - \bar{x}$$

If we take the square of the deviations and compute the average deviation we obtain the variance:

$$\sigma^2 = \frac{1}{N} \sum (x_i - \bar{x})^2$$

Given a sample from a population, the best estimate for the population variance must be correct if we attempt to compute the population variance from the sample:

$$\sigma^2 = \frac{1}{N-1} \sum (x_i - \bar{x})^2$$

### F.3 Standard Error

---

If we were to sample a population multiple times, we could calculate a mean for each sample. The standard deviation for the set of means is called the standard error. The value of the standard error can be calculated using a remarkably simple formula:

$$SE_{\bar{x}} = \frac{\sigma}{\sqrt{N}}$$

Strictly speaking,  $\sigma$  should be the standard deviation of the population but often this is not available and instead, the sample standard deviation is used. The standard error is also a convenient measure of how precise our measurements are, that is how close a set of measurements are to each other.

### F.4 Covariance

---

If the variability of one variable,  $x$ , is influenced by another,  $y$ , then this dependence is measured using the covariance,  $Cov(x, y)$ . The covariance between two variables is defined by:

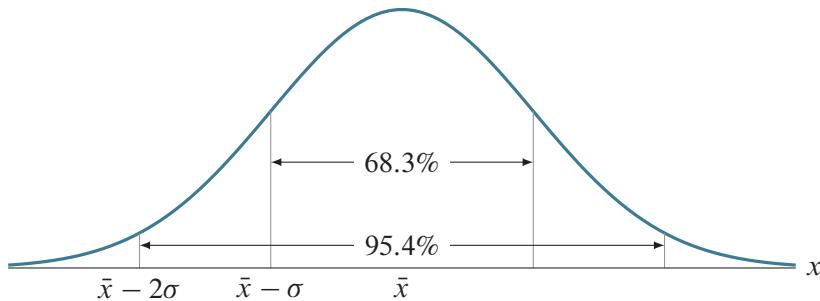
$$Cov(x, y) = \frac{1}{N} \sum [(x_i - \bar{x})(y_i - \bar{y})]$$

A positive covariance means that two variables are positively correlated. A covariance of zero means that two variables are statistically independent.

### F.5 Normal Distribution

---

The normal or Gaussian distribution is a continuous probability distribution that describes the probability of obtaining a given value,  $x$ , when the distribution has mean  $\mu$ , and standard deviation,  $\sigma$ . The probability in a normal distribution is described by the area under the curve such that the total area equals one. The mean corresponds to the peak of the curve (since it is symmetric) and the standard deviation to the width. If a random variable is known to be normally distributed, then the Gaussian curve tells us that there is a 68.3% chance that the value will lie within one standard deviation from the mean (Figure F.1).



**Figure F.1** Normal Distribution: The 68.3% and 95.4% intervals represent one and two standard deviations away from the mean,  $\bar{x}$ .

The equation that defines the Gaussian distribution is given by:

$$f(x) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

## F.6 z-Scores or Standard Scores

Any normal distribution can be standardized so that it has a mean of zero and a standard deviation of one, often denoted  $N(0, 1)$ . For example consider a random variable,  $X$ , with a value 5, that was drawn from a normal distribution with mean ( $\bar{x}$ ) equal to 10, and standard deviation equal to 2 ( $\sigma$ ), denoted  $N(10, 2)$ . We can shift the mean to zero by subtracting 10 and normalize the standard deviation by dividing by 2. The value just computed is called the z-score or standard score.

$$z = \frac{X - \bar{x}}{\sigma} = \frac{5 - 10}{2} = -2.5$$

The z-score tells us that the variate  $X$  is located 2.5 standard deviations to the *left* of the mean. We can therefore describe the z-score as a measure of the divergence of a random variable,  $X$ , from the mean, expressed in terms of the number of standard deviations.

## F.7 Null Hypothesis

The null hypothesis refers to the statement that is to be tested. In the literature the null hypothesis is often referred to by the symbol  $H_0$ . The null hypothesis is assumed to be the true hypothesis and statistical tests will often attempt to determine the probability that the null hypothesis is unlikely. If determined so, then the alternative hypothesis,  $H_1$  is accepted instead.

## F.8 $\chi^2$ Distribution

---

The chi-square distribution describes the distribution of variances drawn independently from a population of normally distributed variates. To be more precise, suppose there is a population that has a normally distributed random variable,  $X$ . The mean of this distribution will be  $\mu$  and the variance:

$$\sigma^2 = \frac{1}{N} \sum_{i=1}^N (X_i - \mu)^2$$

where  $N$  is the size of the population. Let us take a series of independent samples from the population and in each case form the squared standardized score:

$$z^2 = \frac{(X_i - \mu)^2}{\sigma^2}$$

We will call square of the score the standardized  $\chi^2_{(1)}$ , that is:

$$\chi^2_{(1)} = z^2$$

Values for  $\chi^2_{(1)}$  are positive due to the squaring. Since 68% of variances from a normal population will lie between standardized scores of -1 and 1, the bulk of sampled  $\chi^2_{(1)}$  values will also be between 0 and 1 (note the squaring eliminates the negative sign). The distribution of  $\chi^2_{(1)}$  is thus skewed. The distribution of  $\chi^2_{(1)}$  follows the  $\chi^2$  distribution with one degree of freedom.

Let's now consider the distribution of a sum of two independently sampled  $\chi^2_{(1)}$ :

$$\chi^2_{(2)} = \frac{(X_1 - \mu)^2}{\sigma^2} + \frac{(X_2 - \mu)^2}{\sigma^2} = z_1^2 + z_2^2$$

$\chi^2_{(2)}$  also follows a chi-square distribution but this time with two degrees of freedom. Because we are summing two standardized variances, the distribution is less skewed. We can continue this process and define  $\chi^2$  for any number of degrees of freedom. The function that relates the probability density to each value of  $\chi^2$  is given by:

$$f_{\chi^2}(x) = ce^{-x/2}x^{\nu/2-1}$$

where  $\nu$  is the degrees of freedom, and  $c$  is a constant given by the following expression:

$$c = \frac{1}{2^{\nu/2}\Gamma(\nu/2)}$$

where  $\Gamma(n)$  is the Gamma function:

$$\Gamma(n) = (n - 1)!$$

Of particular importance is that the mean of the chi-square distribution is  $\nu$  and the variance  $2\nu$ .

$$E(\chi_\nu^2) = \nu$$

$$\text{Var}(\chi_\nu^2) = 2\nu$$

In other words the  $\chi^2$  distribution is fully described by specifying the degrees of freedom.

## F.9 F-test

---

If two random variables,  $X$  and  $Y$ , have  $\chi^2$  distributions with  $\nu_1$  and  $\nu_2$  degrees of freedom, respectively, then the ratio:

$$F = \frac{X/\nu_1}{Y/\nu_2}$$

will be distributed according to the  $F$  distribution. The  $F$  distribution forms the basis of the  $F$ -test, which allows variances to be compared to determine whether they are significantly different or not. For example, fitting two different models will generate two different  $\chi^2$  values. We can propose the null hypotheses,  $H_0$ , that both  $\chi^2$  are identical.

The test involves computing the  $F$  ratio and looking up the value in an  $F$ -table. If the value falls below the 0.05 critical value, then we accept the null hypothesis. Any differences we see in the two  $\chi^2$  values could easily have come about by chance alone. If, however, the  $F$  value lies above the 0.05 critical value, we would propose that the observed difference in the  $\chi^2$  could not have come about by chance alone and likely represents a real difference.

## F.10 Confidence Intervals

---

Oftentimes we would like to know the likelihood that a given variable will fall within a specified range. That is, we would like some measure of confidence in an estimated value. If someone quoted the statistic that a variable,  $x$ , has a 95% confidence interval of  $x \pm \Delta x$ , that would mean that if we repeatedly measured this variable, 95% of the time the measured value would lie between  $x + \Delta x$  and  $x - \Delta x$ . If the distribution of  $x$  is normal, then the 95% interval is at  $1.96 \sigma$ . For example, if we know that a variable has a mean value of 2.5 and a standard deviation of 0.6, then the 95% confidence interval is given by:

$$\bar{x} \pm 1.96 \sigma = 2.5 \pm 1.96 \times 0.6 = 2.5 \pm 0.3$$

Therefore, if we were to take one more measurement, we could state that the value of the measurement will lie between 2.2 and 2.8, 95% of the time. We can also say that 1 in 20 (5%) of the time, the variable will lie outside this range by chance.

Alternatively, we could obtain an entire sample of measurements and compute the mean of the sample. With a new sample, what can we say about the likely value for the mean of

that sample? Given the original standard deviation, the mean of a new sample will have a confidence limit of:

$$\bar{x}_{95\%} \pm 1.96 \text{ SE}_{\bar{x}}$$

where  $\text{SE}_{\bar{x}}$  is the standard error. That is, the mean of the new sample will have a mean  $\pm$  the standard error. For example, imagine that the mean for a sample of nine data points is 4.0 with a standard deviation of 2.0. Given this information, the standard error can be computed to be:  $\text{SE}_{\bar{x}} = \sigma / \sqrt{n} = 2/3 = 0.66667$ . Therefore the 95% confidence interval on the mean is:

$$\bar{x}_{95\%} \pm 1.96 \times 0.66667 = 4.0 \pm 1.31$$

That is, if we draw a new sample, 95% of the time the mean will lie within the above range.

## F.11 Bootstrapping

---

Assume we wish to estimate the 95% confidence interval for the mean of a population. The problem is we don't have the population, only a sample from the population. We can use bootstrapping to get an estimate for the confidence interval, or more precisely we can use bootstrapping to generate a distribution that resembles the population from which we can estimate a confidence interval. **Bootstrapping** is the act of generating a distribution by sampling. Let us assume for argument sake that our sample from the population is:

$$4, 5, 7, 3, 7, 1$$

We will now resample with replacement from the original sample. Replacement means not removing the sampled value from the sample set, so it is possible to sample the same value again. An example of a bootstrap sample is:

$$7, 3, 1, 4, 1, 7$$

The new sample should have the same number of elements as the original sample. Let's say we create 200 samples in this way. For each sample we compute the mean so we have 200 means. This is our bootstrap sample of means.

At this point we can compute some interesting statistics from our 200 means. For example, what is the 95% confidence interval for the mean of the original population? If we assume our sample of means has the same statistical structure as the population, we can use the 200 means to compute the 95% confidence interval. To do this we must first rank the means in ascending order and then use the 97.5% and 2.5% percentiles as the interval, the middle 95% of all bootstrap sample means.

Listing F.1 shows Python code to bootstrap a sample of 30 values drawn from a normal distribution.

```
import numpy as np
```

```

import random, pylab

sampleSize = 30
s = np.random.normal(0.2, 1, sampleSize)

p = [] ; n = 10000
means = np.zeros (shape=n)
for k in range (n):
    l = np.zeros(shape=sampleSize)
    for i in range (sampleSize):
        r = random.randint (0, sampleSize-1)
        l[i] = s[r]
    p.append (l)
    means[k] = np.mean (l)
pylab.hist (means, 50)

```

**Listing F.1** Bootstrapping Script.

## F.12 Maximum Likelihood

---

To introduce maximum likelihood, consider the problem of estimating the probability,  $p$ , of getting a heads when flipping a coin. Let's say we flip a coin ten times and obtain the following result: HTHHTTHHHH where H represents heads and T tails. The probability of obtaining this sequence is related to  $p$ , which we can state as:

$$P(\text{HTHHTTHHHH}|p)$$

This reads:  $P$  is the probability of seeing the sequence HTHHTTHHHH given  $p$ , the probability of flipping a heads. The probability,  $P$ , can be computed using the AND rule, meaning what is the probability of obtaining a H and a T and a H, etc.? Given that the probability of throwing tails is  $(1 - p)$ , we obtain:

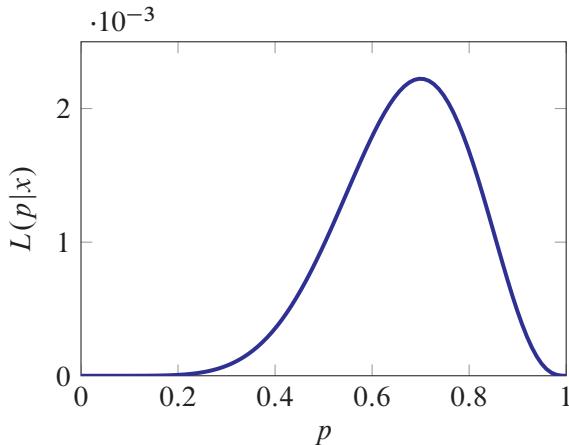
$$\begin{aligned} P(\text{HTHHTTHHHH}|p) &= p(1 - p)p(1 - p)(1 - p)p(1 - p)p \\ &= p^7(1 - p)^3 \end{aligned}$$

Recall the expression  $P(\text{HTHHTTHHHH}|p)$  reads: what is the probability of seeing this particular sequence of heads and tails given  $p$ ? However in the original question, we wanted to know what  $p$  was, *given* a sequence of coin throws. We should therefore ask is what is the *likelihood* of a particular value of  $p$  given the collected data. We should look at the equation as a function of  $p$  instead of what coin throw we see, that is:

$$L(p|x) = L(p|\text{HTHHTTHHHH}) = p^7(1 - p)^3$$

The expression  $L(p|x)$  reads: what is the likelihood of  $p$  given a set of heads and tails? If we vary  $p$ , we'll get different likelihoods, this is shown in Table F.1.

$p$	$L(p, x) = p^7(1 - p)^3$
0.1	$7.29 \times 10^{-8}$
0.2	$6.55 \times 10^{-6}$
0.3	$7.5 \times 10^{-5}$
0.4	0.000354
0.5	0.000977
0.6	0.00179
0.7	0.00222
0.8	0.00168
0.9	0.000478
1.0	0

**Table F.1** Likelihood calculation.**Figure F.2** Maximum Likelihood plot, see Table F.1.

We can see from the table (F.1) that the likelihood reaches a maximum at around 0.7. We therefore conclude that given the data HTHHTTHHHH, the most likely value for  $p$  is 0.7.

Things become more interesting if we generalize the previous analysis by assuming we make  $n$  flips of the coin, and we obtain  $x$  heads and therefore  $n - x$  tails. In this case the likelihood is:

$$L(p|x) = p^x(1 - p)^{n-x}$$

To maximize the likelihood, let us first take the log to make it easier to differentiate:

$$\ln(L(p)) = x \ln p + (n - x) \ln(1 - p)$$

Differentiating the log expression yields:

$$\frac{d \ln L(p)}{dp} = \frac{x}{p} - \frac{n - x}{1 - p}$$

which gives us the maximum. Setting this to zero, and solving for  $p$  gives the final result:

$$p = \frac{x}{n}$$

This tells us that the most likely value for  $p$  is simply equal to the fraction of heads we see in the sequence.

## Maximum Likelihood and Least Squares

Assume  $X$  is a continuous random variable whose probability density function is given by  $f(x)$  and depends on a parameter,  $p$ . If we carry out an experiment  $n$  times, we will obtain a sample of  $n$  numbers:

$$x_1, x_2, \dots, x_n$$

Assuming that the  $n$  random variables are independent, the probability the  $n$  data points will arise is given by the product of the individual probabilities:

$$f(x_1, \dots, x_n | p) = f(x_1) f(x_2) \dots f(x_n)$$

Or in terms of likelihood:

$$L(p | x_1, \dots, x_n) = f(x_1, \dots, x_n | p) = \prod_{i=1}^n f(x_i | p)$$

We now maximize the likelihood by differentiating with respect to  $p$ , set the derivative to zero, and solve for  $p$ :

$$\frac{\partial L}{\partial p} = 0$$

We can use this generalization to show that the sum of squares yields the maximum likelihood for the unknown parameters in the model. Assume that the experimental data is normally distributed with standard deviation,  $\sigma_i$ . Let there be  $n$  data points,  $(x_i, y_i)$ . The probability of making the observed measurement,  $y_i$ , given that  $y(x_i)$  is the model estimate, is given by the normal distribution:

$$P_i = \frac{1}{\sigma_i \sqrt{2\pi}} \exp\left(\left[-\frac{1}{2} \frac{y_i - y(x_i)}{\sigma_i}\right]^2\right)$$

The likelihood function is given by the product:  $\prod P_i$ , so that:

$$L(p) = \prod_{i=1}^n \left( \frac{1}{\sigma_i \sqrt{2\pi}} \right) \exp\left(-\frac{1}{2} \sum_{i=1}^n \left[ \frac{y_i - y(x_i)}{\sigma_i} \right]^2\right)$$

We now seek the maximum value for  $L(p)$ . We note that the first term is a constant while the exponential term is maximized when the sum in the exponential is minimized. The term in the exponential we wish to minimize is:

$$\sum_{i=1}^n \left[ \frac{y_i - y(x_i)}{\sigma_i} \right]^2$$

which is of course the  $\chi^2$  sums of squares. Therefore, the maximum likelihood is equivalent to minimizing the sum of squares. Equation (9.1) is therefore justified on more formal grounds.

---

## Further Reading

---

1. Bevington, PR (1969), Data reduction and error analysis for the physical sciences. McGraw-Hill.
2. Berendsen, HJC. (2011) A student's guide to data and error analysis. Cambridge: Cambridge University Press.
3. For something different: Freedman D, Pisani R, Purves R. Statistics (1998) Norton & Company; 3rd edition.
4. Mandel J. (1984) The statistical analysis of experimental data. Dover Publications.
5. Manly, BFJ. (1997) Randomization, Bootstrap and Monte Carlo Methods in Biology. Chapman & Hall.

**Table F.2** Standard Normal Probabilities, Negative:

Example: The area swept out from  $-\infty$  to  $-1$  standard deviations in a standard normal curve ( $\mu = 0; \sigma = 1$ ) is 0.159 units. The area swept out to  $-0.55$  is 0.291 units. Note that the area swept out from  $-\infty$  to  $0$  is 0.5 units, representing half the area of the normal curve. Data from <http://www.stat.tamu.edu/stat30x/zttables.php>

<b>z</b>	<b>0</b>	<b>0.01</b>	<b>0.02</b>	<b>0.03</b>	<b>0.04</b>	<b>0.05</b>	<b>0.06</b>	<b>0.07</b>	<b>0.08</b>	<b>0.09</b>
<b>-3.8</b>	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
<b>-3.7</b>	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
<b>-3.6</b>	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
<b>-3.5</b>	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
<b>-3.4</b>	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
<b>-3.3</b>	0.001	0.001	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000
<b>-3.2</b>	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
<b>-3.1</b>	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
<b>-3</b>	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
<b>-2.9</b>	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.001	0.001
<b>-2.8</b>	0.003	0.003	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.002
<b>-2.7</b>	0.004	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003
<b>-2.6</b>	0.005	0.005	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004
<b>-2.5</b>	0.006	0.006	0.006	0.006	0.006	0.005	0.005	0.005	0.005	0.005
<b>-2.4</b>	0.008	0.008	0.008	0.008	0.007	0.007	0.007	0.007	0.007	0.006
<b>-2.3</b>	0.011	0.010	0.010	0.010	0.010	0.009	0.009	0.009	0.009	0.008
<b>-2.2</b>	0.014	0.014	0.013	0.013	0.013	0.012	0.012	0.012	0.011	0.011
<b>-2.1</b>	0.018	0.017	0.017	0.017	0.016	0.016	0.015	0.015	0.015	0.014
<b>-2</b>	0.023	0.022	0.022	0.021	0.021	0.020	0.020	0.019	0.019	0.018
<b>-1.9</b>	0.029	0.028	0.027	0.027	0.026	0.026	0.025	0.024	0.024	0.023
<b>-1.8</b>	0.036	0.035	0.034	0.034	0.033	0.032	0.031	0.031	0.030	0.029
<b>-1.7</b>	0.045	0.044	0.043	0.042	0.041	0.040	0.039	0.038	0.038	0.037
<b>-1.6</b>	0.055	0.054	0.053	0.052	0.051	0.050	0.049	0.048	0.047	0.046
<b>-1.5</b>	0.067	0.066	0.064	0.063	0.062	0.061	0.059	0.058	0.057	0.056
<b>-1.4</b>	0.081	0.079	0.078	0.076	0.075	0.074	0.072	0.071	0.069	0.068
<b>-1.3</b>	0.097	0.095	0.093	0.092	0.090	0.089	0.087	0.085	0.084	0.082
<b>-1.2</b>	0.115	0.113	0.111	0.109	0.108	0.106	0.104	0.102	0.100	0.099
<b>-1.1</b>	0.136	0.134	0.131	0.129	0.127	0.125	0.123	0.121	0.119	0.117
<b>-1</b>	0.159	0.156	0.154	0.152	0.149	0.147	0.145	0.142	0.140	0.138
<b>-0.9</b>	0.184	0.181	0.179	0.176	0.174	0.171	0.169	0.166	0.164	0.161
<b>-0.8</b>	0.212	0.209	0.206	0.203	0.201	0.198	0.195	0.192	0.189	0.187
<b>-0.7</b>	0.242	0.239	0.236	0.233	0.230	0.227	0.224	0.221	0.218	0.215
<b>-0.6</b>	0.274	0.271	0.268	0.264	0.261	0.258	0.255	0.251	0.248	0.245
<b>-0.5</b>	0.309	0.305	0.302	0.298	0.295	0.291	0.288	0.284	0.281	0.278
<b>-0.4</b>	0.345	0.341	0.337	0.334	0.330	0.326	0.323	0.319	0.316	0.312
<b>-0.3</b>	0.382	0.378	0.375	0.371	0.367	0.363	0.359	0.356	0.352	0.348
<b>-0.2</b>	0.421	0.417	0.413	0.409	0.405	0.401	0.397	0.394	0.390	0.386
<b>-0.1</b>	0.460	0.456	0.452	0.448	0.444	0.440	0.436	0.433	0.429	0.425
<b>0</b>	0.500	0.496	0.492	0.488	0.484	0.480	0.476	0.472	0.468	0.464

**Table F.3** Standard Normal Probabilities, Positive:

# G

## *Modeling Standards and Databases*

### **G.1 Introduction**

---

The last 10 years have seen a significant increase in the number of simulation tools, all of which use different formats to store models. It was soon realized that some form of standardization for model exchange was necessary. As a result, two proposed standards emerged: CellML [65] and SBML [76] (Systems Biology Markup Language). CellML is primarily a notation for representing biochemical models in mathematical form. SBML on the other hand uses a biologically inspired notation to represent networks from which a mathematical model can be generated. Each standard has its strengths and weaknesses, but SBML has a simpler structure compared to CellML, and as a result has more software support. Many software tools support import and export of SBML. Both standards have very active communities with intracellular models being primarily the domain of SBML and for physiological models CellML. Here we focus on SBML.

### **SBML**

SBML is based on XML and closely follows the way existing modeling packages represent models. For example, SBML represents biochemical networks as a list of chemical transformations. It employs specific and different elements to represent spatial compartments, molecular species, and parameters. In addition, SBML also has a provision for rules which can be used to represent constraints, derived values, and general math.

SBML ([sbml.org](http://sbml.org)), like any standard, has evolved with time. Major revisions of the stan-

dard are captured in levels, while minor modifications and clarifications are captured in versions. An example of a major change within the standard would be the use of MathML in level two of SBML, whereas level one encoded infix (common algebra) strings to denote reaction rates and rules. The most recent level of SBML is level three where new functionality can be supported through extension packages.

## G.2 Graphical Layout

---

Graphical modeling applications [10] routinely enhance computational models by layout annotations. The SBML community devised a common standard on how to embed the layout information within SBML, called the Layout Extension. This extension [50] allows a model to store the size and dimension of all model elements, along with textual annotations and reactions. LibSBML has been modified to provide access to all elements of the Layout Extension. Several reference implementations also exist [10, 32].

In addition to the layout extension mentioned previously, the community has also introduced the Systems Biology Graphical Notation (SBGN) (<http://sbgn.org>) that aims to standardize the visual language of computational models. While this standard is still in development and, strictly speaking, is independent of the SBML effort, experience in other fields such as electrical engineering has demonstrated the essential need for standardizing the visual notation for representing models as diagrams.

## G.3 MIRIAM

---

Model Definition Languages such as SBML and CellML target the exchange of models. They aim to pass quantitative computational models from one software tool to another. Both communities agreed upon the Minimum Information Requested In the Annotation of biochemical Models (MIRIAM, [95]). These annotations make it possible to carry out searches for models with specific attributes in model repositories. This enables researchers to identify biological phenomena captured by a biochemical model and most importantly, it facilitates model reuse and composition.

In order for a model to be MIRIAM compliant, the model has to be encoded in a standard format such as SBML. Furthermore, it needs to be tied to a reference description, describing the properties and results obtained from the model. Parameters of the computational model have to be provided so it can be loaded into a simulation environment to reproduce the expected results. Other required information includes the model, the creator of the model, the date and time of the last modification, as well as a statement about terms of distribution.

## G.4 SBO – Systems Biology Ontology

In order to assign meaning to model constituents, an ontology specific to Systems Biology was developed: The Systems Biology Ontology (SBO, <http://www.ebi.ac.uk/sbo/>). It consists of five controlled vocabularies and two relationships: *is-part-of* and *is-a*. Qualifying model participants such as enzymes, macromolecules, metabolites, or small species such as ions, makes it easier to generate meaning from a model. It also makes the generation of standard visual notations such as SBGN possible. Moreover, it provides help on how to interpret the model computationally, as the SBO allows describing a model as continuous, discrete or logical. One could even go a step further, making explicit kinetic rate laws in a model obsolete. This could be done by referencing the appropriate ontology identifier (e.g. tagging a reaction as following Henri-Michaelis Menten enzyme kinetics and specifying the parameters). The SBO is community driven and new terms or modifications to the existing ontology can be requested by the community.

## G.5 Other Ontologies and Formats

The most recent developments in the CellML and SBML communities revolve around the creation of ontologies and refining exchange semantics. Apart from classifying model constituents with an appropriate ontology, one current area of interest is describing the dynamical behavior of a model. The “Terminology for the Description of Dynamics” (TEDDY,<sup>1</sup>) provides a rich ontology to describe and quantify what kinds of behavior a computational model can exhibit (e.g. the characteristics of a model could describe bifurcation behavior whereas the functionality of a model could be described as oscillations or switch behavior). However, knowing that a model exhibits interesting behavior is not usually enough. More information is needed in order to recreate that behavior. The “Minimum Information About a Simulation Experiment” (MIASE,<sup>2</sup>) project focuses on this problem. MIASE helps to describe the simulation algorithms and the simulation tools used along with all needed parameter settings. Towards this end, one can use the Kinetic Algorithm Ontology (KiSAO) which relates simulation algorithms and methods to each other. As these ontologies are still currently under development, it will be interesting to see how they evolve and are adopted by the community.

Lastly, we should mention BioPAX [101] which stands for Biological Pathway Exchange. BioPAX is an XML based format that will act as a bridge between different pathway databases and data. In relation to modeling software, BioPAX may offer a means to embed rich annotation data into an SBML or CellML model. Some of this capability is being addressed to a limited extent by the new ontologies being developed at EBI in Cambridge, UK. However, BioPAX may offer a useful complementary way to bind pathway data to

<sup>1</sup><http://www.ebi.ac.uk/compneur-srv/teddy/>

<sup>2</sup><http://www.ebi.ac.uk/compneur-srv/miase/>

computational models.

## G.6 Human Readable Formats

---

SBML and CellML are formats that use XML to represent information. One advantage to using XML is that there is much software available to assist in reading and manipulating XML based data. However, XML is not suited for human consumption; it is designed strictly to be read by computer software. In order for humans to build and read models, human-readable formats are required. Often these are text-based or graphical in nature. With respect to text based formats, there has been a long tradition to using human readable formats for representing biochemical models, starting with BIOSSIM [49]. Other examples of early human readable formats include works by Park [132] and Burns [20]. In recent years simulators such as SCAMP [152] and METAMOD [74] also introduced human-readable formats to define models. Both software tools were subsequently developed into Jarnac and PySCeS, respectively.

Other formats of interest include languages that support modular development of models by Smith and Sauro [165] at the University of Washington (called Antimony), and Michael Pederson at the University of Edinburgh [134]. Blinov, Faeder, Goldstein and Hlavacek developed BioNetGen [15] which is a rules based format for representing systems with multiple states. Cyto-Sim incorporates an interesting human-readable language for representing biochemical systems. The SBML community [184] has also developed a human-readable script called SBML-shorthand. This notation maps directly onto SBML but is much easier to hand write SBML. The shorthand is also much less verbose and uses infix to represent expressions rather than MathML. Finally, we should mention a Lisp based language called little b (<http://www.littleb.org/>) being developed at Harvard University. The aim of little b is to allow biologists to build models quickly and easily from shared parts.

## G.7 Databases

---

Along with the standardization of model representation, there has been a desire to create model repositories where models published in journals can be stored and retrieved. There are currently five repositories with varying degrees of quality and usability. The most promising is the UK based BioModels Database, which at the current time (July 2013) holds over nine hundred and sixty three curated and working models that can be downloaded in standard SBML and other formats. BioModels also has the great benefit of providing programmatic access to its database via web services, which allows any software program to access the database seamlessly across the internet. Models stored in the BioModels Database are curated, meaning that models will reproduce the author's original intention. In addition, the models are liberally annotated, so model components can be referenced from other database sources.

Another large database has been assembled by the CellML community [98]. From their site one can convert the CellML into standard C code for compilation into a working model.

The JSim group at the University of Washington has a large database of physiological models <http://nsr.bioeng.washington.edu/Models/> stored in the mathematical language used by the JSim simulation application.

Another useful database is the JWS online database developed by Brett Olivier and Jacky Snoep [126] which has over seventy working models. JWS allows export in both SBML and the script format PySCeS which can be easily translated to other formats such as Jarnac script.

Another database called, DOQCS <http://doqcs.ncbs.res.in/> focuses on signaling networks and contains over two hundred models. Models in DOQCS can only be downloaded in Genesis format [13] however, which limits portability to other frameworks.



# H

## *Modeling with Python*

This appendix provides a brief description of the Python programming language, the Antimony reaction network format, and libRoadRunner.

**Python** Python is an easy to learn general, purpose interactive programming language. It has similar usability characteristics to Matlab or Basic. As such it is a good language to use for doing pathway simulations and is easily learned by new users. In recent years Python has also become widely used as a general purpose scientific programming language and now supports many useful libraries and tools for modelers. All the scripts we provide in this book are written in Python.

**Antimony** SBML has become a de facto standard for exchanging models of biological pathways. Any tool we use should support SBML. However SBML is a computer readable language and is not easy for humans to read or write. Instead more human readable formats have been developed. This text book uses the Antimony pathway description language [165]. Models can be described in Antimony then converted to SBML and vice versa.

**libRoadRunner** To support SBML from within Python, we developed a C/C++ simulation library called libRoadRunner [166] that can read and run models based on SBML. In order to use libRoadRunner within Python, we also provide a Python interface that makes it easy to carry out simulations with Python.

**Spyder** Integration of the various tools including Python is achieved using spyder2 (<https://code.google.com/p/spyderlib/>). Spyder2 offers a Matlab like experience in a friendly, cross-platform environment.

## H.1 Introduction to Python

---

One great advantage of the Python language is that it runs on many computer platforms including Windows Mac and Linux. Python can be freely downloadable from the Python web site ([python.org](http://python.org)). To execute Python code we will need use a Python IDE (Integrated Development Environment). In the Python world there are many IDEs to choose from, ranging from very simple consoles to sophisticated development systems that include documentation, debuggers and other visual aids. In this book we use the spyder2 cross-platform IDE. (<https://code.google.com/p/spyderlib/>).

To make things even easier we have packaged up everything you need into a distribution we call Tellurium. The version does not interfere with any existing Python you might have installed.

The best way to learn Python is to download a copy and start using it. We have prepared installers that install all the components you need. These can be found at [tellurium.analogmachine.org](http://tellurium.analogmachine.org). The Tellurium distribution includes some additional helper routines which can make life easier for new users. To get your copy of Tellurium go to the web site [tellurium.analogmachine.org](http://tellurium.analogmachine.org), and click on the link called *Downloads*. For windows, download and run the Windows installer. For the Mac and Linux versions there are additional instructions that include downloading Anaconda and installing Tellurium via pip install tellurium. Refer to the web site [tellurium.analogmachine.org](http://tellurium.analogmachine.org) for detailed instructions.

One Windows, once Tellurium is installed, go to the start menu, find Tellurium and select the application call Tellurium spyder. If successful you should see something like the screen shot in Figure H.1, but without the plotting window. The screen-shot shows three important elements, on the left we see an editor: where models can be edited. On the lower right is the Python console, where Python commands can be entered. At the top right we show plotting window that illustrates some output from a simulation. For those familiar with IPython, the latest version of spyder2 supports the IPython console directly.

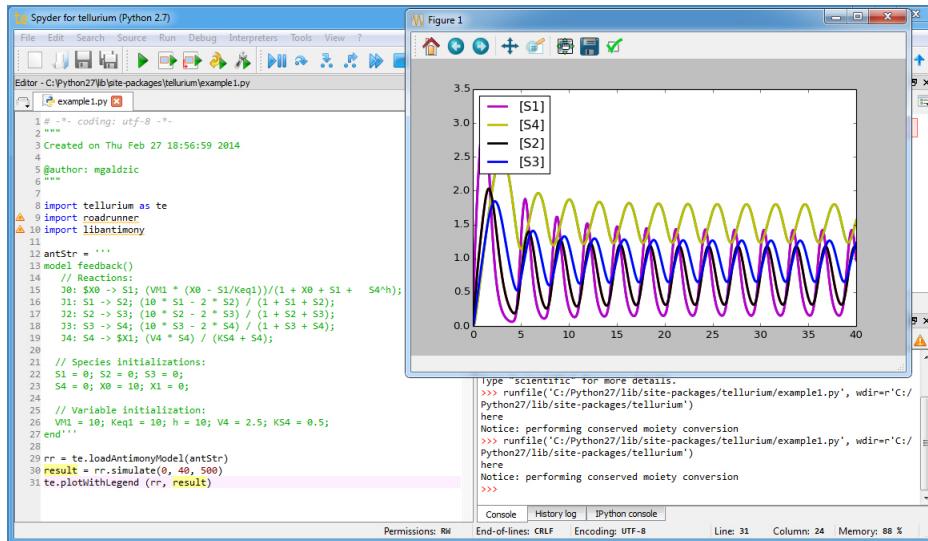
Start the Tellurium IDE, and focus on the Python console at the bottom right of the application. A screen-shot of the console is shown in Figure H.2.

The >>> symbol marks the place where you can type commands. The following examples are based on Python 2.7. To add two numbers, say  $2 + 5$ , we would type the following:

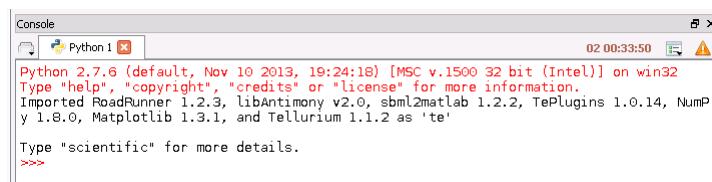
```
>>> print 2 + 5
7
>>>
```

**Listing H.1** Simple Arithmetic

Just like Matlab or Basic we can assign values to variables and use those variables in other calculations:



**Figure H.1** Screen-shot of Tellurium, showing editor on the left, Python console bottom right and plotting window top-right.



**Figure H.2** Screen-shot of Tellurium, focusing on the Python console.

```
>>> a = 2
>>> b = 5
>>> c = a + b
>>> print c
7
>>>
```

### Listing H.2 Assigning values to variables

The types of values we can assign to variables include integers, floating point numbers, Booleans (True or False), strings and complex numbers.

```
>>> a = 2
>>> b = 3.1415
>>> c = False
>>> d = "Hello Python"
```

```
>>> e = 3 + 6j  
>>>
```

### Listing H.3 Different kinds of values

Many functions in Python are accessible via modules. For example to compute the sine of a number we can't simply type `sin (30)`. Instead we must first load the math module. We can then call the sin function:

```
>>> import math  
>>> print sin (3.1415)  
9.265358966049026e-05  
>>>
```

### Listing H.4 Importing modules (libraries) into Python

In Tellurium, we preload some libraries including the math library.

## Repeating Calculations

One of the commonest operations in computer programming is iteration. We can illustrate this with a simple example that loops ten times, each time printing out the loop index. This example will allow us to introduce the IDE editor. The editor is the panel on the left side of the IDE. In the editor we can type Python code, for example we could type:

```
a = 4.0  
b = 8.0  
c = a/b  
print "The answer is:", c
```

### Listing H.5 Writing a simple program in the IDE editor

When we've finished typing this in the editor window, we can save our little program to a file (Select Menu: File/Save As...) and run the program by clicking on the green arrow in the tool bar of the IDE (Figure H.3). If we run this program we will see:

```
The answer is: 0.5  
>>>
```

### Listing H.6 Writing a simple program in the IDE editor

The IDE allows a user to have many program files open at once, each program file is given its own tab so that it is easy to move from one to another. This is useful if one is working on multiple models at the same time.



**Figure H.3** Screen-shot of Tellurium, focusing on the Toolbar with the run button circled.

We will now use the editor to write the simple program that loops ten times, shown in Figure H.7:

```
for i in range (10):
    print i,
```

**Listing H.7** A simple loop in python

This will generate the sequence:

```
0 1 2 3 4 5 6 7 8 9
```

**Listing H.8** Result from simple loop program

There are a number of new concepts introduced in this small looping program. The first line contains the `for` keyword, can be translated into literal English as “for all elements in a list, do this”. The list is generated from the `range()` function and in this case generates a list of 10 numbers starting at 0. `i` is the loop index and within the loop, `i` can be used in other calculations. In this case, we will just print the value of `i` to the console. Each time the program loops it extracts the next value from the list and assigns it to `i`.

Two things are important to note in the print line. The first and most important is that the line has been indented four spaces. This isn’t just for aesthetic reasons but is functional. It tells Python what code should be executed *within* the loop. To elaborate we could add more lines to the loop, such as:

```
for i in range (10):
    a = i
    b = a*2
    print b,
print "Finished Loop"
```

**Listing H.9** A simple loop illustrating multiple statements

In the example shown in Figure H.9 has three indented lines. The indents means that these three lines will be executed within the loop. The last line which prints a message, is not indented and therefore will not be executed within the loop. This means we only see the message appear once, right at the end. The output for this little program is shown below.

```
0 2 4 6 8 10 12 14 16 18 Finished Loop
```

Another important point worth noting is the use of the `,` after the loop print statement. The comma is used to suppress a newline. This is why the output appears on one line only. If we had left out the comma, each print statement would be on its own line.

A final word about `range()`. Range takes up to three arguments. In the example we only gave one argument, 10. A single argument means: create a list starting at zero, incrementing one for each item until the incremented value reaches 10. A second argument such as `range(5, 10)` means start the list at 5 rather than zero. Finally, a third argument can be used to specify the increment size. For example the command `range(1, 10, 2)` yields the list:

```
[1, 3, 5, 7, 9]
```

The easiest way to try out the various options in `range` is to type them at the console to get immediate feedback.

The use of variables, printing results, importing libraries and looping are probably the minimum concepts one needs to start using Python. However there are a huge range of resources online to help learn Python. Of particular interest is the codecademy web site (<http://www.codecademy.com/>). This site offers an interactive means to learn Python and other programming languages.

## H.2 Describing Reaction Networks using Antimony

---

The code shown in the panel below illustrates the description of a very simple model using the Antimony syntax [165] followed by two lines of Python that uses libRoadRunner to run a simulation of the model. This section briefly describes Antimony syntax. A more detailed description of Antimony can be found at <http://antimony.sourceforge.net/index.html>.

```
import tellurium as te

r = te.loada ('''
S1 -> S2; k1*S1;
S1 = 10; k1 = 0.1
''')

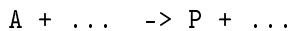
r.simulate (0, 50, 100)
r.plot()
```

**Listing H.10** Simple model Antimony and simulated using libRoadRunner

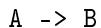
The main purpose of Antimony is to make it simple to specify complex reaction networks

using a familiar chemical-reaction notation.

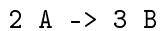
A chemical reaction can be an enzyme catalyzed reaction, a binding reaction, a phosphorylation, a gene expressing a protein or any chemical process that results in the conversion of one of more species (reactants) to a set of one or more other species (products). In Antimony, reactions are described using the notation:



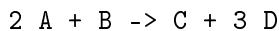
where the reactants are on the left side and products on the right side. The left and right are separated by the  $\rightarrow$  symbol. For example:



describes the conversion of reactant A into product B. In this case one molecule of A is converted to one molecule of B. The following example shows non-unity stoichiometry:

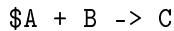


which means that two molecules of A react to form three molecules of B. Bimolecular and other combinations can be specified using the + symbol, that is:



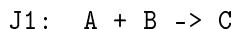
tells us that two molecules of A combine with one molecule of B to form one molecule of C and three molecules of D.

To specify species that do not change in time (boundary species), add a dollar character in front of the name, for example:

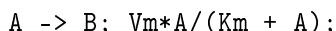


means that during a simulation, the concentration of A is fixed.

Reactions can be named using the syntax J1:. For example:



means the reaction has a name, J1. Named reaction are useful if you want to refer to the flux of the reaction; kinetic rate laws come immediately after the reaction specification. If only the stoichiometry matrix is required, it is not necessary to enter a full kinetic law, a simple  $\dots \rightarrow S1; v;$  is sufficient. Here is an example of a reaction that is governed by a Michaelis-Menten rate law:



Note the semicolons. Here is a more complex example involving multiple reactions:

```
MainFeed: $X0 -> S1; Vm*X0/(Km + X0);
TopBranch: S1 -> $X1; Vm1*S1/(Km1 + S1);
BottomBranch: S1 -> $X2; Vm2*S1/(Km2 + S1);
```

There is no need to pre-declare the species names shown in the reactions or the parameters in the kinetic rate laws. Strictly speaking, declaring the names of the floating species is optional. However, this feature is for more advanced users who wish to define the order

of rows that will appear in the stoichiometry matrix. Normal use need not pre-declare the species names. To pre-declare parameters and variables see the example below:

```
const Xo, X1, X2; // Boundary species
var S1;           // Floating species

MainFeed:   $X0 -> S1; Vm*X0/(Km + X0);
TopBranch:  S1 -> $X1; Vm1*S1/(Km1 + S1);
BottomBranch: S1 -> $X2; Vm2*S1/(Km2 + S1);
```

We can load an Antimony model into libRoadRunner using the short-cut command `loada`. For example:

```
r = te.loada ('''
    const Xo, X1, X2; // Boundary species
    var S1;           // Floating species

    MainFeed:   $X0 -> S1; Vm*X0/(Km + X0);
    TopBranch:  S1 -> $X1; Vm1*S1/(Km1 + S1);
    BottomBranch: S1 -> $X2; Vm2*S1/(Km2 + S1);
''')
```

To reference model properties and methods, the property or method must be preceded with the roadrunner variable. e.g. `r.S1 = 2.3;`

When loaded into libRoadRunner the model will be converted into a set of differential equations. For example, the model:

```
$Xo -> S1; v1;
S1 -> S2; v2;
S2 -> $X1; v3;
```

will be converted into:

$$\begin{aligned}\frac{dS_1}{dt} &= v_1 - v_2 \\ \frac{dS_2}{dt} &= v_2 - v_1\end{aligned}$$

Note that there are no differential equations for  $X_o$  and  $X_1$ , because they are fixed and do not change in time. If the reactions have non-unity stoichiometry, this is taken into account when the differential equations are derived.

### H.2.1 Initialization of Model Values

To initialize the concentrations and parameters in a model, we can add assignments after the network is declared, for example:

```
MainFeed:    $X0 -> S1; Vm*X0/(Km + X0);
TopBranch:   S1 -> $X1; Vm1*S1/(Km1 + S1);
BottomBranch: S1 -> $X2; Vm2*S1/(Km2 + S1);

X0 = 3.4;  X1 = 0.0;
S1 = 0.1;
Vm = 12; p.Km = 0.1;
Vm1 = 14; p.Km1 = 0.4;
Vm2 = 16; p.Km2 = 3.4;
```

## H.3 Using libRoadRunner in Python

libRoadRunner is a high-performance simulator [166] that can simulate models described using SBML. In order to use Antimony with libRoadRunner it is necessary to first convert an Antimony description into SBML and then load the SBML into libRoadRunner. Tel-luirum provides a handy routine called `loadAntimonyModel` to help with this task (The short-cut name is `loada`). To load an Antimony model we first assign an Antimony description to a string variable, for example:

```
model = '''
S1 -> S2; k1*S1;

S1 = 10; k1 = 0.1;
'''
```

We now use the `loadAntimonyModel` (`model`) or `loada` to load the model into libRoadRunner.

```
>>> r = te.loadAntimonyModel (model)
```

**Listing H.11** Loading an Antimony model

In this book we generally use the short-cut command as follows:

```
r = te.loada ('''
S1 -> S2; k1*S1;

S1 = 10; k1 = 0.1;'''
```

```
    ...)  
>>>
```

### **Listing H.12** Loading an Antimony model using the short-cut command

Note that `loadAntimonyModel` and `loada` are part of the Tellurium Python package supplied with the Tellurium installer. If the Tellurium packages hasn't been loaded, use the following command to load the Tellurium package:

```
>>> import tellurium as te
```

### **Listing H.13** Importing the Tellurium Package

#### H.3.1 Time Course Simulation

Once a model has been loaded into `libRoadRunner`, performing a simulation is straight forward. To simulate a model, we use the `libRoadRunner simulate` method. This method has many options but for everyday use, four options will suffice. The following panel illustrates a number examples of how to use `simulate`.

```
>>> result = r.simulate ()  
>>> result = r.simulate (0, 10)  
>>> result = r.simulate (0, 10, 100)  
>>> result = r.simulate (0, 10, 100, ['time', 'S1'])
```

### **Listing H.14** Calling the simulate method

Argument	Description
1st	Start Time
2nd	End Time
3rd	Number of Points
4th	Selection List

Let us focus on the forth version of the `simulate` method that takes four arguments. This call will run a time-course simulation starting at time zero, ending at time 10 units, and generating 100 points. The results of the run are deposited in the matrix variable, `result`. At the end of the run, the `result` matrix will contain columns corresponding to the time column and all the species concentrations as specified by the forth argument. The forth argument can be used to change the columns that are returned from the `simulate` method. For example:

```
>>> result = r.simulate (0, 10, 1000, ['S1'])
```

will return a matrix 1,000 rows deep and one column wide that corresponds to the level of species S1.

Note that the special variable `Time` is available and represents the independent variable ‘time’ in the model.

Finally we plot the results.

```
result = r.simulate (0, 10, 1000, ['Time', 'S1', 'J1', 'J2', 'J3']);  
r.plot()
```

or if we are not interested in the result data itself we can use the libRoadRunner plot:

It is possible to set the output column selections separately using the command:

```
r.selections = ['time', 'S1']
```

This can save some typing each time a simulation needs to be carried out. By default the selection is set to time as the first column followed by all molecular species concentrations. As such it is more common to simply enter the command:

```
>>> result = r.simulate (0, 10, 50)
```

In fact, even the start time and end time and number of points are optional and if missing, `simulate` will revert to its defaults.

```
>>> result = r.simulate()
```

### H.3.2 Plotting Simulation Results

Tellurium comes with Matplotlib, which is a common plotting package used by many Python users. To simplify its use we provide two simple plotting calls:

```
te.plot (array)  
te.plotWithLegend (rr, array)
```

The first takes the array generated by a call to `simulate` and uses the first column as the *x* axis and all subsequent columns as *y* axis data. The second call takes the roadrunner variable as well as the array and does the same kind of plot but this time adds a legend to the plot. We will use the first plotting command in the next section where we merge multiple simulations together.

### H.3.3 Applying Perturbations to a Simulation

Often in a simulation, we may wish to perturb a species or parameter at some point during the simulation and observe what happens. One way to do this in Tellurium is to carry out two separate simulations where a perturbation is made in between the two simulations. For example, let's say we wish to perturb the species concentration for a simple two step pathway and watch the perturbation decay. First, we simulate the model for 10 time units; this gives us a transient and then a steady state.

```
import numpy # Required for vstack
import tellurium as te

r = te.loada '''
$X0 -> S1; k1*X0;
S1 -> $X1; k2*S1;

X0 = 10; k1 = 0.3; k2 = 0.15;
'''

m1 = r.simulate (0, 40, 50)
```

We then make a perturbation in S1 as follows:

```
r.S1 = r.S1 * 1.6
```

which increases S1 by 60%. We next carry out a second simulation:

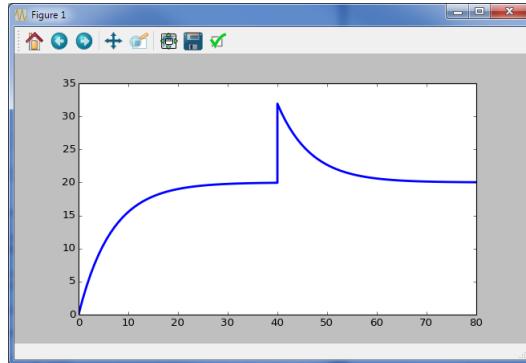
```
m2 = r.simulate (40, 80, 50)
```

Note that we set the time start of the second simulation to the end time of the first simulation. Once we have the two simulations we can combine the matrices from both simulations using the Python command `vstack`

```
% Merge the two result array together
m = numpy.vstack ((m1, m2))
```

Finally, we plot the results, screen-shot shown in Figure H.4.

```
te.plotArray (m)
```



**Figure H.4** Screen-shot from Matplotlib showing effect of perturbation in S1.

#### H.3.4 Steady State and Metabolic Control

To evaluate the steady-state first make sure the model values have been previously initialized, then enter the following statement at the console.

```
>>> r.getSteadyState()
```

This statement will attempt to compute the steady state and return a value indicating how effective the computation was. It returns the norm of the rate of change vector (i.e.  $\sqrt{\sum(dydt^2)}$ ). The closer this is to zero, the better the approximation to the steady state. Anything less than  $10^{-4}$  usually indicates that a steady state has been found.

Once a steady state has been evaluated, the values of the metabolites will be at their steady state values, thus S1 will equal the steady state concentration of S1.

The fluxes through the individual reactions can be obtained by either referencing the name of the reaction (e.g. J1), or via the short-cut command rv. The advantage to looking at the reaction rate vector is that the individual reaction fluxes can be accessed by indexing the vector (see example below). **Note that indexing is from zero.**

```
>>> print r.J1, r.J2, r.J3
3.4, ...
>>> for i in range (0, 2):
...     print r.rv()[i]
3.4
etc
->
```

The following commands are used to compute the various coefficients defined in metabolic control analysis [83, 78]. To compute control coefficients use the statement:

getCC (Dependent Measure, Independent parameter)

The dependent measure is an expression usually containing flux and metabolite references, for example, S1, J1. The independent parameter must be a simple parameter such as a Vmax, Km, ki, boundary metabolite (X0), or a conservation total such as cm\_xxxx. Examples include:

```
r.getCC ('J1', 'Vmax1')
r.getCC ('J1', 'Vm1') + rr.getCC ('J1', 'Vm2')
r.getCC ('J1', 'X0')
r.getCC ('J1', 'cm_xxxx')
```

To compute elasticity coefficients use the statement:

```
getEE (Reaction Name, Parameter Name)
```

For example:

```
r.getEE ('J1', 'X0')
r.getEE ('J1', 'S1')
```

Since getCC and getEE are built-in functions, they can be used alone or as part of larger expressions. Thus, it is easy to show that the response coefficient is the product of a control coefficient and the adjacent elasticity by using:

```
R = r.getCC ('J1', 'X0')
print R - r.getCC ('J1', 'Vm') * r.getEE ('J1', 'X0')
```

To obtain the conservation matrix for a model use the model method, getConservationMatrix. Note that in the Antimony text we use the var word to predeclare the species so that we can set up the rows of the stoichiometry matrix in a certain order if we wish. This allows us to obtain conservation matrices with only positive terms.

```
import tellurium as te

r = te.loada '''
var ES, S1, S2, E;

J1: E + S1 -> ES; v;
J2: ES -> E + S2; v;
J3: S2 -> S1; v;
'''

print r.getConservationMatrix()
print r.fs()

# Output
[[ 1.  1.  1.  0.]
 [ 1.  0.  0.  1.]]
```

```
['ES', 'S1', 'S2', 'E']
```

The result given above indicates that the conservation relations,  $ES + S1 + E$  and  $E + ES$  exist in the model. As a result, Tellurium would generate two internal parameters of the form `cm` corresponding to the two relations.

### H.3.5 Other Model Properties of Interest

There are a number of predefined objects associated with a reaction network model which might also be of interest. For example, the stoichiometry matrix, `sm`, the rate vector `rv`, the species levels vector and `dv` which returns the rates of change.

```
print r.sm()
print r.rv()
print r.sv()
print r.dv()
```

The names for the parameters and variables in a model can be obtained using the short-cuts:

```
print r.fs() # List of floating species names
print r.bv() # List of boundary species names
print r.ps() # List of parameter names
print r.rs() # List of reaction names
print r.vs() $ List of compartment names
```

The Jacobian matrix can be returned using the command: `r.getFullJacobian()`.

## H.4 Generating SBML and Matlab Files

---

Tellurium can import and export standard SBML [76] as well as export Matlab scripts for the current model. To load a model in SBML, load it directly into libRoadRunner. For example:

```
>>> r = roadrunner.RoadRunner ('mymodel.xml')
>>> result = r.simulate (0, 10, 100)
```

There are two ways to retrieve the SBML, one can either retrieve the original SBML loaded using `r.getSBML()` or retrieve the *current* SBML using `r.getCurrentSBML()`. Retrieving the current SBML can be useful if the model has been changed. To save the SBML to a file we can use the Tellurium helper function `saveToFile ()`, for example:

```
>>> te.saveToFile ('mySBMLModel.xml', r.getCurrentSBML())
```

To convert an SBML file into Matlab, use the `getMatlab` method:

```
import tellurium as te

r = te.loada '''
S1 -> S2; k1*S1;
S2 -> S3; k2*S2;
S1 = 10; k1 = 0.1; k2 = 0.2;
'''

# Save the SBML
te.saveToFile ('model.xml', r.getSBML())

# Save the Matlab
te.saveToFile ('model.mat', r.getMatlab())
```

## H.5 Exercise

Figure H.5 shows a two-gene circuit with a feedforward loop. Assume the following rate laws for the four reactions:

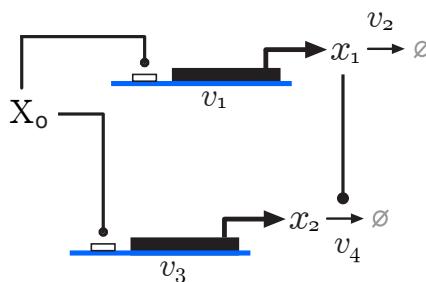
$$v_1 = k_1 X_o$$

$$v_2 = k_2 x_1$$

$$v_3 = k_3 X_o$$

$$v_4 = k_4 x_1 x_2$$

Assume that all rate constants are equal to one and that  $X_o = 1$ . Assume  $X_o$  is a fixed species.



**Figure H.5** Two-gene circuit with feedforward loop.

1. Use Tellurium to model this system.
2. Run a simulation of the system from 0 to 10 time units.
3. Next, change the value of  $X_o$  to 2 (double it) and rerun the simulation for another 10 time units from where you left off in the last simulation. Combine both simulations and plot the result, that is time on the x-axis, and  $X_o$  and  $x_2$  on the y-axis.
4. What do you see?
5. Write out the differential equations for  $x_1$  and  $x_2$ .
6. Show algebraically that the steady state level of  $x_2$  is independent of  $X_o$ .

This copy belongs to Eric Freeman

# Bibliography

- [1] Alberty, R. A. *Thermodynamics of biochemical reactions*. Wiley. com, 2005.
- [2] Aldrich, J. “RA Fisher and the making of maximum likelihood 1912-1922”. In: *Statistical Science* 12.3 (1997), pp. 162–176.
- [3] Alon, U. *An Introduction to Systems Biology: Design Principles of Biological Circuits (Chapman & Hall/Crc Mathematical and Computational Biology Series)*. Chapman & Hall/CRC, 2006.
- [4] Aparicio, O. et al. “Chromatin Immunoprecipitation for Determining the Association of Proteins with Specific Genomic Sequences In Vivo”. In: *Current Protocols in Molecular Biology* 69.1 (2004), pp. 21.3.1–21.3.33.
- [5] Barabási, A. L. and Oltvai, Z. N. “Network biology: understanding the cell’s functional organization”. In: *Nat Rev Genet* 5.2 (2004), pp. 101–113.
- [6] Barabasi, A. *Linked*. Plume, 2003.
- [7] Baralla, A., Mentzen, W. I., and De La Fuente, A. “Inferring gene networks: dream or nightmare?” In: *Annals of the New York Academy of Sciences* 1158.1 (2009), pp. 246–256.
- [8] Barnett, V. and Lewis, T. *Outliers in statistical data*. Vol. 3. Wiley New York, 1994.
- [9] Bennett, B. D. et al. “Absolute metabolite concentrations and implied enzyme active site occupancy in Escherichia coli”. In: *Nature chemical biology* 5.8 (2009), pp. 593–599.
- [10] Bergmann, F. T., Vallabhajosyula, R. R., and Sauro, H. M. “Computational tools for modeling protein networks”. In: *Current Proteomics* 3.3 (2006), pp. 181–197.
- [11] SBW-a modular framework for systems biology. Winter Simulation Conference. WSC ’06 Proceedings of the 38th conference on Winter simulation, 2006, pp. 1637–1645.
- [12] Bevington, P. R. and Robinson, D. K. *Data reduction and error analysis for the physical sciences*. Vol. 2. McGraw-Hill New York, 1969.
- [13] Bhalla, U. S. “Use of Kinetikit and GENESIS for modeling signaling pathways.” In: *Methods Enzymol* 345 (2002), pp. 3–23.
- [14] Biondi, E. G. et al. “Regulation of the bacterial cell cycle by an integrated genetic circuit.” In: *Nature* 444.7121 (2006), pp. 899–904.

- [15] Blinov, M. L. et al. “BioNetGen: software for rule-based modeling of signal transduction based on the interactions of molecular domains”. In: *Bioinformatics* 20.17 (2004), pp. 3289–3291.
- [16] Bode, A. M. and Dong, Z. “Post-translational modification of p53 in tumorigenesis.” In: *Nat Rev Cancer* 4.10 (2004), pp. 793–805.
- [17] Brett, D. et al. “Alternative splicing and genome complexity”. In: *Nature genetics* 30.1 (2002), pp. 29–30.
- [18] Brilli, M. et al. “The diversity and evolution of cell cycle regulation in alpha-proteobacteria: a comparative genomic analysis”. In: *BMC Systems Biology* 4.1 (2010), p. 52.
- [19] Burns, J. “Steady states of general multi-enzyme networks and their associated properties. Computational approaches.” In: *FEBS Lett* 2 Suppl 1 (1969), S30–S33.
- [20] Burns, J. A. “Studies on Complex Enzyme Systems”. <http://www.sys-bio.org/jim-burns-thesis/>. PhD thesis. University of Edinburgh, 1971.
- [21] Cai, L., Friedman, N., and Xie, X. S. “Stochastic protein expression in individual cells at the single molecule level”. In: *Nature* 440.7082 (2006), pp. 358–362.
- [22] Cao, Y., Gillespie, D., and Petzold, L. “Accelerated stochastic simulation of the stiff enzyme-substrate reaction”. In: *Journal of Chemical Physics* 123.14 (2005).
- [23] Cash, J. R. and Karp, A. H. “A variable order Runge-Kutta method for initial value problems with rapidly varying right-hand sides”. In: *ACM Trans. Math. Softw.* 16.3 (Sept. 1990), pp. 201–222.
- [24] Chance, B. “The Kinetics of the Enzyme-Substrate Compound of Peroxidase”. In: *Journal of Biological Chemistry* 151.2 (1943), pp. 553–577.
- [25] Chen, K. C. et al. “Integrative analysis of cell cycle control in budding yeast”. In: *Molecular biology of the cell* 15.8 (2004), pp. 3841–3862.
- [26] Chickarmane, V., Kholodenko, B. N., and Sauro, H. M. “Oscillatory dynamics arising from competitive inhibition and multisite phosphorylation”. In: *J Theor Biol* 244.1 (2007), pp. 68–76.
- [27] Chickarmane, V. et al. “Transcriptional dynamics of the embryonic stem cell switch”. In: *PLoS Comput Biol* 2.9 (2006).
- [28] Chylek, L. A. et al. “Rule-based modeling: a computational approach for studying biomolecular site dynamics in cell signaling systems”. In: *Wiley Interdisciplinary Reviews: Systems Biology and Medicine* 6.1 (2014), pp. 13–36.
- [29] Clarke, B. L. *Stability of complex reaction networks*. Vol. 42. Adv. Chem. Phys. Wiley, New York, 1980, pp. 1–213.
- [30] Cohen, P. “The regulation of protein function by multisite phosphorylation—a 25 year update”. In: *Trends in Biochemical Sciences* 25.12 (2000), pp. 596–601.

- [31] Cohen, S. D. and Hindmarsh, A. C. “CVODE, a Stiff/Nonstiff ODE Solver in C”. In: *Comput. Phys.* 10 (1996), pp. 138–143.
- [32] Deckard, A., Bergmann, F. T., and Sauro, H. M. “Supporting the SBML layout extension.” In: *Bioinformatics* 22.23 (2006), pp. 2966–2967.
- [33] R. C. Dickson and M. D. Mendenhall, eds. *Signal Transduction Protocols*. Vol. 284. Methods in Molecular Biology. Humana Press, 2nd Edition, 2004.
- [34] Dormand, J. R. and Prince, P. J. “A family of embedded Runge-Kutta formulae”. In: *Journal of computational and applied mathematics* 6.1 (1980), pp. 19–26.
- [35] Draper, N. R. and Smith, H. *Applied Regression Analysis*. Wiley, 3rd edition, 1998.
- [36] Efron, B. and Tibshirani, R. “Bootstrap methods for standard errors, confidence intervals, and other measures of statistical accuracy”. In: *Statistical science* (1986), pp. 54–75.
- [37] Egbert, R. G. and Klavins, E. “Fine-tuning gene networks using simple sequence repeats”. In: *Proceedings of the National Academy of Sciences* 109.42 (2012), pp. 16817–16822.
- [38] Elowitz, M. et al. “Stochastic gene expression in a single cell”. In: *Science Signalling* 297.5584 (2002), p. 1183.
- [39] Entus, R., Aufderheide, B., and Sauro, H. M. “Design and implementation of three incoherent feed-forward motif based biological concentration sensors”. In: *Systems and Synthetic Biology* 10.1007/s11693-007-9008-6 (2007).
- [40] Eunen, K. van et al. “Testing biochemistry revisited: how in vivo metabolism can be understood from in vitro enzyme kinetics”. In: *PLoS computational biology* 8.4 (2012), e1002483.
- [41] Faeder, J. R. et al. “Rule-based modeling of biochemical networks.” In: *Complexity* 10 (2005), pp. 22–41.
- [42] Fell, D. *Understanding the Control of Metabolism*. London: Portland Press., 1997.
- [43] Ferrell, J. E. and Xiong, W. “Bistability in cell signaling: How to make continuous processes discontinuous, and reversible processes irreversible”. In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 11.1 (2001), pp. 227–236.
- [44] Fields, S and Song, O. “A novel genetic system to detect protein-protein interactions.” In: *Nature* 340.6230 (1989), pp. 245–246.
- [45] Friedland, A. E. et al. “Synthetic gene networks that count.” eng. In: *Science* 324.5931 (2009), pp. 1199–1202.
- [46] Gama-Castro, S. et al. “RegulonDB (version 6.0): gene regulation model of Escherichia coli K-12 beyond transcription, active (experimental) annotated promoters and Textpresso navigation.” In: *Nucleic Acids Res* 36.Database issue (2008), pp. D120–D124.

- [47] Gardner, T. S. and Collins, J. J. “Neutralizing noise in gene networks”. In: *Nature* 405 (2000), pp. 520–521.
- [48] Garfinkel, D. “A Machine-Independent Language for the Simulation of Complex Chemical and Biochemical Systems.” In: *Comput. Biomed. Res.* 2 (1968), pp. 31–44.
- [49] Garfinkel, D et al. “Computer applications to biochemical kinetics”. In: *Annu Rev Biochem* 39 (1970), pp. 473–498.
- [50] Gauges, R. et al. “A Model Diagram Layout Extension for SBML”. In: *Bioinformatics* 22.15 (2006), pp. 1879–1885.
- [51] Gavin, A.-C. et al. “Proteome survey reveals modularity of the yeast cell machinery.” In: *Nature* 440.7084 (2006), pp. 631–636.
- [52] Gear, C. W. *Numerical Initial Value Problems in Ordinary Differential Equations*. Prentice-Hall, Inc. Englewood Cliffs, New Jersey, 1971.
- [53] Gekas, V. and Lopez-Leiva, M. “Hydrolysis of lactose: a literature review”. In: *Process biochemistry* 20.1 (1985), pp. 2–12.
- [54] Gibson, M. A. and Bruck, J. “Efficient Exact Stochastic Simulation of Chemical Systems with Many Species and Many Channels”. In: *J. Phys. Chem. A* 104.9 (2000), pp. 1876–1889.
- [55] Gillespie, D. T. “A General Method for Numerically Simulating the Stochastic Time Evolution of Coupled Chemical Reactions”. In: *J. Comp. Phys.* 22 (1976), pp. 403–434.
- [56] Gillespie, D. T. “Exact Stochastic Simulation of Coupled Chemical Reactions”. In: *J. Phys. Chem.* 81 (1977), pp. 2340–2361.
- [57] Goldberg, D. “Genetic Algorithms in optimization, search and machine learning”. In: *Addison Wesley, New York. Eiben AE, Smith JE (2003) Introduction to Evolutionary Computing. Springer. Jacq J, Roux C (1995) Registration of non-segmented images using a genetic algorithm. Lecture notes in computer science* 905 (1989), pp. 205–211.
- [58] Golding, I. et al. “Real-time kinetics of gene activity in individual bacteria”. In: *Cell* 123.6 (2005), pp. 1025–1036.
- [59] Goodyear, C. and Silverman, G. “Phage-Display Methodology for the Study of Protein-Protein Interactions: Overview”. In: *Cold Spring Harbor Protocols* 2008.9 (2008).
- [60] M. de Graauw, ed. *Phospho-Proteomics*. Vol. 527. Methods in Molecular Biology. Humana Press, 2009.
- [61] Hairer, E., Nørsett, S. P., and Wanner, G. *Solving ordinary differential equations*. Vol. 2. Springer, 1991.

- [62] Hansen, P. C., Pereyra, V., and Scherer, G. *Least squares data fitting with applications*. JHU Press, 2012.
- [63] Harris, L. A. et al. “BioNetGen 2.2: advances in rule-based modeling”. In: *Bioinformatics* 32.21 (2016), pp. 3366–3368.
- [64] Hatzimanikatis, V. and Bailey, J. E. “Effects of spatiotemporal variations on metabolic control: approximate analysis using (log)linear kinetic models.” eng. In: *Biotechnol Bioeng* 54.2 (1997), pp. 91–104.
- [65] Hedley, W. J. et al. *CellML Specification*. Available via the World Wide Web at <http://www.cellml.org>. 2001.
- [66] Heijnen, J. J. “Approximative kinetic formats used in metabolic network modeling.” eng. In: *Biotechnol Bioeng* 91.5 (2005), pp. 534–545.
- [67] Heinrich, R., Rapoport, S. M., and Rapoport, T. A. “Metabolic regulation and mathematical models.” In: *Prog. Biophys. Molec. Biol.* 32 (1977), pp. 1–82.
- [68] Henry, C. S. et al. “High-throughput generation, optimization and analysis of genome-scale metabolic models”. In: *Nature biotechnology* 28.9 (2010), pp. 977–982.
- [69] Henry, C. S. et al. “High-throughput generation, optimization and analysis of genome-scale metabolic models”. In: *Nature biotechnology* 28.9 (2010), pp. 977–982.
- [70] Herrera, F., Lozano, M., and Verdegay, J. L. “Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis”. In: *Artificial intelligence review* 12.4 (1998), pp. 265–319.
- [71] Hindmarsh, A. C. “ODEPACK, a systematized collection of ode solvers in scientific computing.” In: *Scientific Computing*. Ed. by R. Stepleman. North-Holland, Amsterdam, 1983, pp. 55–64.
- [72] Hoefnagel, M. et al. “Time dependent responses of glycolytic intermediates in a detailed glycolytic model of *Lactococcus lactis* during glucose run-out experiments”. In: *Molecular biology reports* 29.1-2 (2002), pp. 157–161.
- [73] Hofmeyr, J.-H. “Metabolic regulation: a control analytic perspective.” eng. In: *J Bioenerg Biomembr* 27.5 (1995), pp. 479–490.
- [74] Hofmeyr, J.-H. and Merwe, K. J. van der. “METAMOD: software for steady-state modelling and control analysis of metabolic pathways on the BBC microcomputer.” In: *Comp. Appl. Biosci.* 2 (1986), pp. 243–249.
- [75] Hoops, S et al. “COPASI—a COMplex PAthway SImlator”. In: *Bioinformatics* 22.24 (2006), pp. 3067–3074.
- [76] Hucka, M. et al. “The Systems Biology Markup Language (SBML): A Medium for Representation and Exchange of Biochemical Network Models”. In: *Bioinformatics* 19 (2003), pp. 524–531.
- [77] Huerta, A. M. et al. “RegulonDB: a database on transcriptional regulation in *Escherichia coli*.” In: *Nucleic Acids Res* 26.1 (1998), pp. 55–59.

- [78] Ingalls, B. *Mathematical Modeling in Systems Biology: An Introduction*. MIT Press, 2013.
- [79] Ito, T. et al. “A comprehensive two-hybrid analysis to explore the yeast protein interactome.” In: *Proc Natl Acad Sci U S A* 98.8 (2001), pp. 4569–4574.
- [80] Jabbari, S., Heap, J. T., and King, J. R. “Mathematical modelling of the sporulation-initiation network in *Bacillus subtilis* revealing the dual role of the putative quorum-sensing signal molecule PhrA”. In: *Bulletin of mathematical biology* 73.1 (2011), pp. 181–211.
- [81] Jeong, H. et al. “Lethality and centrality in protein networks.” In: *Nature* 411.6833 (2001), pp. 41–42.
- [82] Kacser, H. and Burns, J. A. “The Control of Flux”. In: *Rate Control of Biological Processes*. Ed. by D. D. Davies. Vol. 27. Symp. Soc. Exp. Biol. Cambridge University Press, 1973, pp. 65–104.
- [83] Kacser, H, Burns, J., and Fell, D. “The control of flux”. In: *Biochemical Society Transactions* 23.2 (1995), pp. 341–366.
- [84] Karp, P. D. et al. “Multidimensional annotation of the *Escherichia coli* K-12 genome”. In: *Nucleic Acids Res* 35.22 (2007), pp. 7577–7590.
- [85] Kashtan, N. et al. *Mfinder tool guide*. 2002.
- [86] Keseler, I. M. et al. “EcoCyc: a comprehensive database of *Escherichia coli* biology.” eng. In: *Nucleic Acids Res* 39.Database issue (2011), pp. D583–D590.
- [87] Kim, K. H., Qian, H., and Sauro, H. M. “Sensitivity regulation based on noise propagation in stochastic reaction networks”. In: *arXiv preprint arXiv:0805.4455* (2008).
- [88] Kirkpatrick, S., Jr., D. G., and Vecchi, M. P. “Optimization by simulated annealing”. In: *science* 220.4598 (1983), pp. 671–680.
- [89] Kitano, H et al. “Using process diagrams for the graphical representation of biological networks”. In: *Nat Biotechnol* 23.8 (2005), pp. 961–966.
- [90] Kroeze, W. K., Sheffler, D. J., and Roth, B. L. “G-protein-coupled receptors at a glance”. In: *Journal of Cell Science* 116.24 (2003), pp. 4867–4869.
- [91] Krogan, N. J. et al. “Global landscape of protein complexes in the yeast *Saccharomyces cerevisiae*.” In: *Nature* 440.7084 (2006), pp. 637–643.
- [92] Kuzmić, P. “Program DYNAFIT for the analysis of enzyme kinetic data: application to HIV proteinase”. In: *Analytical biochemistry* 237.2 (1996), pp. 260–273.
- [93] Larson, D. R., Singer, R. H., and Zenklusen, D. “A single molecule view of gene expression”. In: *Trends Cell Biol* 19.11 (2009), pp. 630–637.
- [94] Le Novère, N. et al. “The systems biology graphical notation”. In: *Nature biotechnology* 27.8 (2009), pp. 735–741.

- [95] Le Novère, N. et al. “Minimum Information Requested In theAannotation of biochemical Models (MIRIAM).” In: *Nature biotechnology* 23.12 (2005), pp. 1509–1515.
- [96] Lee, T. I. et al. “Transcriptional regulatory networks in *Saccharomyces cerevisiae*.” In: *Science* 298.5594 (2002), pp. 799–804.
- [97] Leroux, A. E. et al. “Dissecting the Catalytic Mechanism of *Trypanosoma brucei* Trypanothione Synthetase by Kinetic Analysis and Computational Modelling”. In: *Journal of Biological Chemistry* (2013).
- [98] Lloyd, C. M. et al. “The CellML Model Repository”. In: *Bioinformatics* (2008).
- [99] Longabaugh, W., Davidson, E., and Bolouri, H. “Visualization, documentation, analysis, and communication of large-scale gene regulatory networks.” In: *Biochim Biophys Acta* (2008).
- [100] Longabaugh, W., Davidson, E., and Bolouri, H. “Computational representation of developmental genetic regulatory networks”. In: *Developmental Biology* 283.1 (2005), pp. 1–16.
- [101] Luciano, J. S. and Stevens, R. D. “e-Science and biological pathway semantics”. In: *BMC Bioinformatics* 8 Suppl 3 (2007), 8 Suppl 3: S3.
- [102] Maarleveld, T. R. et al. “Basic concepts and principles of stoichiometric modeling of metabolic networks”. In: *Biotechnology Journal* (2013).
- [103] Macek, B. et al. “Phosphoproteome Analysis of *E. coli* Reveals Evolutionary Conservation of Bacterial Ser/Thr/Tyr Phosphorylation”. In: *Molecular & Cellular Proteomics* 7.2 (2008), p. 299.
- [104] MacNamara, S. et al. “Stochastic chemical kinetics and the total quasi-steady-state assumption: Application to the stochastic simulation algorithm and chemical master equation”. In: *Journal of Chemical Physics* 129.9 (2008).
- [105] Mangan, S et al. “The incoherent feed-forward loop accelerates the response-time of the gal system of *Escherichia coli*”. In: *J Mol Biol* 356.5 (2006), pp. 1073–1081.
- [106] Manninen, T et al. “Discrete stochastic simulation of cell signaling: comparison of computational tools”. In: *Conf Proc IEEE Eng Med Biol Soc* 1 (2006), pp. 2013–2016.
- [107] Manning, G. et al. “The Protein Kinase Complement of the Human Genome”. In: *Science* 298 (2000), pp. 1912–1934.
- [108] Mardis, E. R. “ChIP-seq: welcome to the new frontier.” In: *Nat Methods* 4.8 (2007), pp. 613–614.
- [109] Markevich, N. I., Hoek, J. B., and Kholodenko, B. N. “Signaling switches and bistability arising from multisite phosphorylation in protein kinase cascades”. In: *J. Cell Biol.* 164 (2004), pp. 353–9.

- [110] Marquardt, D. "An algorithm for least-squares estimation of nonlinear parameters". In: *J. Soc. Ind. Appl. Math* 11.2 (1963), pp. 431–441.
- [111] Mattick, J. "RNA regulation: a new genetics?" In: *Pharmacogenomics J* 4 (2004), pp. 9–16.
- [112] Maus, C., Rybacki, S., and Uhrmacher, A. M. "Rule-based multi-level modeling of cell biological systems". In: *BMC Systems Biology* 5.1 (2011), p. 166.
- [113] McCollum, J. M. et al. "The sorting direct method for stochastic simulation of biochemical systems with varying reaction execution behavior." In: *Comput Biol Chem* 30.1 (2006), pp. 39–49.
- [114] McLachlan, G., Do, K.-A., and Ambroise, C. *Analyzing microarray gene expression data*. Vol. 422. Wiley.com, 2004.
- [115] Medley, J. K. et al. "Tellurium notebooks: An environment for reproducible dynamical modeling in systems biology". In: *PLoS computational biology* 14.6 (2018), e1006220.
- [116] Milo, R. et al. "Network motifs: simple building blocks of complex networks." In: *Science* 298 (2002), pp. 824–7.
- [117] Milo, R. et al. "On the uniform generation of random graphs with prescribed degree sequences". In: *eprint arXiv: cond-mat/0312028* (2003).
- [118] Milo, R. et al. "Superfamilies of Evolved and Designed Networks". In: *Science* 303.5663 (2004), p. 1538.
- [119] Milo, R. et al. "BioNumbers - the database of key numbers in molecular and cell biology". In: *Nucleic acids research* 38.suppl 1 (2010), pp. D750–D753.
- [120] Monod, J., Wyman, J., and Changeux, J. P. "On the Nature of Allosteric Transitions: A Plausible Model". In: *J Mol Biol* 12 (1965), pp. 88–118.
- [121] Montgomery, D. C., Peck, E. A., and Vining, G. G. *Introduction to linear regression analysis*. Vol. 821. Wiley, 2012.
- [122] Müller, T. et al. "Parameter identification in dynamical models of anaerobic waste water treatment". In: *Mathematical Biosciences* 177 (2002), pp. 147–160.
- [123] Myers, C. J. et al. "iBioSim: a tool for the analysis and design of genetic circuits". In: *Bioinformatics* 25.21 (2009), pp. 2848–2849.
- [124] Nelder, J. and Mead, R. "A Simplex Method for Function Minimization". In: *The Computer Journal* 7.4 (1965), p. 308.
- [125] Olivier, B. G., Rohwer, J. M., and Hofmeyr, J. H. "Modelling cellular systems with PySCeS." In: *Bioinformatics* 21 (2005), pp. 560–1.
- [126] Olivier, B. and Snoep, J. "Web-based kinetic modelling using JWS Online". In: *Bioinformatics* 20.13 (2004), pp. 2143–2144.

- [127] Ozbudak, E. M. et al. “Multistability in the lactose utilization network of *Escherichia coli*”. In: *Nature* 427.6976 (2004), pp. 737–740.
- [128] Ozbudak, E. et al. “Regulation of noise in the expression of a single gene”. In: *Nature genetics* 31.1 (2002), pp. 69–73.
- [129] Pahle, J. “Biochemical simulations: stochastic, approximate stochastic and hybrid approaches”. In: *Briefings in bioinformatics* 10.1 (2009), pp. 53–64.
- [130] Palsson, B. O. *Systems Biology: Properties of Reconstructed Networks*. Cambridge University Press, 2007.
- [131] Palsson, B. *Systems biology: simulation of dynamic network states*. Cambridge University Press, 2011.
- [132] Park, D. J. M. and Wright, B. E. “METASIM, A General Purpose Metabolic Simulator for Studying Cellular Transformations.” In: *Comput. Progm. Biomed.* 3 (1973), pp. 10–26.
- [133] Paulsson, J., Berg, O. G., and Ehrenberg, M. “Stochastic focusing: fluctuation-enhanced sensitivity of intracellular regulation”. In: *Proc Natl Acad Sci U S A* 97.13 (2000), pp. 7148–7153.
- [134] Pedersen, M. and Plotkin, G. “A Language for Biochemical Systems”. In: *Computational Methods in Systems Biology*. in press, <http://homepages.inf.ed.ac.uk/s0677975/papers/lbs.pdf>. Springer, 2008.
- [135] Phillips, R. and Milo, R. “A feeling for the numbers in biology”. In: *Proceedings of the National Academy of Sciences* 106.51 (2009), pp. 21465–21471.
- [136] Phillips, R. et al. “Physical biology of the cell”. In: *American Journal of Physics* 78 (2010), p. 1230.
- [137] Phizicky, E. et al. “Protein analysis on a proteomic scale”. In: *Nature* 422.6928 (2003), pp. 208–215.
- [138] Press, W. H. et al. *Numerical Recipies in C. The Art of Scientific Computing*. Cambridge University Press, Cambridge, 1988.
- [139] Ptacek, J. and Snyder, M. “Charging it up: global analysis of protein phosphorylation”. In: *Trends in Genetics* 22.10 (2006), pp. 545–554.
- [140] Ptacek, J. et al. “Global analysis of protein phosphorylation in yeast”. In: *Nature* 438.7068 (2005), pp. 679–684.
- [141] Ramsey, S., Orrell, D., and Bolouri, H. “Dizzy: stochastic simulation of large-scale genetic regulatory networks”. In: *Journal of bioinformatics and computational biology* 3.02 (2005), pp. 415–436.
- [142] Rao, C. and Arkin, A. “Stochastic chemical kinetics and the quasi-steady-state assumption: Application to the Gillespie algorithm”. In: *Journal of Chemical Physics* 118.11 (2003), pp. 4999–5010.

- [143] Rawlings, J. B. and Ekerdt, J. G. *Chemical reactor analysis and design fundamentals*. Nob Hill Pub., 2002.
- [144] Reich, J. G. and Selkov, E. E. *Energy metabolism of the cell*. London: Academic Press, 1981.
- [145] Ren, B. et al. “Genome-wide location and function of DNA binding proteins.” In: *Science* 290.5500 (2000), pp. 2306–2309.
- [146] Ribeiro, A. S. and Lloyd-Price, J. “SGN Sim, a stochastic genetic networks simulator”. In: *Bioinformatics* 23.6 (2007), pp. 777–779.
- [147] Sanft, K., Gillespie, D., and Petzold, L. “Legitimacy of the stochastic Michaelis-Menten approximation”. In: *Systems Biology, IET* 5.1 (2011), pp. 58–69.
- [148] Sauro, H. M. *Enzyme Kinetics for Systems Biology*. Ambrosius Publishing. First Edition, 2011.
- [149] Sauro, H. M. *Enzyme Kinetics for Systems Biology*. Ambrosius Publishing. 2nd Edition, 2012.
- [150] Sauro, H. M. “Jarnac: A System for Interactive Metabolic Analysis”. In: *Animating the Cellular Map: Proceedings of the 9th International Meeting on BioThermoKinetics*. Ed. by J.-H. S. Hofmeyr, J. M. Rohwer, and J. L. Snoep. Stellenbosch University Press, 2000.
- [151] Sauro, H. M. and Barrett, J. “in vitro Control Analysis of Metabolic Pathways; experimental and analytical developments.” In: *Molecular and Cellular Biochemistry* 145 (1995), pp. 141–150.
- [152] Sauro, H. M. and Fell, D. A. “SCAMP: A metabolic simulator and control analysis program.” In: *Mathl. Comput. Modelling* 15 (1991), pp. 15–28.
- [153] Sauro, H. M. and Ingalls, B. “Conservation analysis in biochemical networks: computational issues for software writers”. In: *Biophys Chem* 109.1 (2004), pp. 1–15.
- [154] Sauro, H. M. and Kholodenko, B. N. “Quantitative analysis of signaling networks”. In: *Prog Biophys Mol Biol.* 86 (2004), pp. 5–43.
- [155] Sauro, H. M. et al. “Next Generation Simulation Tools: The Systems Biology Workbench and BioSPICE Integration”. In: *OMICS* 7(4) (2003), pp. 355–372.
- [156] Sauro, H. and Bergmann, F. “Standards and ontologies in computational systems biology”. In: *Essays Biochem* 45 (2008), pp. 211–222.
- [157] Sauro, H. M. et al. “libRoadRunner: A High Performance SBML Compliant Simulator”. In: *bioRxiv* (2013).
- [158] Schleich, K. and Lavrik, I. N. “Mathematical modeling of apoptosis”. In: *Cell Communication and Signaling* 11.1 (2013), p. 44.
- [159] Schreiber, F. and Schwobbermeyer, H. “MAVisto: a tool for the exploration of network motifs”. In: *Bioinformatics* 21.17 (2005), pp. 3572–3574.

- [160] Seshasayee, A. S. N. et al. “Transcriptional regulatory networks in bacteria: from input signals to output responses”. In: *Current Opinion in Microbiology* 9.5 (2006), pp. 511–519.
- [161] Shen-Orr, S. S. et al. “Network motifs in the transcriptional regulation network of *Escherichia coli*”. In: *Nature Genetics* 31 (2002), pp. 64–68.
- [162] Siegal, M. L., Promislow, D. E. L., and Bergman, A. “Functional and evolutionary inference in gene networks: does topology matter?” In: *Genetica* 129.1 (2007), pp. 83–103.
- [163] Singer, S. and Singer, S. “Efficient Implementation of the Nelder-Mead Search Algorithm”. In: *Applied Numerical Analysis & Computational Mathematics* 1.2 (), pp. 524–534. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/anac.200410015>.
- [164] Smith, G. P. “Filamentous fusion phage: novel expression vectors that display cloned antigens on the virion surface”. In: *Science* 228.4705 (1985), pp. 1315–1317.
- [165] Smith, L. P. et al. “Antimony: a modular model definition language”. In: *Bioinformatics* 25.18 (2009), pp. 2452–2454.
- [166] Somogyi, E. T. et al. “libRoadRunner: a high performance SBML simulation and analysis library.” ENG. In: *Bioinformatics (Oxford, England)* (2015).
- [167] Stolovitzky, G., Prill, R. J., and Califano, A. “Lessons from the DREAM2 Challenges”. In: *Annals of the New York Academy of Sciences* 1158.1 (2009), pp. 159–195.
- [168] Storn, R. and Price, K. “Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces”. In: *Journal of global optimization* 11.4 (1997), pp. 341–359.
- [169] Straume, M and Johnson, M. “Analysis of residuals: criteria for determining goodness-of-fit.” In: *Methods in enzymology* 210 (1992), p. 87.
- [170] Straume, M. and Johnson, M. L. “Monte Carlo method for determining complete confidence probability distributions of estimated model parameters”. In: *Essential Numerical Computer Methods* (2010), p. 55.
- [171] Sundararaj, S. et al. “The CyberCell Database (CCDB): a comprehensive, self-updating, relational database to coordinate and facilitate in silico modeling of *Escherichia coli*”. In: *Nucleic acids research* 32.suppl 1 (2004), pp. D293–D295.
- [172] Taft, R. and Mattick, J. “Increasing biological complexity is positively correlated with the relative genome-wide expansion of non-protein-coding DNA sequences”. In: *Arxiv preprint q-bio.GN/0401020* (2004).
- [173] Teusink, B. et al. “Can yeast glycolysis be understood in terms of in vitro kinetics of the constituent enzymes? Testing biochemistry”. In: *Eur. J. Biochem* 267 (2000), pp. 5313–5329.

- [174] Thiele, I. and Palsson, B. Ø. “A protocol for generating a high-quality genome-scale metabolic reconstruction”. In: *Nature protocols* 5.1 (2010), pp. 93–121.
- [175] Thron, C. “A model for a bistable biochemical trigger of mitosis”. In: *Biophysical chemistry* 57.2 (1996), pp. 239–251.
- [176] Toledo, F. and Wahl, G. M. “Regulating the p53 pathway: in vitro hypotheses, in vivo veritas.” In: *Nat Rev Cancer* 6.12 (2006), pp. 909–923.
- [177] Tyson, J. J., Chen, K. C., and Novak, B. “Sniffers, buzzers, toggles and blinkers: dynamics of regulatory and signaling pathways in the cell”. In: *Current Opinion in Cell Biology* 15 (2003), pp. 221–231.
- [178] Uetz, P. et al. “A comprehensive analysis of protein-protein interactions in *Saccharomyces cerevisiae*.” In: *Nature* 403.6770 (2000), pp. 623–627.
- [179] Visser, D. and Heijnen, J. J. “Dynamic simulation and metabolic re-design of a branched pathway using linlog kinetics.” In: *Metabolic Engineering* 5 (2003), pp. 164–76.
- [180] Wang, E. et al. “Alternative isoform regulation in human tissue transcriptomes”. In: *Nature* 456 (2008), pp. 470–476.
- [181] Warner, J. R. “The economics of ribosome biosynthesis in yeast”. In: *Trends in biochemical sciences* 24.11 (1999), pp. 437–440.
- [182] Wernicke, S. and Rasche, F. “FANMOD: a tool for fast network motif detection”. In: *Bioinformatics* 22.9 (2006), pp. 1152–1153.
- [183] Westerhoff, H. V. and Van Dam, K. *Thermodynamics and control of biological free-energy transduction*. Elsevier Amsterdam et al., 1987.
- [184] Wilkinson, D. J. *SBML Shorthand*. <http://www.staff.ncl.ac.uk/d.j.wilkinson/software/sbml-sh/>. 2007.
- [185] Wilkinson, D. J. *Stochastic Modelling for Systems Biology*. Chapman & Hall/CRC Press, Boca Raton, Florida, 2nd edition, 2012.

This copy belongs to Eric Freeman

# ***History***

## 1. VERSION: 1.00 (Gildas)

**Date:** 2014-18-3

**Author(s):** Herbert M. Sauro

**Title:** Essentials of Biochemical Modeling

**Modification(s):** First edition, first printing

## 2. VERSION: 1.01 (Vortiporius)

**Date:** 2014-1-5

**Author(s):** Herbert M. Sauro

**Title:** Essentials of Biochemical Modeling

**Modification(s):** Minor corrections to most chapters, thanks to Joseph Hellerstein.

## 3. VERSION: 1.02 (Caninus)

**Date:** 2014-8-6

**Author(s):** Herbert M. Sauro

**Title:** Essentials of Biochemical Modeling

**Modification(s):** Correction to figure in exercise 12 in chapter 3

## 4. VERSION: 1.03 (Malgo)

**Date:** 2014-20-8

**Author(s):** Herbert M. Sauro

**Title:** Systems Biology: An Introduction to Pathway Modeling

**Modification(s):** Republished under a new title and new ISBN number. All modeling scripts converted to Python

## 5. VERSION: 1.04 (Cadfan)

**Date:** 2014-1-8

**Author(s):** Herbert M. Sauro

**Title:** Systems Biology: An Introduction to Pathway Modeling

**Modification(s):** LaTeX typo on page 75 fixed.

6. VERSION: 1.05 (Cunedda)

**Date:** 2014-23-12

**Author(s):** Herbert M. Sauro

**Title:** Systems Biology: An Introduction to Pathway Modeling

**Modification(s):** Minor clarification to linearization on page 90.

7. VERSION: 1.06 (Owain)

**Date:** 2015-24-1

**Author(s):** Herbert M. Sauro

**Title:** Systems Biology: An Introduction to Pathway Modeling

**Modification(s):** Fixed typo in Taylor series second order approximation of sine on page 339 and fixed equation typo for normalized propensity function on page 141.

8. VERSION: 1.07 (Bede)

**Date:** 2015-1-8

**Author(s):** Herbert M. Sauro

**Title:** Systems Biology: An Introduction to Pathway Modeling

**Modification(s):** Added index entries for fast reactions and fixed typo on page 123. Rewrote the introduction to the chapter on stoichiometry networks. Modified Tellurium scripts to match the latest version where the need to include ‘model’ to reference variables and parameters has been relaxed.

9. VERSION: 1.08 (Ceolfrith)

**Date:** 2015-14-9

**Author(s):** Herbert M. Sauro

**Title:** Systems Biology: An Introduction to Pathway Modeling

**Modification(s):** Added more detailed algorithm for the simplex. Correct small error in the Differential evolution algorithm.

10. VERSION: 1.09 (Sigfrith)

**Date:** 2015-14-12

**Author(s):** Herbert M. Sauro

**Title:** Systems Biology: An Introduction to Pathway Modeling

**Modification(s):** Fixed code typo in the python script Steady state band detector

11. VERSION: 1.1 (Eosterwine)

**Date:** 2016-4-08

**Author(s):** Herbert M. Sauro

**Title:** Systems Biology: An Introduction to Pathway Modeling

**Modification(s):** Fixed typo on page 214, Chap 10: 95.5% should be 97.5%. Page 210, ‘contingence’ should be ‘confidence’. Chap 9: Rewritten section on Levenberg-Marquardt method to fix some errors and added new section on Gauss-Newton. Minor formatting issues fixed.

12. VERSION: 1.11 (Benedict Biscop)

**Date:** 2016-1-11

**Author(s):** Herbert M. Sauro

**Title:** Systems Biology: An Introduction to Pathway Modeling

**Modification(s):** Updated script that generates the phase diagram Figure 12.3

13. VERSION: 1.12 (Offa)

**Date:** 2017-17-8

**Author(s):** Herbert M. Sauro

**Title:** Systems Biology: An Introduction to Pathway Modeling

**Modification(s):** The Tellurium scripts 5.1 and 5.2 were wrongly transcribed. A few remaining references to Jarnac removed and substituted with Tellurium. Bad figure references in Chapter 6 fixed and Gillespie scripts updated.

14. VERSION: 1.13 (Egfrid)

**Date:** 2018-21-7

**Author(s):** Herbert M. Sauro

**Title:** Systems Biology: An Introduction to Pathway Modeling

**Modification(s):** Chapter 1 to 9 have been reedited, a few minor typographical corrections and improvements to some explanations.

15. VERSION: 1.14 (Eanbert)

**Date:** 2018-17-8

**Author(s):** Herbert M. Sauro

**Title:** Systems Biology: An Introduction to Pathway Modeling

**Modification(s):** Went through every Tellurium example to make sure the code worked.

Only a couple of instances where they didn't due to changes in the Tellurium syntax. This version corrects those errors. Refreshed all the phase plots with newly generated simulations and included the Python scripts used to generate these. Edited the Appendices and made some corrections. Added Python example of how to plot Q-Q plots in Chapter 10 on parameter estimation. Reformatted the bibliography.

16. VERSION: 1.15 (Eardulf)

**Date:** 2018-20-9

**Author(s):** Herbert M. Sauro

**Title:** Systems Biology: An Introduction to Pathway Modeling

**Modification(s):** Fixed some typos and made some of the notation more consistent throughout the book. Revised chapter 9 on optimization methods: 1) Fixed some minor errors in the Levenberg-Marquardt method; 2) Added a new figure describing differential evolution and 3) include an example of fitting a SBML model to data using scipy.

This copy belongs to Eric Freeman

# Index

## Symbols

$K_a$	36
$K_d$	36
$K_m$	330
$V_m$	330
$N$	54
$\bar{x}$	349
$\chi^2$	172, 352
$\sigma$	350
$\sin(x)$	89
$c$	134
$h$	134
$v_i$	34

## A

absolute sensitivities	246
accurate	73
activator	42
additivity	87
allosteric constant	334
allosteric control	3
allostery	335
Alon	323
alternative splicing	17
AMP	62
anabolic	3
analytical solutions	104
Antimony	44
approximations	93
association constant	36
ATP	3, 62
Avogadro's constant	85

## B

bait protein	7
bell curve	350
Bevington	358

bifurcation plot	267, 275
bimodal	309
BioModels	364
BioNetGen	364
bionumbers	323
BIONUMBR斯	20
BioPAX	363
Biotapestry	46
bistable	268, 270
bistable system	308
Boltzmann probability	187
bootstrap	215
bootstrapping	354
boundary	69
boundary variables	78
BRENDA	20
Briggs-Haldane	330
Brownian motion	81
brute force fitting	174
BSD	117
building a model	84
Burns	247
bursting	302, 303

## C

Calvin cycle	2
cAMP	13
Carsonella ruddii	16
catabolic	3
Cdc2-Cdc13	10
CellDesigner	118, 155
CellML	361
chatter	308
chemical equilibrium	35
chi-square	172, 352
ChIP-chip	13

- ChIP-seq ..... 13  
 clamp ..... 78  
 classification of models ..... 86  
 Claude Bernard ..... 156  
 closed ..... 70  
 cluster plots ..... 222  
 coenzyme A ..... 63  
 cofactors ..... 3  
 combination of molecules ..... 134  
 competitive ..... 332  
 competitive inhibition ..... 332  
 complex number ..... 266  
 compute steady state ..... 233  
 confidence interval ..... 353  
 confidence intervals ..... 212  
 conjugate Pair ..... 266  
 conserved cycles ..... 60  
 constants ..... 77  
 continuous variable ..... 81  
 cooperativity ..... 333  
 COPASI ..... 115, 118, 155, 196  
 covariance ..... 350  
 covariance matrix ..... 212  
 cross-sectional area ..... 161  
 crossover ..... 190  
 cumulative probability function ..... 139  
 curvature ..... 308  
 CVODE ..... 114  
 CyberCell ..... 20  
 cybercell ..... 323
- D**
- damped Newton method ..... 240  
 Darren Wilkinson ..... 145  
 data for models ..... 96  
 database ..... 361  
 databases ..... 364  
 dependent variable ..... 78  
 deterministic ..... 80  
 differential evolution ..... 191  
 diffusion ..... 159  
 diffusion coefficient ..... 160  
 dimensional analysis ..... 85
- dimensions ..... 85  
 direct method ..... 140  
 discrete events ..... 45  
 discrete variable ..... 81  
 disequilibrium ratio ..... 37  
 dissociation constant ..... 36  
 distributed models ..... 87  
 disturbance ..... 256  
 Dizzy ..... 140  
 DOQCS ..... 365  
 dynamic models ..... 86
- E**
- EcoCyc ..... 1, 18, 20  
 eigenvalues ..... 258  
 eigenvectors ..... 223  
 elasticity ..... 98, 275, 337  
 elasticity matrix ..... 260  
 elasticity values ..... 98  
 elementary reaction ..... 39  
 elementary reactions ..... 39  
 elitism ..... 189  
 endergonic ..... 62  
 enzyme action ..... 330  
 enzyme inhibitor complex ..... 332  
 enzyme kinetics ..... 329  
 enzyme-reactant complex ..... 330  
 equilibration model ..... 301  
 equilibrium ..... 149  
 equilibrium approximation ..... 123  
 equilibrium constant ..... 36  
 Euler method ..... 106  
 evolutionary algorithms ..... 189  
 exclusive model ..... 334  
 explicit regulation ..... 58
- F**
- F-test ..... 353  
 falsifiability ..... 73  
 fast proceses ..... 123  
 FBA ..... 86  
 Fell ..... 247  
 Fick's first law ..... 160

- FindRoot ..... 243  
first reaction method ..... 140  
first-order ..... 85  
fitness landscape ..... 174  
fixed points ..... 227  
fixed species ..... 45  
flux ..... 153  
flux balance analysis ..... 86  
forcing functions ..... 82  
Frank Herbert ..... ix  
Frankenstein ..... ix  
fsolve ..... 243  
functional parts ..... 38
- G**
- gate ..... 303  
Gauss-Newton method ..... 179  
Gaussian distribution ..... 350  
gene networks ..... 58  
genetic algorithm ..... 189  
Gepasi ..... 240  
Gillespie ..... 134  
Gillespie algorithm ..... 140  
Gillespie on a grid ..... 142  
Gillespie SSA ..... 140  
global and local searches ..... 194  
global error ..... 106, 110  
global minimum ..... 175  
Glycolysis ..... 2  
goodness of fit ..... 207  
Goodsell ..... 19  
GPL license ..... 117  
gradient ..... 160  
gradient search ..... 182  
graphical layout ..... 362  
Greg Bear ..... ix  
GSL library ..... 117, 187
- H**
- Haldane relationship ..... 331  
Hessian ..... 179, 213, 223  
Heun method ..... 110  
heuristic model ..... 72
- Hill equations ..... 337  
HIV Proteinase ..... 219  
homeostasis ..... 156, 244  
homogeneity ..... 88  
human ..... 11
- I**
- iBiosim ..... 118  
implicit regulation ..... 58  
independent variable ..... 79  
index ..... 78  
Ingalls ..... 147  
inhibitor ..... 42  
initial rate ..... 330  
ion channel ..... 303  
irreversible bistability ..... 277  
isolated ..... 70
- J**
- Jacobian ..... 181  
Jacobian matrix ..... 241, 257, 261  
Jacobian of biochemical system ..... 260  
Jarnac ..... 44, 240  
JDesigner ..... 53  
JWS online ..... 365
- K**
- Kacser ..... 247  
KEGG ..... 1, 97  
kinetic data ..... 97  
kinetic mechanism ..... 329  
kinetic order ..... 98  
KiSAO ..... 363  
Klipp ..... 156  
Kuzmic ..... 196
- L**
- lac repressor ..... 13  
large number of molecules ..... 49  
layout extension ..... 362  
leastsq ..... 197  
Levenberg-Marquardt ..... 182  
libSBML ..... 362  
lin-log ..... 97, 98

- lin-log approximation ..... 98  
 linear approximation ..... 90, 97  
 linear model ..... 86, 87  
 linear pathway ..... 51  
 linear time invariant systems ..... 87  
 linearization ..... 89  
 little b ..... 364  
 lmdif ..... 198  
 lmfit library ..... 183  
 LSODA ..... 115, 116  
 LTI ..... 87  
 lumped models ..... 87
- moiety conserved cycles ..... 60  
 molarity ..... 85  
 molecular weight ..... 85  
 moles ..... 85  
 Monod, Wyman, Changeaux ..... 96  
 Monte Carlo simulations ..... 213  
 multicompartment systems ..... 159  
 multiple steady states ..... 268  
 mutations ..... 190  
 MWC model ..... 334  
 Mycoplasma genitalium ..... 16

**M**

- mass-action kinetics ..... 35  
 mass-action ratio ..... 37  
 mass-balance equation ..... 46  
 Mathematica ..... 150  
 Mathematical models ..... 72  
 Matlab ..... 52, 155  
 Matlab solvers ..... 115  
 Maxima ..... 150  
 maximal velocity ..... 330  
 McCollum ..... 140  
 mean ..... 349  
 mechanistic details ..... 43  
 median ..... 349  
 membrane ..... 163  
 metabolic control analysis ..... 247  
 metabolic reconstruction ..... 97  
 metabolism ..... 3  
 MetaCyc ..... 97  
 MIASE ..... 363  
 Michaelis-Menten kinetics ..... 330  
 Milo ..... 323  
 minipack ..... 198  
 MIRIAM ..... 362  
 model composition ..... 45  
 model fitting ..... 169  
 model variables ..... 77  
 models ..... 71  
 modified Euler ..... 110  
 moiety ..... 61

**N**

- NAD ..... 3  
 NAD/NADH ..... 63  
 negative cooperativity ..... 333  
 negative feedback ..... 156  
 Nelder-Mead ..... 184  
 network topology ..... 261  
 Newton algorithm ..... 239  
 Newton-Raphson method ..... 234  
 next reaction method ..... 140  
 NLEQ2 ..... 240  
 noise propagation ..... 307  
 non-elementary reactions ..... 42  
 non-unity stoichiometry ..... 41  
 nonlinear model ..... 86, 87  
 normal distribution ..... 350  
 numbers ..... 323  
 numerical solution ..... 104

**O**

- observability problem ..... 222  
 Occam ..... 74  
 ode15s ..... 116  
 ode45 ..... 115, 116  
 odepack ..... 115  
 ontologies ..... 363  
 open ..... 70  
 open source ..... 117  
 operating point ..... 91  
 operator site ..... 268  
 optimization ..... 174

---

Oscill8.....	278	quasi-equilibrium .....	151		
outputs.....	79				
overfitting.....	210				
<b>P</b>					
Pahle .....	140	R state .....	334		
Palsson .....	65	radioisotopes .....	3		
parameter .....	77	random telegraph model .....	303		
particular model .....	72	rapid-equilibrium .....	43		
PathwayDesigner .....	118	rate constant .....	36, 85		
percentile values .....	216	rate of change .....	34		
periodic behavior .....	263	reaction kinetics .....	33		
periodic solutions .....	265	reaction rate .....	34		
permeability coefficient.....	160	reaction rates .....	54, 85		
perturbations .....	231	reduced chi-square .....	172		
phase plot .....	262	reference state .....	98		
phase portrait.....	262	regular intervals .....	142		
Phillips .....	323	regular time grid .....	142		
phosphate .....	63	regulatory link .....	43		
phosphofructokinase.....	3	RegulonDB .....	1, 13		
phosphorylation.....	10	relative sensitivities .....	247		
positive cooperativity.....	333	relaxed.....	334		
positive feedback loop.....	268	replacement .....	354		
positive feedback: stability.....	274	residuals .....	203		
PottersWheel .....	196	response times .....	21		
predictability .....	73	reversibility .....	40, 55		
predictive .....	73	reversible .....	35		
prey protein .....	7	reversible rate law .....	331		
probability .....	134	RK4 .....	110		
probability distribution function .....	138	roadRunner .....	114		
product inhibition .....	330, 333	roadrunner .....	240		
proof .....	73	robustness .....	156, 244		
propensity function .....	135	roulette wheel selection .....	189		
proteasome .....	10	Rum1 .....	10		
protein kinase .....	9	Runge-Kutta .....	110		
protein pore .....	163	<b>S</b>			
pure competitive inhibitor.....	332	saddle node .....	264		
PySCeS .....	44, 140, 240, 364, 365	SBGN .....	46, 362		
Python .....	196	SBML .....	52, 97, 118, 361		
<b>Q</b>					
quadratic function.....	345	SBML-shorthand .....	364		
quality of fit .....	210	SBO .....	363		
		SBSI .....	196		
		SCAMP .....	240		
		science fiction .....	ix		

- SciLab ..... 117  
SciPy ..... 196  
Scipy ..... 187  
second-order ..... 85  
sensitivity measures ..... 246  
sigmoid response ..... 333  
signaling networks ..... 57  
simplex ..... 184  
simulated annealing ..... 187  
simulation model ..... 74  
small effect ..... 94  
software ..... 117  
software: bifurcation ..... 275  
software: curve fitting ..... 196  
software: steady state ..... 240  
software:stochastic ..... 140  
software:time course ..... 117  
spiral trajectories ..... 265  
square root ..... 235  
SSA ..... 140  
SSA: multiple reactions ..... 141  
stability ..... 244, 255  
stable ..... 256  
stable node ..... 264  
standard deviation ..... 212, 349  
standard error ..... 350  
standard scores ..... 351  
standards ..... 361  
static models ..... 86  
stationary state ..... 227  
steady state ..... 152, 227  
steady state: analytical ..... 229  
steady state: computation ..... 233  
steady state: graphical ..... 228  
Steuer ..... 156  
stochastic ..... 80  
stochastic bursting ..... 302, 303  
stochastic chatter ..... 308  
stochastic defocusing ..... 308  
stochastic focusing ..... 305  
stochastic kinetics ..... 133  
stochastic mean ..... 143  
stochastic processes ..... 143  
stochastic rate constant ..... 134, 135  
stochastic trajectory ..... 143  
stochastic variance ..... 143  
stochastic: events ..... 147  
stochkit ..... 140  
stoichiometric amount ..... 33  
stoichiometric coefficient ..... 34, 50  
stoichiometric network ..... 39  
stoichiometry matrix ..... 54  
STRING ..... 1  
Stucki ..... 156  
SuBliMinaL ..... 97  
sum of propensities ..... 142  
sundials ..... 114, 240  
superposition ..... 88  
surroundings ..... 69  
synthetic biology ..... ix  
synthetic data set ..... 216  
system ..... 69  
system behavior ..... 149  
system equation ..... 63  
systems biology ontolog ..... 363
- T**
- T state ..... 334  
Taylor Series ..... 343  
Taylor series ..... 89  
telegraph model ..... 133  
Tellurium ..... 53, 64, 118, 154, 155  
temperature ..... 188  
tense ..... 334  
text representation ..... 44  
thermodynamic equilibrium ..... 152  
thermodynamic properties ..... 98  
time dependent ..... 150  
time invariant ..... 86  
time invariant models ..... 86  
time scale ..... 60  
time to reaction ..... 137  
Torrielli Law ..... 75  
tournament selection ..... 189  
trajectory ..... 143

- transcription factor ..... 58, 268  
transient ..... 155  
truncation error ..... 106  
two dimensional system ..... 93  
Tyson ..... 277

**U**

- Ullah ..... 148  
uncompetitive inhibition ..... 332  
uniform random number ..... 139  
unit balancing ..... 85  
units ..... 85  
unscaled elasticity ..... 337  
unstable node ..... 264

**V**

- validation ..... 73  
Van der Pol equations ..... 111  
variance ..... 172, 349  
VCell ..... 196

**W**

- Warner ..... 323  
water tank model ..... 74  
weighing ..... 172  
well-stirred reactor ..... 47  
Whisk fern ..... 16  
Wilkinson ..... 145, 148  
working hypothesis ..... 72

**X**

- X-gal ..... 8  
XML ..... 118

**Y**

- yeast ..... 6  
Yeast two-hybrid ..... 7

**Z**

- z-score ..... 351

