

Ronald Gieschke · Daniel Serafin

# Development of Innovative Drugs via Modeling with MATLAB

A Practical Guide

 Springer

# Development of Innovative Drugs via Modeling with MATLAB

Ronald Gieschke · Daniel Serafin

# Development of Innovative Drugs via Modeling with MATLAB

A Practical Guide

Ronald Gieschke  
Daniel Serafin  
Pharma Research and Early Development  
F. Hoffmann-La Roche Ltd  
Basel  
Switzerland

ISBN 978-3-642-39764-6      ISBN 978-3-642-39765-3 (eBook)  
DOI 10.1007/978-3-642-39765-3  
Springer Heidelberg New York Dordrecht London

Library of Congress Control Number: 2013944198

© Springer-Verlag Berlin Heidelberg 2014

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed. Exempted from this legal reservation are brief excerpts in connection with reviews or scholarly analysis or material supplied specifically for the purpose of being entered and executed on a computer system, for exclusive use by the purchaser of the work. Duplication of this publication or parts thereof is permitted only under the provisions of the Copyright Law of the Publisher's location, in its current version, and permission for use must always be obtained from Springer. Permissions for use may be obtained through RightsLink at the Copyright Clearance Center. Violations are liable to prosecution under the respective Copyright Law. The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Printed on acid-free paper

Springer is part of Springer Science+Business Media ([www.springer.com](http://www.springer.com))



# Preface

About 20 years ago, the pharmaceutical industry started to consider mathematical model-based drug development as a means to streamline the execution of drug development programs. This constituted a new discipline, pharmacometrics, grounded in pharmacology and statistics.

We have written this book to describe our learning and experiences in the emergent field of pharmacometrics applied to drug discovery and clinical development. We did not aim to compete with the many excellent contributions in this field, both theoretical and practical, but rather wanted to highlight specific areas that we have repeatedly encountered. This book may also give answers to those who wonder to what kind of problems mathematics is actually applied in the modeling and simulation department of a large pharmaceutical company. Parts of this book require some familiarity with mathematical notation and elementary calculus including the concept of a differential equation.

Besides providing some concepts behind drug discovery and development, we have added many exercises (with our solutions) designed to be solved symbolically or programmatically. As a programming language, we chose MATLAB (Version 2012b) as we found it best suited to the diversity of problems we faced.

Our final message is that pharmacometrics could have an even larger impact on drug discovery and development if it were consistently applied to diseases and their potential treatment targets, thus merging with another quantitative discipline, systems biology, to form quantitative systems pharmacology.

The book is organized into nine chapters. *Background of Pharmacologic Modeling* (Chap. 1) introduces two topics which underpin later chapters: the emergence, role, and tasks of the pharmaceutical industry as a healthcare provider; and the philosophy of modeling and simulation. Regarding modeling, we start with a *First Example of a Computational Model* (Chap. 2) from oncology to introduce physiologic, pharmacologic, and computational concepts that are explained and detailed in later chapters. *Differential Equations in MATLAB* (Chap. 3) provides the numerical and symbolic treatment of ordinary differential equations with time and state event scheduling. *Pharmacologic Modeling* (Chap. 4) is about dynamic concepts in relationship to drugs. This entails the modeling of drug concentrations and related body responses over time. *Disease Modeling* (Chap. 5) adds another component. As drugs work on a diseased human body, a model-based understanding of how the body functions under the disease will be of value to learn how

to optimally use drugs. This requires the integration of pharmacologic drug models and disease models. *Population Analyses* (Chap. 6) refers to parameter estimation in pharmacologic and/or disease models aimed at characterizing the sources (such as gender, age) of interindividual variability. This has a direct impact on how drugs are prescribed. *Clinical Trial Simulations* (Chap. 7) use population analysis results to forecast the statistical distribution of clinical trial outcomes under different design scenarios, thus enabling the generation of optimal designs that maximize the probability of successful trials. It is worth noting that trials at the very late stage of drug development are extremely-resource intensive and that a failed trial can be financially crippling.

Traditionally, mathematical models are written as explicit or implicit equations, as in Chap. 3. There is, however, a trend for an alternative modeling approach, *Graphics-Based Modeling* (Chap. 8), where models are built using symbols indicating flows between compartments rather than using mathematical notation. As we have to sell our modeling to decision-making non-modelers, the advantage of the graphically based approach is obvious. *Outlook* (Chap. 9) summarizes the current prospects for modeling and simulation in innovative drug development.

The accompanying material (MATLAB programs and data sets) can be retrieved from MATLAB Central.

This book would not have been possible without internal and external support. Internally, we thank our senior management and departmental colleagues for their reviews and comments on the manuscript. Special thanks to Nicolas Frey and James Lu for their extensive reviews and for testing some of the MATLAB scripts, and to Antonello Caruso and Bruno Reigner for reviewing part of the manuscript. Externally, we very much enjoyed collaborating with a number of persons from different institutions: with senior representatives from In Silico Biosciences, Inc. who showed us (Sect. 5.2.3.) that for theoretical work on neurons to be useful it must be embedded in a wider disease and treatment context; with Sam Roberts and Sven Mesecke from the software developer, MathWorks, who helped us with many programming issues and made us aware that a MATLAB program should follow some formal standards; and with Britta Mueller and Jutta Lindenborn from our publisher, Springer, who provided help for and answers to technical and legal questions. Final thanks are given to our families for their love and understanding.

We are looking forward to sharing our experience with readers.

Basel, May 2013

Ronald Gieschke  
Daniel Serafin  
Pharma Research and Early Development  
F. Hoffmann-La Roche Ltd  
Basel  
Switzerland

# Contents

<b>1</b>	<b>Background of Pharmacologic Modeling</b>	<b>1</b>
1.1	The Pharmaceutical Value Chain	1
1.1.1	Maturation of Medical Concepts and Drug Treatment	2
1.1.2	Emergence of the Pharmaceutical Industry	3
1.1.3	How are Drugs Discovered?	4
1.1.4	How are Drugs Developed?	6
1.1.5	Drug Regulation	9
1.2	Modeling and Simulation	10
1.2.1	Examples of Models	10
1.2.2	Modeling and Scientific Understanding	11
1.2.3	Mathematical Models	11
1.2.4	The Modeling Process	14
1.2.5	Application of Models	16
1.2.6	Validation of Models	18
	References	19
<b>2</b>	<b>First Example of a Computational Model</b>	<b>21</b>
2.1	Problem Description	21
2.2	Conceptual and Mathematical Modeling	22
2.3	Computational Model	26
2.4	Computational Model in MATLAB	29
2.5	Simulation Results	35
2.6	Comments	37
2.7	Exercises	38
	References	38
<b>3</b>	<b>Differential Equations in MATLAB</b>	<b>39</b>
3.1	Concepts	39
3.2	Numerical Solution of Ordinary and Partial Differential Equations	41
3.2.1	Basic Numerical Algorithms	41
3.2.2	Ordinary Differential Equation	41
3.2.3	Delay Differential Equations	47
3.2.4	Differential Algebraic Equation	52

3.2.5	Scheduling Time Events . . . . .	55
3.2.6	Scheduling State Events. . . . .	61
3.2.7	Partial Differential Equation. . . . .	68
3.3	Analytical Solutions. . . . .	70
3.3.1	Analytical Solutions by Elementary Methods . . . . .	70
3.3.2	Analytical Solutions Using Laplace Transforms . . . . .	72
3.4	Parameter Estimation in Differential Equations . . . . .	78
3.4.1	Nonlinear Regression. . . . .	79
3.5	Comments . . . . .	82
3.6	Exercises . . . . .	83
	References . . . . .	85
<b>4</b>	<b>Pharmacologic Modeling . . . . .</b>	<b>87</b>
4.1	Pharmacokinetics. . . . .	87
4.1.1	General Concepts . . . . .	87
4.1.2	Empirical PK Modeling. . . . .	91
4.1.3	Physiologically-based Pharmacokinetics. . . . .	105
4.1.4	Example: Tamiflu <sup>®</sup> . . . . .	112
4.2	Pharmacodynamics . . . . .	114
4.2.1	General Concepts . . . . .	114
4.2.2	Receptor Binding . . . . .	116
4.2.3	Pharmacodynamic Models . . . . .	118
4.2.4	Example: Saquinavir . . . . .	120
4.3	Time Course of Drug Response. . . . .	123
4.3.1	General Concepts . . . . .	123
4.3.2	PK-PD Relationships. . . . .	123
4.3.3	K-PD Analyses. . . . .	128
4.3.4	Example: Ibuprofen . . . . .	130
4.4	Exercises . . . . .	139
	References . . . . .	140
<b>5</b>	<b>Drug-Disease Modeling . . . . .</b>	<b>143</b>
5.1	Terms and Concepts . . . . .	143
5.2	Central Nervous System Disorders. . . . .	146
5.2.1	Physiologic Background. . . . .	147
5.2.2	Hodgkin and Huxley Mathematical Model. . . . .	153
5.2.3	Application to Neuronal Networks and CNS Disorders . . . . .	178
5.3	Renal Anemia . . . . .	180
5.3.1	Disease Definition. . . . .	180
5.3.2	Physiologic Model . . . . .	180
5.3.3	Pathophysiologic Model. . . . .	183
5.3.4	Drug-Disease Model . . . . .	184

5.4	Diabetes Mellitus. . . . .	191
5.4.1	Disease Definition. . . . .	191
5.4.2	Physiologic Model . . . . .	191
5.5	Viral Infection. . . . .	195
5.5.1	Definition. . . . .	195
5.5.2	Viral Kinetics . . . . .	196
5.6	Disease Progression Modeling. . . . .	199
5.6.1	General Idea. . . . .	199
5.6.2	Alzheimer's Disease . . . . .	201
5.7	Exercises . . . . .	203
	References . . . . .	203
<b>6</b>	<b>Population Analyses. . . . .</b>	<b>207</b>
6.1	Terms and Concepts . . . . .	207
6.2	Model-based Analysis of Clinical Data . . . . .	208
6.3	Population Approach . . . . .	209
6.3.1	Pharmacostatistical Model . . . . .	209
6.3.2	Estimation Issues in Pharmacostatistical Models. . . . .	212
6.4	Population Analyses in MATLAB. . . . .	213
6.4.1	Data . . . . .	213
6.4.2	Exploratory Analysis. . . . .	215
6.4.3	Pharmacostatistical Model Development . . . . .	217
6.4.4	Covariate Inclusion . . . . .	224
6.4.5	The Stochastic Approximation Algorithm . . . . .	235
6.5	Exercises . . . . .	237
	References . . . . .	237
<b>7</b>	<b>Clinical Trial Simulation . . . . .</b>	<b>239</b>
7.1	Terms and Concepts . . . . .	239
7.2	Execution of Trial Protocols . . . . .	240
7.3	A Basic Clinical Trial Simulator . . . . .	243
7.4	Example: ESA Replacement . . . . .	250
7.4.1	PK Model . . . . .	250
7.4.2	PD Model . . . . .	251
7.4.3	Computational Model . . . . .	253
7.4.4	Trial Simulations . . . . .	264
7.5	Exercises . . . . .	271
	References . . . . .	271
<b>8</b>	<b>Graphics-based Modeling. . . . .</b>	<b>273</b>
8.1	SimBiology. . . . .	273
8.1.1	Model Components . . . . .	273
8.1.2	The SimBiology Desktop. . . . .	275
8.1.3	The SimBiology Command Line. . . . .	284

8.2	Simulink . . . . .	290
8.2.1	One-Compartment Model (Simulink). . . . .	291
8.2.2	Two-Compartment Model (Simulink) . . . . .	295
8.2.3	Example: Ibandronate . . . . .	299
8.2.4	Simulink Model as a Multi-process . . . . .	299
8.2.5	Simulink Models: Summary and Impact . . . . .	303
8.3	Exercises . . . . .	303
	References . . . . .	303
<b>9</b>	<b>Outlook . . . . .</b>	<b>305</b>
9.1	A Visionary Concept . . . . .	305
9.2	R&D Improvements Based on Mathematical Modeling . . . . .	306
9.2.1	In Silico Target Validation. . . . .	307
9.2.2	Model-based Drug Development. . . . .	308
9.2.3	Systems Biology . . . . .	308
9.2.4	Quantitative Systems Pharmacology . . . . .	309
9.3	Prerequisites and Obstacles. . . . .	309
9.3.1	Education. . . . .	309
9.3.2	Computational Tools . . . . .	310
9.3.3	Mindset . . . . .	310
	References . . . . .	312
	<b>Appendix A: Hints to MATLAB Programs . . . . .</b>	<b>315</b>
	<b>Appendix B: Solution to Exercises . . . . .</b>	<b>349</b>
	<b>Index . . . . .</b>	<b>395</b>

# Abbreviations

AD	Alzheimer's disease
ADP	Adenosine diphosphate
AIDS	Acquired immune deficiency syndrome
AP	Action potential
ATP	Adenosine triphosphate
AUC	Area under the curve
BACE	Beta-secretase
BC	Basket cell
BID	Bis in die (twice daily)
BMD	Bone mineral density
BSV	Between subject variability
CERN	Conseil européen pour la recherche nucléaire
CI	Confidence interval
CKD	Chronic kidney disease
CNS	Central nervous system
COMT	Catechol-o-methyltransferase
CTS	Clinical trial simulation
DA	Dopamine
DAE	Differential algebraic equation
DAT	Dopamine uptake transporter
DDE	Delay differential equation
DDMoRe	Drug disease model resources
DNA	Deoxyribonucleic acid
DP	Disease progression
EEG	Electroencephalography
EIH	Entry-into-human
EM	Expectation-maximization
EMA	European Medicines Agency (formerly EMEA)
ESA	Erythropoietin stimulating agent
FDA	Food and Drug Administration
fMRI	Functional magnetic resonance imaging
GIT	Gastrointestinal tract
GOF	Goodness of fit
GUI	Graphical user interface

Hb	Hemoglobin
HH	Hodgkin-Huxley
HIV	Human immunodeficiency virus
ICH	International conference on harmonization
ICI	Imperial Chemical Industries
IDR	Indirect response
IMI	Innovative medicine initiative
IO	Input–output
IV	Intravenous
IVP	Initial value problem
K-PD	Kinetics of pharmacodynamic response
LS	Life span
MAD	Multiple ascending dose
MATLAB	Matrix laboratory
MBDD	Model-based drug development
MTD	Maximum tolerated dose
NDA	New drug application
NIH	National Institute of Health
NLME	Nonlinear mixed effect
NLMEM	Nonlinear mixed effect model
NPD	Naïve pooled data
OAD	Once a day
ODE	Ordinary differential equation
OFV	Objective function value
OS	One stage
PBPK	Physiologically based pharmacokinetics
PD	Pharmacodynamics
PDE	Partial differential equation
PK	Pharmacokinetics
PMO	Postmenopausal osteoporosis
PO	Per os (oral)
POC	Proof of concept
QSP	Quantitative systems pharmacology
R&D	Research and development
RBC	Red blood cell
RNA	Ribonucleic acid
RNG	Random number generator
SAD	Single ascending dose
SAEM	Stochastic approximation of expectation-maximization
SB	Systems biology
SBML	Systems biology mark-up language
SC	Subcutaneous
SDE	Stochastic differential equation
SE	Standard error
SIAM	Society for industrial and applied mathematics



SQV	Saquinavir
TID	Ter in die (three times daily)
TS	Two stage
TT	Turnover time
US	United States
VK	Virus kinetics
VPC	Visual predictive check
WHO	World Health Organization

# Document Conventions

Mathworks	Normal text in the document
<PSI>	Indicates placeholder
<i>Introduction to...</i>	Important terms, book/chapter titles
<b>ode45</b>	MATLAB source code text
<a href="http://www.mathworks.com">http://www.mathworks.com</a>	Internet address
<b>A, b</b>	Matrices, vectors
$X, \zeta$	Mathematical functions, variables
.	Multiplication
o	Element-wise multiplication of two vectors
% .....	A block of function statements already specified
% .....	
% .....	
<b>%OUTPUTNAME = ...</b>	A help block already specified

All programs (scripts and function) presented in this book are available on the MATLAB Central portal, at the following link: <http://www.mathworks.ch/matlabcentral/fileexchange/40813-rgdspracticalguide>

# Chapter 1

## Background of Pharmacologic Modeling

The pharmaceutical industry, which emerged during the second half of the nineteenth century, operates nowadays in a sensitive environment. Providing patients with differentiated medicines that can be afforded by health care systems is proving more and more difficult. Today's medical treatment concepts are largely determined by drugs that act on predefined targets whereas previously a more phenomenological approach was taken. Drug development is a lengthy and resource-intensive process subject to tight regulatory supervision with emphasis on the safety of medicines. Due to high attrition rates at the different stages of drug research and development (R&D), pharmaceutical productivity is declining. To support the complex process of turning a treatment concept into a marketable drug, industry specialists increasingly advocate the consistent application of mathematical modeling based on realistic models of pharmacokinetics, pharmacodynamics, and disease biology.

Mathematical modeling and computer simulation are methodologies that have proven their value on countless occasions in many disciplines. The pharmaceutical industry adopted modeling approaches first for time versus drug concentration data in individual subjects, then time-response data. Today, models are built for data on subject populations and biological systems. Based on this information, we can perform simulations in advance of a clinical trial that help to optimize the trial design for a given purpose. Mathematical models have the ability to integrate information. They can also generate knowledge and understanding. Yet they should never be regarded as a substitute for one's brain but rather as one element contributing to rational discussion of a specific issue.

### 1.1 The Pharmaceutical Value Chain

*Dosis facit venenum* (Paracelsus, 1493–1541)

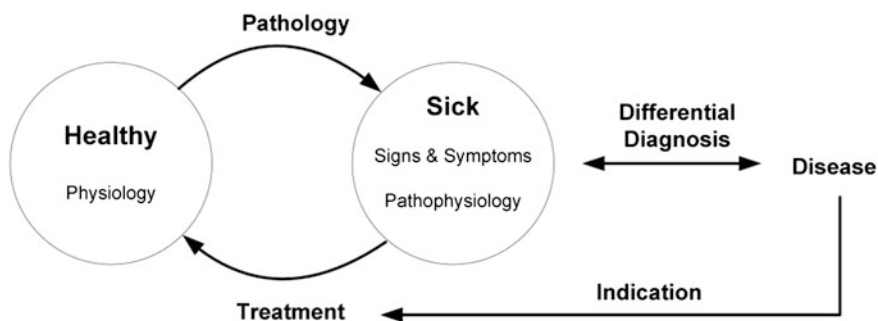
A *pharmacon* (Greek: *φάρμακον*) is a biologically active substance with beneficial or detrimental effects. Drugs or pharmaceuticals are molecules given to humans to

mitigate an ailment or cure a disease. They are predominantly developed, produced, and marketed by the pharmaceutical industry.

### *1.1.1 Maturation of Medical Concepts and Drug Treatment*

Since ancient times, substances have been used to counteract medical symptoms, such as pain, or to improve physiologic functioning beyond normal limits, as currently exemplified by the many doping cases. Drug treatment rationale has changed in line with evolving theories of human physiology. From Hippocrates (ca. 460–370 BC) until the eighteenth century, four humors (blood, black bile, yellow bile, and phlegm) were considered to constitute a healthy human state when in balance, and a diseased state when in imbalance. Salt water, for example, was applied as an emetic in order to correct for an elevated humor. In the middle of the nineteenth century, the German pathologist Rudolf Virchow (1821–1902) created a new disease paradigm: cellular pathology. Cells, the fundamental units of all living, were considered as the origin of diseases. Around the beginning of the twentieth century, Paul Ehrlich (1854–1915), a German physician and biologist, added an important idea to the notion of cellular pathology. He postulated that (chemo)-receptors in the protoplasm of the cell chemically bind certain substances thereby eliciting their pharmacological effects. Hence, cellular pathology and receptor theory became the cornerstones of modern medicine and pharmacology.

Nowadays, medicines play an undisputable and important role in the prophylaxis and treatment of disease. Figure 1.1 presents a general scheme of the processes involved in treating signs and symptoms as expressions of disease. A previously healthy person may notice symptoms (nausea, headache, etc.) prompting him to consult a physician. The physician will explore the person's medical status by assessing objective measures (signs) such as vital signs (blood pressure, pulse rate, etc.), and laboratory values (blood concentrations of a number



**Fig. 1.1** Processes involved in treating disease. Pathological signs and symptoms may lead to the diagnosis of a disease and/or to symptomatic treatment. Treatment indicated for the disease is given to restore the biological system to health. It may itself produce signs and symptoms

of substances such as sodium, potassium, glucose). In a differential diagnostic process, the physician has to weigh all relevant information (signs, symptoms, patient's medical history, etc.) to reach a diagnosis usually coupled to an established disease entity. It is important to recognize that the notion of "disease" changes over time. In an effort of standardization, the World Health Organization (WHO) maintains a regularly updated medical classification of diseases comprising more than 10,000 codes.

For a large number of diseases, treatment modalities have been developed and it is the duty of the physician to determine a treatment indication (drug, surgery, radiation, etc.). Successful treatment restores the patient to health or at least to a less diseased state. In addition to this rational treatment process, symptomatic treatment can be administered that falls outside differential diagnosis process and is based on signs and symptoms alone.

Figure 1.1 also emphasizes the role of disease pathology. The process by which health turns into disease is the subject of etiology (why) and pathogenesis (how), both of which are areas of pathology. Normal body function is the subject of physiology, and diseased body function that of pathophysiology. Understanding the pathophysiology of a disease is essential for its treatment.

### ***1.1.2 Emergence of the Pharmaceutical Industry***

The pharmaceutical industry emerged in the second half of the nineteenth century, both in Europe and the US [1]. Companies were founded with different goals in mind. These included the production and marketing of standardized dosages with and without purification of natural products originating from minerals, herbs, and animals (US: Squibb (1858), Wyeth (1860), SmithKline (1865), Eli Lilly (1876), Upjohn (1886), Bristol-Myers (1887); Europe: Beecham (1842), Schering (1851), F. Hoffmann - La Roche (1896)); bulk production of fine chemicals for delivery to pharmacies (US: Pfizer (1849), Merck (1891)); trading in healthcare products and/or food (US: Nathan (1873), Johnson & Johnson (1886)); and the commercialization of drugs largely synthesized by organic chemistry from coal tar (Europe: Geigy (1857), Ciba (1859), Bayer (1863), Hoechst (1863), Sandoz (1888)). After the foundation of the Swedish companies Pharmacia (1911) and Astra (1913), the pharmaceuticals infrastructure was completed so that no new major company emerged thereafter.

The American Civil War (1861–1865), World War I (1914–1918), and especially World War II (1939–1945) created big demands for painkillers, antiseptics, and antibiotics. The discovery of penicillin by Alexander Fleming (1881–1955) in 1928 triggered research efforts and bulk production of antibiotics, financially supported by the US and British governments, and created large incomes for those companies with the respective technological resource (Pfizer, Merck, Squibb, and Glaxo (created in 1947 from Nathan)).

Between 1945 and 1965 drugs were discovered for a large number of diseases (such as antipsychotics, antineoplastic agents, antihypertensives), mainly by serendipity. These were prescription drugs (i.e., obtainable by a doctor's prescription only) and were generally reimbursed by state or health care insurance funds, thus providing companies with a sound financial basis.

By around 1965 the number of new products had declined [2], with many companies diversifying into other technologies (such as medical instrumentation) or expanding their nonprescription (over the counter) business. This lasted until the mid-1970s when companies largely recognized that their future lay in prescription drugs and therefore abandoned diversification.

New biosciences emerged in the 1970s (microbiology, biochemistry) and 1980s (biotechnology, molecular biology, genetic engineering). They had a profound impact on the pharmaceutical industry in that they offered new drug discovery routes. Companies unable to create the requisite knowledge base, especially in biotechnology, looked for competent partners. A large number of biotech startups emerged, notably Genentech (1976), Biogen (1978), Amgen (1980), Genzyme (1981), Immunex (1981), and Chiron (2006).

Mergers and acquisitions have since shaped the current chapter in the history of the pharmaceutical industry, driven by the reduction in R&D budgets—the new biosciences have yet to create many more new drugs [2]—and growing financial pressure from health care providers. Table 1.1 summarizes the mergers and acquisitions involving the 10 leading prescription drug companies.

More information on pharmaceutical mergers and acquisitions is available on the internet [25–37].

### 1.1.3 How are Drugs Discovered?

Before the emergence of the pharmaceutical industry, new drugs resulted mainly from the purification of natural products reputed for their clinical effects.

**Table 1.1** Major mergers and acquisitions in the pharmaceutical industry to 2011

Company	Mergers and acquisitions
Astra	Zeneca
Bristol-Myers	Squibb
Eli Lilly	–
Glaxo	SmithKline, Beecham, Wellcome
Johnson & Johnson	Janssen
Merck	Schering-Plough, Sharp and Dohme
Novartis	Sandoz, Ciba, Geigy, Chiron
Pfizer	Wyeth, Pharmacia Upjohn, Warner Lambert, Parke-Davis
Roche	Genentech, Boehringer Mannheim, Syntex
Sanofi	Genzyme, Aventis, Rhone-Poulenc Rorer, Synthelabo, Hoechst Marion

For example, Friedrich Sertuerner (1783–1841) isolated the painkiller morphine from opium in 1815 [3]. With the emergence of the pharmaceutical industry, purification of compounds still remained a major source of new drugs. Examples are aspirin by Bayer in 1899, and insulin by Eli Lilly in 1922. A major industrial success was the mass production of penicillin whose antibacterial properties were accidentally discovered by Fleming in 1928. About 10 years later, Howard Florey (1898–1968) and Ernst Chain (1906–1979) undertook clinical tests with penicillin and paved the way for its broad application following World War II. In the 1950s, a number of drugs were discovered for mental disorders (depression, schizophrenia, anxiety). All were largely the result of chance: chemical structures were being investigated for other purposes when alert biologists or physicians noticed relevant effects on the central nervous system (CNS) in animal and/or clinical testing. The 1960s inaugurated a more rational approach, drug discovery by design, thanks to the work of James Black (1924–2010) at Imperial Chemical Industries (ICI, from which Zeneca was created in 1993). Based on the pharmacologic characterization of epinephrine on the circulatory system, he developed the antihypertensive propranolol, one of the world's best-selling drugs. In the 1970s, at Smith Kline & French, he went on to develop the antiulcer compound cimetidine based on his insight into the role of histamine in this disease. Decades later, with the advent of genomics, it became possible to genetically engineer microorganisms so that they produced protein drugs (such as insulin and erythropoietin).

Today's researcher follow two screening strategies, one target-based, the other phenotype-based. Both rely on automated high-throughput experimental procedures involving genes, proteins, and cells. Target-based screening explores how a large number of chemicals interfere with a molecular target (gene, protein) thought to be related to a disease. However, from identifying a target to discovering a compound that acts on it in the desired way can be a lengthy and expensive process, as exemplified by beta-secretase 1 (BACE1) inhibition, a target for Alzheimer's disease (AD) [4]. In some instances, molecular modeling—the computational representation of molecular interactions—has been successfully applied to the design and production of new chemical entities (NCE) [5]. Suggestions for targets, mainly proteins, often come from academic and biotech research. Their number has rapidly increased (especially since the genomics revolution), but at the same time their validation often remains unclear. Validation means providing evidence that engaging the target will produce benefit in patients with the disease.

Phenotype screening investigates the effects of compounds on cells and tissues. Its primary concern is not the molecular mode of action of the test chemical, but rather with its impact on the overall cell function (cell growth, etc.) thought to be implicated in the pathology of a disease. Compounds generated by target-based screening can usually be optimized more easily than those from phenotype screening where the molecular mode of action needs to be elucidated after screening.

Any promising compound, or hit, is then refined to meet requirements regarding target specificity, i.e., avoidance of interaction with other targets, target sensitivity, i.e., interacting with the target in low, nanomolar concentrations, and favorable

animal pharmacokinetics. It is often unclear which of many hits to move forward into development. The ability to rank compounds by clinical potential at this early stage would be highly desirable.

The weak point in current screening methods is that the target-disease relationship is often insufficiently explored [6]. The important question for the pharmaceutical industry is to know as soon as possible whether a hit—a molecule that binds to a target—has drug potential and under which conditions. Working on poorly validated targets carries the risk of attrition at a later stage of development, most likely when the drug has to demonstrate that it is clinically effective. Companies trying to be the first in delivering a compound with a novel mode of action (“first in class”) take a larger risk of attrition than those trying to optimize existing therapies (“best in class”).

Two types of agents are currently being pursued as drugs: low-molecular weight agents derived from medicinal chemistry via defined routes of synthesis; and high-molecular weight agents (i.e., antibodies) derived by biotechnology and thus exhibiting slightly varying compositions.

### 1.1.4 How are Drugs Developed?

The goal of drug development is to turn a molecule (compound, drug, or hit) delivered by the discovery process into a medicine. This involves providing drug regulators with a disease indication and a respective dosage regimen. The drug should be safe, i.e., it should not have harmful effects, and effective, i.e., it should make the patient better. Drug development is organized as shown in Fig. 1.2.

#### Preclinical Research

After a lead compound has been identified, its suitability as a drug therapy is evaluated in a series of tests that focus on biology, pharmacology, toxicology, and pharmacokinetics. Preclinical research includes laboratory work and studies to

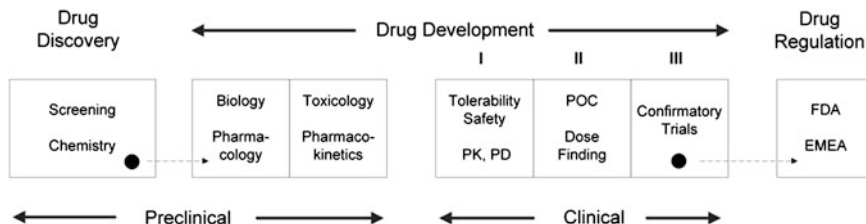


Fig. 1.2 The drug development process from screening to submission



determine the feasibility of manufacturing the compound in a form that can be administered to humans, ideally as tablets.

An important aspect of drug treatment is to avoid undue risk due to adverse effects. Benefit has to be carefully balanced against risk. Molecules that have passed pharmacological testing and are considered potential candidates for human testing undergo thorough multispecies toxicology screening, together with pharmacokinetics studies to characterize their absorption, distribution, metabolism, and elimination (detailed in [Sect. 4.1](#)).

## **Clinical Research**

The clinical research process consists of three main phases, each with unique characteristics and priorities. As testing progresses from Phase I through Phase III, it expands knowledge and understanding of a drug's attributes, safety, and efficacy. Both drug companies and regulatory authorities carefully assess the data gathered during each phase to reach decisions as to a drug's safety and efficacy.

### **Phase I**

Entry-Into-Human (EIH): maybe the most keenly anticipated phase of drug development as it is the first time that the molecule is given to man. EIH studies are generally designed as single ascending dose (SAD) trials. Small numbers of subjects take the drug only once, and are closely monitored for subjectively experienced and objectively measurable drug effects. The next higher dose is administered to the next group of subjects only if there are no safety concerns. SAD trials provide information about a drug's maximum tolerated dose (MTD), i.e., the dose defining an upper dose limit in later trials. In general, SAD studies are conducted in healthy volunteers, but for certain drugs, e.g., cancer agents, in patients only. The starting dose is calculated from preclinical safety and pharmacokinetic studies. Preclinical safety results may also determine the highest acceptable dose or drug concentration in humans. The next step is to conduct a multiple ascending dose (MAD) trial designed and evaluated like an SAD trial except that repeated doses are given, say once daily over two weeks, to each subject (healthy volunteer or patient). MAD trials deliver information on the MTD for a given dosing frequency and treatment duration, and on potential time-dependent drug characteristics (e.g., drug accumulation, tolerance). In general, the results of SAD studies inform the selection of dose strengths in MAD trials.

### **Phase II**

Proof of Concept (POC) and dose finding: Phase I studies often give no indication as to whether a drug will hold its promise for the selected target and disease. How much target engagement (e.g., receptor occupancy, enzyme inhibition, transporter

blockade) is needed to produce a desired effect in a patient population with a given disease? Which dosage regimen will result in adequate target engagement? If the disease is heterogeneous, which disease subpopulation will best respond to the drug? And will the drug still be safe? These are typical questions to be answered in Phase II. POC studies explore drug-disease mechanisms by making use of biomarkers indicative of drug action. For example, glucose would be a biomarker for the effect of an antidiabetic drug. The investigation of effective dosage regimens is another task in Phase II. Often, these studies are done in patient populations that are too small for deriving statistically significant results. Ultimately, Phase II should provide dosage regimens requiring confirmation in Phase III trials.

### **Phase III**

Phase III consists of huge studies, often comprising up to several thousand patients, recruited from dozens of countries across more than one continent, consuming several 100 million dollars and taking many years to complete. Their goal is to confirm that an experimental drug regimen is safe and effective. Phase II dosage regimens are administered unless there are compelling reasons to the contrary. Such reasons might be generated from clinical trial simulation (CTS). Due to the large patient numbers, it is reasonable to investigate sources of inevitable variability in drug response (population analyses). Bodies such as the Food and Drug Administration (FDA) in the US and European Medicines Agency (EMA) in Europe regulate Phase III studies.

### **Regulation**

Before a drug can gain market authorization, results from preclinical and clinical research need to be compiled and submitted to regulators according to their requirements. In general, this is not the regulator's first dealing with the drug concerned. Companies often solicit advice on different issues (safety, efficacy assessments, etc.) during the drug development program.

### **Post-approval**

Post-approval drug safety data will be systematically collected and analyzed, e.g., to get information about rare adverse events. Another goal is to compare the new drug with alternate therapies to establish a competitive advantage. Such studies can also help to raise awareness of the new product in the medical and scientific communities. Once the drug is approved for market authorization, there will be a period of several years during which the drug can be sold under patent protection, i.e., no imitator can reproduce the drug and sell it for a cheaper price. This period ends 18 years after the patent was filed (around molecule discovery) and results in 6–8 years exclusivity, depending on the preceding development time.

What can be said about the process described above? For about 50 years it produced a flourishing industry. Without caring too much about molecular targets, pharmaceutical scientists stumbled, often by chance, on compounds that worked in a number of diseases. Subsequently, these compounds were more and more refined as their role in the body became transparent. The targeted drug approach emerged but in parallel with a decrease in productivity [7]. A number of counter-measures have been proposed, such as testing the mechanism of drug action in Phase I and using adaptive study designs allowing more flexible dose selection. One may argue that our knowledge of targets and disease is too fragmentary to deliver rational drug development programs. The advent of the genomics era did not reverse this deficit, but the contrary appears to be the case. Our knowledge of biological systems is exponentially increasing while our ability to integrate all these facts is decreasing. Table 1.2 offers a gross summary of key figures for preclinical and clinical development phases.

### 1.1.5 Drug Regulation

Drug regulation was first established in the US in 1906 when the FDA had to supervise substances scheduled for ingestion by humans. In 1938, the FDA ruled for the first time that new drugs had to demonstrate safety before they could be sold and that certain drugs could only be taken if prescribed by a medical expert. This supported the pharmaceutical industry which was better able to cope with these requirements than less legitimate drug producers.

In Europe, there was practically no regulation of prescription drugs until 1961. This changed with the thalidomide disaster [8]. Promoted as a sleeping pill and morning sickness remedy, thalidomide caused over 10,000 birth defects. Drug regulation changed thereafter in that manufactures were required to adequately describe the risks and benefits of prescription drugs.

In 1962, the FDA added a new amendment requiring proof of efficacy and accurate disclosure of side effects for new medications (the Kefauver Harris

**Table 1.2** Drug development stages, attrition rates, duration, and costs. Numbers refer to one entity entering the respective phase. For example, to move a single product through Phase I takes 1.5 years, costs \$15 M and succeeds in 54 % of cases. The probability of a successful launch is the product of all success rates, and equals 7 %. Thus, for one successful launch 14 compounds need to enter preclinical development. If this is taken into consideration, development costs rise from \$260 M to \$784 M. This is the price paid for attrition (= 1-success rate) at the different development stages [23]

	Biology pharmacology	Toxicology DMPK	Phase I	Phase II	Phase III	Submission launch
Success rate	85 %	69 %	54 %	34 %	70 %	91 %
Duration (y)	2	1	1.5	2.5	2.5	1.5
Costs (\$M)	10	5	15	40	150	40

Amendment). Likewise, the 1964 Declaration of Helsinki put greater ethical demands on clinical research, clearly cementing the difference between production of scientific prescription medicines and other chemicals. The Hatch-Waxman Act of 1984 regularized generic drug production in the US.

Drug manufacturers must follow the laws of each country in which they want to conduct research and/or market their products. Manufacturers who operate in the international market usually follow the most stringent regulations. Since the 1990s, the International Conference on Harmonization (ICH), an organization of both regulatory agencies and pharmaceutical manufacturing organizations, has worked to rationalize and harmonize regulations throughout the world. For more information, see [38].

## 1.2 Modeling and Simulation

*Science is built up of facts, as a house is built of stones;  
but an accumulation of facts is no more a science  
than a heap of stones is a house (Henri Poincare, 1854–1912)*

### 1.2.1 Examples of Models

Broadly speaking, modeling is undertaken for a specific purpose. Examples of the power and versatility of the modeling approach might include:

- A town map as a concise representation of the layout of streets and buildings
- A scale model as a miniaturized representation, e.g., to test a car's behavior in a wind tunnel and optimize its profile
- The DNA double helix, discovered by James Watson (1928) and Francis Crick (1916–2004), was a conceptual molecular model accounting for many mysteries of biology
- The model of the atom by Niels Bohr (1885–1962) is a mathematical model based on a few physical assumptions that explains why electromagnetic radiation is emitted at discrete values
- The mathematical model named after Alan Hodgkin (1914–1988) and Andrew Huxley (1917–2012) showed how neuronal ion channel dynamics relate to the propagation of membrane potentials (details in [Sect. 5.2](#)).

All the above models simplify the complex real-life phenomena they try to describe or to explain. Some are concrete (town map, scale model, double helix), others are abstract (the Bohr and Hodgkin-Huxley models). Interestingly, the DNA helix and ion channels were experimentally confirmed decades after their original presentation.

### 1.2.2 Modeling and Scientific Understanding

There is an ample evidence that the current productivity of pharmaceutical R&D is insufficient to sustain the business model. Our understanding of disease biology remains relatively rudimentary causing current R&D to over-rely on serendipity and empiricism. As will be shown below, (mathematical) modeling can aid understanding by providing biologically realistic computer models. These models can further be used for the consistent description of experimental data, the planning of new experiments to support modeling assumptions, and the prediction of dose-related drug effects at early and late stages of drug development.

How is modeling related to scientific understanding? According to de Regt [9], a phenomenon can be understood if there is a theory to represent it. Mathematical models based on theoretical subject matter have provided powerful representations. Electrodynamics and biophysics theory enabled Hodgkin and Huxley to formulate a mathematical model that closely represented the experimental data. It is important for the theories involved to be intelligible to scientists, who should be able to recognize the qualitatively characteristic consequences of the theories themselves without performing exact calculations [9].

### 1.2.3 Mathematical Models

*Essentially, all models are wrong, but some are useful*  
George E.P. Box (1919–2013)

The above citation by Box [10] is one of the most frequently quoted about mathematical models, not only within the pharmaceutical modeling world. To appreciate why all models are wrong, we need to know what (mathematical) models are and how they are developed.

Any model, mathematical or otherwise, is a substitute for something. A model aims to describe aspects of reality by consciously abstracting unnecessary detail and making crucial assumptions. Consequently, models are approximations or caricatures of reality (and therefore wrong), but may be able to describe the phenomena or system for which they have been built (in which case such models are useful).

To take a simple example: we wish to describe the behavior of drug concentrations in the body over time after single intravenous dosing. We simplify the problem by assuming that (a) the body can be treated as a single container, (b) drug input is instantaneous, (c) the drug is immediately and homogeneously distributed throughout the body, and (d) elimination obeys a first-order process, i.e., the drug elimination rate is proportional to the amount in the body. These may be crude assumptions but the resulting models have often proven to deliver useful approximations.

Mathematical models are characterized by equations that describe relationships between variables. In this book, we concentrate on equations that describe the time dependency of quantities of interest such as drug concentrations in the body. As we explain below, models can be classified into different types depending on their formal mathematical structure.

### Linear versus Nonlinear Models

A model  $y = f(x, p)$ , where  $x$  is the independent variable and  $p$  a parameter, is linear with regard to a parameter  $p$  if it can be expressed in the form

$$f(x, p) = a(x) + b(x) \cdot p \quad (1.1)$$

with functions  $a$  and  $b$  that do not depend on  $p$ .

A more readily checkable criterion is the following:

$$\frac{d^2 f}{dp^2} = 0 \quad (1.2)$$

Example: The model

$$y = \frac{a \cdot x}{b + x}$$

is linear with respect to  $a$ , but nonlinear with respect to  $b$  or  $x$ .

Linear models are mathematically much better explored than nonlinear models. This proves to be useful when considering linear approximations to nonlinear models.

### Dynamic versus Static Models

Dynamic models describe dependent variables as functions of time,  $t$ , and model parameters,  $p$ , either explicitly (algebraic equation) or implicitly (differential equation), i.e.,

$$\text{explicitly : } y = f(t, p), \quad \text{implicitly : } \frac{dy}{dt} = f(t, y, p), y(t_0) = y_0 \quad (1.3)$$

Static models have no time dependency. Sometimes they are derived from dynamic models at equilibrium, i.e., by setting

$$\frac{dy}{dt} = 0 \quad (1.4)$$

Example: The model

$$\frac{dy}{dt} = a - b \cdot y ; \quad y(0) = 0 \quad (1.5)$$

is a dynamic model which at equilibrium yields the static relationship  $y_{eq} = a/b$ , independently of the initial condition.

## Deterministic versus Stochastic Models

Deterministic models always provide the same output for the same input. Stochastic models contain one or more random variables and provide for the repeatedly identical inputs generally different outputs. The value, or realization, a random variable takes is the inverse of its cumulative probability density function at a randomly (!) chosen argument between 0 and 1. Random number generating algorithms are well established and widely available in software packages (such as MATLAB). Random variables usually occur in regression models that estimate parameters from data, as detailed in [Chap. 6](#) (*Population Analysis*) or in simulation models that create data from a population model, as worked out in [Chap. 7](#) (*CTS*). The most important density function is normal or Gaussian density,  $N(\mu, \sigma^2)$ , defined as

$$f(x) = \frac{1}{\sigma \cdot \sqrt{2 \cdot \pi}} e^{\frac{-(x-\mu)^2}{2 \cdot \sigma^2}} \quad (1.6)$$

and characterized by mean  $\mu$  and variance  $\sigma^2$  (Carl Friedrich Gauss, 1777–1855).

Example 1: The equation

$$y = f(t, p) + \varepsilon ; \quad \varepsilon \sim N(\mu, \sigma^2) \quad (1.7)$$

defines a stochastic model.

Example 2: The equation

$$\frac{dy}{dt} = f(t, p, \varepsilon) ; \quad \varepsilon \sim N(\mu, \sigma^2) \quad (1.8)$$

is a stochastic differential equation (SDE), where the stochasticity occurs within the function  $f$  defining the right-hand side of the equation. SDEs are used to describe processes that are inherently random, e.g., diffusion.

In the following, we will concentrate on nonlinear and dynamic models of both deterministic and stochastic type. [Chapter 3](#) (*Differential Equations in MATLAB*) details their specification further.

### 1.2.4 The Modeling Process

Before describing the modeling process in more detail, we should distinguish between two categories of models, unrelated to the formal type of mathematical equations: models of systems and models of data [11].

Models of systems, also called mechanistic models, make assumptions about the mechanisms underlying system behavior. They refer to different natural sciences, such as biophysics, biochemistry, physiology, and pharmacology. These models are in general nonlinear, dynamic, and high-dimensional (in terms of variables and parameters). An example is the Hodgkin-Huxley model presented in [Sect. 5.2](#). Models of systems are applied to data mainly to check whether the model reflects trends in the data. Parameters are estimated with caution to avoid endangering the biological soundness of the model. Depending on their mechanistic detail, mechanistic models may be very hard to develop. But they offer a distinct advantage over empirical models in that they are potentially suited to extrapolation beyond the data with which they were built. As will be described in more detail in [Chap. 5 \(Drug-Disease Modeling\)](#), system models related to disease and disease progression are of ultimate value for the pharmaceutical industry.

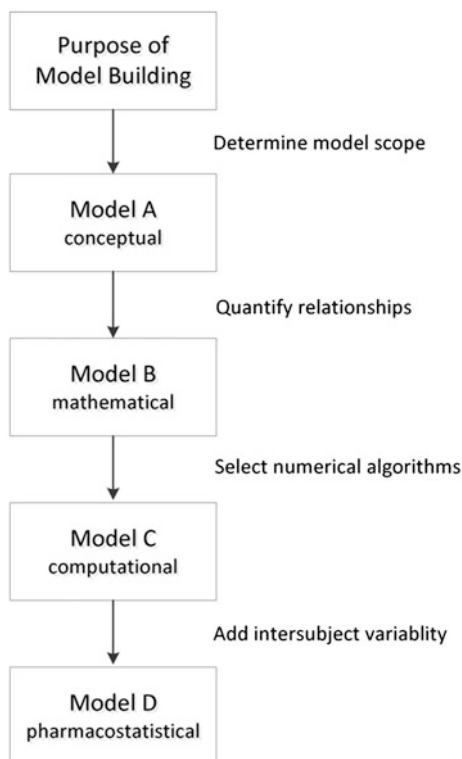
Models of data, also called empirical models, come in two versions. The first are purely empirical models that make no assumptions about the system or physical processes that could have generated the data. These kinds of models are often linear and static. Although they generally provide reasonable descriptions of the data, they readily fail when used for predictions outside the scope of the data. Even more importantly, they prove unable to generate testable biological hypotheses. Typical models include splines, polynomials, and the sum of exponentials. The second type are semi-empirical or semi-mechanistic models that let the data decide how many mechanistic elements (e.g., PK compartments) are to be assembled ([Sect. 4.1](#)). In contrast to models of systems, models of data are subject to (nonlinear) regression to determine their parameter values. Semi-empirical models are currently the most widely applied models in pharmacometrics. They may take days to months to develop.

Figure 1.3 describes the general model-building process.

Model-building is a collaborative and iterative process between individuals with different mindsets. Typical stakeholders include biologists, toxicologists, pharmacologists, clinical pharmacologists, pharmacists, biomarker experts, and physicians. The first step is to create a high level of understanding among the parties involved about the scope of the modeling, its potential simplification, and the basic relationships between elements of interest. This yields conceptual model A.

Guided by the conceptual model, corresponding mathematical descriptions are formulated. Variables and parameters are chosen, and the model is put in context with its purpose. The outcome is mathematical model B. Before proceeding to a computational form of model B, its mathematical properties should be evaluated, such as parameter sensitivity, identifiability, and qualitative behavior (see [Chap. 3](#)). In order to create the computational model, hardware and software needs to be





**Fig. 1.3** The model-building process. Starting from the modeling purpose, a conceptual model is created (Model A) containing essential model elements and their possible associations. Next, relationships between model elements are quantified, yielding a mathematical description (Model B). Model B deserves some investigation regarding its mathematical characteristics before it is implemented as a computational model (Model C). Finally, if the model is needed in a population context, intersubject variability needs to be added, yielding a pharmacostatistical model (Model D). Each modeling stage requires careful examination of its assumptions, with return to a previous model-building stage always being possible

considered, along with numerical methods, their stability, and computational runtime. This eventually brings us to computational model C that can be used to represent model outcomes. This model should be demonstrated to the stakeholders to benefit from their feedback. Depending on the modeling purpose, it may be necessary to extend model C to a pharmacostatistical or population parameter estimation model D (see [Chap. 6](#)). This is a two-level model which on the first (deterministic) level describes mathematical relationships for a single experimental unit (e.g., a single patient), and on the second level the variability of model parameters across experimental units. To create a parameter estimation model, we have to define the data to be used, the model parameters to be estimated, and the random variables to be added to the mathematical model. Furthermore, we need to

specify an objective function. Execution of an estimation model requires selection of an optimization algorithm. After execution, proper convergence, quality of parameter estimates (value and precision), and overall goodness of fit have to be checked. Should these checks yield acceptable results, we arrive at pharmacostatistical model D, required for CTS ([Chap. 7](#)).

It is important to note that the modeling process is iterative and reuses data and assumptions. Data may be of questionable quality and assumptions may be somewhat unsupported. In biology, natural laws are mathematically less developed than in physics, thus compounding the weakness of such models. Therefore, it is prudent to avoid language such as “the model decided”, but rather to use the modeling process to enhance judgment and intuition [[12](#)].

### ***1.2.5 Application of Models***

The pharmaceutical industry develops mathematical models for the (main) purposes described below.

#### **Summary**

A concise summary of relationships between observed variables is required in the form of a parametric model condensing raw data into a set of parameters. This can be applied to an individual subject ([Chap. 4](#)) or—currently the main application (population PK/PD/disease modeling, [Chap. 6](#))—to a population of subjects. Population parameter estimates may help to decide whether to adjust a dosage regimen for a given covariate ([Chap. 6](#)).

#### **Prediction**

This comprises the prediction of pharmacokinetic behavior for different candidate molecules during preclinical development. A standard modeling application is the forecasting of human pharmacokinetics from animal results based on physiologic pharmacokinetic models ([Chap. 4](#)), in order to support dose selection for trials in man. Other predictions include repeated dose concentrations based on single dose information, and, oral administration concentrations based on intravenous data.

#### **Clinical Trial Simulation**

Clinical trials can be interpreted as random experiments and their outcome as a random variable. A clinical trial simulation (CTS) ([Chap. 7](#)) is a Monte Carlo simulation with an underlying population model to determine the distribution of

this random variable before the trial is actually performed. CTS helps to explore the many aspects of trial procedures and may contribute to a more appropriate design.

## Optimal Design

Assessment schedules in clinical trial protocols are often empirically designed without taking into account the pharmacokinetics or pharmacodynamics of the drug. A less empirical way to choose sampling times for quantities of interest is to use an optimality criterion (such as minimal parameter estimation error for a selected parameter) and the model-derived Fisher Information Matrix (Ronald Fisher, 1890–1962) [13]. Another type of optimal design occurs in chemical engineering where the goal is to maximize yield or minimize costs under predefined constraints [14–16].

## Control

Keeping a dynamic system under control means being able to adjust input quantities (e.g., a dosage regimen) in order to maintain response variable(s) within a certain range. Trying to control a dynamic system without a mathematical model may result in the “hunting” phenomenon nicely exemplified by the treatment of anemia with erythropoietin (Chap. 7) [17]. More elaborate control systems apply closed-loop architectures, where the system itself provides feedback control on its input. A fascinating, but as yet unrealized, example of such a system has been described for the treatment of refractory major depressive disorder [18].

## Exploration

Use of a computational model to answer “what if” questions provides an excellent learning tool. For complex models, the consequences of changing parameters or initial conditions can be difficult to anticipate. Such model explorations deepen our understanding of the system (Chap. 5). A famous example of such a model was given by 1960 investigation by the Club of Rome into the behavior of the Earth’s finite resources assuming exponential growth for a number of dimensions (world population, industrial production) [19].

The above goals can be combined, e.g., a trial can be simulated with different dose adjustment rules to keep a response variable under control (Chap. 7, CTS), or the effects of initial feed compositions and other process parameters (temperature, pressure) can be investigated on the equilibrium, conversions, and concentrations of methanol synthesis components [20].

Outside the pharmaceutical industry, modeling and simulation continue to be successfully applied to a wide variety of problems in different scientific areas [21].

Weather forecasts, earthquake and tsunami warnings, the World3 model investigating aspects of the Earth's ecosystem, production of the Boeing 777, replacement of actual by virtual crash testing, throughput capacities of railway stations, Google's search engine, traffic light control, artificial organs (heart, pancreas), assessment of cooperative behavior [22], and three-dimensional image reconstruction are all the result of mathematical modeling and simulation. Finally, physics has to a large extent turned into a mathematical discipline to understand and master large- and small-scale phenomena.

Comparing model-predicted outcomes with actual observations has led to many new findings. Some models have required tweaking in order to comply with adjustments in subject matter theory, while other models, assumed correct, have led to the postulation of new observable entities. The interplay between model and experiment forms an important element in innovative drug development based on modeling and simulation.

### ***1.2.6 Validation of Models***

Models have been applied to many situations. Following confrontation with experimental data, they may be abandoned or refined. If a computational model is correctly implemented technically, then discrepancies with the experimental data are a call to action. The assumptions behind the model and data need checking, the model may need to be revised or the experimental data discarded, as happened with an experiment at the European Organization for Nuclear Research (CERN) when it was originally claimed that there were particles capable of moving faster than light. If true, this would have caused a major revision of physics. Eventually, the experimental equipment turned out to be defective [24]. But models can also fail (catastrophically). Economic models completely failed to predict the economic crisis of 2008, prompting serious debate on the modeling approach as such, and on the assumptions behind the failed models.

One way to validate a model for data (such as a pharmacostatistical model) is to apply the model prospectively to data from another experiment. A less stringent procedure is data splitting where data from one experiment are split into model-building and model-validation parts. A further procedure is to use the model and its parameters to create a large number of longitudinal predictions that are then compared to observed outcomes (see Visual Predictive Check in [Chap. 6](#)).

Systems models can be validated by observable predictions, if available. This is the case for physiologically based pharmacokinetic (PBPK) models that predict plasma drug concentrations in different organs. Unobservable model predictions can at least be correlated with experimental data, e.g., to check whether they are better at explaining variability than other predictors. Systems models may also be checked by how mechanisms are implemented, which theories are applied, and the degree of granularity within different parts of the model.

Furthermore, we should keep in mind that according to Karl Popper (1902–1994), models cannot be verified, but only falsified. And it may help to stress that decisions should not be based on the outcome of a single model, but on thorough discussion supported by a mathematical model. We would like to end this chapter with another quote from Box [10]:

*Remember that all models are wrong;  
the practical question is how wrong do they have to be to not be useful.*

A useful model is one that solves a problem and helps us to reach an informed decision. For example: How likely is it that our drug will work? Can we be confident in our dosing regimen? Are we addressing the right patient population?

## References

1. Chandler AD (2005) Shaping the industrial century: the remarkable story of the modern chemical and pharmaceutical industries. Harvard University Press, Cambridge
2. LaMattina JL (2011) The impact of mergers on pharmaceutical R&D. *Nat Rev Drug Discov* 10(559):560
3. Drews J (2000) Drug discovery: a historical perspective. *Science* 287:1960–1964
4. John V (2010) BACE: lead target for orchestrated therapy of Alzheimer's disease. Wiley, New York
5. Talele TT, Khedkar SA, Rigby AC (2010) Successful applications of computer aided drug discovery: moving drugs from concept to the clinic. *Curr Topics Med Chemistry* 10:127–141
6. Swinney DC, Anthony J (2011) How were new medicines discovered. *Nat Rev Drug Discov* 10:507–519
7. Lindsay MA (2005) Finding new drug targets in the twenty-first century. *Drug Discov Today* 10:1683–1687
8. Avorn J (2011) Learning about the safety of drugs – a half-century of evolution. *N Engl J Med* 365:2151–2153
9. de Regt HW, Dieks D (2003) A contextual approach to scientific understanding. <http://philsci-archive.pitt.edu/1354/>. Accessed 13 March 2013
10. Box GEP, Draper NR (1987) Empirical model-building and response surfaces. Wiley, New York
11. Di Stefano JJ, Landaw EM (1984) Multiexponential, multicompartmental, and noncompartmental modeling. I. Methodological limitations and physiological interpretations. *Am J Physiol - Regul Integr Comp Physiol* 246:651–664
12. Sterman JD (1991) A skeptic's guide to computer models. <http://www.systems-thinking.org/simulation/skeptics.pdf> Accessed 13 March 2013
13. Bazzoli C, Retout S, Mentre F (2010) Design evaluation and optimisation in multiple response nonlinear mixed effect models: PFIM 3.0. *Comput Methods Programs Biomed* 98:55–65
14. Serafin D, Skrzypek J, Grzesik M (1989) Optimization of directly cooled multi-stage adiabatic reactors I. *Chem Process Eng* 3:389–403
15. Serafin D, Skrzypek J, Grzesik M (1989) Optimization of directly cooled multi-stage adiabatic reactors II. *Chem Process Eng* 4:641–653
16. Serafin D, Skrzypek J, Grzesik M (1990) Optimization of directly cooled multi-stage adiabatic reactors III. *Chem Process Eng* 2:493–499
17. Garred LJ, Pretlac R (1991) Mathematical modeling of erythropoietin therapy. *Am Soc Artif Intern Organs Trans* 37:M457–M459

18. Ward MP, Irazoqui PP (2010) Evolving refractory major depressive disorder diagnostic and treatment paradigms: toward closed-loop therapeutics. *Front Neuroeng* 3:7.1-7.15
19. Meadows DH, Randers J, Meadows DL (2004) Limits to growth: The 30-year update. Chelsea Green Publishing Company, With River Junction
20. Skrzypek J, Lachowska M, Serafin D (1990) Methanol synthesis from CO<sub>2</sub> and H<sub>2</sub>: Dependence of equilibrium conversions and exit equilibrium concentration of components on the main process variables. In: *Chemical Engineering Science*, vol 45. Pergamon Press, Oxford, pp 89-96
21. Kaufmann WJ, Smarr LL (1993) Supercomputing and the transformation of science. Scientific American Library, New York
22. Axelrod R (2006) The Evolution of Cooperation: Revised Ed. Basic Books, New York
23. Paul SM, Mytelka DS, Dunwiddie CT, Persinger CC, Munos BH, Lindborg SR, Schacht AL (2010) How to improve R&D productivity: the pharmaceutical industry's grand challenge. *Nat Rev Drug Discov* 9:203-214

### Internet references (Accessed 13 Mar 2013):

24. <http://press.web.cern.ch/press/pressreleases/releases2011/pr19.11e.html>
25. <http://www.astrazeneca.com/Home>
26. <http://www.bms.com/ourcompany/Pages/history.aspx>
27. <http://www.lilly.com/about/heritage/Pages/heritage.aspx>
28. <http://www.gsk.com/about-us/our-history.html>
29. <http://www.jnj.com/connect/about-jnj/company-history/>
30. <http://www.merckgroup.com/en/company/history/history.html>
31. <http://www.merck.com/about/our-history/home.html>
32. <http://www.novartis.com/about-novartis/company-history/index.shtml>
33. [http://www.pfizer.com/about/history/1849\\_1899.jsp](http://www.pfizer.com/about/history/1849_1899.jsp)
34. [http://www.roche.com/about\\_roche/milestones.htm](http://www.roche.com/about_roche/milestones.htm)
35. [http://en.sanofi.com/our\\_company/history/history.aspx](http://en.sanofi.com/our_company/history/history.aspx)
36. <http://www.bayer.com/en/1863-1881.aspx>
37. <http://www.pharmaphorum.com/2010/09/17/a-history-of-the-pharmaceutical-industry/>
38. <http://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfcfr/CFRSearch.cfm?fr=314.50/>

## Chapter 2

# First Example of a Computational Model

Based on prior knowledge and ad hoc assumptions, a computational model is built that mimics the effects of a virtual drug on colorectal tumor size and on palmar and plantar skin. Models for tumor growth and skin turnover are combined with pharmacokinetic (PK) and pharmacodynamic (PD) models to assess the impact of two alternative dosing regimens on efficacy and safety. Both regimens deliver the same cumulative drug amount, but one regimen employs a continuous schedule while the other allows for temporary drug discontinuation. Interindividual variability is introduced on PK and PD parameters and Monte Carlo simulations are performed in treatment groups of 50 subjects. Such simulations can contribute to the assessment of the benefit/risk ratio of an intended drug treatment.

### 2.1 Problem Description

As an introductory example, let us consider a computational model for a virtual oral anticancer drug [1]. The purpose of the model is to address the following key question:

Which of the following two potential dosage regimens is more suitable with regard to efficacy/safety events? An intermittent regimen consisting of a 12-week treatment divided into four 3-week cycles, with each cycle comprising a 1,500 mg dose given twice daily (BID) during the first 2 weeks followed by 1 week without treatment, or a continuous regimen consisting of a 12-week treatment with a 1,000 mg dose given BID? Efficacy is to be assessed by drug effects on colorectal tumor size, and safety by drug effects on high turnover skin tissues (i.e., the palms of the hands and soles of the feet).

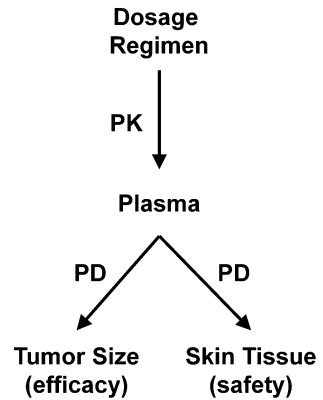
## 2.2 Conceptual and Mathematical Modeling

Figure 2.1 depicts an initial conceptual model for the above problem. The key elements are the dosage regimen, either intermittent or continuous; the resulting drug concentrations in plasma (PK); and the effects on tumor size and skin tissue (PD), which represent efficacy and safety outcomes of drug treatment. We will derive conceptual and mathematical models for each of these parts.

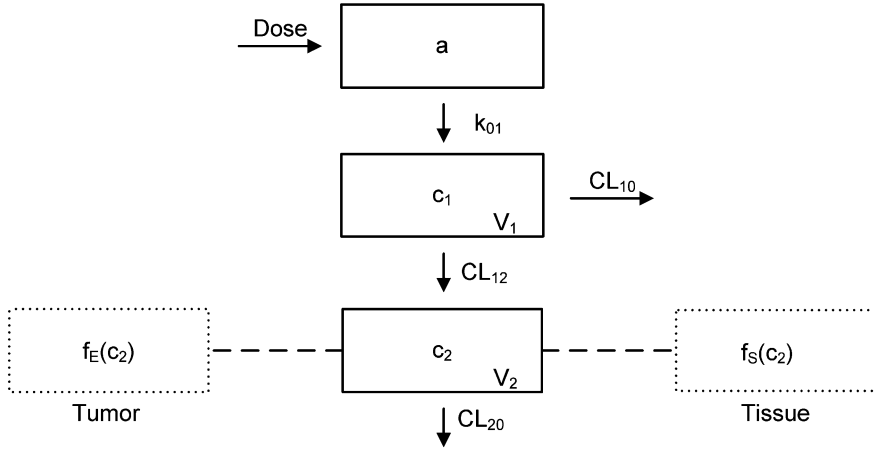
The next level of the conceptual model details PK by specifying the compartments, indicated by solid squares, involved in drug transport through the body, such as the amount  $a$  of drug at the absorption site, the plasma drug concentration  $c_1$ , and the plasma concentration  $c_2$  of a drug metabolite (Fig. 2.2).

Arrows indicate drug movement between compartments. Regarding the mathematical description of drug amounts in each compartment, mass balance is a guiding principle and, in this case, we assume all mass transport rates from a given compartment to be proportional to the drug amount in that compartment, i.e., to obey first-order kinetics. The products  $r_{01} := k_{01} \cdot a$ ;  $r_{10} := CL_{10} \cdot c_1$ ;  $r_{12} := CL_{12} \cdot c_1$ ; and  $r_{20} := CL_{20} \cdot c_2$  in Fig. 2.2 designate drug amount rates from the compartments describing dose amounts,  $a$ , and plasma drug concentrations,  $c_1$ , and  $c_2$ , respectively. The terms  $CL_{10}$ ,  $CL_{20}$ , and  $CL_{12}$ , named clearances, are the proportionality constants between rate of drug elimination and drug concentration, respectively. Another important proportionality constant is that between drug amount and drug concentration in a given compartment, named volume of distribution,  $V$ . In our example, drug that enters plasma at (time-dependent) rate  $r_{01}$  is measured as the concentration requiring application of a volume term,  $V_1$ . A similar consideration applies to drug metabolite formed at rate  $r_{12}$  into volume  $V_2$ . Overall PK can be described mathematically by the ordinary differential equations (ODE) system (2.1):

**Fig. 2.1** Conceptual model (high level) for drug action integrating PK (drug concentration in plasma) and PD (tumor size and skin tissue)







**Fig. 2.2** Conceptual model for plasma pharmacokinetics. The model considers:  $a$ —drug amount,  $k_{01}$ —first-order absorption rate constant,  $c_1$ —parent drug concentration,  $CL_{10}$ —parent drug elimination clearance,  $CL_{12}$ —metabolic clearance,  $c_2$ —active metabolite concentration and  $CL_{20}$ —metabolite elimination clearance,  $V_1$  and  $V_2$ —volumes of distribution

$$\left. \begin{aligned} \frac{da}{dt} &= -k_{01} \cdot a \\ \frac{dc_1}{dt} &= \frac{k_{01}}{V_1} \cdot a - \left( \frac{CL_{12}}{V_1} + \frac{CL_{10}}{V_1} \right) \cdot c_1 \\ \frac{dc_2}{dt} &= \frac{CL_{12}}{V_2} \cdot c_1 - \frac{CL_{20}}{V_2} \cdot c_2 \end{aligned} \right\} \quad (2.1)$$

with initial values:

$$a(0) = Dose, \quad c_1(0) = 0, \quad c_2(0) = 0 \quad (2.2)$$

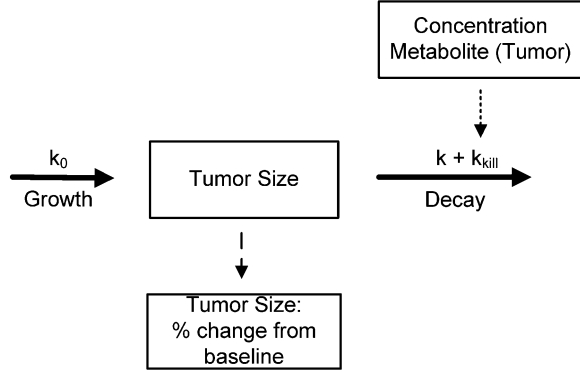
where  $t$ —time (independent variable);  $a$ —drug amount;  $Dose$ —given dose;  $c_1$  and  $c_2$ —parent drug and metabolite concentrations;  $k_{01}$ —first-order elimination rate constant;  $CL_{10}$ ,  $CL_{20}$ , and  $CL_{12}$ —parent drug and metabolite elimination clearances, and parent drug metabolic clearance;  $V_1$  and  $V_2$ —parent and metabolite volumes of distribution.

Our conceptual model combined the PK and PD of our virtual drug. As a ‘bridge’ between PK and PD we will use the Hill equation (Archibald Hill, 1886–1977) describing drug effect (PD) as a static function of drug concentration (PK), i.e.,

$$E = E_0 \pm \frac{E_{\max} \cdot c^x}{EC_{50}^x + c^x} \quad (2.3)$$

where  $E$  means drug effect;  $E_0$ —baseline value;  $c$ —drug concentration value;  $x$ —power value (Hill coefficient),  $E_{\max}$ —maximum drug effect value;  $EC_{50}$ —drug

**Fig. 2.3** Conceptual model for drug action on tumor size. *Bold solid lines* indicate growth and decay of the tumor.  $k_0$  is the tumor growth rate constant and  $k$  the tumor decay rate constant. Drug concentrations affect tumor size by increasing the rate constant  $k$  by value  $k_{kill}$  (*dotted line*). The calculation of the relative change of tumor size from baseline is indicated by the *dashed line*



potency, i.e., the value at which half maximum effect is achieved. We will use Eq. (2.3) in PD models considering the drug effects on both efficacy (reduction in tumor growth) and safety (damage to epidermis).

The mathematical description of tumor growth corresponding to the conceptual model shown in Fig. 2.3 assumes that three rate constants impact tumor size:  $k_0$ , the tumor growth rate constant, on tumor growth;  $k$ , the tumor decay rate constant, on (untreated) tumor decay, and  $k_{kill}$ , the drug-induced increase in tumor decay rate. The mathematical equation of this process is derived from the Gompertz growth model (Benjamin Gompertz, 1779–1865) [2], and is expressed in (2.4)

$$\frac{dn}{dt} = \left[ k_0 \cdot \log \frac{n_{00}}{n} - (k + k_{kill}) \right] \cdot n \quad (2.4)$$

with initial value  $n_0$  for tumor size  $n$ :

$$n(0) = n_0 \quad (2.5)$$

where  $k_0$  is the tumor growth rate constant;  $n_{00}$ —tumor growth limiting value;  $k$ —tumor decay rate constant,  $k_{kill}$ —drug-induced increase in tumor decay rate constant. In order to describe  $k_{kill}$  we apply (2.3). With the assignments

$$\begin{aligned} E &\equiv k_{kill}, & E_0 &\equiv 0, & E_{max} &\equiv k_{kill_{max}}, \\ c &\equiv f_E(c_2), & x &\equiv 1, & EC_{50} &\equiv kc_{kill_{50}} \end{aligned} \quad (2.6)$$

we obtain parameter  $k_{kill}$ :

$$k_{kill} = \frac{k_{kill_{max}} \cdot f_E(c_2)}{kc_{kill_{50}} + f_E(c_2)} \quad (2.7)$$

The function  $f_E$  provides a means to account for the possibility that another concentration profile than  $c_2$  might act on  $k_{kill}$ . In this case, we have chosen for simplicity  $f_E(c_2) := tufac \cdot c_2$  where  $tufac$  is a constant.

Similarly, we can get the mathematical model for describing the drug effects on healthy skin tissue (epidermis). We assume the conceptual model shown in Fig. 2.4, which was proposed by Weinstein et al. [3].

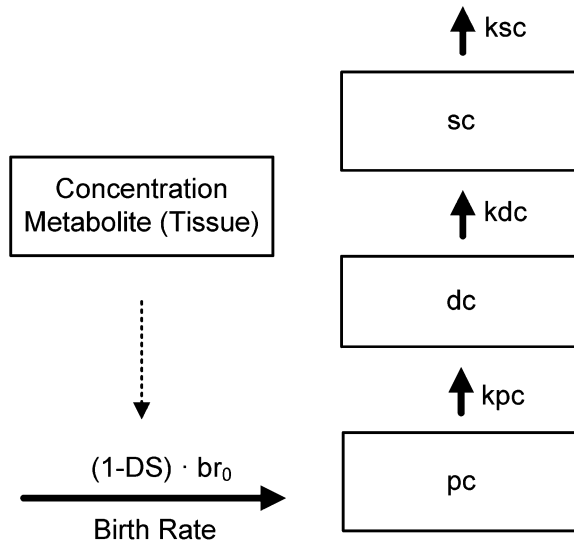
Mathematically, the three-compartment epidermis model leads to the following three differential equations:

$$\left. \begin{aligned} \frac{d}{dt}pc &= br_0 \cdot (1 - DS) - kpc \cdot pc \\ \frac{d}{dt}dc &= kpc \cdot pc - kdc \cdot dc \\ \frac{d}{dt}sc &= kdc \cdot dc - ksc \cdot sc \end{aligned} \right\} \quad (2.8)$$

with initial values:

$$pc(0) = pc_0, \quad dc(0) = dc_0, \quad sc(0) = sc_0 \quad (2.9)$$

where  $br_0$  means initial birth rate;  $kpc$ —elimination rate constant of the proliferative compartment;  $kdc$ —elimination rate constant of the differentiated compartment,  $ksc$ —elimination rate constant of the stratum corneum compartment. Cell turnover times are 6, 4 and 9 days and steady-state cell values are 27,000, 18,000,



**Fig. 2.4** Conceptual model for drug action on skin (epidermis). **Bold solid lines** indicate skin growth and decay.  $br_0$  is the birth rate of new epidermal cells. Epidermal cells sequentially pass three compartments, named proliferative ( $pc$ ), differentiated ( $dc$ ), and stratum corneum ( $sc$ ) compartments with respective first-order rate constants,  $kpc$ ,  $kdc$ , and  $ksc$ . Each compartment is characterized by a cell turnover time, and by the cell number at steady state. Drug concentrations in skin tissue produce an effect,  $DS$ , equivalent to a decrease in birth rate of new epidermal cells

and 40,500 cells/mm<sup>2</sup> for *pc*, *dc*, and *sc*, respectively. The elimination rate constants are the inverse of the turnover times. To describe *DS* we again use the Hill equation (2.3):

$$DS = \frac{DS_{\max} \cdot f_S(c_2)}{DS_{50} + f_S(c_2)} \quad (2.10)$$

Like  $f_E$  before,  $f_S$  is a function of plasma concentrations  $c_2$  and acts on  $br_0$ . In this case, we have chosen for simplicity  $f_S(c_2) := skfac \cdot c_2$  where *skfac* is a constant.

Having created the mathematical model from the conceptual model (Fig. 2.2), the next step is to provide a computational model which can be implemented in MATLAB.

### 2.3 Computational Model

For the purpose of our model, namely to simulate different trial design scenarios with respect to treatment efficacy and safety, (2.1–2.10) should be formulated as one ODE system, according to the following general formulation:

$$\frac{dy}{dt} = \mathbf{f}(t, \mathbf{y}, \mathbf{u}, \boldsymbol{\varphi}); \quad t \in [0; T] \quad (2.11)$$

with initial value:

$$\mathbf{y}(0) = \mathbf{y}_0 \quad (2.12)$$

where  $\mathbf{y}$  is a vector of dependent variables,  $t$ —time,  $\mathbf{u}$ —a vector of input (control) variables,  $\boldsymbol{\varphi}$ —a vector of model parameters. The ODE is to be integrated from time 0 to time  $T$ , where  $T$  is given by the context.

First, we show that (2.1–2.10) is easily transformed to the general form (2.11), using the following assignments:

$$\mathbf{y} \equiv \begin{bmatrix} a \\ c_1 \\ c_2 \\ n \\ pc \\ dc \\ sc \end{bmatrix}, \quad \mathbf{f} \equiv \begin{bmatrix} -k_{01} \cdot a + \mathbf{u} \\ \frac{k_{01}}{V_1} \cdot a - \left( \frac{CL_{12}}{V_1} + \frac{CL_{10}}{V_1} \right) \cdot c_1 \\ \frac{CL_{12}}{V_2} \cdot c_1 - \frac{CL_{20}}{V_2} \cdot c_2 \\ \left[ k_0 \cdot \log \frac{n_{00}}{n} - (k + k_{\text{kill}}) \right] \cdot n \\ br_0 \cdot (1 - DS) - kpc \cdot pc \\ kpc \cdot pc - kdc \cdot dc \\ kdc \cdot dc - ksc \cdot sc \end{bmatrix}, \quad \mathbf{u} \equiv [Dose \cdot \delta(t)] \quad (2.13)$$

In the above equation,  $\delta(t)$  stands for the Dirac function (Paul Dirac, 1902–1984) and as initial values we have:

$$\mathbf{y}(0) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ n_0 \\ pc_0 \\ dc_0 \\ sc_0 \end{bmatrix}; \quad \mathbf{u} \equiv [Dose \cdot \delta(t)] \Leftrightarrow \mathbf{y}(0) = \begin{bmatrix} Dose \\ 0 \\ 0 \\ n_0 \\ pc_0 \\ dc_0 \\ sc_0 \end{bmatrix}; \quad \mathbf{u} = \mathbf{0} \quad (2.14)$$

Equation (2.14) means that vector  $\mathbf{u}$  can be omitted if *Dose*, linked to variable *a*, is considered as the initial value of this variable. Based on (2.13), and including (2.7) and (2.10) for  $k_{kill}$  and *DS*, respectively, the vector of model parameters  $\boldsymbol{\varphi}$  can be now defined as:

$$\boldsymbol{\varphi} \equiv [k_{01} \ CL_{12} \ CL_{10} \ CL_{20} \ V_1 \ V_2 \ kc_{kill_{50}} \ DS_{50} \ k_0 \ n_{00} \ k \ k_{kill_{max}} \ tufac \ br_0 \ DS_{max} \ skfac \ kpc \ kdc \ ksc]^T \quad (2.15)$$

Furthermore, a trial simulation has to be conducted on a population of many subjects, and if so, then at least some elements of the  $\boldsymbol{\varphi}$  vector change from subject to subject. This intersubject variability is an important factor and has also to be considered in the trial simulation procedure. In order to generate variability of model parameters we will use normal and log-normal distributions, the most typical distributions applied in pharmacologic models. If a model parameter (elements of the vector  $\boldsymbol{\varphi}$ ), for example  $k_{01}$ , is assumed to have a normal distribution, then it can be generated from the following formula:

$$\left. \begin{array}{l} \varepsilon \sim \mathbf{N}(0, 1) \\ k_{01} = \mu + \sigma \cdot \varepsilon \end{array} \right\} \Leftrightarrow k_{01} \sim \mathbf{N}(\mu, \sigma^2) \quad (2.16)$$

In this case, the formula (2.16) generates normally distributed variability of the parameter  $k_{01}$  with the typical value (mean value)  $\mu$  and standard deviation  $\sigma$  (variance  $\sigma^2$ ). The normally distributed variability,  $\mathbf{N}(\mu, \sigma^2)$ , is characterized by the Gaussian probability density function:

$$f(x) = \frac{1}{\sigma \cdot \sqrt{2 \cdot \pi}} e^{\frac{-(x-\mu)^2}{2 \cdot \sigma^2}} \quad (2.17)$$

If another model parameter, for example  $DS_{50}$ , is assumed to have a log-normal distribution then we use the formula:

$$\left. \begin{aligned} \varepsilon &\sim \mathbf{N}(0, 1) \\ \mu^* &= \ln \frac{\mu^2}{\sqrt{\mu^2 + \sigma^2}} \\ \sigma^* &= \sqrt{\ln \frac{\mu^2 + \sigma^2}{\mu^2}} \\ DS_{50} &= \exp(\mu^* + \sigma^* \cdot \varepsilon) \end{aligned} \right\} \Leftrightarrow DS_{50} \sim \ln \mathbf{N}(\mu, \sigma^2) \quad (2.18)$$

where formula (2.18) generates a log-normally distributed parameter  $DS_{50}$  with the typical value (mean value)  $\mu$  and standard deviation  $\sigma$ .

Usability of (2.16)–(2.18) consists in easy generation of random numbers with the proper distribution; at first, random numbers with standard normal distribution  $\mathbf{N}(0, 1)$  are generated, and then converted to normal  $\mathbf{N}(\mu, \sigma^2)$  or lognormal distribution  $\ln \mathbf{N}(\mu, \sigma^2)$ .

Input vector  $\mathbf{u}$  has only one element:

$$\mathbf{u} \equiv [Dose \cdot \delta(t)]. \quad (2.19)$$

Six model parameters, linked to the PK model in (2.1), are normally distributed. Two parameters, linked to the efficacy and safety models in (2.4) and (2.8), have a lognormal distribution.

The domain of independent variable, time  $t$ , needs to be considered with regard to the dosage regimen. For a single dose no further comment is necessary. In the case of a multiple dosage regimen, the time domain has to be divided into a proper number of intervals, linked to each time of dosing. It means also that initial values of dependent variables for the next dose time have to be derived from the end value in the state where a new dose is given, so if a dose was given  $M$  times then:

$$\left. \begin{aligned} \frac{dy}{dt} &= \mathbf{f}(\mathbf{y}, t, \mathbf{u}, \varphi); \quad t \in [0; T] \\ [0; T] &= \bigcup_{m=1}^{M+1} [t_{m-1}; t_m]; \quad t_0 = 0; \quad t_{M+1} = T \end{aligned} \right\} \quad (2.20)$$

with initial values:

$$\mathbf{y}(0) = \begin{bmatrix} Dose \\ 0 \\ 0 \\ n_0 \\ pc_0 \\ dc_0 \\ sc_0 \end{bmatrix}, \quad \mathbf{y}(t_m) = \lim_{t \rightarrow t_m^-} \begin{bmatrix} a(t) + Dose \\ c_1(t) \\ c_2(t) \\ n(t) \\ pc(t) \\ dc(t) \\ sc(t) \end{bmatrix}; \quad m = 1, 2, \dots, M-1 \quad (2.21)$$

In (2.21),  $\mathbf{y}(t_m)$  is the one-sided limit of the right-hand side, i.e.,  $t$  approaches  $t_m$  from the left (indicated by  $t_m^-$ ).

## 2.4 Computational Model in MATLAB

To evaluate the model described by (2.1–2.10), a MATLAB program was created (Listings 2.1–2.6). It consists of a collection of functions. To launch a simulation the primary function **concmmod()** has to be called with proper input parameters described in Listing 2.1.

**Listing 2.1** Program **concmmod.m** (only the primary function is shown)

```
function concmmod(regimenType, numSubjects)
%CONCMOD Model Tumor Growth
%   CONCMOD(REGIMENTYPE, NUMSUBJECTS) models tumor growth and creates
%   graphs of the tumor growth and epidermis over time. |REGIMENTYPE|
%   must be either 'intermittent' or 'continuous'. |NUMSUBJECTS| is
%   the number of subjects.
%
%   Example:
%
%   Model tumor growth with an intermittent regimen and 10 subjects
%   concmmod('intermittent',10)

% Check input arguments and provide default values if necessary
error(nargchk(0, 2, nargin));      %#ok<*NCHKN>

if nargin < 2
    numSubjects = 10;
end
if nargin < 1
    regimenType = 'intermittent';
end

% Reset the random number generator to the default
rng default; rand(100);

% Calculate dosing times and amount based on regimen
[doseTimes, doseAmount] = doseSchedule(regimenType);

% Set up figures for plotting results
tumorFigure = figure;
tumorAxes   = axes; set(tumorAxes,'FontSize',14)
xlabel('time [h]')
ylabel('Number of tumor cells (relative to baseline)')
title('Tumor Growth')
xlim([0, doseTimes(end)]); hold on; grid on;
epidermisFigure = figure;
epidermisAxes   = axes; set(epidermisAxes,'FontSize',14)
xlabel('time [h]')
ylabel('% change (relative to baseline)')
title('Epidermis')
xlim([0, doseTimes(end)]); hold on; grid on; ylim([50, 100]);
```

```

set([tumorFigure; epidermisFigure], ...
    'units', 'normalized', ...
    {'Position'}, {[0.1, 0.5, 0.3, 0.4]; [0.6, 0.5, 0.3, 0.4]});

% Simulate system for each subject
for subjectID = 1:numSubjects

    % Initialize parameters and set initial values
    p = initializeParams;
    y0 = [doseAmount, p.c10, p.c20, p.n0, p.pc0, p.dc0, p.sc0];

    % Allocate variables to store results
    timePoints = [];
    tumorGrowth = [];
    epidermis = [];

    % Simulate system for each treatment period
    for dose = 1:(length(doseTimes)-1)

        % Set time interval for this treatment period
        tspan = [doseTimes(dose), doseTimes(dose+1)];

        % Call Runge-Kutta method
        [t,y] = ode45(@derivatives, tspan, y0, [], p);

        % Record values for plotting
        timePoints = [timePoints ; t];           %#ok
        tumorGrowth = [tumorGrowth; y(:,4)/p.n0]; %#ok
        epidermis = [epidermis ; 100*y(:,7)/p.sc0]; %#ok

        % Reset initial values for next treatment period
        % and add next dose
        y0 = y(end,:);
        y0(1) = y0(1) + doseAmount;
    end

    % Plot results for this subject
    plot(tumorAxes, timePoints, tumorGrowth, 'Color', 'black')
    plot(epidermisAxes, timePoints, epidermis, 'Color', 'black')
    drawnow
end

% save graphs as TIFF file
print(tumorFigure, '-r900', '-dtiff', ...
    ['tumor', '_', regimenType])
print(epidermisFigure, '-r900', '-dtiff', ...
    ['epidermis', '_', regimenType])
end

```



**Listing 2.2** Function **derivatives**

```

function dydt = derivatives( ~, y, p)
%DERIVATIVES Compute the right-hand side of the ODE.
%   DYDT = DERIVATIVES(T, Y, P) calculates |DYDT|, the right-hand
%   side of the ODE model, at points defined by the vector of
%   dependent variables |Y|, time |T|, and with parameters |P|.

amount = y(1);      % drug amount
c1      = y(2);      % parent drug concentration
c2      = y(3);      % active metabolite concentration
n       = y(4);      % tumor growth
pc      = y(5);      % cells in proliferative compartment
dc      = y(6);      % cells in differentiated compartment
sc      = y(7);      % cells in stratum corneum compartment

% PK Model
dAmountdt = -p.k01*amount;
dc1dt      = p.k01/p.V1*amount - (p.CL12/p.V1 + p.CL10/p.V1)*c1;
dc2dt      = p.CL12/p.V2*c1 - p.CL20/p.V2*c2;

% Efficacy Model
kkill = hillEffect(p.tumFactor*c2, 0, p.kkillMax, p.kkill50, 1);
dndt  = (p.k0*log(p.nn00/n) - (p.k + kkill))*n;

% Toxicity Model
DS    = hillEffect(p.skFactor*c2, 0, p.DSMax, p.DS50, 1);
br1   = p.br0*(1-DS); % birth rate;
dpcdt = br1 - p.kpc*pc;
ddcdt = p.kpc*pc - p.kdc*dc;
dscdt = p.kdc*dc - p.ksc*sc;

% Derivatives vector of ODE system
dydt = [ dAmountdt;      % drug amount
         dc1dt;          % parent drug concentration
         dc2dt;          % active metabolite concentration
         dndt;           % tumor growth
         dpcdt;          % changes in proliferative compartment
         ddcdt;          % changes in differentiated compartment
         dscdt;          % changes in stratum corneum compartment
       ];
end

```

**Listing 2.3** Function `doseSchedule`

```

function [doseTimes, doseAmount] = doseSchedule(regimenType)
%DOSESCHEDULE Creates a vector of dosing times and amounts of drug.
% [DOSETIMES, DOSEAMOUNT] = DOSESCHEDULE(REGIMENTYPE) returns
% a vector |DOSETIMES| of dosing times and a scalar |DOSEAMOUNT|
% of the dosing amount. |REGIMENTYPE| must be either 'intermittent'
% or 'continuous'.

treatmentWeeks = 12;
cycleWeeks     = 3;
daysInWeek    = 7;
hoursInDay     = 24;
initialTime    = 0;
% treatment time [hours]
endTime = treatmentWeeks * daysInWeek * hoursInDay;
% cycle time [hours]
cycleTime = cycleWeeks * daysInWeek * hoursInDay;

switch regimenType
    case 'intermittent'
        % intermittent treatment
        dailyDoses = 2;          % BID
        doseAmount = 1500;       % 1 dose in [mg]
        timeOnDrug = cycleTime - daysInWeek * hoursInDay;
    case 'continuous'
        % continuous (over 12 weeks) treatment
        dailyDoses = 2;          % BID
        doseAmount = 1000;       % 1 dose in [mg]
        timeOnDrug = cycleTime - hoursInDay/dailyDoses;
    % case 'other' % placeholder for other regimen(s)
    otherwise
        messageID = 'GieschkeBook:concmmod:UnknownRegimen';
        messageStr = ['Unknown regimen '%s''. The regimen must ',...
            'be either 'intermittent' or 'continuous''];
        error(messageID, messageStr, regimenType);
end

% create vector of treatment times
initialCycleTimes = ...
    initialTime : cycleTime : (endTime - cycleTime);
doseTimesWithinCycle = ...
    initialTime : hoursInDay/dailyDoses : timeOnDrug;
doseTimes = reshape(...
    repmat(doseTimesWithinCycle', 1, length(initialCycleTimes)) + ...
    repmat(initialCycleTimes, length(doseTimesWithinCycle), 1), ...
    1, length(initialCycleTimes) * length(doseTimesWithinCycle));
doseTimes = [doseTimes, endTime]; % add the end time of treatment

end

```

**Listing 2.4** Function `initializeParams`

```

function params = initializeParams
%INITIALIZEPARAMS create initial values for model parameters
%   PARAMS = INITIALIZEPARAMS returns a structure |PARAMS| containing
%   initial values for model parameters, including variability when
%   necessary.

% PK parameters
pkCV = 0.3;
lim1 = 0.3;
lim2 = 5.0;

k01 = 0.7;      % first order elimination
params.k01 = variability('varnorm', k01, k01*pkCV, 1, lim1);
CL12 = 10;      % metabolite clearances
params.CL12 = variability('varnorm', CL12, CL12*pkCV, 1, lim2);
CL10 = 80;      % parent drug clearances
params.CL10 = variability('varnorm', CL10, CL10*pkCV, 1, lim2);
CL20 = 60;      % metabolite elimination clearances
params.CL20 = variability('varnorm', CL20, CL20*pkCV, 1, lim2);
V1 = 30;        % metabolite volumes of distribution
params.V1 = variability('varnorm', V1, V1*pkCV, 1, lim2);
V2 = 150;       % metabolite elimination volumes of distribution
params.V2 = variability('varnorm', V2, V2*pkCV, 1, lim2);
params.c10 = 0;   % initial value: parent drug concentration
params.c20 = 0;   % initial value: active metabolite concentration

% Efficacy parameters (tumor growth parameters)
params.k0 = 4.2e-5; % doubling time 105 days
params.k = 0;      % natural elimination rate
params.tumFactor = 12; % tumor factor
params.nn00 = 1e12; % tumor growth limiting value
params.n0 = 1e9;   % tumor growth initial value
params.kkillMax = 0.05; % max effect (Hill Eq.)
kkill150 = 100;    % concentration linked to 50% of max effect
kkill150CV = 1.5;
params.kkill150 = variability('varlog', ...
    kkill150, kkill150CV*kkill150, 1);

% Toxicity parameters (epidermis)
params.skFactor = 8.0; % toxicity factor
params.DSMax = 0.8;   % max effect (Hill Eq.)
DS50 = 10.0;          % concentration linked to 50% of max effect
DS50CV = 0.8;
params.DS50 = variability('varlog', ...
    DS50, DS50CV*DS50, 1);

% proliferative compartment (pc)
ttpcd = 6;            % cell cycle time in pc (days)
ttpc = ttpcd*24;      % cell cycle time in pc (hours)
params.kpc = 1/ttpc;  % app. elimination rate at steady state
growthFactor = 1.0;   % growth factor in pc
pcTot = 27000;        % number of cells in pc

```

```

params.pc0 = growthFactor*pcTot;    % cells in pc at time 0
params.br0 = params.pc0/ttpc;        % initial birth rate

% differentiated compartment (dc)
ttddc = 9;                          % transit time in dc (days)
ttcdc = ttddc*24;                   % transit time in dc (hours)
params.kdc = 1/ttcdc;               % app. elimination rate at steady state
params.dc0 = params.br0*ttcdc;      % cells in dc at time 0

% stratum corneum compartment (sc)
ttscd = 7;                          % transit time in sc (days)
ttsc = ttscd*24;                   % transit time in sc (hours)
params.ksc = 1/ttsc;               % app. elimination rate at steady state
params.sc0 = params.br0*ttsc;      % cells in sc at time 0

end

```

The function **variability()** was introduced to generate variability (Listing 2.5) that can be log-normally distributed or right-censored normally distributed random numbers. The MATLAB Statistics Toolbox already has implementations **lognrnd()** and **normrnd()** but due to the context in which they have to be used it is more comfortable to introduce a new function where the properly distributed variability can be selected.

#### Listing 2.5 Function **variability**

```

function x = variability(distribution, m, s, num, lim)
%VARIABILITY Generate variability.

% X = VARIABILITY(DISTRIBUTION, M, S, NUM, LIM) returns a vector
% of random numbers for variability, specified by |DISTRIBUTION|
% with the mean value |M| and standard deviation |S|. |NUM| is the
% length of the vector |X|.
% |DISTRIBUTION| can be 'varlog' or 'varnorm':
% 'varlog' generates random numbers |X| ~ logN(|M|, |S|).
% |M| stands for mean value of the lognormal distribution
% |S| - standard deviation of the lognormal distribution
% 'varnorm' generates normally distributed random numbers |X|,
% where |X| ~ N(|M|, |S|), right-censored by |LIM|.

switch distribution
case 'varlog'
    %lognormal distribution
    mu = log(m^2/sqrt(m^2 + s^2)); % where: N(|mu|, |sigma|)
    sigma = sqrt(log(1 + (s/m)^2)); % where: N(|mu|, |sigma|)
    x = lognrnd(mu, sigma, 1, num);
case 'varnorm'
    %lognormal distribution
    mu = m;
    sigma = s;
    x = max(normrnd(mu, sigma, 1, num), lim);
otherwise
    % placeholder for other distributions
end

end

```

**Listing 2.6** Function **hillEffect**

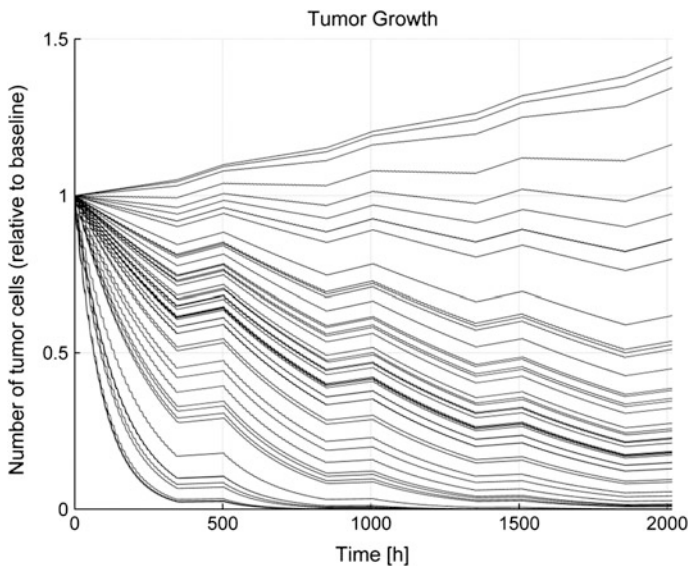
```
function E = hillEffect(c, E0, Emax, EC50, n)
%HILLEFFECT Compute drug effect based on the Hill equation.
% E = HILLEFFECT(C, E0, EMAX, EC50, N) computes drug effect |E|,
% based on the Hill equation as a function of concentration |C|,
% and with the following concentration-response parameters:
% |E0| - baseline response, |EMAX| - maximum effect,
% |EC50| - concentration related to 50% of max. effect,
% and |N| - Hill coefficient of sigmoidicity.

E = E0 + Emax.*c.^n./(EC50.^n+c.^n);

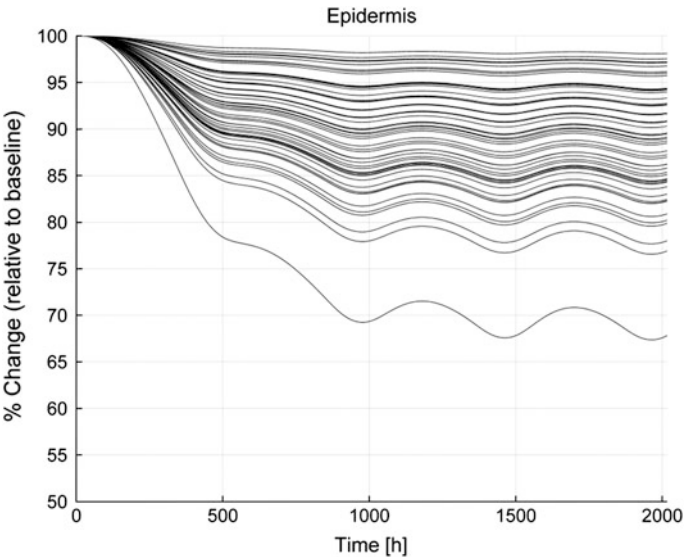
end
```

**2.5 Simulation Results**

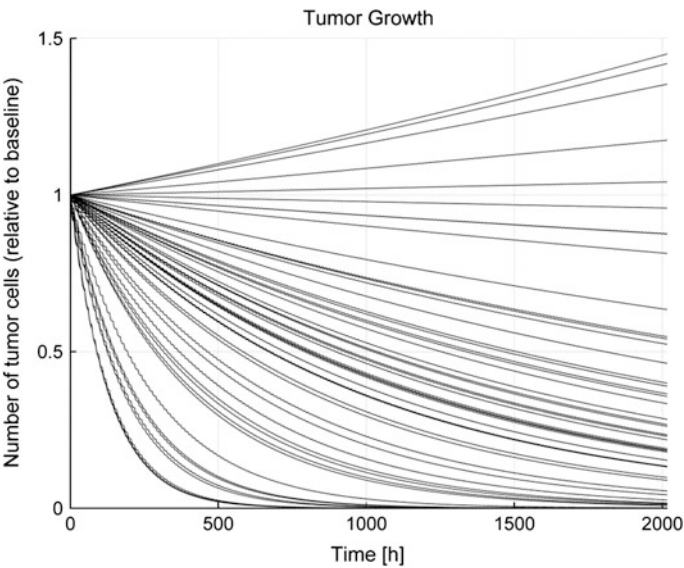
Based on the **concmmod.m** program the trial was simulated. Figures 2.5, 2.6, 2.7, and 2.8 illustrate the effects of different treatment regimens on efficacy and safety in a cohort of 50 randomly selected subjects. Both regimens appeared to reduce tumor size to a similar extent. However, continuous drug administration had stronger adverse effects on the epidermis than intermittent administration.



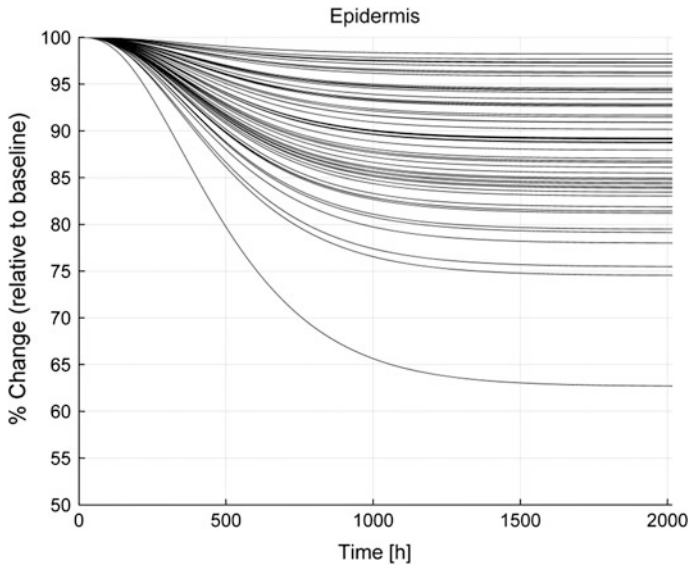
**Fig. 2.5** Effect of intermittent treatment on tumor growth in 50 simulated subjects. Treatment over 12 weeks with 3-week cycles (14 days on, 7 days off)



**Fig. 2.6** Effect of intermittent treatment on epidermis (stratum corneum cells) in 50 simulated subjects. Treatment over 12 weeks with cycling: 14 days on, 7 days off



**Fig. 2.7** Effect of continuous (12-week) treatment on tumor growth in 50 simulated subjects



**Fig. 2.8** Effect of continuous (12-week) treatment on epidermis (stratum corneum cells) in 50 simulated subjects

More thorough statistical analysis would require repeating this simulation many times and determining an overall study outcome for each simulation at relevant times (e.g., at 3, 6, 9, 12 weeks after start of treatment), together with the distribution of outcomes over all replicates. We leave it to the reader to conduct such an analysis.

## 2.6 Comments

An additional question could be asked: What is the impact on the utility (safety/efficacy) of applying a dose adjustment rule based on the occurrence of adverse events? For example, a 5 % reduction in stratum corneum cells could prompt a dose reduction of a given percentage. We leave such a study as an exercise (it requires ODE state event triggering, shown in [Chap. 3](#)).

## 2.7 Exercises

### Exercise 2.1

Rewrite the `concmold.m` program applying vectorization. In this implementation you will have to introduce element-wise vector operators `./`, `.*`, `.^` instead of standard operators `/`, `*`, `^` where necessary, and remove all loops `for`.

### Exercise 2.2

Visualize (parent and metabolite) drug concentrations after first intake.

### Exercise 2.3

Determine outcome distribution in tumor size reduction over 100 replicates in 50 patients.

### Exercise 2.4

Show that the non-negative function defined by (2.17) is a density function, i.e.,

$$\int_{-\infty}^{\infty} f(x)dx = 1 \quad (2.22)$$

Show that this equation is fulfilled, both manually and using MATLAB.

## References

1. Gieschke R, Reigner BG, Steimer JL (1997) Exploring clinical study design by computer simulation based on pharmacokinetic/pharmacodynamic modeling. *Int J Clin Pharmacol Ther* 25:469–474
2. Ingram D, Bloch RF (1986) *Mathematical methods in medicine*. Wiley, New York
3. Weinstein GD, McCulough JL, Ross P (1984) Cell proliferation in normal epidermis. *J Investig Dermatol* 82:623–628



## Chapter 3

# Differential Equations in MATLAB

Differential equations play an important role in describing time-dependent quantities such as drug concentrations in the human body following drug treatment. In this chapter we introduce the main types of differential equations used in the pharmacologic modeling of drug concentrations and effects. We show how they are treated in MATLAB, from both the symbolic and numeric viewpoints. Several interrelated quantities are often considered together, giving rise to differential equation systems also called dynamic systems [1]. In pharmacologic modeling, dynamic systems representing the PK of a drug receive external input due to drug treatment. Forcing (driving, or control) functions are the mathematical construct for modeling this phenomenon. Dynamic systems may be steered or controlled in two ways: either by scheduling times for input ('time scheduling') or by specifying events that should impact on input ('event scheduling'). We will show how both types of scheduling can be used with MATLAB. Empirical models described by differential equations usually contain model parameters to be determined from data. This regression problem can be solved in MATLAB, as shown at the end of this chapter.

### 3.1 Concepts

Differential equations started their triumphant march with the work of Isaac Newton (1642–1727) who established the field of classical mechanics by inventing differential and integral calculus. Classical mechanics deals with the motion, i.e., the change in position over time, of bodies under the influence of external forces. Differential equations became the predominant mathematical construct to describe any quantities that change over time. If the quantities are interrelated, we refer to them as a dynamic system. In ecology, the interactions between two species, predator and prey, were independently investigated about 100 years ago by Alfred Lotka (1880–1949) and Vito Volterra (1860–1940) by means of nonlinear dynamic systems.

Differential equations are equations that contain derivatives of an unknown function  $\mathbf{y}$ , as shown below:

$$\left. \begin{aligned} \frac{d\mathbf{y}}{dt} &= \mathbf{f}(t, \mathbf{y}) ; \quad t \in [t_0 ; t_f] \\ \mathbf{y}(t_0) &= \mathbf{y}_0 \end{aligned} \right\} \quad (3.1)$$

Here,  $\mathbf{y}$  refers to a vector-valued function of time  $t$ ,  $\mathbf{y} = \mathbf{y}(t)$ . The right-hand side of (3.1) specifies a vector-valued function  $\mathbf{f}$  depending on  $\mathbf{y}$  and  $t$ .  $\mathbf{y}$  is a solution of this equation if its derivative with respect to time matches  $\mathbf{f}$  for the interval  $[t_0; t_f]$ .

$$\left. \begin{aligned} \frac{d\mathbf{y}(t)}{dt} &\equiv \mathbf{f}(t, \mathbf{y}(t)) ; \quad t \in [t_0 ; t_f] \\ \mathbf{y}(t_0) &= \mathbf{y}_0 \end{aligned} \right\} \quad (3.2)$$

Without further constraints, differential equations have no unique solution. The second condition in (3.1) poses a constraint on  $\mathbf{y}$  by specifying a value,  $\mathbf{y}_0$ , at initial time  $t_0$ . For this initial value problem (IVP) uniqueness and existence of a solution are guaranteed for a wide class of functions  $\mathbf{f}$ . A differential equation is said to be autonomous if it does not explicitly depend on time, as given below:

$$\left. \begin{aligned} \frac{d\mathbf{y}}{dt} &= \mathbf{f}(\mathbf{y}) ; \quad t \in [t_0 ; t_f] \\ \mathbf{y}(t_0) &= \mathbf{y}_0 \end{aligned} \right\} \quad (3.3)$$

Autonomous differential equations occur naturally in many applications, such as in basic PK and PD models, as shown in [Chap. 4, Pharmacologic Modeling](#). They are closely linked to more formally defined dynamic systems whose qualitative behavior (stability, limit cycles) has undergone rich theoretical development [1].

Equation (3.1) defines a first-order explicit ordinary differential equation (ODE) system. The order of a differential equation is the order of the highest occurring derivative. Explicit means that the equation is solved for the highest occurring derivative, as opposed to an implicit ODE, shown below:

$$\left. \begin{aligned} \mathbf{g}\left(t, \mathbf{y}, \frac{d\mathbf{y}}{dt}\right) &= 0 ; \quad t \in [t_0 ; t_f] \\ \mathbf{y}(t_0) &= \mathbf{y}_0 \end{aligned} \right\} \quad (3.4)$$

The term ordinary indicates that  $\mathbf{y}$  involves only one independent variable,  $t$ . If  $\mathbf{y}$  depends on more than one independent variable, such as  $\mathbf{y} = \mathbf{y}(t, x)$ , the respective differential equation is called a partial differential equation (PDE) as it now involves partial derivatives. PDEs occur in many natural science applications, and an example is worked out in [Sect. 5.2.2, Hodgkin and Huxley Mathematical Model](#). If  $\mathbf{y}$  is one-dimensional, we speak of an ODE equation otherwise we use the term ODE system. In the following, we will mainly deal with explicit ODE systems.

### 3.2 Numerical Solution of Ordinary and Partial Differential Equations

#### 3.2.1 Basic Numerical Algorithms

Occasionally, a differential equation can be solved symbolically, providing profound insight into the mathematical properties of its solution. Often, however, one has to resort to numerical techniques which became rather sophisticated over time. Leonard Euler (1707–1783) can be credited for having invented the basic idea. Equation (3.1) says that the slope of function  $\mathbf{y}$  at initial point  $(t_0, \mathbf{y}_0)$  is  $\mathbf{f}(t_0, \mathbf{y}_0)$ . Thus,  $\mathbf{y}_1 := \mathbf{y}_0 + \mathbf{f}(t_0, \mathbf{y}_0) \cdot h$ , for some (small)  $h$ , could be a good approximation for  $\mathbf{y}(t_1)$ ,  $t_1 := t_0 + h$ . Taking  $(t_1, \mathbf{y}_1)$  as the next initial point, this procedure could be iterated (Euler method). This basic idea works in some, but not many, cases. One problem is to find an appropriate step-size  $h$ . If  $h$  is too small, calculation of  $\mathbf{f}$  could be very resource-intensive; if it is too large, the approximation may be bad. Modern algorithms determine the step-size  $h$  based on the function  $\mathbf{f}$  and use a more refined calculation for  $\mathbf{y}_1$ . Such algorithms were pioneered by Carl Runge (1856–1927) and Martin Kutta (1867–1944): Runge–Kutta methods for ODE solving are now broadly available in many software packages.

MATLAB uses a suite of ODE solvers (Table 3.1). It is difficult to say in advance which solver would be the best for the currently analyzed problem, but it is recommended to start with `ode45` [2, 3].

#### 3.2.2 Ordinary Differential Equation

The standard ODE problem is an IVP, as specified below:

**Table 3.1** List of MATLAB ODE solvers

Solver	Problem type	Order of accuracy	When to use
<code>ode45</code>	Nonstiff	Medium	Most of the time. This should be the first solver you try (Runge-Kutta algorithm)
<code>ode23</code>	Nonstiff	Low	For problems with crude error tolerances or for solving moderately stiff problems
<code>ode113</code>	Nonstiff	Low to high	For problems with stringent error tolerances or for solving computationally intensive problems
<code>ode15s</code>	Stiff	Low to medium	If <code>ode45</code> is slow because the problem is stiff
<code>ode23s</code>	Stiff	Low	If using crude error tolerances to solve stiff systems and the mass matrix is constant
<code>ode23t</code>	Moderately Stiff	Low	For moderately stiff problems if you need a solution without numerical damping
<code>ode23tb</code>	Stiff	Low	If using crude error tolerances to solve stiff systems

**Table 3.2** List of parameters passed to and returned by MATLAB ODE solvers

Formal parameters	Meaning
<T>	Returned values of the independent variable at which the solution was computed
<Y>	Returned solution array, where each column represents a dependent variable within <b>y</b>
<sol> . <b>x</b>	Independent variable within structure <sol> returned by the ODE function; alternative to <T> if structure <sol> is used
<sol> . <b>y</b>	Solution array, where each column represents a dependent variable within <b>y</b> ; alternative to <Y> if structure <sol> is used
<sol> . <b>yp</b>	Structure as above for dependent variables but contains approximated values of derivatives of dependent variables <b>y</b>
<sol> . <b>solver</b>	Name of solver used
<solver>	<b>ode45</b>   <b>ode23</b>   <b>ode113</b>   <b>ode15 s</b>   <b>ode23 s</b>   <b>ode23t</b>   <b>ode23tb</b>
<odefun>	Name of ODE function to be defined as the right-hand side <b>f(t,y)</b> of the differential equations. This function has a header: <output>=<odefun> (<t> , <y>, ...) where <t> is time and <y>—solution vector
<tspan>	Interval of the solution, i.e., [ <i>t</i> <sub>0</sub> ; <i>t</i> <sub>f</sub> ]
<y0>	Initial value vector to be provided to <solver> for dependent variables, i.e., <b>y</b> <sub>0</sub>
<options>	Structure with optional parameters for integration properties and handling events
...	Additional parameters

$$\left. \begin{aligned} \frac{dy}{dt} &= \mathbf{f}(t, \mathbf{y}) ; \quad t \in [t_0 ; t_f] \\ \mathbf{y}(t_0) &= \mathbf{y}_0 \end{aligned} \right\} \quad (3.5)$$

In MATLAB, this could be solved using the obvious syntax:

```
[t, y] = ode45(@f, [t0 tf], y0);
```

In general, the syntax for calling an ODE solver in MATLAB is:

```
[<T>, <Y>] = <solver>(@<odefun>, <tspan>, <y0>, <options>, ...);
```

or alternatively:

```
<sol> = <solver>(@<odefun>, <tspan>, <y0>, <options>, ...);
```

The meaning of the above parameters is explained in Table 3.2. In our context, the independent variable is time, *t*.

Calling the ODE solver with the function handle @ in front of <odefun> is the preferred method in MATLAB. Using @, we can use a function name as an argument in the calling statement.

### Basic ODE Examples

To illustrate how MATLAB handles ODEs, let us consider the following situation for an invented drug. Details about the derivation of the equations as well as the pharmacokinetic parameters are described in [Chap. 4, Pharmacologic Modeling](#). The clearance of the drug is assumed to be  $CL = 2.16$  L/h and the volume of distribution  $V = 12.6$  L. A dose of  $Dose = 200$  mg is to be given as an infusion over 2 h ( $t_R = 2$  h). We would like to simulate drug concentrations over time until  $t_f = 15$  h. The differential equation is as follows:

$$\frac{dc}{dt} = \frac{r(t)}{V} - \frac{CL}{V} \cdot c; \quad t \in [0; 15] \quad (3.6)$$

where

$$r(t) = \begin{cases} \frac{Dose}{t_R}; & t \in [0; t_R) \\ 0 & ; \quad t \geq t_R \end{cases} \quad (3.7)$$

and the initial value for the concentration is:

$$c(0) = 0. \quad (3.8)$$

Equations (3.6)–(3.8) define an ODE IVP which includes the infusion rate as a forcing function,  $u(t) := r(t)$ . A forcing function is a function that is part of an ODE and depends only on time, in general notation:

$$\left. \begin{aligned} \frac{dy}{dt} &= \mathbf{f}(t, \mathbf{y}, \mathbf{u}); \quad t \in [t_0; t_f] \\ \mathbf{y}(t_0) &= \mathbf{y}_0 \end{aligned} \right\} \quad (3.9)$$

Forcing functions,  $\mathbf{u}$ , are part of PK models to describe drug input.

The MATLAB implementation for solving (3.6)–(3.8) could look like the one in Listing 3.1. The source code in the listing is limited to statements considering only this particular model and no processing of the ODE output has been coded. Instead, we have used a **TO DO** placeholder to indicate a section for later output specifications.

**Listing 3.1** Program `infus1draft.m`—preliminary implementation

```
function infus1draft()
    p.V = 12.6;
    p.CL = 2.16;
    [t,c] = ode45(@derivatives, 0:0.01:15, 0, [], p);
    %
    % TO DO: additional statement for visualization
    %
end

function dcdt = derivatives(t, c, p)
    dcdt = r(t)/p.V - p.CL/p.V*c(1);
end

function rt = r(t)
    rt = 200/2*(t < 2);
end
```

Comparing the code in Listing 3.1 to the general syntax for an ODE call illustrates how formal parameters are replaced by current parameters. The model parameters *CL* and *V* enter the `ode45` call via a MATLAB structure **p** which is a convenient way to organize the input for the solver. Parameters used in this example to call the `ode45` solver are shown in Table 3.3.

Having done the step from the mathematical model, expressed in (3.6)–(3.8), to an initial computational model, as in Listing 3.1, we can now complete the source code to get a more detailed MATLAB implementation. In this step, the placeholder, `%TO DO:...`, is replaced with code for visualization (graphics) including comments. Additionally, some comment lines are included to enhance the handling of the program. Details regarding the recommended format for these statements can be found in [2], and in Appendix A, *Hints to MATLAB Programs*. The updated source code is shown in Listing 3.2.

**Table 3.3** List of parameters used to call the `ode45` solver in Listing 3.1

Formal parameter	Current parameter	Meaning
<T>	t	Time
<Y>	c	Concentration
<solver>	ode45	ODE solver name
<odefun>	derivatives	Right-hand side in (3.6)
<tspan>	0:0.01:15	Time domain, [0; 15], with constant evaluation time step, 0.01
<y0>	0	Concentration initial value
<options>	[]	No option is used
...	p	A structure passing additional model parameters ( <i>V</i> , <i>CL</i> ) into the function

**Listing 3.2** Program **infus1inter.m**—full implementation

```

function infus1inter()
%INFUS1INTER() One-Compartment Model with Infusion
%   INFUS1INTER() computes plasma concentration-time course and
%   creates graphics output for one-compartment model. A 200 [mg]
%   dose was given as 2-h infusion. Time courses of plasma
%   concentration and infusion rate are plotted. The function has
%   two parameters, volume (V) and clearance (CL)

% PK parameter(s) specification
p.V = 12.6;      % Volume of distribution
p.CL = 2.16;     % Clearance

% Calling the differential equation solver (Runge-Kutta Method)
[t,c] = ode45(@derivatives, 0:0.01:15, 0, [], p);

% Generating graphs
subplot(3, 1, 1);
plot(t, r(t), 'Color', 'k', 'LineWidth', 2)
title(['One-Compartment Model with Infusion', 10], 'FontSize', 15)
ylim([0 110])
ylabel(['Infusion' 'Rate [mg/h]'], 'FontSize', 12)
subplot(3, 1, 2:3);
plot(t, c, 'Color', 'k', 'LineWidth', 2)
xlabel('Time [h]', 'FontSize', 12)
ylabel('Concentration [mg/L]', 'FontSize', 12)
% save graphs as TIFF file
print('-r900', '-dtiff', 'infus1inter')

end

function dcdt = derivatives(t, c, p)
%DERIVATIVES Compute the right-hand side of the ODE.
%   DCDT = ...

dcdt = r(t)/p.V - p.CL/p.V*c;

end

function rt = r(t)
%R Specifies Infusion Rate
%   RT = R(T) returns infusion rate |RT| at time |T|

rt = 200/2 * (t < 2);

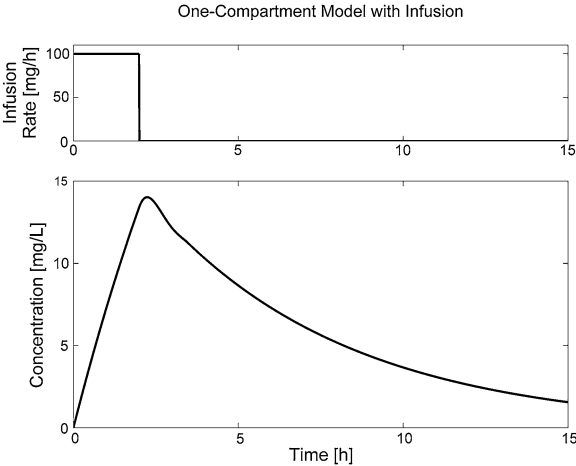
end

```

The output generated by program **infus1inter** (Listing 3.2) is shown in Fig. 3.1. To better visualize the analyzed model, the time course of the infusion rate was added to the graph.

Surprisingly, the discontinuous change in the infusion rate at time  $t = 2$  h is not reflected in the corresponding concentration–time profile which looks too smooth

**Fig. 3.1** Time course of plasma concentrations for a one-compartment model with drug infusion input. The upper graph shows the infusion rate and the lower graph the time course of plasma concentrations for the time interval [0, 15]



**Table 3.4** List of options to control error properties of ODE solvers

Option	Value {Default value}	Meaning
RelTol	Value > 0 {1e-3}	Relative error tolerance that applies to all components of the solution vector, <b>y</b> . This tolerance is a measure of the error relative to the size of each solution component. Roughly, it controls the number of correct digits in all solution components, except those smaller than thresholds AbsTol(i)
AbsTol	Value > 0 {1e-6}	Absolute error tolerances that apply to the individual components of the solution vector
NormControl	on {off}	Control error relative to norm of solution. This property requests that the solvers control the error in each integration step with $\text{norm}(\mathbf{e}) \leq \max(\text{RelTol} * \text{norm}(\mathbf{y}), \text{AbsTol})$ . By default the solvers use a more stringent component-wise error control

at this time. One way to verify the accuracy of the solution is to pass error control options to the ODE solver, thereby overriding default values for the <options> parameter. These options are described in [2] and Table 3.4.

There are two ways to declare options:

```
<options> = odeset ( '<option1>' , <val1> , '<option2>' , <val2> , ... ) ;
```

or

```
<options> = odeset ;  
<options> . <option1> = <val1> ;  
<options> . <option2> = <val2> ;
```



In both cases a structure `<options>` with fields `<option1>` , `<option2>` ... and values `<val1>` , `<val2>` ... will be created. For example:

```
options = odeset('RelTol',1e-6, 'AbsTol',1e-6);  
options = odeset('RelTol',1e-3, 'AbsTol',1e-4);
```

or, if the other declaration is used:

```
options = odeset;  
options.RelTol = 1e-3; options.AbsTol = 1e-4;  
options.RelTol = 1e-6; options.AbsTol = 1e-6;
```

These and additional cases were implemented in MATLAB. The source of the respective function, **infus1comstep**, is available at MATLAB Central. The output is shown in Fig. 3.2, where differences in time courses of plasma concentrations become evident at times shortly after the infusion stop (infusion rate jumped from 10 to 0 at time  $t = 2$  h). The exact solution is only met if error control parameters are set as in condition B (Fig. 3.2), 'AbsTol' = 1e-6 and 'RelTol' = 1e-6. Inaccurate solutions of ODEs (in this case **ode45**) might affect model parameter estimation, which is a standard task in PK and PD analyses.

The time of infusion stop is also an example of a “time event”. Events scheduling for ODEs will be described later, in *Scheduling Time Events* and *Scheduling State Events*.

### 3.2.3 Delay Differential Equations

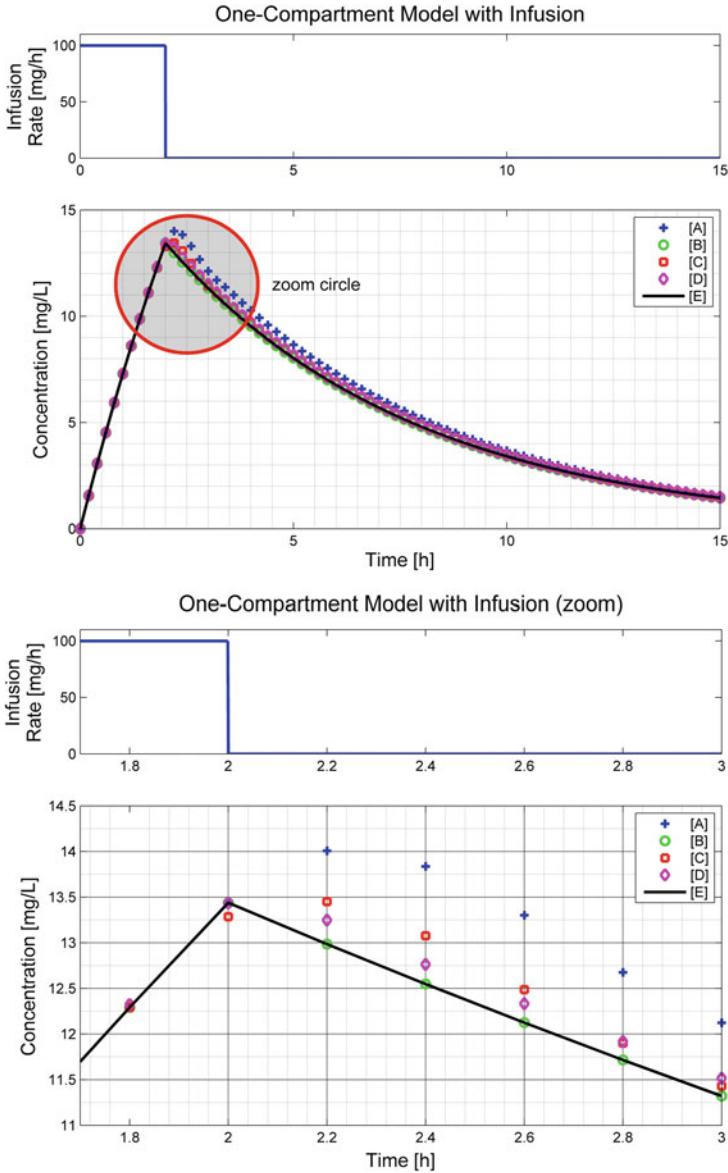
Some biological phenomena are naturally described by a delay differential equation (DDE). A DDE has the general form:

$$\left. \begin{aligned} \frac{dy}{dt} &= f(t, y_i(t - \tau_j)); & t \in [t_0, t_f], & \quad i \in \{1, 2 \dots m\}, \quad j \in \{1, 2 \dots n\} \\ y(t) &= h(t); & t \in [-\tau_{\max}; t_0] \end{aligned} \right\} \quad (3.10)$$

where  $\mathbf{y} = [y_1, y_2 \dots y_m]^T$  is a vector of dependent variables,  $\boldsymbol{\tau} = [\tau_1, \tau_2 \dots \tau_n]^T$  a vector of non-negative real numbers, and  $\tau_{\max}$  the largest of these numbers. The syntax for calling a DDE solver in MATLAB is:

```
<sol> = <solver>(@ddefun,<lags>,<history>,<tspan>,<options>,...);
```

`<sol>` is a structure returned by the solver `<solver>`. The fields of the `<sol>` structure and the meaning of solver arguments are explained in Table 3.5.



**Fig. 3.2** Time course of plasma concentrations created for different error tolerances, specified by parameters 'RelTol' (relative tolerance) and 'AbsTol' (absolute tolerance): **A**—'RelTol' =  $1e-3$ , 'AbsTol' =  $1e-6$ ; **B**—'RelTol' =  $1e-6$ , 'AbsTol' =  $1e-6$ ; **C**—'RelTol' =  $1e-3$ , 'AbsTol' =  $1e-4$ ; **D**—'RelTol' =  $1e-3$ , 'AbsTol' =  $1e-7$ ; **E**—exact (analytical) solution. Upper panel: solution over time interval  $[0, 15]$ . Lower panel: solution for time interval  $[1.9; 3.0]$  including infusion stop time,  $t = 2$  h

**Table 3.5** List of parameters passed to or returned by MATLAB DDE solvers

Formal parameters	Meaning
<sol> <b>.x</b>	Independent variable
<sol> <b>.y</b>	Solution array with column vector representing dependent variables
<sol> <b>.yp</b>	Solution array with column vector representing approximated values of derivatives of dependent variables
<sol> <b>.solver</b>	Name of the used solver
<solver>	<b>dde23</b>   <b>ddesd</b>
<ddefun>	Name of the DDE function defining the right-hand side of the DDE. This function has the header: <output>= <b>&lt;ddefun&gt;</b> (<t> , <y> , <z> , ...) where <t> is time, <y>—solutions vector and <z>—an array explained below
<lags>	Vector $\tau$ providing values of delays in the DDE
<history>	Name of the function that specifies values of the dependent variable in time interval $[-\tau_{\max}; t_0]$
<tspan>	Time interval for which the DDE is to be solved, $[t_0; t_f]$
<options>	Structure with optional parameters for integration properties and handling events
...	Additional (model) parameters

The function <ddefun> defines the right-hand side of the DDE. It must have as input an array, <Z>, containing values for all elements of the vector <y>, at all the delayed arguments; for practical applications we can use the schema shown in Table 3.6, where  $\mathbf{Z}(\mathbf{i}, \mathbf{j}) \equiv y_i(t - \tau_j)$ .

The elements of <solver> are briefly described in Table 3.7, based on [2].

### Introductory DDE Example

The following example describes a one-compartment PK model where drug is absorbed according to a first-order process with bioavailability  $F$  and eliminated with constant clearance and delayed drug concentrations:

$$\left. \begin{aligned} \frac{da(t)}{dt} &= -k_a \cdot a(t) \\ \frac{dc(t)}{dt} &= k_a \cdot \frac{F}{V} \cdot a(t) - \frac{CL}{V} \cdot c(t - t_{\text{lag}}) \end{aligned} \right\}; \quad t \geq 0 \quad (3.11)$$

**Table 3.6** Dependency between array <Z>  $\equiv \mathbf{Z}$  and solutions  $y_i$  at delayed time points  $(t - \tau_j)$ 

$\tau$	$\tau_1$	$\tau_2$	...	$\tau_n$
<b>y</b>				
$y_1$	$\mathbf{Z}(1, 1)$	$\mathbf{Z}(1, 2)$	...	$\mathbf{Z}(1, n)$
$y_2$	$\mathbf{Z}(2, 1)$	$\mathbf{Z}(2, 2)$	...	$\mathbf{Z}(2, n)$
...	...	...	...	...
$y_m$	$\mathbf{Z}(m, 1)$	$\mathbf{Z}(m, 2)$	...	$\mathbf{Z}(m, n)$

**Table 3.7** List of MATLAB DDE solvers

Solver	Problem type	Order of accuracy	When to use
dde23	Solves DDE with limited type of delays	As might be expected for the specified error tolerances	All delays are constant
ddesd	Solves DDE with general delays	As might be expected for the specified error tolerances	Delays are functions of time

where  $a(t)$  is the drug amount at the site of absorption,  $k_a$ —absorption rate constant,  $c(t)$  and  $c(t-t_{\text{lag}})$ —drug concentrations at times  $t$  and  $(t-t_{\text{lag}})$ ,  $F$ —drug bioavailability,  $CL$ —clearance, and  $V$ —volume of distribution. The model (3.11) has the form of a DDE system.

To solve (3.11), the MATLAB DDE solver **dde23** is used. Details of the implementation are presented in Listing 3.3.

### Listing 3.3 Program **DDEmar.m**

```
function DDEmar
%DDEMAR solves a PK model with time delay
% DDEMAR is a simple example that solves a PK model equation
% described in form of a DDE. The computation is repeated for
% several delay values. For comparison, the solutions are plotted
% in the same figure.

% PK Parameters
p.F = 1;
p.CL= 2.16;
p.V = 12.6;
p.ka = 0.1;
% parameters for calling |dde23|
p.c0 = 0; % init. concentration
p.A0 = 100; % init. drug amount [microgram]
tspan = [0 60];
options = ddeset('RelTol',1e-6, 'AbsTol', 1e-6);
alag = {3 2 1}; % possible delays
a = axes; set(a,'FontSize', 15)
for si = struct('alag', alag, 'col', {'b', 'g', 'r'})
    sol = dde23(@derivatives, si.alag, @history, tspan, options, p);
    t = sol.x;
    y = sol.y;
    plot(t, (y(2,:)),si.col, 'LineWidth', 2); hold on
end
```

```

% compare with analytical solution without delay (alag=0)
Dose = p.A0; F = p.F; ke = p.CL/p.V; ka = p.ka; V = p.V;
yana = (F*Dose/V)*(ka/(ka-ke))*(exp(-ke*t) - exp(-ka*t));
plot(t, yana, 'k', 'LineWidth', 2);

legend(['t_{lag}=' num2str(alag{1})], ['t_{lag}=' ...
    num2str(alag{2})], ['t_{lag}=' num2str(alag{3})], ...
    't_{lag}=0', 'Location', 'Best')
title('Concentration-Time Course');
xlabel('Time [h]')
ylabel('Concentration [\mu g/L]')
print('-r900', '-dtiff', 'd demar')

end

function dydt = derivatives(~, Y, Z, p)
%DERIVATIVES Compute the right-hand side of the DDE.
% DYDT = DERIVATIVES(T, Y, Z, P) calculates |DYDT|, the right-hand
% side of the DDE model, at points defined by the vector |Y|, |T|,
% and with parameters |P|. Matrix |Z| corresponds to all possible
% delays and contains solutions |Y| at the delayed argument. Thus,
% |Z| has the number of rows equal to the length of |Y|, and the
% number of columns equal to the number of delays. For example:
% Z(2,1) means y(2) at time |T|-clag; clag is a delay.

% PK Model
a = y(1); % amount
%c = y(2); % y(2) at t is not used directly
clag = Z(2,1); % y(2) at delayed time point (t-clag)
dadt = -p.ka*a;
dcdt = (p.ka/p.V*p.F*a - p.CL/p.V*clag);
dydt = [ dadt
        dcdt ];

end

function yhist = history (~, p)
%HISTORY Provide the history of the solution
% YHIST = HISTORY(T, P) specifies the time profiles of dependent
% variables at times |T| prior to the initial point. This function
% is a form of 'initial condition' for the DDE. |P| stands for
% additional parameters which can be passed to the function.

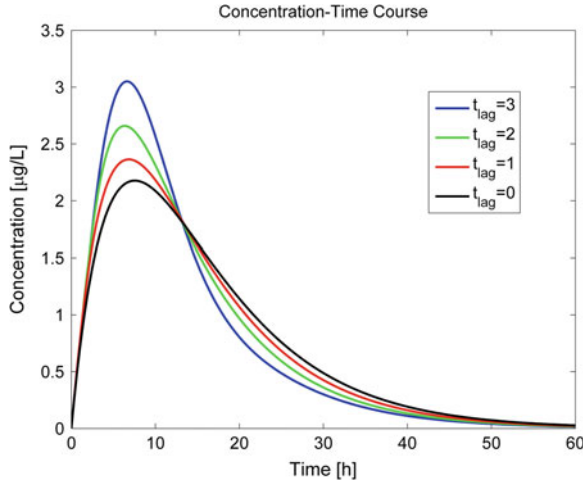
yhist = [p.A0 p.c0];

end

```

The output from the program **DDEmar.m** is visualized in Fig. 3.3.

A real example of a process exhibiting delays will be presented in [Chap. 5, Drug-Disease Modeling](#), with the hematopoietic system.



**Fig. 3.3** Simulation of drug concentrations over time for a one-compartment model. Drug input and drug elimination follow a first-order process but drug elimination is delayed by different lag times. DDE model equations are shown in (3.11). DDE solver **dde23** was applied. The case  $t_{\text{lag}} = 0$  was plotted using the analytical solution of (3.11) as the DDE solver cannot be applied for a model without delay

### 3.2.4 Differential Algebraic Equation

Consider the following equations which describe a time-dependent infusion rate into a one-compartment PK model with first-order elimination. Drug concentrations are linked to an  $E_{\text{max}}$  model. For details see [Chap. 4, Pharmacologic Modeling](#).

$$\begin{bmatrix} \frac{dc}{dt} \\ E \end{bmatrix} = \begin{bmatrix} \frac{r(t)}{V} - k \cdot c \\ \frac{E_{\text{max}} \cdot c}{c + E_{50}} \end{bmatrix}; \quad t \in [0; 15] \quad (3.12)$$

The initial values and PK-PD parameters are

$$c(0) = 0; \quad E_{\text{max}} = 20; \quad E_{50} = 10; \quad r(0) = 1; \quad k = 0.1; \quad V = 6 \quad (3.13)$$

Interestingly, (3.12) contains both a differential equation (representing PK), and an algebraic equation (representing PD). In order to solve such equations one may first solve the differential equation and enter the solution into the algebraic equation in a second step. Although this appears practical, issues arise when the algebraic equation triggers another differential equation. In order to obtain the time profile of the algebraic equation together with the solution of the differential equations, MATLAB offers the specification of a differential algebraic equation (DAE) system, i.e., a system of equations containing both differential and algebraic equations. This is accomplished by use of the **Mass** option in the following way:

```
<options> = odeset ( 'Mass', <Mass-matrix> );
```

or alternatively:

```
<options> = odeset ;  
<options> .Mass = <Mass-matrix> ;
```

In the above statements, <Mass-matrix> corresponds to a singular matrix  $M$ , and the general form of the DAE given by:

$$M(t, y) \cdot \frac{dy}{dt} = f(t, y) \quad (3.14)$$

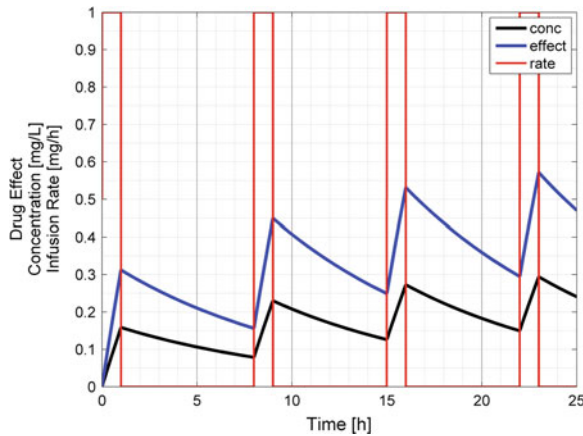
To apply a MATLAB solver to the model presented in (3.12), we rewrite the model as follows:

$$\begin{bmatrix} \frac{dc}{dt} \\ E \end{bmatrix} = \begin{bmatrix} \frac{r(t)}{V} - k \cdot c \\ \frac{E_{\max} \cdot c}{c + E_{50}} \end{bmatrix} \Leftrightarrow \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} \frac{dc}{dt} \\ \frac{dE}{dt} \end{bmatrix} = \begin{bmatrix} \frac{r(t)}{V} - k \cdot c \\ \frac{E_{\max} \cdot c}{c + E_{50}} - E \end{bmatrix} \quad (3.15)$$

Comparing (3.14) with (3.15) shows that

$$M = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}; \quad y = \begin{bmatrix} c \\ E \end{bmatrix}; \quad \frac{dy}{dt} = \begin{bmatrix} \frac{dc}{dt} \\ \frac{dE}{dt} \end{bmatrix}; \quad f(y, t) = \begin{bmatrix} \frac{r(t)}{V} - k \cdot c \\ \frac{E_{\max} \cdot c}{c + E_{50}} - E \end{bmatrix} \quad (3.16)$$

Based on this form, a MATLAB implementation of the PK-PD model, **emaxDAE.m**, was created (Listing 3.4). Figure 3.4 shows the graph produced by program **emaxDAE.m**.



**Fig. 3.4** The graph produced by program **emaxDAE.m** (and the linked functions). It shows the time courses of the drug concentrations and drug effects calculated by the Hill equation. The infusion rate of 1 mg/h is given over 1 h, and repeated every 8 h

Listing 3.4 Program `emaxDAE.m`

```

function emaxDAE()
%EMAXDAE
%   EMAXDAE() schedules PK-PD events, using a DAE solver. Drug
%   effect should be observed within a given observation time.
%   Infusion rate is a control variable.

% PK-PD parameters
p.Rate = 1;      % infusion rate (mg/h)
p.V     = 6;      % volume of distribution (L)
p.k     = 0.1;    % elimination rate constant (1/h)
p.Emax  = 20;     % max effect (in Hill equation)
p.E50   = 10;     % drug conc. linked to 50% of maximum effect (mg/l)

% Calling ODE
M = [ 1 0      % specifies Mass matrix
      0 0 ];
options = odeset('Mass', M, 'RelTol',1e-6, 'AbsTol', 1e-6);
tspan = [0 25];
% Initial value for the dependent variables
c0 = 0;
E0 = hilleffect(c0, 0, p.Emax, p.E50, 1);
[t,c] = ode15s(@derivalgeb, tspan, [c0 E0], options, p);
% Generate graphs
set(axes,'FontSize',14)
plot(t, c(:,1), 'Color', 'k', 'LineWidth', 3)
hold on;
plot(t, c(:,2), 'Color', 'b', 'LineWidth', 3)
plot(t, rate(t, p), 'Color', 'r', 'LineWidth', 2)
grid minor; grid on ;
xlabel('Time [h]');
ylabel({'Drug Effect', 'Concentration [mg/L]', ...
        'Infusion Rate [mg/h]'});
legend('conc','effect','rate')
% save graphs as TIFF file
print('-r600', '-dtiff', 'emaxDAE')

end

function massdydt = derivalgeb(t, y, p)
%DERIVALGEB Compute the right-hand side of the DAE.
%   MASSDYDT = DERIVALGEB(T, Y, P) calculates |DYDT|, the right-hand
%   side of the DAE model at points defined by the vector of
%   dependent variables |Y|, time |T|, and with parameters |P|.
%   The right-hand side represents both the differential equations
%   and algebraic equations.

massdydt = [rate(t, p)/p.V - p.k*y(1);
            hilleffect(y(1),0,p.Emax,p.E50,1) - y(2)];

end

function r = rate(t, p)
%RATE specify infusion rate within observation time
%   R = RATE(T, P) calculates the infusion rate value at time |T|

tR = [0 1; 8 9; 15 16; 22 23]'; % rate times (begin end; ... )
r = 0;
for ir = tR

```



```

        r = r + p.Rate*(heaviside(t-ir(1)) - heaviside(t-ir(2)));
    end

    end

    function E = hilleffect(c, E0, Emax, EC50, n)
    % .....
    % .....
    % .....

    end

```

### 3.2.5 Scheduling Time Events

The basic ODE example in [Sect. 3.2.2](#) uses as forcing function an infusion that was stopped 2 h after start. In the following, we will name such events ‘time events’ as they depend only on time. It will be the first group of events we will deal with. The second group will be ‘state events’, caused by the state of one or more dependent variables. Combinations of time and state events are possible. For example, to keep drug effects under control (i.e., within a safe and effective range), it is important to follow the respective state events and adjust drug dosing accordingly. Instead of declaring the infusion rate from above as a constant function of time, we may change its value depending on the value of other system variables. In the following, we will describe how to use MATLAB procedures for handling events, starting with a one-compartment model with infusion described in [Sect. 3.2.2](#).

First, we need to inform MATLAB that it should detect events. This is done by the following statement:

```
<options> = odeset ('Events', @<events>);
```

It tells MATLAB that the option ‘**Events**’ will be used and that events are defined in the function <events>. Furthermore, the statement calling an ODE function has to be used in an extended form. The following syntax applies:

```
[<T>, <Y>, <TE>, <YE>, <IE>] = <solver>(@<odefun>,
    <tspan>, <y0>, <options>, ...);
```

or

```
<sol> = <solver>(@<odefun>, <tspan>, <y0>, <options>, ...);
```

The meaning of the additional parameters in the extended syntax is explained in [Table 3.8](#).

**Table 3.8** List of parameters passed to or returned by MATLAB ODE solvers handling events

Formal parameters	Meaning
<TE>	Time when an event occurred
<YE>	ODE solution at the time of the event
<IE>	Index defining which event occurred
<sol> . <b>xe</b>	Time when an event occurred (if <sol> is used)
<sol> . <b>ye</b>	ODE solution at the time of the event (if <sol> is used)
<sol> . <b>ie</b>	Index defining which event occurred (if <sol> is used)
<options>	Contains the option ' <b>Events</b> '

Similarly, if we want to handle events within DDEs, then the general DDE function call is given by:

```
<sol> = <solver>(@<ddefun>,<lags>,<history>,  
                <tspan>,<options>,...);
```

With and without events, the calling syntax for DDE systems is the same, but the additional information linked to events can only be returned in the structure <sol>. A summary of this additional information is shown in Table 3.9, and is similar to the case with ODE.

In MATLAB, an event is characterized by the zero crossings of a function  $e$  depending on  $t$ . For example, if we were interested only in an event at time  $t = 2$ , the function  $e$  would be  $e(t) = t - 2$ . In case of more than one event, we have to deal with a vector of single event functions. Each single function is addressed by the index of this function vector. The general syntax for defining events in MATLAB is:

```
function [<value>,<isterminal>,<direction>] =  
    <events>(<T>,<Y>,...);
```

where: <event>—name of the event function depending on <T>, <Y>; <value>—vector-valued output from the event function; <isterminal>—vector assigning to each event one of two values: 0 (if the integration is to be continued) or 1 (if the integration is to be stopped); <direction>—vector containing for each event function one of three values: 1 (if the corresponding <value> passes 0 in positive direction) or -1 (if the corresponding <value> passes 0 in negative direction) or 0 (if the corresponding <value> passes 0 in positive or negative direction). The <events> function reports when an event occurred and in variable <IE> returns the respective index, i.e., which event has occurred, to the calling ODE solver.

The following example shows how to implement events in MATLAB (Listing 3.5).

**Table 3.9** List of parameters passed to or returned by MATLAB DDE solvers handling events

Formal parameters	Meaning
<sol> . <b>xe</b>	Time when an event occurred
<sol> . <b>ye</b>	ODE solution at the time of the event
<sol> . <b>ie</b>	Index defining which event occurred
<options>	Contains the option ' <b>Events</b> '

**Listing 3.5** Program **infus1com.m**—final implementation

```

function infus1com()
%INFUS1COM() One-compartment model with infusion
%   INFUS1COM() computes plasma concentration versus time course
%   and creates plots for a one-compartment model. Drug doses were
%   applied as infusions. The function has no parameters.

% PK parameter(s) specification
p.Dose = 200;           % dose [mg]
p.tStop = 2;           % infusion time stop [h]
p.R = p.Dose/p.tStop;  % infusion rate [mg/h]
p.V = 12.6;            % volume of distribution [L]
p.CL = 2.16;           % clearance [L/h]

options = odeset('Events', @fevents, 'RelTol', 1e-3, 'AbsTol', 1e-3);
ie = 99;               % any non-empty value to avoid 't=0' as event
tspan = [0,100];       % integration time interval
c0 = 0;                % initial concentration value

% prepare data for graphs
timePoints = []; concPoints = []; ratePoints = [];
% Call differential equation solver (Runge-Kutta Method)
% and check events
while ~isempty(ie)
    [t,c,te,ce,ie] = ode45(@derivatives, tspan, c0, options, p);
    % collect data to generate graph
    timePoints = [timePoints; t];    %#ok<AGROW>
    concPoints = [concPoints; c];    %#ok<AGROW>
    ratePoints = [ratePoints; p.R*ones(length(t),1)];    %#ok<AGROW>
    tspan = [te(end) 100];
    c0 = ce(end,1);
    % check if an event is detected
    switch ie(end)
        case 1      % event: stop infusion
            p.R = 0;
        case 2      % event: end of observation
            ie = [];
        otherwise
            % no action needed - placeholder for other events
    end;
end

% Generating graphs
subplot(3, 1, 1)
plot(timePoints, ratePoints, '-k', 'LineWidth', 2); hold on
title({' ', '\fontsize{13}One-Compartment Model With Infusion', ''})
ylim([0 110])
ylabel('\fontsize{13}Rate [mg/h]')
subplot(3, 1, 2:3);
plot(timePoints, concPoints, '-k', 'LineWidth', 2)
hold on; grid minor; grid on;
xlabel('\fontsize{13}Time [hour]')
ylabel('\fontsize{13}Concentration [mg/L]')
% save graphs as TIFF file
print('-r900', '-dtiff', 'infus1com')

end

```

```

function dcdt = derivatives(t, c, p)
%DERIVATIVES Compute derivatives of the ODE
% DCDT = ...

dcdt = r(t,p)/p.V - p.CL/p.V*c;

end

function rt = r(~, p)
%R Specifies Infusion Rate
% RT = R(T,P) returns infusion rate |RT| at time |T|, and with
% parameters |P|

% infusion (depends on event)
rt = p.R;

end

function [value, isterminal, direction] = fevents(t, ~, p)
%FEVENTS Specify events for the ODE model.
% [VALUE,ISTERMINAL,DIRECTION] = FEVENTS(T,Y,P) detects the exact
% time points of events specified as a function of time |T|,
% vector of dependent variables |Y|, and a structure of some )
% (specific) other parameters |P|.
%
% Output parameters: |VALUE| - value investigated as event;
% |ISTERMINAL| = 0|1 where 0 stands for 'don't stop', 1 for 'stop
% integration'; |DIRECTION| = -1|0|1 where (-1) means 'negative
% direction only', 0 - 'both direction', 1 - 'positive direction
% only'

% 1. Event: Locate the time to stop infusion
value(1,1) = t - p.tStop; % Detect t = 2 (infusion stop time)
isterminal(1,1) = 1; % Interrupt integration, stop infusion
direction(1,1) = 1; % Positive direction only

% 2. Event: Locate the time to stop observation/integration
value(2,1) = t - 15; % Detect t = 15 (end of observation)
isterminal(2,1) = 1; % Stop integration
direction(2,1) = 1; % Positive direction only

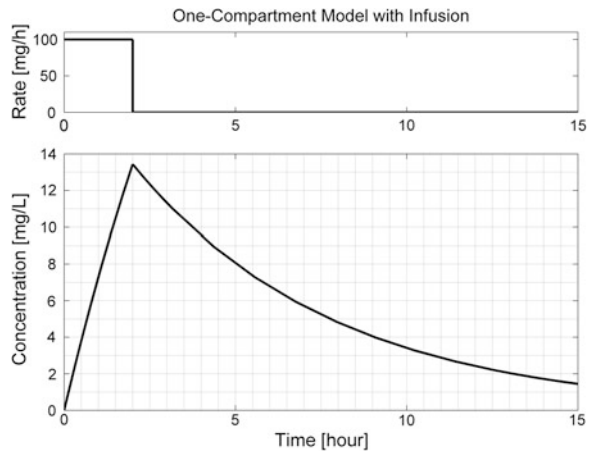
end

```

The result of this implementation is shown in Fig. 3.5. It is worth mentioning that usage of the event option provided a solution with high accuracy despite larger error tolerance values ('AbsTol' = 1e-3, 'RelTol' = 1e-3) compared to Listing 3.2 ('AbsTol' = 1e-6, 'RelTol' = 1e-6). The output given by **infus1com** shows high accuracy and is virtually identical with the exact solution from Fig. 3.2.

In the next example, we consider a system of two differential equations, corresponding to a two-compartment model, where again a dose of 100 mg of a hypothetical drug is given as 2 h infusion. The following PK parameters are used: clearance,  $CL = 13.8$  L/h, central volume of distribution,  $V_l = 14.8$  L,

**Fig. 3.5** Final implementation of the oral one-compartment model. The infusion stop time is handled as a time event



intercompartmental clearance,  $Q = 2.17$  L/h, and peripheral volume of distribution,  $V_2 = 4.23$  L. Drug concentrations over time are determined by the following ODE system:

$$\begin{bmatrix} \frac{dc_1}{dt} \\ \frac{dc_2}{dt} \end{bmatrix} = \begin{bmatrix} \frac{r(t)}{V_1} - (k + k_{12}) \cdot c_1 + k_{21} \cdot \frac{V_2}{V_1} \cdot c_2 \\ k_{12} \cdot \frac{V_1}{V_2} \cdot c_1 - k_{21} \cdot c_2 \end{bmatrix}; \quad t \in [0; 15] \quad (3.17)$$

with initial values

$$c_1(0) = c_2(0) = 0 \quad (3.18)$$

The infusion rate and rate constants are defined as follows:

$$\left. \begin{aligned} r(t) &= \begin{cases} 50; & t \in [0; 2) \\ 0; & t \geq 2 \end{cases} \\ k &= \frac{CL}{V_1}; \quad k_{12} = \frac{Q}{V_1}; \quad k_{21} = \frac{Q}{V_2} \end{aligned} \right\} \quad (3.19)$$

A MATLAB implementation of the model described in (3.17)–(3.19) is shown in Listing 3.6, as program **infus2com.m**.

Listing 3.6 Program `infus2com.m`

```

function infus2com()
%INFUS2COM() Two-compartment model with infusion.
%   INFUS2COM() computes plasma concentration time course and creates
%   graph for two-compartment model. In the treatment, doses were
%   given as infusions to each subject. A graph with time courses of
%   plasma concentration and infusion is generated. The function has
%   no arguments.

p.Rate = 50;           % initial infusion rate = 100/2 [mg/h]
p.CL   = 1.38E+01;     % central clearance
p.V1   = 1.48E+01;     % volume of distribution in central compartment
p.Q    = 2.17E+00;     % inter-compartmental clearance
p.V2   = 4.23E+00;     % volume of distribution peripheral compartment
p.k    = p.CL/p.V1;    % rate constant of elimination
p.k12  = p.Q/p.V1;     % rate constant from central to peripheral
p.k21  = p.Q/p.V2;     % rate constant from peripheral to central
options = odeset('Events',@fevents, 'RelTol',1e-6, 'AbsTol', ...
    [1e-6 1e-6]);
% Initial values
tspan = [0 50];       % max. time domain
c0 = [0 0];           % Initial value for the dependent variable
ie = 99;               % avoid empty events vector
while ~isempty(ie)
    % Calling Runge-Kutta Method
    [t,c,te,ye,ie] = ode45(@derivatives, tspan, c0, options, p);
    % Generate graph
    plot(t, c(:,1), '-', 'Color', 'r', 'LineWidth', 2)
    hold on;
    plot(t, c(:,2), '-.', 'Color', 'b', 'LineWidth', 2)
    tspan = [te(end) 50];
    c0 = [ye(end,1) ye(end,2)];
    % check if an event is detected
    switch ie(end)
        case 1           % event: stop infusion
            p.Rate = 0;
        case 2           % event: end of observation
            ie = [];
        otherwise
            % no action needed
    end;
end
end
% Describe graph
title('Two-Compartment Model with Infusion')
xlabel('Time [h]');
ylabel('Concentration [mg/L]');
text('units','inch', 'position',[1.9 2.7], ...
    'fontsize',11, 'interpreter','latex', 'string',...
    ['$\frac{dc_1}{dt} = \frac{r}{V_1} - (k_{12} + k_{21})c_1 + ' ...
    'k_{21}c_2$']);
text('units','inch', 'position',[1.9 2.2], ...
    'fontsize',11, 'interpreter','latex', 'string', ...
    ['$\frac{dc_2}{dt} = k_{12}c_1 - k_{21}c_2$']);
legend('central compartment','peripheral compartment')
% save graphs as TIFF file
print('-r900', '-dtiff', 'infus2com')
end

```

```

function dcdt = derivatives(~, c, p)
%DERIVATIVES Compute the right-hand side of the ODE.
% DCDT = ...

dcdt = [p.Rate/p.V1 - (p.k+p.k12)*c(1) + p.k21*p.V2/p.V1*c(2)
        p.k12*p.V1/p.V2*c(1) - p.k21*c(2)
        ];
end

function [value, isterminal, direction] = fevents(t, ~, ~)
%FEVENTS Specify events for the ODE model.
% [VALUE,ISTERMINAL,DIRECTION] = ...

% 1. Event: Locate the time to stop infusion
value(1,1) = t - 2;          % Detect t = 2 i.e. time to stop infusion
isterminal(1,1) = 1;         % Stop integration
direction(1,1) = 1;          % Positive direction only

% 2. Event: Locate the time to stop observation / integration
value(2,1) = t - 15;         % Detect t = 15 i.e. end of observation
isterminal(2,1) = 1;         % Stop the integration
direction(2,1) = 1;          % Positive direction only

end

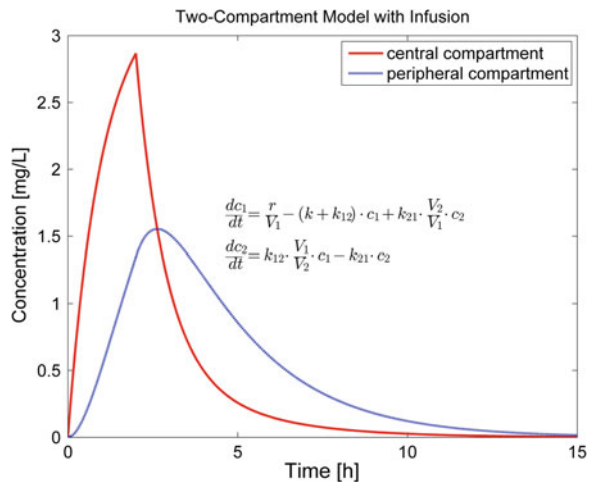
```

Figure 3.6 shows the output generated by the function `infus2com.m`, including the TeX notation of the ODE equations.

### 3.2.6 Scheduling State Events

In this section we deal with the other group of events in ODEs: state events. A state event is detected at the zero-crossings of a prespecified function  $e$  of  $t$  and vector of dependent variables  $\mathbf{y}$ , which can be expressed as:

**Fig. 3.6** Graphs generated by the implementation of the two-compartment model. The graph contains time courses of plasma drug concentrations in both central and peripheral compartments



$$e(t, \mathbf{y}) = 0 \quad (3.20)$$

During the numerical integration, MATLAB evaluates (3.20) using the actual values for  $t$  and  $\mathbf{y}$ . As ‘event location’ we understand a time  $t_e$  that fulfills (3.20) with  $\mathbf{y}$  equal to the corresponding vector of values  $\mathbf{y}_e$ . Any detected event can be used to apply rules changing the underlying ODE. As an example, we consider the two-compartment model (3.17), and the following rules for handling specified events:

- Time Event (event #1): First infusion lasts for 2 h  
Rule: Stop first infusion
- Time Event (event #2): Integration lasts for 30 h  
Rule: Stop integration
- State Event (event #3): Central compartment concentration has increased to 2 mg/L  
Rule: Set infusion rate to 5 mg/h
- State Event (event #4): Central compartment concentration has decreased to 0.5 mg/L  
Rule: Set infusion rate to 50 mg/h

In order to implement this task, we have to modify the block for handling events in program **infus2com.m**. This modification is presented in Listing 3.7 below.



**Listing 3.7** Program **infus2comstev.m**

```

function infus2comstev
%INFUS2COMSTEV Two-compartment model with state events
%   INFUS2COMSTEV implements state events in two-compartment model.
%   Plasma concentration should be kept within a given range by
%   infusion rate: if drug concentration achieves the maximum then
%   decrease infusion; if minimum - infusion has to be increased.

% PK parameters
CL = 1.38E+01;      % clearance
V1 = 1.48E+01;      % volume of distribution in 1st compartment
Q = 2.17E+00;       % intercompartmental clearance
V2 = 4.23E+00;      % volume of distribution in 2nd compartment
Rate = 50;          % infusion rate
% p structure with PK parameters available in several functions
p.Rate = Rate;
p.V1 = V1;
p.V2 = V2;
p.k = CL/V1;
p.k12 = Q/V1;
p.k21 = Q/V2;

options = odeset('Events',@fevents);
tspan = [0 50];
c0 = [0 0]; % Initial value of concentrations
ie = 99;
while ~isempty(ie)
    [t, c, te, ye, ie] = ode45(@derivatives, tspan, c0, options, p);
    plot(t, c(:,1), 'Color','r','LineWidth',2)
    hold on;
    plot(t, c(:,2), 'Color','b','LineWidth',2)
    tspan = [te(end) 50];
    c0 = [ye(end,1) ye(end,2)];
    switch ie(end)
        case 1
            % stop infusion after 2 hours
            p.Rate = 0;
        case 2
            % stop process definitely
            ie = [];
        case 3
            % reduce infusion if conc. has grown to allowed max
            p.Rate = Rate/10;
        case 4
            % allow full infusion rate if conc. has fallen to min
            p.Rate = Rate;
        otherwise
            messageID = 'Book:infus2comstev:UnknownEvent';
            messageStr = 'Unknown event '%s'.';
            error(messageID, messageStr, num2str(ie(end)));
    end
end
end

```

```

hold off;
xlabel('Time [h]');
ylabel('Concentration [mg/L]');
legend('central compartment','peripheral compartment')
title('Two-Compartment Model / Infusion / Multiple Doses')
% save graphs as TIFF file
print('-r900', '-dtiff', 'infus2comstev')

end

function dcdt = derivatives(~, c, p)
%DERIVATIVES Compute the right-hand side of the ODE.
% DCDT = ...

dcdt = [ p.Rate/p.V1 - (p.k + p.k12)*c(1) + p.k21*p.V2/p.V1*c(2)
        p.k12*p.V1/p.V2*c(1) - p.k21*c(2) ]';
end

function [value,isterminal,direction] = fevents(t,y,~)
%FEVENTS Specify events for the ODE model.
% [VALUE,ISTERMINAL,DIRECTION] = ...

value(1,1) = t - 2;           % max time for first infusion
isterminal(1,1) = 1;          % Stop the integration
direction(1,1) = 1;           % Positive direction only

value(2,1) = t - 30;          % End of observation
isterminal(2,1) = 1;          % Stop the integration
direction(2,1) = 1;           % Positive direction only

value(3,1) = y(1) - 2;        % Detect concentration = max while growing
isterminal(3,1) = 1;          % Stop the integration
direction(3,1) = 1;           % Positive direction only

value(4,1) = y(1) - 0.5;      % Detect concentration = min while falling
isterminal(4,1) = 1;          % Stop the integration
direction(4,1) = -1;          % Negative direction only

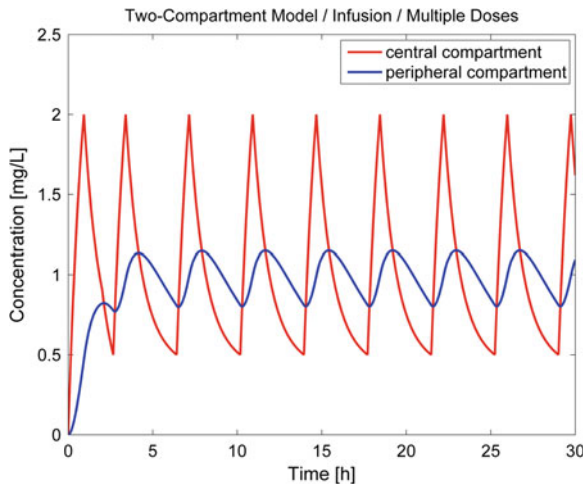
end

```

The output from the above program is visualized in Fig. 3.7.

Another example for scheduling state events related to PK and PD states is given below.

$$\begin{bmatrix} \frac{dc}{dt} \\ E \end{bmatrix} = \begin{bmatrix} \frac{r(t)}{V} - k \cdot c \\ \frac{E_{\max} \cdot c}{c + E_{50}} \end{bmatrix}; \quad t \in [0 ; 25] \quad (3.21)$$



**Fig. 3.7** Concentration-time course in the central and peripheral compartments. The infusion rate is scheduled and depends on state events: at the beginning the infusion rate is 50 mg/h. If the concentration in the central compartment increases to 2 mg/L, the infusion rate is decreased to 5 mg/h. If the concentration decreases to 0.5 mg/L, the infusion rate is set to 50 mg/h. The scheduled events are linked only to the plasma concentration in the central compartment, and the concentration in the peripheral compartment has no direct impact on the scheduled process

The initial values and PK-PD parameters are:

$$c(0) = 0, \quad E_{\max} = 20, \quad E_{50} = 10, \quad r(0) = 1, \quad k = 0.1, \quad V = 6 \quad (3.22)$$

The following events and rules are considered:

- State Event (#1): Drug effect  $E$  has increased to 0.5  
Rule: Stop infusion
- State Event (#2): Drug effect  $E$  has decreased to 0.3  
Rule: Set infusion to 1 mg/h
- Time Event (#3): Simulation lasts for 25 hours  
Rule: Stop simulation.

The model in (3.21) and (3.22) is a combination of two equation types, differential and algebraic, and was already described in Sect. 3.2.4 to demonstrate the DAE solver, but without including event. Assuming the above events and rules, we can handle the differential equation with an ODE solver, and the algebraic equation can be directly used for events specification within the events function. The implementation is demonstrated in Listing 3.8.

Listing 3.8 Program **emaxevent.m**

```

function emaxevent()
%EMAXEVENT
%   EMAXEVENT() schedules PK-PD events. The drug effect should be
%   kept within a given range by infusion rate as control variable,
%   according to the rule: if the drug effect achieves the allowed
%   maximum, then decrease the infusion; if the drug effect
%   declines to the allowed minimum - infusion has to be increased.

% PKPD parameters
p.Rate = 1.0;      % infusion rate
p.V     = 6;       % volume of distribution;
p.k     = 0.1;     % elimination rate
p.Emax  = 20;      % max effect (in Hill equation)
p.E50   = 10;      % concentration linked to 50% of max effect

% Calling ODE
options = odeset('Events', @fevents);
tspan = [0 150];
c0 = 0;           % Initial value for the dependent variable
ie = 99;
while ~isempty(ie)
    [t,c,te,ye,ie] = ode45(@derivatives, tspan, c0, options,p);
    % Generate graph
    plot(t, p.Rate*ones(length(t),1),'Color', 'r', 'LineWidth',3)
    hold on;
    plot(t, c(:,1), 'Color', 'k', 'LineWidth', 3);
    hold on;
    plot(t, hillEffect(c(:,1), 0, p.Emax, p.E50, 1), ...
        'Color', 'b', 'LineWidth', 3)
    tspan = [te(end) 500];
    c0 = ye(end,1);
    R1 = p.Rate;
    % investigate events
    switch ie(end)
        case 1
            % drug effect E has increased to 2
            % stop infusion
            p.Rate = 0;
        case 2
            % drug effect E has decreased to 1
            % restart infusion
            p.Rate = 1;      % [mg/h]
        case 3
            % end of observation
            ie = [];        % no more events
    end
    R2 = p.Rate;
    plot([t(end) t(end)], [R1 R2],'Color', 'r', 'LineWidth',2)
end
grid minor; grid on ;
xlabel('Time [h]');
ylabel({'Drug Effect', 'Concentration [mg/L]', ...
    'Infusion Rate [mg/h]'});
legend('rate','conc','effect')
title('')

```

```

% save graphs as TIFF file
print('-r900', '-dtiff', 'emaxevent')
end
function dcdt = derivatives(~, c, p)
%DERIVATIVES Compute the right-hand side of the ODE.
%   DCDT = ...

dcdt = p.Rate/p.V - p.k*c(1);

end

function [value,isterminal,direction] = fevents(t,y,p)
%FEVENTS Specify events for the ODE model.
%   [VALUE,ISTERMINAL,DIRECTION] = ...

% Event 1: Drug effect E has increased to the limit (upper)
value(1,1) = hillEffect(y(1), 0, p.Emax, p.E50, 1) - 0.5;
isterminal(1,1) = 1;           % Stop integration
direction(1,1) = 1;           % Positive direction only

% Event 2: Drug effect E has increased to the limit (lower)
value(2,1) = hillEffect(y(1), 0, p.Emax, p.E50, 1) - 0.3;
isterminal(2,1) = 1;           % Stop integration
direction(2,1) = -1;          % Negative direction only

% Event 3: End of time integration
value(3,1) = t - 25 ;
isterminal(3,1) = 1;           % Stop integration
direction(3,1) = 1;           % Positive direction only

end

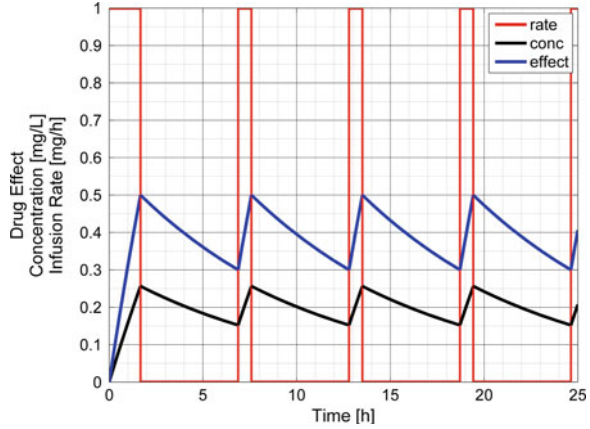
function E = hillEffect(c, E0, Emax, EC50, n)
% .....
% .....
% .....

end

```

In this implementation, the algebraic equation for the drug effect  $E$  is calculated within function **fevents** (see **% Event 1:...** and **% Event 2:...**) and not within **derivatives** as in [Sect. 3.2.4](#). The result of the simulation is shown in [Fig. 3.8](#).

**Fig. 3.8** Drug concentrations and effects over time for a one-compartment PK model linked to an  $E_{\max}$  PD model. Drug infusion rate is scheduled by state events as follows: Initial infusion rate of 1 mg/h is stopped if drug effect exceeds (positive direction) 0.5, and set to 1 mg/h if drug effect falls (negative direction) below 0.3



### 3.2.7 Partial Differential Equation

MATLAB provides a solver, **pdepe**, for a specific parabolic-elliptic partial differential equation (PDE). It solves an initial-boundary problem specified as follows:

$$\left. \begin{aligned} \mathbf{c}\left(x, t, \mathbf{u}, \frac{\partial \mathbf{u}}{\partial x}\right) \circ \frac{\partial \mathbf{u}}{\partial t} &= x^{-\mu} \cdot \frac{\partial}{\partial x} \left[ x^{\mu} \cdot \mathbf{f}\left(x, t, \mathbf{u}, \frac{\partial \mathbf{u}}{\partial x}\right) \right] + \mathbf{s}\left(x, t, \mathbf{u}, \frac{\partial \mathbf{u}}{\partial x}\right); \\ t &\in [t_0; t_f], \quad x \in [a; b] \end{aligned} \right\} \quad (3.23)$$

with initial condition

$$\left. \begin{aligned} \mathbf{u}(x, t_0) &= \mathbf{u}_0(x); \\ x &\in [a; b] \end{aligned} \right\} \quad (3.24)$$

and boundary condition

$$\left. \begin{aligned} \mathbf{p}(x, t, \mathbf{u}) + \mathbf{q}(x, t, \mathbf{u}) \circ \mathbf{f}\left(x, t, \mathbf{u}, \frac{\partial \mathbf{u}}{\partial x}\right) &= \mathbf{0}; \\ t &\in [t_0; t_f], \quad x = a \quad \text{or} \quad x = b \end{aligned} \right\} \quad (3.25)$$

where  $\mathbf{u}$  is the vector of independent variables,  $t$ —time,  $x$ —distance;  $\mu$  is equal to 0 (for slab symmetry), 1 (for cylindrical symmetry), or 2 (for spherical symmetry); the symbol  $\circ$  means the element-wise multiplication of two vectors. The vector functions  $\mathbf{c}(\cdot)$ ,  $\mathbf{f}(\cdot)$ , and  $\mathbf{s}(\cdot)$ , as well as  $\mathbf{p}(\cdot)$  and  $\mathbf{q}(\cdot)$ , have to be specified before calling the PDE solver **pdepe**.

The PDE solver is called according to the following syntax.

**Table 3.10** List of parameters passed to or returned by the MATLAB PDE solver **pdepe**

Formal parameters	Meaning
<code>&lt;sol&gt;</code>	The returned three-dimensional solution array, where dimensions 1 and 2 reference points $t$ and $x$ , respectively, at which the solution $\mathbf{u}$ was computed; dimension 3 references the components of solution vector $\mathbf{u}$
<code>&lt;msym&gt;</code>	A type of symmetry that corresponds to $\mu$
<code>&lt;pdefun&gt;</code>	A function name that defines the components of the PDE, and corresponds to Eq. (3.23). The function is called via a handle, i.e., <code>@&lt;pdefun&gt;</code> , and has a header as described below
<code>&lt;icfun&gt;</code>	A function name that defines the initial conditions, and corresponds to Eq. (3.24). The function is called via a handle, i.e., <code>@&lt;icfun&gt;</code> , and has a header as described below
<code>&lt;bcfun&gt;</code>	A function name that defines the boundary conditions, and corresponds to Eq. (3.25). The function is called via a handle, i.e., <code>@&lt;bcfun&gt;</code> , and has a header as described below
<code>&lt;xmesh&gt;</code>	A vector of $x$ points within the interval $[a; b]$ at which the solution $\mathbf{u}$ has to be computed for every value in <code>tspan</code>
<code>&lt;tspan&gt;</code>	A vector of time point within the time interval $[t_0; t_f]$ at which the solution has to be computed for every value in <code>xmesh</code>
<code>&lt;options&gt;</code>	Structure with optional parameters for integration properties and handling events
...	Additional (model) parameters

```
<sol> = pdepe (<msym>, @<pdefun>, @<icfun>, @<bcfun>,
    <xmesh>, <tspan>, <options>, ...);
```

The meaning of the above parameters is explained in Table 3.10.

Table 3.10 points via three handles to functions which are called internally by **pdepe**. These functions must be specified by the user, according to the following headers:

```
[<c>, <f>, <s>] = <pdefun> (<x>, <t>, <u>, <dudx>, ...);
```

The function `<pdefun>` evaluates the quantities  $\mathbf{c}(\cdot)$ ,  $\mathbf{f}(\cdot)$ , and  $\mathbf{s}(\cdot)$  as described in (3.23), at a point defined by  $x$ ,  $t$ ,  $\mathbf{u}$  and  $\frac{\partial \mathbf{u}}{\partial x}$ .

```
<u> = <icfun> (<x>, ...);
```

The function `<icfun>` evaluates the initial values  $\mathbf{u}_0$  according to (3.24), at a point  $x$ .

```
[<pl>, <ql>, <pr>, <qr>] = <bcfun> (<xl>, <ul>, <xr>,
    <ur>, <t>, ...);
```

The function `<bcfun>` evaluates the boundary condition from (3.25).

Table 3.11 summarizes the above functions, specifying the parameters involved and their meaning.

**Table 3.11** The list of parameters passed to and returned by the functions `<pdefun>`, `<pdefun>` and `<pdefun>`

Formal parameters	Meaning
<code>&lt;x&gt;</code>	A scalar corresponding to a value of $x$
<code>&lt;t&gt;</code>	A scalar corresponding to a time point $t$
<code>&lt;u&gt;</code>	A vector of the solution values at the point specified by $x$ and $t$ , $\mathbf{u}(t, x)$ .
<code>&lt;dudx&gt;</code>	A vector of the partial derivative value of $\mathbf{u}(t, x)$ with respect to $x$ , computed at the point specified by $x$ and $t$
<code>&lt;c&gt;</code> , <code>&lt;f&gt;</code> , <code>&lt;s&gt;</code>	Vectors of output values that correspond to the terms $\mathbf{c}(\cdot)$ , $\mathbf{f}(\cdot)$ and $\mathbf{s}(\cdot)$
<code>&lt;x1&gt;</code> , <code>&lt;xr&gt;</code>	Scalars corresponding to $x = a$ , and $x = b$ , respectively
<code>&lt;ul&gt;</code> , <code>&lt;ur&gt;</code>	Vectors corresponding to the solutions $u(x, t)$ at $x = a$ or $x = b$ , respectively, and for time point $t$
<code>&lt;pl&gt;</code> , <code>&lt;ql&gt;</code>	Vectors of output values that correspond to the terms $\mathbf{p}(\cdot)$ and $\mathbf{q}(\cdot)$ computed at $x = a$ , and for time point $t$
<code>&lt;pr&gt;</code> , <code>&lt;qr&gt;</code>	Vectors of output values that correspond to the terms $\mathbf{p}(\cdot)$ and $\mathbf{q}(\cdot)$ computed at $x = b$
<code>&lt;options&gt;</code>	Structure with optional parameters for integration properties and handling events
...	Additional (model) parameters

The **pdepe** solver will be used to implement the action potential model in MATLAB, in [Sect. 5.2.2](#), *Hodgkin and Huxley Mathematical Model*.

### 3.3 Analytical Solutions

#### 3.3.1 Analytical Solutions by Elementary Methods

Some differential equations can be solved using elementary methods. The equation:

$$\left. \begin{aligned} \frac{dy(t)}{dt} &= -\mathbf{A} \cdot \mathbf{y}(t); \quad t \in [t_0; t_f] \\ \mathbf{y}(t_0) &= \mathbf{y}_0 \end{aligned} \right\} \quad (3.26)$$

where  $\mathbf{A}$  is an  $m \times m$  constant matrix, can be easily solved using the exponential matrix  $e^{-\mathbf{A} \cdot (t-t_0)}$  [4], i.e.,:

$$\left. \begin{aligned} \mathbf{y}(t) &= e^{-\mathbf{A} \cdot (t-t_0)} \cdot \mathbf{y}(t_0) \\ e^{-\mathbf{A} \cdot (t-t_0)} &= \sum_{i=0}^{\infty} \frac{(-\mathbf{A})^i}{i!} \cdot (t-t_0)^i \end{aligned} \right\} \quad (3.27)$$

yielding:

$$\mathbf{y}(t) = \mathbf{y}(t_0) \cdot e^{-(t-t_0)} \quad (3.28)$$



when  $\mathbf{A}$  is the identity matrix,  $\mathbf{A} = \mathbf{I}$ . If the matrix  $\mathbf{A}$  is not the identity matrix, then the solution (3.27) can be obtained using MATLAB function **expm**:

**expm(-A\*(t-t0)\*y0)**

For example, considering a model similar to (3.11) but without delay, we have:

$$A = \begin{bmatrix} k_a & 0 \\ -\frac{F \cdot k_a}{V} & \frac{CL}{V} \end{bmatrix}; \quad t_0 = 0; \quad \mathbf{y} = \begin{bmatrix} a \\ c \end{bmatrix}; \quad \mathbf{y}_0 = \begin{bmatrix} 100 \\ 0 \end{bmatrix} \quad (3.29)$$

and the solution can be easily computed with the following script:

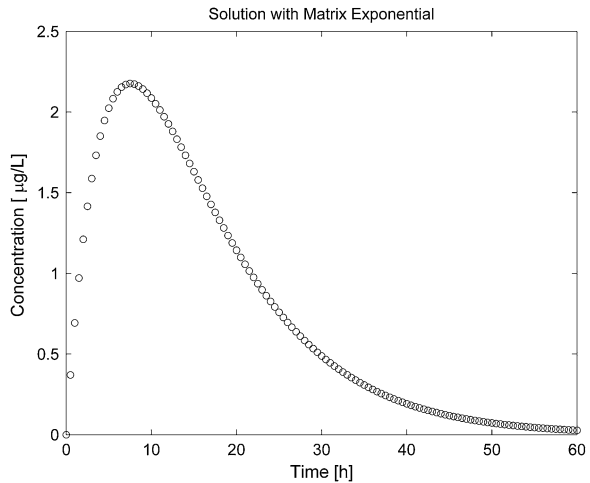
**Listing 3.9** Matrix exponential implementation

```
F = 1; CL = 2.16; V = 12.6; ka = 0.1; t = 0:0.5:60; c0 = 0; a0 = 100;
A = [ka 0; -ka*F/V CL/V];
set(axes,'FontSize', 14); c = t*0;
for ti = t; yi = expm(-A*ti)*[a0;c0]; c(t==ti) = yi(2); end
plot(t,c,'ok'); title('Solution with Matrix Exponential');
xlabel('Time [h]'); ylabel('Concentration [ \mu g/L ]')
```

In this way, we have obtained the solution demonstrated in Fig. 3.9.

In practice, however, such elementary methods are rarely applicable due to the complexity of the equations. A more advanced technique for solving certain types of ODEs is presented next.

**Fig. 3.9** The solution obtained with the matrix exponential method for a one-compartment model with first-order absorption and elimination rates, and without delay; the result is the same as in Fig. 3.3 ( $t_{\text{lag}} = 0$ )



### 3.3.2 Analytical Solutions Using Laplace Transforms

Transforming a system of differential equations to algebraic equations in Laplace form in a complex number domain, and then inverting the Laplace transform solution to a function in time domain, is a general procedure that can be applied to solve linear ODEs [5, 6] (Pierre-Simon de Laplace, 1749–1827). A linear ODE has the general form:

$$\left. \begin{aligned} \frac{dy}{dt} + \mathbf{G}(t) \cdot \mathbf{y} &= \mathbf{u}(t); \quad t \in [t_0; t_f] \\ \mathbf{y}(t_0) &= \mathbf{y}_0 \end{aligned} \right\} \quad (3.30)$$

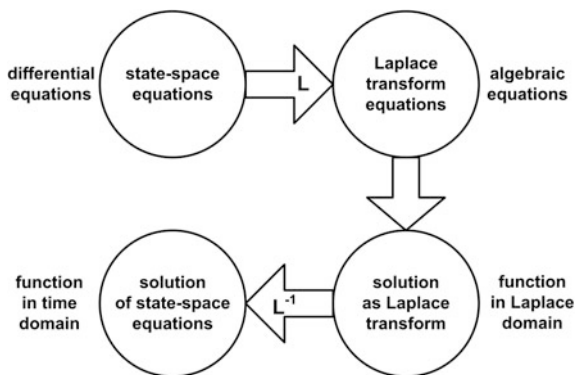
where  $\mathbf{G}$  is an  $m$ -by- $m$  array of functions and  $\mathbf{u}$  an  $m$ -by-1 vector of (forcing) functions all depending only on time,  $t$ . Differential equations in PK and PD models are predominantly linear.

The idea of using the Laplace transform for solving ODEs is visualized in Fig. 3.10.

Elementary analytical methods using the Laplace transform are well described [4, 7, 8], but can be rather tedious to use manually. The Symbolic Math Toolbox from MATLAB provides an easier route to the solutions.

We consider again the one-compartment model described by (3.6) and show how to solve the equation by using Laplace transforms both manually and with MATLAB.

According to the definition given above, (3.6) is a linear ODE with  $\mathbf{y} \equiv c$ ,  $\mathbf{G}(t) \equiv CL/V$ , and  $\mathbf{u}(t) \equiv r(t)/V$ . Taking the Laplace transform of all terms in (3.6) [4, 7], and considering (3.7)–(3.8):



**Fig. 3.10** Application of the Laplace transform to solve differential equations. The method converts a system of differential equations to a simpler system of algebraic equations, and solves it using the formal rules of algebra. The final solution of the original differential equations can then be found by applying the inverse Laplace transform. The entire step can be implemented using MATLAB routines from the Symbolic Math Toolbox

$$\left. \begin{aligned} \mathcal{L}\left\{\frac{d}{dt}c\right\} &= C(s) \cdot s - c(0) \\ \mathcal{L}\left\{\frac{r(t)}{V}\right\} &= \frac{1}{V} \cdot R(s) \\ \mathcal{L}\left\{-\frac{CL}{V} \cdot c\right\} &= -\frac{CL}{V} \cdot C(s) \\ &\text{with } c(0) = 0 \end{aligned} \right\} \quad (3.31)$$

the ODE is converted to an algebraic equation in  $C(s)$ :

$$C(s) \cdot s = \frac{1}{V} \cdot R(s) - \frac{CL}{V} \cdot C(s) \quad (3.32)$$

From (3.32) the transfer function  $K(s)$  for the one-compartment model can be obtained, relating output to input in the Laplace domain:

$$K(s) = \frac{C(s)}{R(s)} \quad (3.33)$$

The transfer function  $K(s)$  characterizes the behavior of the ODE independently of drug input represented by forcing function  $\mathbf{u}$ . For the one-compartment model it can be obtained as:

$$K(s) = \frac{C(s)}{R(s)} = \frac{1}{V} \cdot \frac{1}{s + \frac{CL}{V}} \Rightarrow K(s) = \frac{1}{CL + V \cdot s} \quad (3.34)$$

To find the solution  $c(t)$  for a given dosing regimen,  $r(t)$ , we have to find the Laplace transform of  $r(t)$ . For drug input as specified in (3.7), we would use the following relationship:

$$r(t) = \begin{cases} \text{Rate} ; & t \in [0 ; t_R) \\ 0 ; & t \geq t_R \end{cases} \Leftrightarrow r(t) = \text{Rate} \cdot [H(t) - H(t - t_R)] \quad (3.35)$$

where  $\text{Rate} = \text{Dose}/t_R$  is the infusion rate, and  $H(\cdot)$  the Heaviside function (Oliver Heaviside, 1850–1925). Laplace transformation of infusion  $r(t)$  gives:

$$R(s) = \mathcal{L}\{\text{Rate} \cdot [H(t) - H(t - t_R)]\} = \text{Rate} \cdot \left(\frac{1}{s} - \frac{1}{s} \cdot e^{-t_R \cdot s}\right) \quad (3.36)$$

Using (3.34)–(3.36), the solution in Laplace form  $C(s)$  is:

$$C(s) = R(s) \cdot K(s) \Leftrightarrow C(s) = \text{Rate} \cdot \left(\frac{1}{s} - \frac{1}{s} \cdot e^{-t_R \cdot s}\right) \cdot \frac{1}{s \cdot V + CL} \quad (3.37)$$

The inverse transform of  $C(s)$  [7] gives the solution in the time domain, i.e., concentration as function of time:

$$\mathcal{L}^{-1}\{C(s)\} = c(t) = \frac{Rate}{CL} \cdot [(1 - e^{-\frac{CL}{V}t}) \cdot H(t) - (1 - e^{-\frac{CL}{V}(t-t_R)}) \cdot H(t - t_R)] \quad (3.38)$$

Inserting  $V = 12.6$  L,  $CL = 2.16$  L/h,  $t_R = 2$  h, and  $Rate = 100$  mg/h, concentration  $c(t)$  can be expressed as follows:

$$c(t) = 46.296 \cdot [(1 - e^{-0.1714t}) \cdot H(t) - (1 - e^{-0.1714(t-2)}) \cdot H(t - 2)] \quad (3.39)$$

which is the solution wanted. The graphical representation of this solution is shown in Fig. 3.2e.

Equation (3.38) is not only the general solution for a one-compartment model with infusion but can also be used for an instantaneous (bolus) drug input. It can be shown that for  $t_R \rightarrow 0$ , (3.38) is transformed to a solution for one-compartment model with bolus (instead infusion) input, expressed as:

$$c(t) = \frac{Dose}{V} \cdot e^{-\frac{CL}{V}t} \cdot H(t) \quad (3.40)$$

We leave it to the reader, as an exercise, to conduct this task (see the section *Exercises* at the end of this chapter) as well as to show that (3.40) can be also obtained easily if the Laplace transform function is used. Furthermore, considering a general dosing history  $d(t)$  in which doses  $\{Dose_1, Dose_2, Dose_3 \dots Dose_m\}$  are given at times  $\{t_1, t_2, t_3 \dots t_m\}$ , the above equation can be extended as follows:

$$d(t) = \sum_{i=1}^m Dose_i \cdot \delta(t - t_i) \Rightarrow c(t) = \sum_{i=1}^m \frac{Dose_i}{V} \cdot e^{-\frac{CL}{V}(t-t_i)} \cdot H(t - t_i) \quad (3.41)$$

Proving the above equation is simple using (3.34) and knowing the Laplace transform for multiple doses,  $d(t)$ :

$$\mathcal{L}\{d(t)\} = \mathcal{L}\left\{\sum_{i=1}^m Dose_i \cdot \delta(t - t_i)\right\} = \sum_{i=1}^m Dose_i \cdot e^{-t_i \cdot s} \quad (3.42)$$

In a similar way, we can derive a general formula for the concentration versus time course when drug administration contains multiple infusions with rates  $\{Rate_1, Rate_2, Rate_3, \dots, Rate_m\}$  given over time intervals  $\{(t_1, t_{R1}), (t_2, t_{R2}), (t_3, t_{R3}), \dots, (t_m, t_{Rm})\}$ :

$$\left. \begin{aligned} r(t) &= \sum_{i=1}^m Rate_i \cdot [H(t - t_i) - H(t - t_{Ri})] \\ &\Downarrow \\ c(t) &= \sum_{i=1}^m \frac{Rate_i}{CL} \cdot [(1 - e^{-\frac{CL}{V}(t-t_{Ri})}) \cdot H(t - t_i) - (1 - e^{-\frac{CL}{V}(t-t_{Ri})}) \cdot H(t - t_{Ri})] \end{aligned} \right\} \quad (3.43)$$

Again, proving the above equation is easy using (3.34) and the Laplace transform for multiple infusion,  $r(t)$ :

$$\mathcal{L}\{r(t)\} = \mathcal{L}\left\{\sum_{i=1}^m \text{Rate}_i \cdot [H(t - t_i) - H(t - t_{Ri})]\right\} = \sum_{i=1}^m \frac{(e^{-t_i \cdot s} - e^{-t_{Ri} \cdot s})}{s} \quad (3.44)$$

As the considered model is linear, it is also possible to combine (3.41) and (3.43). More generally, the transform function (3.34) represents a one-compartment model and is independent of dosing. It means that concentration  $c(t)$  can also be derived for other dosing regimens,  $r(t)$ , with the help of the same transform function. Only the Laplace transform of  $r(t)$  has to be inserted, if it exists. For typical dose administrations this assumption is fulfilled.

It is not always easy to perform the Laplace transformation manually as for the one-compartment model. In general, it can be very tedious but MATLAB has built-in routines that support such calculations. A MATLAB implementation of the method using Laplace transforms is contained in Listing 3.10 as function **lap11com()**. It computes symbolically transform function  $K(s)$ , function  $C(s)$  and, finally, concentration  $c(t)$  via the inverse Laplace transform of the symbolic expression  $C(s)$ .

### Listing 3.10 Program **lap11com.m**

```
function c = lap11com()
%LAPL1COM Laplace solution for one-compartment model with infusion.
% C = LAPL1COM() solves the equation dcdt = r/V - k*c using
% first Laplace transform and then inverse Laplace transform.
% The result is a function |C| = f(t) in analytic form.

syms R CL V s t k dcdt Lc c r Lr

% differential equation in analytic form (symbols)
dcdt = sym('diff(c(t),t)');
c = sym('c(t)');
r = sym('r(t)');
equation = dcdt - r/V + CL/V*c;
transformL = laplace(equation, t, s);
% simplify the Laplace form of the equation
transformL = subs(transformL, {'c(0)', 'laplace(c(t), t, s)', ...
    'laplace(r(t), t, s)'}, {0, Lc, Lr}); %ok<NODEF>
transformL = collect(collect(transformL, V), Lc);
% solution as Laplace transform Lc(s)
Lc = solve(transformL, Lc);
Ks = Lc/Lr; % transform function
pretty(Ks)
% specify the input (dosing)
r = R*(heaviside(t) - heaviside(t-2)); % infusion
%r = R*dirac(t); % bolus
Us = laplace(r, t, s);
Ys = Ks*Us;
c = ilaplace(Ys, s, t);
pretty(c)
display(['c(t) = ' char(c)]);

end
```

Besides final solution  $c(t)$ , the procedure also prints the symbolic output of  $K(s)$  and  $C(s)$ —see Listing 3.11. This confirms the result derived manually in (3.34).

**Listing 3.11** Symbolic output of  $K(s)$  given by function `solve1ap11com`

$$\frac{1}{CL + V s}$$

Similarly, function `lap12com()` (shown in Listing 3.12) computes the transform function for a two-compartment model. In this case the function computes both concentrations  $c_1(t)$  and  $c_2(t)$ , and prints the symbolic output (Listing 3.13) of transform function vector  $\mathbf{K}(s)$ :

$$\mathbf{K}(s) = \begin{bmatrix} \frac{C_1(s)}{R(s)} & \frac{C_2(s)}{R(s)} \end{bmatrix}^T \quad (3.45)$$

with elements:

$$\begin{aligned} \frac{C_1(s)}{R(s)} &= \frac{k_{21} + s}{V_1 \cdot (k \cdot k_{21} + k \cdot s + k_{12} \cdot s + k_{21} \cdot s + s^2)} \\ \frac{C_2(s)}{R(s)} &= \frac{k_{12}}{V_2 \cdot (k \cdot k_{21} + k \cdot s + k_{12} \cdot s + k_{21} \cdot s + s^2)} \end{aligned} \quad (3.46)$$

Listing 3.12 Program `lapl2com.m`

```

function c = lapl2com()
%LAPL2COM Laplace solution for two-compartment model with infusion.
% C = LAPL2COM() solves the equation:
% dcdt = [p.R/p.V1-(p.k+p.k12)*c(1)+p.k21*p.V2/p.V1*c(2);
%         p.k12*p.V1/p.V2*c(1)-p.k21*c(2)]';
% using first Laplace transform and then inverse Laplace transform.
% The result is a vector function |C| = c(t) in analytic form.

syms R V1 V2 s t k k12 k21 LC1 c1 LC2 c2 Rt Lr r

dc1dt = sym('diff(c1(t),t)');
dc2dt = sym('diff(c2(t),t)');
c1 = sym('c1(t)');
c2 = sym('c2(t)');
r = sym('r(t)');
%Rt = R*(heaviside(t) - heaviside(t-2));
equation1 = dc1dt - r/V1 + (k+k12)*c1 - k21*V2/V1*c2;
equation2 = dc2dt - k12*V1/V2*c1 + k21*c2;
transformL1 = laplace(equation1, t, s);
transformL2 = laplace(equation2, t, s);
% specify Laplace transform of c(t) as Lc
transformL1 = subs(transformL1, {'c1(0)', 'laplace(c1(t), t, s)', ...
    'laplace(c2(t), t, s)', 'laplace(r(t), t, s)'}, ...
    {0, LC1, LC2, Lr}); %ok<NODEF>
transformL2 = subs(transformL2, {'c2(0)', 'laplace(c1(t), t, s)', ...
    'laplace(c2(t), t, s)'},{0, LC1, LC2});
transformL1 = collect(transformL1, LC1);
transformL2 = collect(transformL2, LC2);
[LC1, LC2] = solve(transformL1, transformL2, LC1, LC2);
Ks = [LC1/Lr; LC2/Lr];
% Show the transform function in 'like algebraic' notation
pretty(Ks)
% specify input (dose administration)
r = R*(heaviside(t) - heaviside(t-2)); % infusion
%r = R*dirac(t); % bolus
Us = laplace(r, t, s);
Ys = Ks*Us;
c = ilaplace(Ys, s, t);
c = simplify(c);
pretty(c)
display(['c1(t) = ' char(c(1))]);
display(['c2(t) = ' char(c(2))]);

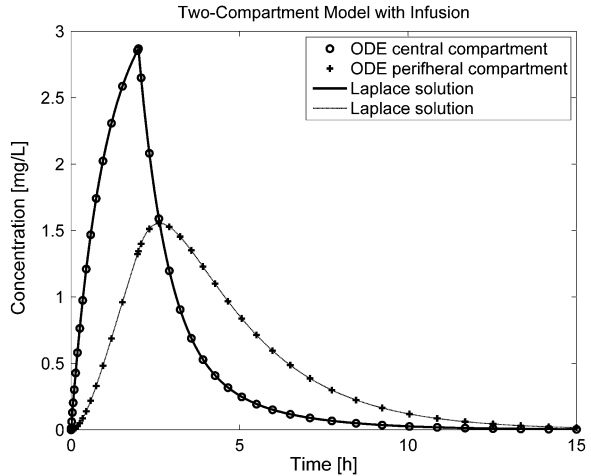
end

```

**Listing 3.13** Symbolic output of  $K(s)$  given by function `lap12com`

```
+--
|                                     +-+
|                                     |
|                                     | k21 + s
|-----+-----+
|                                     |
| V1 (k k21 + k s + k12 s + k21 s + s ) 2
|                                     |
|                                     | k12
|-----+-----+
|                                     |
| V2 (k k21 + k s + k12 s + k21 s + s ) 2
|                                     |
|                                     +-+
+--
```

**Fig. 3.11** The exact solutions were created using the Laplace transforms method. For the purpose of comparison, the graph also contains a numerical solution obtained with ODE solver `ode45` that handles in a proper manner the time event “infusion stop at 2 h after start of infusion”



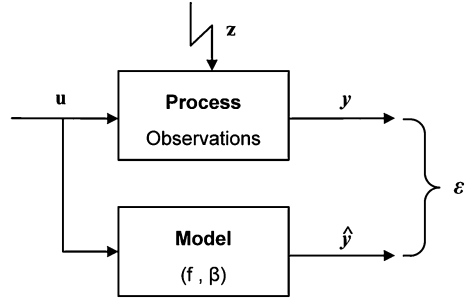
The above procedure can also be directly used to create an analytical solution. An example, `infus2comlap1.m`, is available on MATLAB Central. It produces the graph shown in Fig. 3.11.

### 3.4 Parameter Estimation in Differential Equations

Drug effect modeling is a main objective in pharmacometrics. PK and PD provide a large number of empirical models applicable to observed drug concentrations and effects. Those models have parameters such as volume of distribution,  $V$ , clearance,  $CL$ , maximum effect,  $E_{\max}$ , and concentration at half maximum effect,  $EC_{50}$ , that need to be estimated from prevailing data. Parameter estimation occurs in many contexts, especially in linear or nonlinear regression, and its basic features will be shortly described.



**Fig. 3.12** Parameter estimation: Observations  $\mathbf{y}$ , arising from a process, are described by model predictions  $\hat{\mathbf{y}}$  from a model  $\mathbf{f}$  with parameters  $\boldsymbol{\beta}$ .  $\boldsymbol{\beta}$  is then determined to minimize a given objective function.  $\mathbf{u}$  is the input to the system, and  $\mathbf{z}$  represents random influence on the process



As depicted in Fig. 3.12, we assume a process delivering observations,  $\mathbf{y}$ , over time following input  $\mathbf{u}$ . Observations may be distorted by noise  $\mathbf{z}$  (e.g., measurement error). To model the process, we may rely on a parametric time-dependent function  $\mathbf{f}$  which provides values  $\hat{\mathbf{y}}$  for input  $\mathbf{u}$  and a parameter vector  $\boldsymbol{\beta}$ . The nature of the function  $\mathbf{f}$  and its parametrization are mainly determined by our understanding of the process. In our examples,  $\mathbf{f}$  is often the solution of an ODE:

$$\left. \begin{aligned} \frac{d\mathbf{f}}{dt} &= \mathbf{g}(t, \mathbf{f}, \mathbf{u}, \boldsymbol{\beta}); & t \in [t_0; t_f] \\ \mathbf{f}(t_0) &= \mathbf{f}_0 \end{aligned} \right\} \quad (3.47)$$

Vector  $\boldsymbol{\beta}$  is then determined to minimize distance  $d(\mathbf{y}, \hat{\mathbf{y}})$  between observations,  $\mathbf{y}$ , and predictions,  $\hat{\mathbf{y}}$ . Often, it is practical to derive distance  $d$  from the Euclidean norm  $\|\cdot\|$ , so that the following minimization problem has to be considered:

$$\boldsymbol{\beta} \in \mathbf{R}^m : \min_{\boldsymbol{\beta}} \|\boldsymbol{\varepsilon}\| = \min_{\boldsymbol{\beta}} \|\mathbf{y} - \hat{\mathbf{y}}\|; \quad \|\boldsymbol{\varepsilon}\| = \boldsymbol{\varepsilon}^T \cdot \boldsymbol{\varepsilon} \quad (3.48)$$

Better results are in general achieved if a weight vector  $\mathbf{W}$  is incorporated into the objective function balancing the contribution of the elements of  $\boldsymbol{\varepsilon}$ :

$$\boldsymbol{\beta} \in \mathbf{R}^m : \min_{\boldsymbol{\beta}} \|\boldsymbol{\varepsilon}_w\|; \quad \|\boldsymbol{\varepsilon}_w\| = \boldsymbol{\varepsilon}^T \cdot \mathbf{W} \cdot \boldsymbol{\varepsilon}; \quad \mathbf{W} = \text{diag}(\mathbf{w}); \quad \boldsymbol{\varepsilon} = \mathbf{y} - \hat{\mathbf{y}} \quad (3.49)$$

Vector  $\mathbf{w}$  (elements on the main diagonal of  $\mathbf{W}$ ) contains the weights. Weights can be assigned by the modeler or calculated from the corresponding residuals of vector  $\boldsymbol{\varepsilon}$ . For example, MATLAB uses bisquare weights as the default [3]. A regression procedure solves the tasks described in (3.48) and (3.49), i.e., determines the model parameter estimates so as to minimize the objective function. In general, the model  $\mathbf{f}$  in Fig. 3.12 is nonlinear with regard to model parameters, and thus the regression procedure described in the following section will focus on nonlinear regression in ODEs.

### 3.4.1 Nonlinear Regression

We will now show how parameter estimation in ODEs can be accomplished with MATLAB. The ODE system in (3.50) describes two coupled equations as encountered in PK modeling:

**Table 3.12** Observations

Variable								
Time, $t$	0	1	2	3	4	5	6	12
Amount, $a_2$	0	1	1.5	1.6	1.5	1.4	1.3	1.4

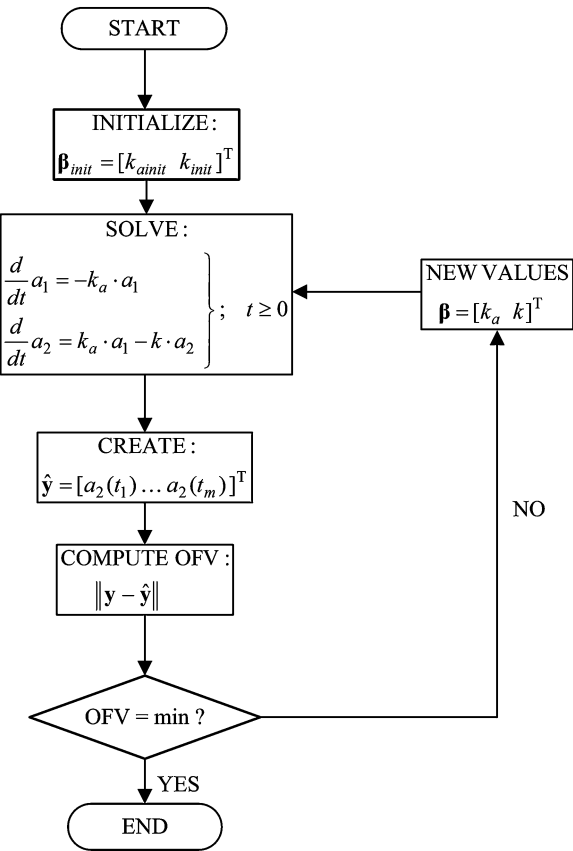
$$\left. \begin{aligned} \frac{da_1}{dt} &= -k_a \cdot a_1, & a_1(0) &= Dose \\ \frac{da_2}{dt} &= k_a \cdot a_1 - k \cdot a_2, & a_2(0) &= 0 \end{aligned} \right\}; \quad t \geq 0 \tag{3.50}$$

Observations are available as shown in Table 3.12.

There are two model parameters to be estimated: absorption rate constant  $k_a$ , and elimination rate constant  $k$ .

The flowchart in Fig. 3.13 shows how to obtain the estimates for  $k_a$  and  $k$ . It shows how to estimate parameters when the objective function is not given in closed form. In order to get values predicted by the model, the ODE system in (3.50) has to be solved. This step needs to be repeated at each iteration while searching for the minimum of the objective function (OFV). A MATLAB implementation of the steps shown in the flowchart (Fig. 3.13) is presented in Listing 3.14.

**Fig. 3.13** Flowchart illustrating the procedure for estimating model parameters in a dynamic model described by a system of two differential equations



**Listing 3.14** Program **nlregr.m**

```

function nlregr()
%NLREGR estimate parameters in PK model
%   %NLREGR() uses the nonlinear least squares method to find model
%   parameter estimates
%
% This function uses plot_ci from MATLAB Central:
% http://www.mathworks.com/matlabcentral/fileexchange/31752-plotci
% Copyright (c) 2011, Zbigniew
% All rights reserved.
% Code covered by the BSD License: http://www.mathworks.com/
% matlabcentral/fileexchange/view_license?file_info_id=31752
%

observations = [ 0  1  2   3   4   5   6  12           % time points
                0  1  1.5 1.6 1.5 1.4 1.3 1.4 ]; % drug amount
tObs = observations(:,1);
aObs = observations(:,2);

% beta estimation
options = statset('Robust', 'on', 'WgtFun', 'bisquare');
[beta, resid, J] = nlinfit(tObs, aObs, @model, [1 2], options);
% objective function
w = (abs(resid)<1).*(1 - resid.^2).^2; % weight function 'bisquare'
ofv = resid'*diag(w)*resid;
% confidence interval for model parameters
paramCI = nlparci(beta,resid,'jacobian',J);
display([beta' paramCI]);
% confidence interval for predictions
tPred = 0:0.02:12;
[aPred,delta] = nlpredci(@model, tPred, beta, resid, 'jacobian', ...
    J, 'simopt', 'on');
[~, iInd] = ismember(tObs,tPred);
% show predictions
display(['time', '      Obs', '      Pred']);
display(num2str([tObs aObs aPred(iInd)]))
predCI = [aPred aPred+delta aPred-delta];
set(axes,'FontSize',15)
plot(tObs, aObs, 'or', 'MarkerSize',5,'MarkerFaceColor','r');
grid on; hold on
rgbfill = [0.3047 0.6992 0.9102];
rgbline = [0.9414 0.9414 0.3359];
% call plot_ci; the function is available on MATLAB Central:
plot_ci(tPred, predCI, 'PatchColor', rgbfill, 'PatchAlpha', 0.5, ...
    'MainLineWidth', 2, 'MainLineColor', rgbline, 'LineWidth', 1, ...
    'LineStyle','-','LineColor', 'k');
legend('observation', 'prediction CI','regression line', ...
    'Location','Best' );
xlabel('Time')
ylabel('amount')
title(['Fitting: ', ' ka=', num2str(beta(1)), ' k=', ...
    num2str(beta(2)), ' OFV(least squares)=', num2str(ofv)])
print('-r1000', '-dtiff', 'nlregression');
end

function dadt = derivatives(~, a, p)

```

```

%DERIVATIVES Compute the right-hand side of the ODE.
%   DADT = ...

% PK one-compartment model (SC administration)
dadt = [-p.ka*a(1);          % drug amount - depot
        p.ka*a(1) - p.k*a(2)]; % drug amount - compartment
end

function a = model(beta, t)
%MODEL Prediction of the drug amount.
%   A = MODEL(BETA, T) computes prediction for the drug amount |A|
%   in one-compartment model at time points given as vector |T|,
%   and with model parameters |BETA|.

dose = 100;
amount0 = 0;
tspan = [0 12];
a0 = [dose amount0];
p.ka = beta(1);      % absorption rate
p.k = beta(2);       % elimination rate
sol = ode45(@derivatives, tspan, a0,[], p);
a12 = deval(sol, t)';
a = a12(:,2);

end

```

**Table 3.13** Model parameter estimates obtained from the **nlregr.m** program

Parameter	Estimates	CI (95 %)
$k_a$	0.0183	[0.0119; 0.0248]
$k$	1.1443	[0.7344; 1.5542]

**Table 3.14** Model prediction results obtained from **nlregr.m** program

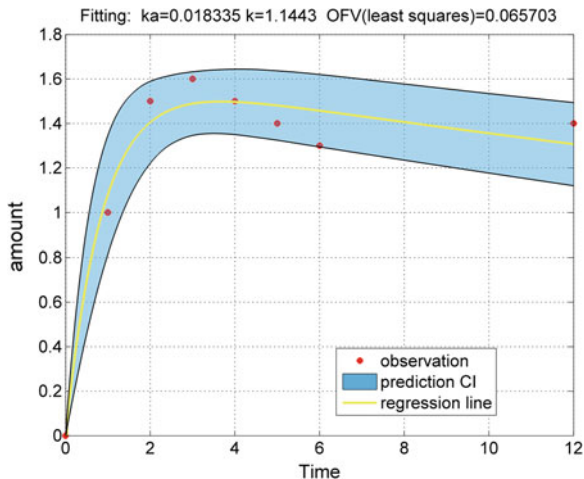
Time	Observation	Prediction
0	0	0
1.0000	1.0000	1.0802
2.0000	1.5000	1.4047
3.0000	1.6000	1.4887
4.0000	1.5000	1.4965
5.0000	1.4000	1.4803
6.0000	1.3000	1.4570
12.0000	1.4000	1.3067

The estimation results are demonstrated in Tables 3.13, 3.14, and Fig. 3.14.

### 3.5 Comments

This chapter presented several MATLAB functions and showed how to apply them to dynamic models. The selection criterion for these functions was their usability for pharmacologic PK-PD modeling and simulation. MATLAB has a number of

**Fig. 3.14** Fitting results generated by program **nlregr.m**. The prediction confidence interval for is generated at the default 95 % level (graph created with function **plot\_ci** from [9])



other tools and functions that can be useful for modeling dynamic systems in many other areas of science, e.g., control theory, electrical circuits, mechanical systems, and automation, to name a few.

## 3.6 Exercises

### Exercise 3.1

Assuming a DDE equation

$$\frac{dx}{dt} = -k \cdot x(t - \tau); \quad t \geq 0 \quad (3.51)$$

and the following history

$$x(t) = x_0; \quad t \in [-\tau; 0] \quad (3.52)$$

find the analytical solution for the DDE defined by (3.51) with history as in (3.52). Furthermore, write a program to solve this equation numerically by using a MATLAB DDE solver. Finally, incorporate the analytical solution into this program and graphically demonstrate that both solutions are identical.

### Exercise 3.2

Assuming DDE equation:

$$\frac{dx}{dt} = -k \cdot x(t - \tau); \quad t \geq 0 \quad (3.53)$$

with the following history:

$$x(t) = a \cdot t; \quad t \in [-\tau; 0] \quad (3.54)$$

find the analytical solution for the DDE defined by (3.53) with history as in (3.54). Furthermore, write a program to solve this equation numerically by using a MATLAB DDE solver. Finally, incorporate the analytical solution into this program and graphically demonstrate that both solutions are identical.

### Exercise 3.3

Modify the `emaxevent.m` program considering the same state events:

- State Event: Drug effect  $E$  has increased to 2  
Rule: Stop infusion
- State Event: Drug effect  $E$  has decreased to 1  
Rule: Set infusion to 1 mg/h
- Time Event: Simulation lasts for 25 hours  
Rule: Stop simulation.

However, use the DAE solver to find the infusion regimen. Compare the output with Fig. 3.8 obtained without the DAE solver.

### Exercise 3.4

Create a function that generates the concentration-time profile following three initial infusions and multiple bolus administrations thereafter. Use a one-compartment model with first-order elimination.

### Exercise 3.5

Find drug concentrations  $c(t)$  after a bolus dose of 200 mg using the transform function  $K(s)$  of the one-compartment model.

### Exercise 3.6

Solve the same problem as in Exercise 3.5, but using MATLAB routines for Laplace transformations.

### Exercise 3.7

Show that the equation for the one-compartment model with infusion and first-order elimination, i.e.,

$$c(t) = \frac{Rate}{CL} \cdot [(1 - e^{-\frac{CL}{V}t}) \cdot H(t) - (1 - e^{-\frac{CL}{V}(t-t_R)}) \cdot H(t - t_R)] \quad (3.55)$$

approaches the analytical solution for a one-compartment model with bolus input given by:

$$c(t) = \frac{D}{V} \cdot e^{-\frac{CL}{V}t} \cdot H(t) \quad (3.56)$$

if the infusion duration  $t_R$  approaches 0.

**Exercise 3.8**

Using a one-compartment model with bolus input, show that the solution of the differential equation describing such a model is the same both for dose handled as the initial value or as an input value.

**References**

1. Meiss JD (2007) Differential dynamical systems. SIAM, Philadelphia
2. The MathWorks (2012) MATLAB Mathematics (R2012b), The MathWorks, Natick
3. Xue D, Chen YQ (2009) Solving applied mathematical problems with MATLAB. Chapman & Hall, Boca Raton
4. Ogata K (1992) System dynamics. Pearson, Englewood Cliffs
5. Föllinger O (2003) Laplace-, Fourier- und z-transformation. Hüthig, Heidelberg
6. Hilderbrand FB (1963) Advanced calculus for application. Prentice-Hall, Englewood Cliffs
7. Sokolnikov IS, Redheffer RM (1996) Mathematics of physics and modern engineering. McGraw-Hill, New York
8. Ogata K (2008) MATLAB for control engineers. Pearson, Englewood Cliffs
9. <http://www.mathworks.com/matlabcentral/fileexchange/31752-plotci>

## Chapter 4

# Pharmacologic Modeling

Pharmacologic modeling deals with pharmacokinetics (PK) and pharmacodynamics (PD). PK describes “what the body does to a drug,” i.e., how it is absorbed, distributed and eliminated, and PD “what a drug does to the body,” i.e., how it interacts with receptors and their signaling pathways up to the whole body level. The time course of drug concentrations and drug effects is modeling outcomes.

PK uses compartmental models either in an empirical way to get a satisfactory fit to experimental data, or in a physiologic manner to create predictive models across species (physiologically-based PK, or PBPK). PD models turn drug concentrations at the pharmacologic site of action, or biophase, into biological effects using direct and/or indirect response mechanisms. Taken together, PK-PD models provide a powerful tool to link dosage regimens to clinical effects and vice versa.

### 4.1 Pharmacokinetics

#### 4.1.1 General Concepts

PK deals with processes that move a drug through the body (Greek: *pharmakon*—drug, *kinesis*—movement), and with the quantitative description of drug concentrations over time at different body locations [1]. Why are we interested in PK? In order to exert its (beneficial or adverse) effects, a drug must achieve effective concentrations at the site of action, which could be in the extracellular space (e.g., a receptor bound to the cell membrane) or in the intracellular space (e.g., an enzyme located in the cytosol). PK investigates the relationship between drug input and drug concentrations at the site of action or target.

As drug concentrations at the target are rarely accessible, reference is made to drug concentrations in blood as assessed in samples from an arm vein. There are three types of drug concentrations that are derived from blood samples: (whole) blood concentrations; plasma concentrations determined after cells (red blood cells, white blood cells, platelets) have been removed from blood; serum



concentrations determined after removal of clotting factors from plasma. Blood cells account for about 45 % of blood volume and drug concentrations in cells can be very different from those outside cells.

Drug movement through the body takes place in four processes: *Absorption*, *distribution*, *metabolism*, and *elimination*.

### Absorption

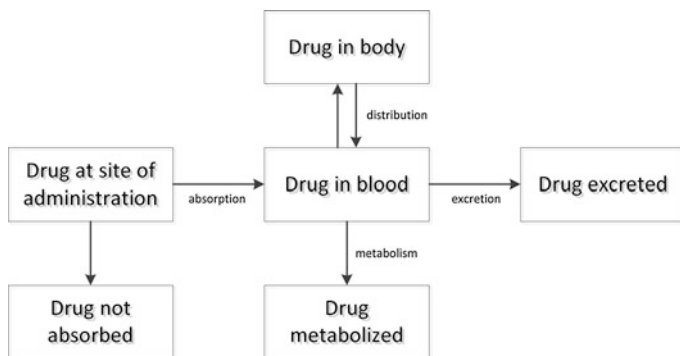
Absorption is the process that moves drug from the site of administration to the location where it is measured, usually an arm vein. There are many potential sites of administration, such as the gastrointestinal tract (following oral, buccal, sublingual, or rectal administration), muscle, subcutaneous tissue, and lungs (following inhalation). In addition, drugs can be administered directly into a vein or artery, thereby circumventing the absorption process following extravascular administration. In the case of extravascular administration, not all administered drug amount may be absorbed.

### Distribution

Distribution refers to processes that move the drug between different locations within the body in a reversible manner. Drug that is absorbed does not stay within the venous blood system but enters other tissues and organs. From there it is moved back, sometimes via routes that differ from the original one, as exemplified by enterohepatic cycling (see [Sect. 4.1.3](#)).

### Elimination

Elimination is the process that moves drug irreversibly out of the site of measurement. Elimination can occur in two forms, as excretion or metabolism. Excretion is the loss of unchanged drug from the body. Metabolism is the conversion of unchanged drug into a metabolite without back-conversion. Any back-conversion of the metabolite into the parent drug would be considered as distribution.



**Fig. 4.1** Conceptual model of the basic processes of pharmacokinetics: Absorption, distribution, metabolism, and elimination (ADME). *Arrows indicate the transport of drug*

## Disposition

Disposition designates processes related to both elimination and distribution.

A respective conceptual model for the processes described above is shown in Fig. 4.1.

To turn the conceptual model into a quantitative model, some general PK terms and principles are introduced [2].

## Volume

Drug administered to the body distributes into a volume that is not directly measurable. However, if we relate drug amounts in the body to venous blood (serum, plasma) drug concentrations, we can infer an apparent space into which drug has been distributed. Thus, we define the volume of distribution,  $V$ , as the ratio between  $a$ , the amount of drug in the body (in units of g or mol), and  $c$ , the concentration of drug in blood (serum, plasma) (in units of g/L or mol/L) at any given time,  $t$ :

$$V(t) := \frac{a(t)}{c(t)} \quad (4.1)$$

In (4.1) no assumptions are made under which conditions drug amount  $a$ , and drug concentrations  $c$  are considered. The distribution volume obtained at steady-state drug concentrations will be different from the one obtained during terminal drug elimination. If we make the assumption that ratios of drug concentrations in the body between any two tissues or fluids are constant over time, then  $V$  is independent of time (see Exercises).

## Compartment

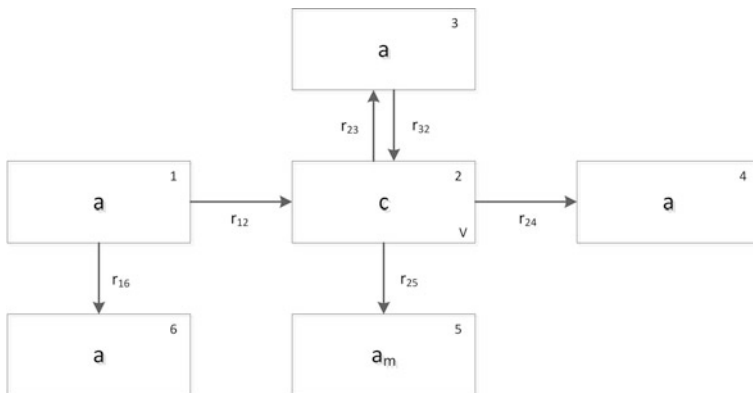
A compartment is a PK modeling construct representing drug amount,  $a$ . Compartments exchange amounts between each other at a certain rate, which means that the total drug amount is conserved (principle of mass balance). Compartments with an assigned volume  $V$  also represent drug concentrations  $c$  according to  $c = a/V$ . In this case, drug amounts are assumed to be homogeneously distributed at all times, and drug amounts entering a compartment are instantaneously mixed.

A compartmental representation of the PK processes described above is shown in Fig. 4.2.

Compartmental representations can be easily translated into mathematical equations. For example, a rate  $r_{12}$  from compartment 1 to compartment 2 causes changes of drug amounts in these compartments according to:

$$\left. \begin{aligned} \frac{da_1}{dt} &= -r_{12}(t) \\ \frac{da_2}{dt} &= r_{12}(t) \end{aligned} \right\} \quad (4.2)$$

De novo drug input into a compartment occurs in two forms. If the drug is administered as an infusion with rate  $r$  over a time interval  $t_{\text{dur}}$ ,  $r$  can be directly applied to the respective compartment, e.g.,



**Fig. 4.2** A multi-compartment model representing PK processes. Compartments are numbered from 1 to 6.  $r_{ij}$  is the rate of drug movement from compartment  $i$  to compartment  $j$ . Drug from compartment 1 (site of absorption) is moved into compartment 2 (blood) where it equilibrates with compartment 3 (distribution), and is eliminated into compartment 4 (excretion) and compartment 5 (metabolism). Compartment 2 has assigned a volume which allows the derivation of drug concentrations;  $a$ —drug amount,  $a_m$ —metabolite amount,  $c$ —drug concentrations

$$\begin{aligned} r_{02} &= r; & 0 \leq t \leq t_{\text{dur}} \\ r_{02} &= 0; & t > t_{\text{dur}} \end{aligned} \quad (4.3)$$

The “0” in  $r_{02}$  indicates that drug is taken from a source that is not further specified in the model. A similar convention applies to elimination compartments. If drug is introduced instantaneously into a compartment at time  $t$  (e.g., as an injection into blood, or an oral dose into the gut), the drug amount in that compartment is updated before drug continues to exchange between compartments. The Dirac delta function ( $\delta$ ) allows an elegant description of this kind of drug input:

$$r_{01} = \text{Dose} \cdot \delta_t \quad (4.4)$$

which specifies that a *Dose* is applied as a bolus at time  $t$  into compartment 1.

It is important to note that compartment volumes and rates are not introduced as physiological quantities, and one should be cautious in their physiological interpretation.

If we let  $r_{\text{abs}}(t)$  be the rate at which a given dose is absorbed, and  $r_{\text{elim}}(t)$  the rate of elimination, then we get the relationship:

$$F \cdot \text{Dose} = \int_0^{\infty} r_{\text{abs}}(t) \, dt = \int_0^{\infty} r_{\text{elim}}(t) \, dt \quad (4.5)$$

$F$  is the fraction of the *Dose* that is absorbed, and is referred to as bioavailability.

### Clearance

Another important concept is to relate drug transport rates to drug concentrations. If  $r_{12}$  is the transport rate of drug amounts from compartment 1 to compartment 2, and  $c_1$  the drug concentrations in compartment 1, then the ratio:

$$CL_{12}(t) := \frac{r_{12}(t)}{c_1(t)} \quad (4.6)$$

is called clearance of drug from compartment 1 to compartment 2. Both volume and clearance cannot be measured directly and must be inferred from concentration data under additional assumptions (see below).

If we apply the above definition of clearance to (4.5), we get:

$$F \cdot Dose = \int_0^{\infty} r_{elim}(t) dt = \int_0^{\infty} CL_{elim} \cdot c(t) dt \quad (4.7)$$

If  $CL_{elim}$  is constant over time, it can be factored out of the integral yielding:

$$\int_0^{\infty} CL_{elim} \cdot c(t) dt = CL_{elim} \int_0^{\infty} c(t) dt = CL_{elim} \cdot AUC \quad (4.8)$$

or

$$F \cdot Dose = CL_{elim} \cdot AUC \quad (4.9)$$

where  $AUC$  is the area under the concentration time curve,

$$AUC = \int_0^{\infty} c(t) dt \quad (4.10)$$

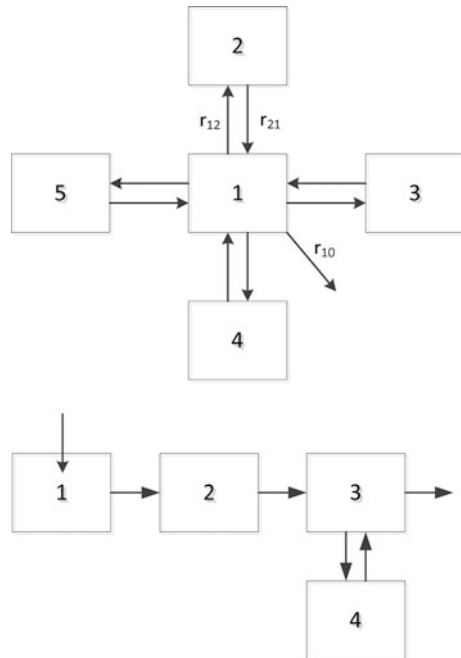
Thus, the  $AUC$  for the central compartment depends only on  $Dose$ , bioavailability  $F$ , and the elimination clearance  $CL$ , but not on intercompartmental processes.

### 4.1.2 Empirical PK Modeling

Two approaches to PK modeling will be introduced, the empirical approach and the PBPK approach. Both use PK compartments but with different intentions. Empirical PK modeling aims to develop a model for (concentration) data, whereas PBPK modeling emphasizes a systems approach by respecting human or animal anatomy and physiology.

In empirical PK modeling an increasing number of compartments is assembled until a satisfactory fit is reached. Figure 4.3 shows two multi-compartment PK models that often occur in empirical PK analyses. One is the mammillary model

**Fig. 4.3** Mammillary (*upper panel*) and catenary (*lower panel*) multi-compartment models. The mammillary model is a multi-compartment disposition model with no extravascular input specified.  $r_{12}$  denotes a drug rate from compartment 1 to compartment 2, etc. Rate  $r_{10}$  denotes a sink out of compartment 1. Drug amounts transported via  $r_{10}$  need to be integrated if mass balance is to be checked. In the catenary models drug is administered to compartment 1 and sequentially transported to compartment 3 from where it is distributed (to compartment 4) and eliminated



describing drug disposition and the other one a catenary model describing drug transport or absorption.

In the mammillary model, drug is assumed to be placed into the central compartment from where it reversibly distributes to peripheral compartments, which are not connected to each other. Measured drug concentrations (e.g., from an arm vein) are referred to the central compartment (#1) which has a stationary volume  $V_1$  assigned. In the catenary absorption model, compartments are lined up in a chain where each compartment is connected to only one neighbor. Catenary and mammillary models are often combined, e.g., to describe drug concentrations after oral input and taking into account the different sections of the gastrointestinal tract.

To describe concentrations in the body, drug rates between compartments need to be mathematically represented. An often made assumption is that rates obey a first-order reaction law. First-order reaction laws are biophysically supported for a number of natural phenomena such as diffusion. Transport of a substance from compartment A to compartment B is said to obey a first-order reaction law (first-order kinetics) if transport rate  $r_{AB}$  is proportional to the amount of substance in compartment A,  $a_A$ , with proportionality constant,  $k_{AB}$ .

$$r_{AB} = k_{AB} \cdot (a_A)^1 \quad (4.11)$$

Similarly, transport of a substance from compartment A to compartment B is said to obey a zero-order reaction law (zero-order kinetics) if the transport rate  $r_{AB}$  is constant.

$$r_{AB} = k_{AB} \cdot (a_A)^0 = k_{AB} \quad (4.12)$$

A more complex rate law is named Michaelis–Menten kinetics (Leonor Michaelis, 1875–1945; Maud Menten, 1879–1960). For this form of kinetics, the transport rate is concentration-dependent, according to:

$$r_{AB} = \frac{v \cdot c_A}{(c_A + k_m)} \quad (4.13)$$

where  $k_m$  is the Michaelis–Menten constant, and  $v$  the maximum transport rate. Large concentrations ( $c_A \gg k_m$ ) result in zero-order kinetics with rate  $v$ , low concentrations ( $c_A \ll k_m$ ) in first-order kinetics with rate constant  $v/k_m$ .

If all drug rates in a PK model that refer to disposition processes are first-order, the superposition principle holds as follows: Assume  $c_1(t)$  and  $c_2(t)$  are two functions describing drug concentrations following drug input into any of the disposition compartments at rates  $r_1(t)$  and  $r_2(t)$ , then  $c_1(t) + c_2(t)$  describes drug concentrations following input into the same compartment at rate  $r_1(t) + r_2(t)$ . PK is said to be linear if  $AUC$  increases proportionally with dose. However, this does not imply that all transport rates are first-order.

If in the mammillary model drug elimination is of first-order,

$$r_{10} = k_{10} \cdot a_1 \quad (4.14)$$

we have the following relationship for clearance,  $CL$ :

$$CL = \frac{r_{10}}{c_1} = k_{10} \cdot \frac{a_1}{c_1} = k_{10} \cdot V_1 \quad (4.15)$$

In the following, the main PK models encountered with empirical PK modeling are introduced.

### One-compartment PK model with IV bolus input and first-order elimination

A straightforward PK model is the one-compartment model with IV drug input and first-order elimination. Its compartmental representation is shown in Fig. 4.4.

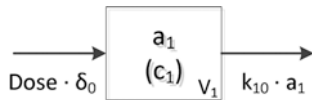
The Dirac delta function  $\delta_0$  indicates that the amount in the compartment is to be increased by  $Dose$  at time zero. It can easily be generalized to multiple doses. For example,

$$\sum_{i=1}^n Dose \cdot \delta_{t_i} \quad (4.16)$$

specifies repeated administration of the same dose at times  $t_i$ ,  $i = 1, \dots, n$ .

The mathematical equations corresponding to Fig. 4.4 are

$$\frac{da_1}{dt} = -k_{10} \cdot a_1; \quad a_1(0) = Dose \quad (4.17)$$



**Fig. 4.4** One-compartment model, bolus input of amount  $Dose$  and first-order drug elimination with elimination rate constant  $k_{10}$ ,  $a_1$ , and  $c_1$ —drug amount and concentration in compartment #1,  $V_1$ —volume assigned to compartment #1. Instead of indicating the drug rate  $k_{10} \cdot a_1$ , often only the rate constant  $k_{10}$  is shown for first-order processes

and

$$c_1 = \frac{a_1}{V_1} \quad (4.18)$$

For multiple doses, (4.17) is transformed to

$$\left. \begin{aligned} \frac{da_1}{dt} &= -k_{10} \cdot a_1 \\ a_1(t_i) &= a_1^-(t_i) + Dose \end{aligned} \right\} \quad (4.19)$$

for  $i = 1, \dots, n$ . The term  $a_1^-(t_i)$  in (4.19) denotes the left-sided limit of  $a_1$  at time  $t_i$  taking into account the discontinuity in amounts due to dosing. Equations (4.17) and (4.18) have a known explicit solution:

$$c_1 = \frac{Dose}{V_1} \cdot e^{-k_{10} \cdot t} \quad (4.20)$$

The solution for multiple dosing can easily be obtained using the superposition principle. Applied to (4.20) the solution can be written as:

$$c_1 = \frac{Dose}{V_1} \sum_{i=1}^n H(t - t_i) \cdot e^{-k_{10} \cdot (t - t_i)} \quad (4.21)$$

$H$  denotes the Heaviside function which is defined for all real numbers as

$$H(x) = 1, \text{ for } x \geq 0; \quad H(x) = 0, \text{ for } x < 0 \quad (4.22)$$

Equation (4.17) is often written in terms of concentrations. If we divide both equations by the (constant) compartmental volume  $V_1$  and consider (4.15), we arrive at

$$V_1 \cdot \frac{dc_1}{dt} = -CL_{10} \cdot c_1; \quad c_1(0) = \frac{Dose}{V_1} \quad (4.23)$$

Equations (4.17) and (4.18) are equivalent to (4.23), however, they emphasize different things. The former refers to rate constants and clearly separates the (unobservable) dynamics of drug amounts from (observable) drug concentrations. The latter has no explicit representation of drug amounts and emphasizes the use of concentrations and clearance terms.

The one-compartment PK model described in this section can be implemented in MATLAB in various ways. Equations (4.17) and (4.18) suggest first solving an ODE for drug amounts and then the algebraic equation for concentrations (see Listing 4.1). It is also possible to do both tasks in one step (see Listing 4.2). Equation (4.20), the explicit solution of (4.17) and (4.18), presents drug concentrations in function of time and can be immediately plotted (see Listing 4.3). Equation (4.23) is an implicit ODE and can be solved in MATLAB without prior division by  $V_1$  (see Listing 4.4). All these listings produce output corresponding to Fig. 4.5.

**Listing 4.1** Program **PK1IV1a.m**: sequential calculation of drug amounts and drug concentrations for a one-compartment PK model with IV input and first-order elimination

```
function PK1IV1a
%PK1IV1a Sequential calculation of drug amounts and concentrations.
%   PK1IV1a sequentially calculates drug amounts and drug
%   concentrations for a one-compartment PK model with IV
%   input and first-order elimination.

clear; clc; close all;
p.V = 5.0;      % L
p.k10 = 1.0;    % mg/h
dose = 100;     % mg
timeEnd = 10;   % h

[t,y] = ode45(@derivatives, [0 timeEnd], dose, [], p);
a = y;
c = a/p.V;
plot(t,a,'-r',t,c,'b-','lineWidth',2)
legend('Drug amounts [mg]','Drug concentrations [mg/L]')
xlabel('Time [h]')
ylabel('Value')
print('-dtiff','-r900','PK1IV1a.tif')

end

function dydt = derivatives(~, y, p)
%DERIVATIVES Compute the right-hand side of the ODE.
%   DYDT = ...

a1 = y(1);
da1 = - p.k10*a1;
dydt = da1;

end
```



**Listing 4.2** Program **PK1IV1b.m**: simultaneous calculation of drug amounts and drug concentrations for a one-compartment PK model with IV input and first-order elimination. The second equation defined in PK1IVb is treated as algebraic equation by using the **Mass** option in **odeset**

```
function PK1IV1b
%PK1IV1B Simultaneous calculation of drug amounts and concentrations.
%   PK1IV1B calculates drug amounts and drug concentrations for
%   a one-compartment PK model with IV input and first-order
%   elimination. The second equation defined in PK1IVb is treated
%   as algebraic equation by using the Mass option in odeset.

clear; clc; close all;
p.V = 5.0;          % L
p.k10 = 1.0;        % mg/h
dose = 100;         % mg
timeEnd = 10;       % h

options = odeset;
options.Mass = [1 0; 0 0];

[t, y] = ode15s(@derivalgeb,[0 timeEnd],[dose; dose/p.V],options,p);
a = y(:,1);
c = y(:,2);
plot(t,a,'-r',t,c,'b-','lineWidth',2)
legend('Drug amounts [mg]','Drug concentrations [mg/L]')
xlabel('Time [h]')
ylabel('Value')

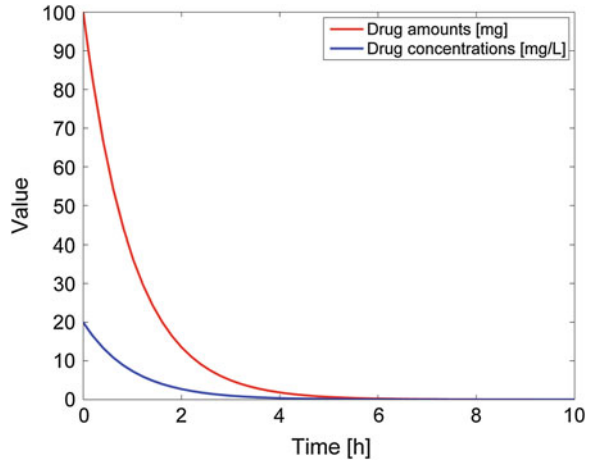
end

function dydt = derivalgeb(~, y, p)
%DERIVALGEB Compute the right-hand side of the DAE.
%   DYDT = ...

a1 = y(1);
c1 = y(2);
da1 = - p.k10*a1; % ode
dc1 = c1 - a1/p.V; % algebraic equation: dc1 = 0
dydt = [da1; dc1];

end
```

**Fig. 4.5** Drug amounts and drug concentrations for a one-compartment PK model with first-order elimination after a single intravenous bolus dose



**Listing 4.3** Program **PK1IV1c.m**: graphical presentation of the explicit solution (drug amounts and drug concentrations) for a one-compartment PK model with IV input and first-order elimination

```
function PK1IV1c
%PK1IV1c explicit solution for drug amounts and concentrations.
% PK1IV1c produced a graph with profiles for drug amounts and
% drug concentrations based on the explicit solution for a one-
% compartment PK model with IV input and first-order elimination

clear; clc; close all;
p.V = 5.0;      % L
p.k10 = 1.0;    % mg/h
dose = 100;     % mg
timeEnd = 10;  % h

t = 0:0.1:timeEnd;
c = dose/p.V*exp(-p.k10*t);
a = c*p.V;
plot(t,a,'-r',t,c,'b-','lineWidth',2)
legend('Drug amounts [mg]','Drug concentrations [mg/L]')
xlabel('Time (h)')
ylabel('Value')

end
```

**Listing 4.4** Program **PK1IV1d.m**: calculation of drug concentrations for a one-compartment PK model with IV input and first-order elimination from an implicit ODE using the **Mass** option in **odeset**

```
function PK1IV1d
%PK1IV1D Calculation of drug concentrations using Mass option.
% PK1IV1D calculates drug concentrations for a one-compartment PK
% model with IV input and first-order elimination. The procedure
% is based on an implicit ODE with the Mass option.

clear; clc; close all;
p.V = 5.0; % L
p.CL10 = 5.0; % L/h
dose = 100; % mg
timeEnd = 10; % h

options = odeset;
options.Mass = p.V;

[t, y] = ode15s(@derivalgeb, [0 timeEnd], dose/p.V, options, p);
c = y;
a = c*p.V;
plot(t,a,'-r',t,c,'b-','lineWidth',2)
legend('Drug amounts [mg]','Drug concentrations [mg/L]')
xlabel('Time [h]')
ylabel('Value')

end

function dydt = derivalgeb(~, y, p)
%DERIVALGEB Compute the right-hand side of the DAE.
% DYDT = ...

c1 = y(1);
dc1 = - p.CL10*c1;
dydt = dc1;

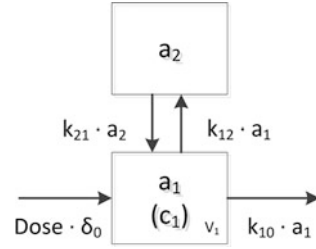
end
```

An interesting feature of the one-compartment PK model described in this section can be seen when considering the logarithm of drug amounts or concentrations. From the explicit solution (4.20) it is obvious that

$$\log c_1 = \log \frac{Dose}{V_1} - k_{10} \cdot t \quad (4.24)$$

indicating a linear relationship of  $\log c_1$  with time  $t$ . It is good practice in empirical PK model-building to plot the logarithm of drug concentration data against time. If a single straight line emerges, one can safely assume a one-compartment model.

**Fig. 4.6** Two-compartment PK model with bolus input into the central compartment (#1) and first-order disposition



### Two-compartment PK model with IV bolus input and first-order elimination

The two-compartment PK model with IV bolus input and first-order elimination is the simplest PK model that can represent drug distribution. Its compartmental representation is shown in Fig. 4.6.

The mathematical equations corresponding to Fig. 4.6 are

$$\left. \begin{aligned} \frac{da_1}{dt} &= -k_{10} \cdot a_1 - k_{12} \cdot a_1 + k_{21} \cdot a_2; & a_1(0) &= Dose \\ \frac{da_2}{dt} &= k_{12} \cdot a_1 - k_{21} \cdot a_2; & a_2(0) &= 0 \\ c_1 &= \frac{a_1}{V_1} \end{aligned} \right\} \quad (4.25)$$

Equation (4.25) is a linear ODE system for which algebraic solutions can be derived. In analogy to the one-compartment model, solutions for the two-compartment model are sums of two exponentials

$$c_1(t) = A \cdot e^{-\lambda_1 \cdot t} + B \cdot e^{-\lambda_2 \cdot t} \quad (4.26)$$

which have to fulfill the initial condition,  $c_1(0) = A + B = Dose/V_1$ , and, by convention,  $\lambda_2 < \lambda_1$ . The constants  $k_{ij}$  of (4.25) are related to the constants  $A$ ,  $B$ ,  $\lambda_1$ ,  $\lambda_2$  of (4.26), as follows:

$$\left. \begin{aligned} \lambda_1 &= 0.5 \cdot (ks + \sqrt{ks^2 - 4 \cdot k_{21} \cdot k_{10}}) \\ \lambda_2 &= 0.5 \cdot (ks - \sqrt{ks^2 - 4 \cdot k_{21} \cdot k_{10}}) \\ A &= \frac{Dose}{V_1} \cdot \frac{(k_{21} - \lambda_1)}{(\lambda_2 - \lambda_1)} \\ B &= \frac{Dose}{V_1} \cdot \frac{(k_{21} - \lambda_2)}{(\lambda_1 - \lambda_2)} \end{aligned} \right\}; \quad ks := k_{12} + k_{21} + k_{10} \quad (4.27)$$

Now we will show that the volume of distribution for this two-compartment model will change over time. In the previous chapter, we defined the volume of distribution  $V$  as the ratio of drug amount in the body to drug concentrations in blood, i.e., compartment #1. Initially, at time 0, all drug is in compartment 1, thus  $V(0) = V_1$ . At any time  $t > 0$ , the amount of drug in the body can be calculated as

the difference between the administered *Dose* and the amount eliminated,  $A_{\text{elim}}(t)$ .  $A_{\text{elim}}(t)$  is easily calculated as:

$$A_{\text{elim}}(t) = \int_0^t k_{10} \cdot a_1(t) \, dt = \int_0^t CL \cdot c_1(t) \, dt \quad (4.28)$$

Using the relationship  $Dose = CL \cdot AUC$ , we get for  $V$

$$V(t) = \frac{Dose - \int_0^t CL \cdot c_1(t) \, dt}{c_1(t)} = \frac{CL \cdot (AUC - \int_0^t c_1(t) \, dt)}{c_1(t)} \quad (4.29)$$

Concentrations  $c_1(t)$  have reached their terminal phase values,  $c_{1T}(t)$ , as soon as they follow a single exponential decline to a sufficient degree. As  $\lambda_2 < \lambda_1$ , this has always been achieved at times  $t$  larger than some time  $T$ . For the two-compartment model the terminal phase concentrations are given by:

$$c_{1T}(t) = B \cdot e^{-\lambda_2 \cdot t}, \quad t > T \quad (4.30)$$

The volume of distribution  $V$  during the terminal phase is then calculated as:

$$\begin{aligned} V(t) &= \frac{CL \cdot (AUC - \int_0^t c_1(t) \, dt)}{c_1(t)} = \frac{CL \cdot \int_t^\infty c_{1T}(t) \, dt}{c_{1T}(t)} \\ &= \frac{CL \cdot \int_t^\infty B \cdot e^{-\lambda_2 \cdot t} \, dt}{B \cdot e^{-\lambda_2 \cdot t}} = \frac{CL}{\lambda_2} \\ &= \frac{Dose}{AUC \cdot \lambda_2} = \frac{k_{10} \cdot V_1}{\lambda_2} > V_1 \end{aligned} \quad (4.31)$$

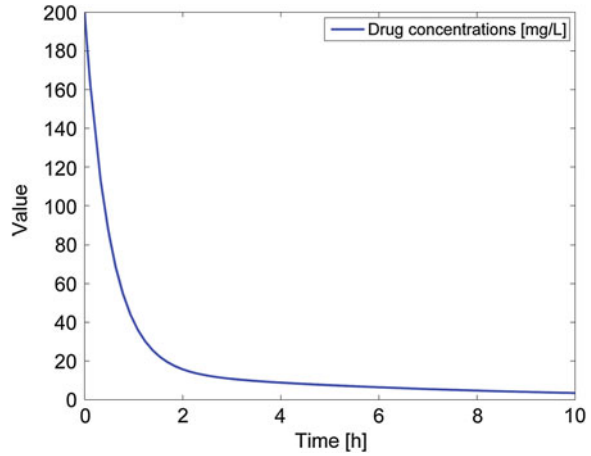
Another volume term can be derived if we consider a constant rate input into the two-compartment model in (4.32).

$$\left. \begin{aligned} \frac{da_1}{dt} &= r_{\text{in}} - k_{10} \cdot a_1 - k_{12} \cdot a_1 + k_{21} \cdot a_2; & a_1(0) &= 0 \\ \frac{da_2}{dt} &= k_{12} \cdot a_1 - k_{21} \cdot a_2; & a_2(0) &= 0 \\ c_1 &= \frac{a_1}{V_1} \end{aligned} \right\} \quad (4.32)$$

At steady state, the derivatives vanish by definition, yielding

$$\left. \begin{aligned} r_{\text{in}} &= k_{10} \cdot a_{1\text{ss}} \\ k_{12} \cdot a_{1\text{ss}} &= k_{21} \cdot a_{2\text{ss}} \end{aligned} \right\} \quad (4.33)$$

**Fig. 4.7** Drug concentrations for a two-compartment PK model with first-order elimination after single intravenous bolus input into the eliminating compartment



Therefore, the volume  $V$  at steady state is

$$V_{ss} = \frac{a_{1ss} + a_{2ss}}{\frac{a_{1ss}}{V_1}} = V_1 \cdot \left(1 + \frac{k_{12}}{k_{21}}\right) > V_1 \quad (4.34)$$

Drug amounts and concentrations for the two-compartment model with IV bolus input and first-order disposition are calculated in Listing 4.5 and respective output is presented in Fig. 4.7.

**Listing 4.5** Program **PK2IV1.m**: calculation of drug amounts and drug concentrations for a two-compartment PK model with IV bolus input and first-order elimination

```
function PK2IV1
%PK2IV1 Solve two-compartment model with bolus.
% PK2IV1 calculates drug amounts and drug concentrations for
% a two-compartment PK model with IV bolus input and first-order
% elimination. The Mass option is applied.

clear; clc; close all;
p.V = 5.0;      % L
p.k10 = 1.0;    % mg/h
p.k12 = 0.8;    % mg/h
p.k21 = 0.3;    % mg/h
dose = 1000;    % mg
timeEnd = 10;   % h

options = odeset;
options.Mass = [1 0 0; 0 0 0; 0 0 1];

[t, y] = ode15s(@derivalgeb,[0 timeEnd],[dose;dose/p.V;0], ...
    options,p);
[t, y]      %#ok<NOPRT>
plot(t,y(:,1),'-r',t,y(:,2),'b-',t,y(:,3),'g-','lineWidth',2)
legend('Central Compartment: Drug amounts [mg]',...
    'Central Compartment: Drug concentrations [mg/L]',...
    'Peripheral Compartment: Drug amounts [mg]')
xlabel('Time (h)')
ylabel('Value')
figure()
plot(t,y(:,2),'b-','lineWidth',2)
legend('Drug concentrations [mg/L]')
xlabel('Time [h]')
ylabel('Value')
print('-dtiff','-r900','PK2IV1.tif')

end

function dydt = derivalgeb(~, y, p)
%DERIVALGEB Compute the right-hand side of the DAE.
% DYDT = ...

a1 = y(1);
c1 = y(2);
a2 = y(3);
da1 = - p.k10*a1 - p.k12*a1 + p.k21*a2;
dc1 = c1 - a1/p.V;
da2 = p.k12*a1 - p.k21*a2;
dydt = [da1; dc1; da2];

end
```



**Fig. 4.8** One-compartment PK model, with extravascular first-order drug absorption and first-order drug elimination (number of compartments are determined by disposition, not by absorption)

Following intravenous bolus input, logarithmic drug concentrations versus time profiles often present two slopes, the first steeper than the second. This can be interpreted as a distribution phenomenon where drug is moving out of the central compartment into a peripheral compartment and returning at respective rates.

### One-compartment PK model with first-order drug absorption and first-order elimination

Until now, we considered only PK models where drug was administered intravenously. For extravascular drug administration, such as oral dosing, we have to consider how the drug is absorbed. In many cases, empirical PK models treat oral drug absorption adequately as a first-order process. Drug is placed into an absorption compartment (bolus input!) from where it is absorbed at a rate proportional to the present amount. The one-compartment PK model with first-order drug absorption and first-order drug elimination is the simplest PK model representing drug concentrations after oral dosing. Its compartmental representation is shown in Fig. 4.8.

The mathematical equations corresponding to Fig. 4.8 are

$$\left. \begin{aligned} \frac{da_0}{dt} &= -k_{01} \cdot a_0; & a_0(0) &= F \cdot Dose \\ \frac{da_1}{dt} &= k_{01} \cdot a_0 - k_{10} \cdot a_1; & a_1(0) &= 0 \\ c_1 &= \frac{a_1}{V_1} \end{aligned} \right\} \quad (4.35)$$

or in terms of concentrations

$$\left. \begin{aligned} \frac{da_0}{dt} &= -k_{01} \cdot a_0; & a_0(0) &= F \cdot Dose \\ V_1 \cdot \frac{dc_1}{dt} &= k_{01} \cdot a_0 - CL_{10} \cdot c_1; & c_1(0) &= 0 \end{aligned} \right\} \quad (4.36)$$

Equation (4.36) can be easily integrated yielding the following equation for drug concentrations  $c_1$ :

$$c_1(t) = \frac{F \cdot Dose}{V_1} \cdot \frac{k_{01}}{k_{01} - k_{10}} \cdot (e^{-k_{10} \cdot t} - e^{-k_{01} \cdot t}) \quad (4.37)$$

where  $k_{10} = CL_{10}/V_1$ . Solution (4.37) presents a structural identifiability problem, i.e., different sets of parameters  $k_{01}$ ,  $k_{10}$ , and  $V_1$  can result in the same function  $c_1(t)$ . This is realized if the following substitutions are made

$$k_{01} \rightarrow k_{10}, \quad k_{10} \rightarrow k_{01}, \quad V_1 \rightarrow V_1 \cdot \frac{k_{10}}{k_{01}} \quad (4.38)$$



Thus, if we fit this model to data, the above two solutions can emerge. Only with additional assumptions, such that  $V_1$  reflects plasma (or blood) volume, can uniqueness be assured.

Drug amounts and concentrations for the one-compartment PK model with first-order absorption and first-order elimination are calculated in Listing 4.6 and respective output is presented in Fig. 4.9.

**Listing 4.6** Program **PK1PO1.m**: example of a one-compartment model with absorption and elimination. The model has an ODE form

```
function PK1PO1()
%PK1PO1 Solve one-compartment model with absorption and elimination.
%   PK1PO1 calculates and produces graphs for the drug amount at
%   absorption site and drug concentration in one-compartment model
%   with first-order drug absorption and first-order elimination.

p.k01 = 0.5;    % 1/h
p.V = 1.0;     % L
p.CL = 2.0;    % L/h
p.F = 1;       % L/h
dose = 100*p.F; % mg

timeEnd = 12;
[t,y] = ode45(@derivatives, 0:0.01:timeEnd, [dose; 0], [], p);
[AX,H1,H2] = plotyy(t,y(:,1),t,y(:,2),'plot');
% Set colors for lines
set(H1,'LineWidth',2,'Color','blue')
set(H2,'LineWidth',2,'Color','red')
% Set colors for amount(t)
set(get(AX(1),'Ylabel'),'String', ...
    'Drug Amount at Site of Absorption','Color', 'blue', ...
    'FontSize', 15)
set(AX(1),'YColor', 'blue', 'FontSize', 15)
% Set colors for concentration(t)
set(get(AX(2),'Ylabel'),'String','Drug Concentrations','Color', ...
    'red', 'FontSize', 15)
set(AX(2),'YColor', 'red', 'FontSize', 15)
title('One-Compartment - Oral Input')
xlabel('Time [h]')
legend('drug amount at site of absorption [mg]', ...
    'drug concentrations [mg/L]')
print('-dtiff','-r900','PK1PO1.tif')

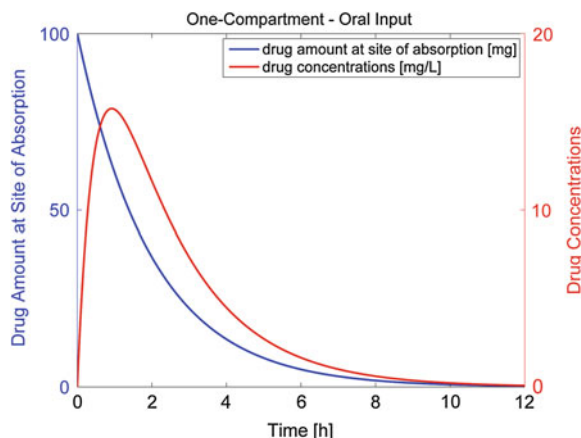
end

function dydt = derivatives(~, y, p)
%DERIVATIVES Compute the right-hand side of the ODE.
%   DYDT = ...

dy1 = - p.k01*y(1);
dy2 = p.k01*y(1) - p.CL/p.V*y(2);
dydt = [dy1; dy2];

end
```

**Fig. 4.9** Drug amounts and drug concentrations for a one-compartment PK model with first-order absorption and first-order elimination



Phenomena underlying oral absorption are complex; tablet disintegration, dissolution, gastrointestinal transport, absorption windows, pH dependencies, transit times in the different sections of the gastrointestinal tract (GIT), and the like. Models have been developed that can account for these phenomena [3, 4].

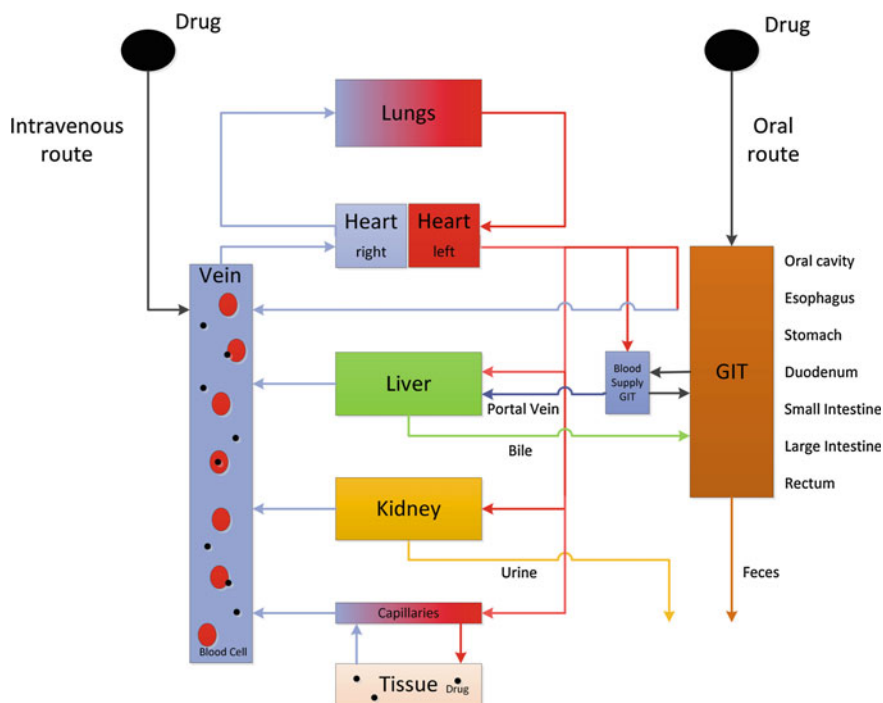
### 4.1.3 Physiologically-based Pharmacokinetics

#### A Systems Approach to PK

Besides data-driven empirical PK modeling, a systems-oriented approach to PK can be applied: PBPK, designed to describe drug concentrations in blood and other organs by utilizing physiological parameters such as blood flow and organ volumes as well as compartments representing different tissues/organs ([5, 6]; Fig. 4.10).

The (left) heart pumps blood into the arterial system from where it supplies each organ with an organ-specific blood flow (blood flow values and relative organ sizes are tabulated in [1]). Following uptake of nutrients organs release blood into the venous system. Blood collected in the venous system is then transported to the (right) heart which pumps the blood to the lungs where it is oxygenated and moved to the (left) heart.

Drug input may be intravenous, in general as an injection or infusion, or oral, into the GIT. Other administration routes include the lungs (inhalation), skin (ointments), muscle (injection), and eye (drops). Orally administered drugs can be absorbed at different parts of the GIT. Drug absorbed from the oral cavity or rectum bypass the liver. All other parts of the GIT (mainly stomach, duodenum, small and large intestine) transport blood (and drug) into the liver via the portal vein. There are transporters in gut wall mucosa cells that take up drug from the gut and may excrete drug back into the gut. Relatively high drug concentrations are processed by the liver during first pass of the drug. The liver may excrete drug into the bile and thence duodenum. Thus, drug could be reabsorbed resulting in enterohepatic cycling.



**Fig. 4.10** Physiologically-based pharmacokinetics (PBPK). Key organs involved in PK processes are shown. *Red and blue arrows* indicate arterial and venous blood flow. Drugs administered by the intravenous route enter blood circulation directly. Drugs administered by the oral route enter blood circulation mainly via the portal vein, i.e. after absorption from the gastrointestinal tract (GIT) and first-pass through the liver. The liver may transport drug back to the GIT via the bile (enterohepatic cycling). Drug may be bound to or taken up by red blood cells or bound to other plasma binding proteins. Free drug may be eliminated from the body via excretion in the urine or metabolism in the liver. Only free drug is considered to occupy receptors and to elicit a biological response

The two major organs of drug elimination are the liver and kidneys. The liver with its extensive enzymatic equipment is the major metabolizing organ of the body. Well-perfused kidneys filter small drugs and excrete them with the urine.

The tissue compartment indicated in Fig. 4.10 represents any organ or tissue that is implicated in the biological response of interest. Drug uptake into the tissue could be limited by blood perfusion or by blood-tissue permeability.

The venous blood compartment is important as it is the standard source of blood samples. Drug concentrations can be measured in blood, plasma (blood without blood cells), or serum (plasma without clotting proteins).

### Organ Models

The PBPK approach is a compartmental approach where organs are represented as shown in Fig. 4.11.



**Fig. 4.11** Organ model as used in PBPK:  $c_A$  ( $c_V$ )—drug concentration in the arterial (venous) system,  $Q$ —blood flow through the organ with volume  $V$  and partition coefficient  $R$ ,  $c$ —organ drug concentration

Blood flow,  $Q$ , transports drug from the arterial system to the organ at rate  $Q \cdot c_A$ . Drug leaves the organ at rate  $Q \cdot c_V$ . The organ is characterized by two parameters, i.e., volume  $V$ , and partition coefficient  $R$ .  $R$  relates drug concentration in the organ to venous drug concentration,  $R = c/c_V$ . With these definitions, the following differential equation holds for  $c$ , under the assumption of mass balance:

$$V \frac{dc}{dt} = Q \cdot \left( c_A - \frac{c}{R} \right); \quad c(0) = c_0 \quad (4.39)$$

If the rate of drug elimination by the organ,

$$R_{\text{elim}} = Q \cdot (c_A - c_V) \quad (4.40)$$

is related to the rate of drug entering the organ,  $Q \cdot c_A$ , we get the extraction ratio  $ER$  of the organ specifying the efficiency of drug elimination ( $ER = 0$ , no elimination;  $ER = 1$ , complete elimination).

$$ER = \frac{c_A - c_V}{c_A} = 1 - \frac{c_V}{c_A} \quad (4.41)$$

According to (4.6) organ clearance is defined as the rate of drug elimination over the organ related to incoming drug concentration

$$CL = \frac{Q \cdot (c_A - c_V)}{c_A} = Q \cdot ER \quad (4.42)$$

Thus, drug organ clearance is linked to physiological quantities, blood flow,  $Q$ , and extraction ratio,  $ER$ , which might be changed by disease states.

A basic PBPK model, representing blood and one organ, is depicted in Fig. 4.12 and respective output is shown in Fig. 4.13.

The differential and algebraic equations describing this model are shown in Eq. 4.43.

$$\left. \begin{aligned} V_b \cdot \frac{dc_b}{dt} &= Q \cdot (c_V - c_A), & c_b(0) &= c_{b0} \\ V \cdot \frac{dc}{dt} &= Q \cdot (c_A - c_V), & c(0) &= c_0 \\ c_V &= \frac{c}{R}, & c_A &= c_b \end{aligned} \right\} \quad (4.43)$$

The respective MATLAB code is as follows (Listing 4.7):

**Listing 4.7** Program **pbpk1.m**

```

function pbpk1
%PBPK1 Solve a basic PBPK model, representing blood and 1 organ.
%   PBPK1 calculates the time courses of the drug concentration
%   in blood and in 1 organ considered in a PKPB model, and produces
%   the respective graphs.

p.Q = 2;
p.V = 5;
p.R = 1;
p.Vb = 10;
p.C = 0;
p.Cb = 10;

timeEnd = 12;
[t, y]=ode45(@derivatives,[0 timeEnd],[p.Cb; p.C],[],p);
plot(t,y)
hold on
p.R = 2;
[t, y]=ode45(@derivatives,[0 timeEnd],[p.Cb; p.C],[],p);
plot(t,y)

xlabel('Time [h]');
ylabel('Drug COncentrations [mg/L]');
legend('Blood','Organ','Location','South')
text(10,7.2,'R_{Organ} = 1')
text(10,9.2,'R_{Organ} = 2')
text(10,4.2,'R_{Organ} = 2')
print('-dtiff','-r800','pbpk1.tif')

end

function dydt = derivatives(~, y, p)
%DERIVATIVES Compute the right-hand side of the ODE.
%   DYDT = ...

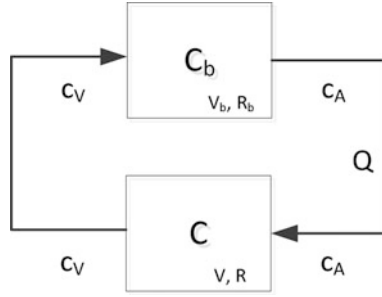
Cb = y(1);
C = y(2);
Ca = Cb;
Cv = C/p.R;
dydt(1) = p.Q*(Cv - Ca)/p.Vb;
dydt(2) = p.Q*(Ca - Cv)/p.V;
dydt = [dydt(1); dydt(2)];

end

```

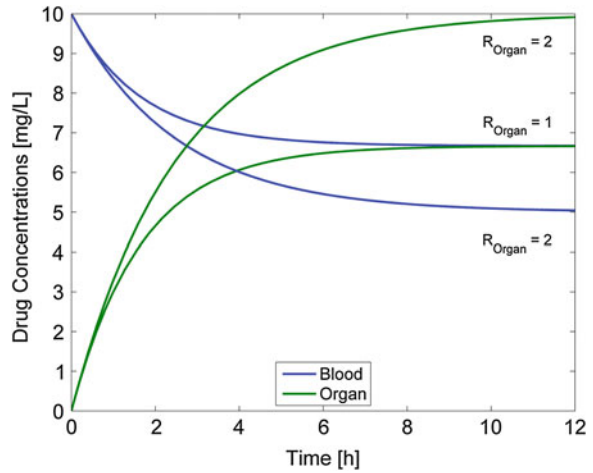
A more realistic PBPK model is created if we add a drug-eliminating organ to the model described before (Fig. 4.14).

The following equations describe this model:

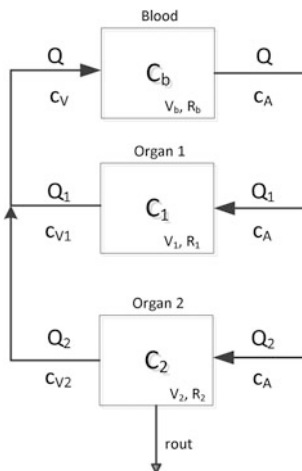


**Fig. 4.12** Basic PBPK model:  $c_A$  ( $c_V$ )—drug concentration in the arterial (venous) system,  $Q$ —blood flow through the organ with volume  $V$  and partition coefficient  $R$ ,  $c$ —organ drug concentration.  $C_b$ —concentration in blood with volume  $V_b$  and partition coefficient  $R_b$

**Fig. 4.13** Basic PBPK model. Impact of  $R_{\text{organ}}$ : at steady state, the ratio of drug concentrations in blood and organ is determined by the organ partition coefficient  $R_{\text{organ}}$



$$\left. \begin{aligned}
 V_b \cdot \frac{dc_b}{dt} &= Q \cdot (c_V - c_A); & c_b(0) &= c_{b0} \\
 V_1 \cdot \frac{dc_1}{dt} &= Q_1 \cdot (c_A - c_{V1}); & c_1(0) &= 0 \\
 V_2 \cdot \frac{dc_2}{dt} &= Q_2 \cdot (c_A - c_{V2}) - r_{\text{out}}; & c_2(0) &= 0 \\
 Q &= Q_1 + Q_2 \\
 c_A = c_b, & \quad c_{V1} = \frac{c_1}{R_1}, & c_{V2} = \frac{c_2}{R_2}, & \quad c_V = \frac{Q_1 \cdot c_{V1} + Q_2 \cdot c_{V2}}{Q}
 \end{aligned} \right\} \quad (4.44)$$



**Fig. 4.14** Basic PBPK model including drug elimination:  $c_A$  ( $c_{V1}$ ,  $c_{V2}$ ,  $c_V$ )—drug concentration in the arterial (venous) system;  $Q$ ,  $Q_1$ ,  $Q_2$ —blood flow through the organs with volume  $V_b$ ,  $V_1$ ,  $V_2$ , and partition coefficient  $R_b$ ,  $R_1$ ,  $R_2$ , respectively;  $c_b$ ,  $c_1$ ,  $c_2$ —drug concentrations in blood and organs.  $r_{out}$ —drug elimination rate from organ 2

Organ 2 eliminates drug at rate  $r_{out}$  which needs to be specified in terms of  $c_2$ , e.g.,  $r_{out} = CL \cdot c_2$  where  $CL$  is an organ clearance. The MATLAB implementation is shown in Listing 4.8 and the respective output in Fig. 4.15.

**Listing 4.8** Program **pbpk3.m**: PBPK model with two organs (one eliminating organ)

```
function pbpk3
%PBPK3 Solve a PBPK model with 2 organs.
%   PBPK3 calculates the time courses of the drug concentration
%   in blood and in a PBPK model representing blood and 2 organs,
%   and produces the respective graphs.

%parameters
%flows
p.Q1 = 2;
p.Q2 = 1;
%volumes
p.Vb = 5;
p.V1 = 4;
p.V2 = 4;
%partition coefficients
p.R1 = 0.5;
p.R2 = 2;
%elimination parameters
p.CL = 1;
%initial values
p.cbi = 10;
p.c1i = 0;
p.c2i = 0;
```

```

timeEnd = 24;
[t, y] = ode45(@derivatives,[0 timeEnd],[p.cbi; p.c1i; p.c2i],[],p);
plot(t,y, 'LineWidth', 2)
hold on
p.CL=0;
[t, y] = ode45(@derivatives,[0 timeEnd],[p.cbi; p.c1i; p.c2i],[],p);
plot(t,y,'-.')
xlabel('Time [h]');
ylabel('Drug Concentrations [mg/L]');
legend('Blood','Organ 1 - non-eliminating', ...
'Organ 2 - eliminating')
print('-dtiff','-r900','pbpk3.tif')

end
function dydt = derivatives(~, y, p)
%DERIVATIVES Compute the right-hand side of the ODE.
%   DYDT = ...

cb = y(1);
c1 = y(2);
c2 = y(3);
Q = p.Q1+p.Q2;
c1v = c1/p.R1;
c2v = c2/p.R2;
ca = cb;
cv = (c1v*p.Q1+c2v*p.Q2)/Q;
rout = p.CL*c2;
dcbdt = Q*(cv - cb)/p.Vb;
dc1dt = p.Q1*(ca - c1v)/p.V1;
dc2dt = (p.Q2*(ca - c2v)-rout)/p.V2;
dydt = [dcbdt; dc1dt; dc2dt];

end

```

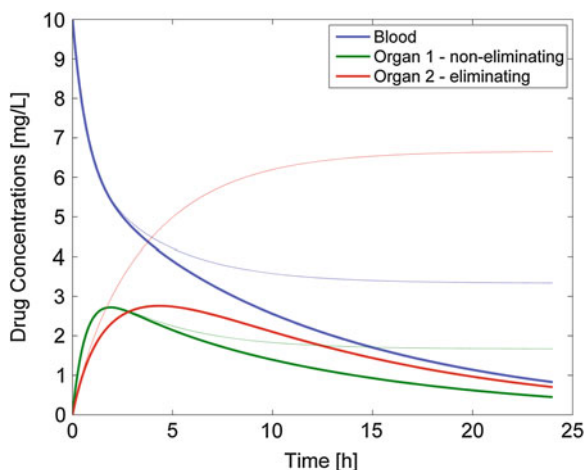
According to the general scheme (Fig. 4.10), more compartments have to be added to the model shown in Fig. 4.14. A special case is presented by the GIT which delivers blood to the liver and not directly into the venous system.

Modeling drug absorption via the GIT is an important aspect of PBPK. Respective models are available on MATLAB Central, [7, 8].

### Role in Drug Development

PBPK models are used in preclinical research to predict drug concentrations in plasma and various organs based on parameters that may be physiological (volumes, blood flows) or compound-specific (partition coefficients, intrinsic clearance). Clinical applications of PBPK models comprise assessment of drug–drug interactions, PK in children (absorption of pediatric drug formulations) and PK in subjects with renal and/or hepatic impairment. In general, PBPK model parameters represent averages (often from a small number of measurements), the literature data, or even educated guesses. Some PBPK model parameters might be very well characterized in terms of central tendency and dispersion (e.g., blood flow in humans). When predicting drug concentrations such knowledge should be incorporated, thus





**Fig. 4.15** PBPK model with eliminating organ. Two scenarios are presented: **a** drug-eliminating organ is active and drug concentrations approach zero (*bold lines*), **b** drug-eliminating organ is blocked and drug concentrations approach non-zero steady state (*thin lines*)

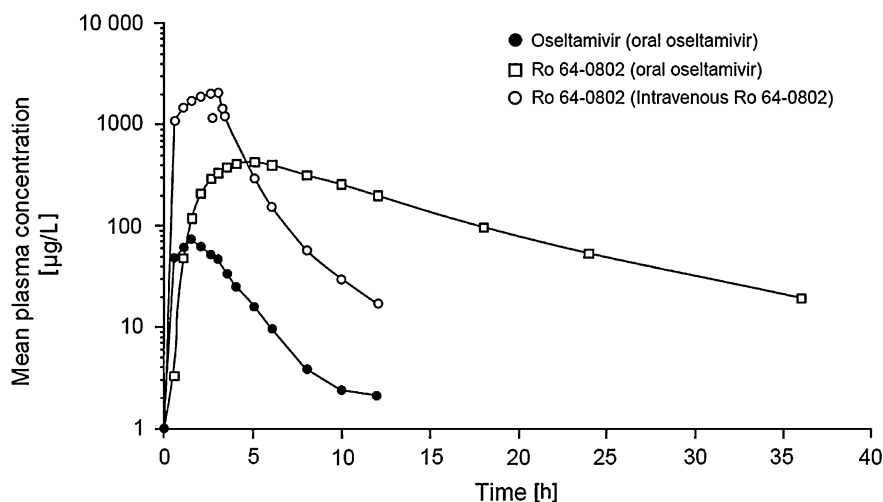
addressing variability. Other PBPK model parameters might not be known enough to assign a firm statistical distribution. When predicting drug concentrations such fuzzy knowledge should be incorporated, thus addressing uncertainty.

#### 4.1.4 Example: *Tamiflu*<sup>®</sup>

Tamiflu (oseltamivir) is a potent, selective, oral neuraminidase inhibitor for the treatment and prophylaxis of influenza. Plasma concentrations of the active metabolite, oseltamivir carboxylate, are increased in the presence of probenecid [9] suggesting that the combination could allow for the use of reduced doses of oseltamivir. To investigate this idea, we developed a population pharmacokinetic model and simulated the pharmacokinetics of candidate combination regimens of oral oseltamivir (45 and 30 mg twice a day) plus oral probenecid (500 mg for every 6 h) [10]. Here, we describe the development of the structural PK model.

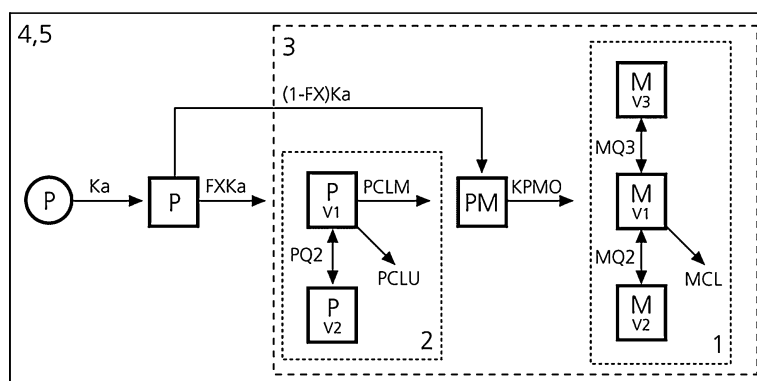
He et al. [9] summarized the clinical pharmacokinetics of oseltamivir and oseltamivir carboxylate and presented the graph shown in Fig. 4.16. Both molecules are eliminated quickly from plasma, and the rate-limiting step appears to be the formation of the metabolite.

Brennan et al. [11] provided other PK data, namely *AUC* values for oseltamivir and oseltamivir carboxylate after 75 mg oral and 100 mg IV dosing. *AUC* values were 140 and 540 ng/mL·h for oseltamivir, and 3060 and 3840 ng/mL·h for the active metabolite. It appears that oseltamivir after oral dosing undergoes a distinct



**Fig. 4.16** Mean plasma concentration–time profiles of oseltamivir and its active metabolite Ro 64-0802 following oral administration of oseltamivir 150 mg and intravenous administration of Ro 64-0802 150 mg to healthy volunteers ( $n = 12$ ) [9]

first-pass effect in the liver where it is carboxylated. When building the PK model, we have to consider this first-pass effect as well as the rate-limited carboxylate release. The actual model was constructed as shown in Fig. 4.17.



**Fig. 4.17** Flow chart illustrating the development of the empirical PK model for oseltamivir and oseltamivir carboxylate. Numbers 1–5 indicate the sequence of model-building steps. Abbreviations:  $P$ —prodrug (oseltamivir),  $M$ —metabolite (oseltamivir carboxylate),  $K_a$  ( $1/h$ )—absorption rate constant,  $FX$ —fraction of drug absorbed as prodrug,  $PQ_2$  ( $L/h$ )—distributional clearance of  $P$  between compartment 1 and 2,  $PCLU$  ( $L/h$ )—urinary clearance of  $P$  from compartment 1,  $PCLM$  ( $L/h$ )—metabolic clearance of  $P$  from compartment 1,  $PM$ —prodrug conversion compartment,  $KPMO$  ( $1/h$ )— $P$  to  $M$  conversion rate,  $MCL$  ( $L/h$ )—total elimination clearance of  $M$  from compartment 1,  $MQ_2$ ,  $MQ_3$  ( $L/h$ )—distributional clearance of  $M$  between compartments 1 and 2, 3.  $PV_1$ ,  $PV_2$  ( $L$ )—volume of distribution of  $P$  in compartments 1 and 2,  $MV_1$ ,  $MV_2$ ,  $MV_3$  ( $L$ )—volume of distribution of  $M$  in compartments 1, 2, and 3 [10]

The following steps were performed:

- Step 1: PK model-building started with IV data for the metabolite, oseltamivir carboxylate, thereby allowing the characterization of disposition without interfering absorption processes. Logarithmic drug concentrations in Fig. 4.16 suggested (at least) a two-compartment disposition model, but actual data supported a three-compartment disposition model.
- Step 2: Similar considerations yielded a two-compartment model for the prodrug, oseltamivir, following IV dosing. Urine data helped to differentiate between urinary and metabolic prodrug clearance.
- Step 3: Next, prodrug data following IV administration were used in conjunction with metabolite data to establish parameters for the metabolic clearance of prodrug, conversion into metabolite, and release of metabolite from the conversion compartment. From Fig. 4.16 it is apparent that this conversion is the rate-limiting step for the kinetics of the metabolite.
- Step 4: Absorption processes following single oral dosing of the prodrug were added to the model. The first-pass effect, which turns prodrug into metabolite during first passage through the liver, was quantified in the range of 80 %.
- Step 5: Confirming the model structure following multiple oral dosing of the prodrug.

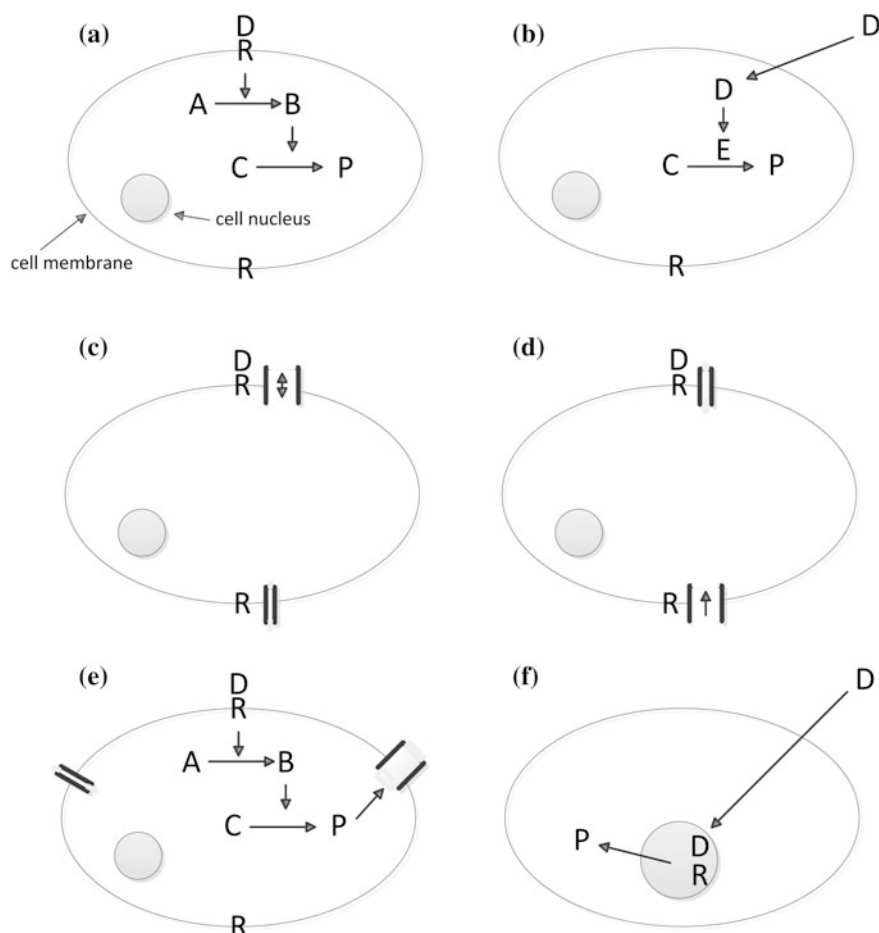
## 4.2 Pharmacodynamics

### 4.2.1 General Concepts

Pharmacodynamics (PD, Greek: *dynamis*—power) is the study of drug effects on biological systems like the human body. This comprises the mode of drug action, the time course of drug response (beneficial or adverse), the relationship between drug concentrations and drug response, and the dose–response relationship. Also, the impact of other determinants (disease state, demographics, etc.) on PD is of high practical importance.

Before a drug can exert any effect, it must bind to a receptor or target. Receptors are in general proteins and located within cells, within cell membranes, or outside cells. Capillary blood flow transports a drug to virtually all of the more than  $10^{13}$  cells of the human body where it binds especially to those receptors for which it has high affinity. Once a drug D is bound to a receptor R to form a drug-receptor complex DR, a cellular pharmacologic response may be elicited. Figure 4.18 shows examples of cellular pharmacologic responses.

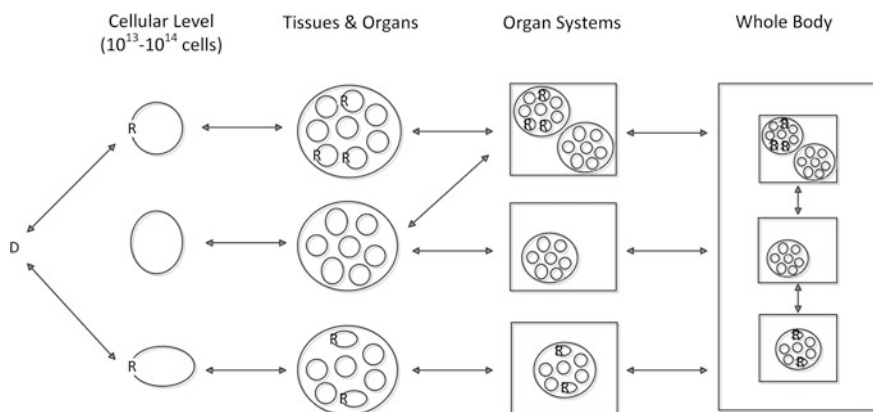
Cellular pharmacologic responses are then propagated to tissues, organs, organ systems, and to the whole body (Fig. 4.19). Signal integration occurs at all these levels according to prevailing (patho)-physiologic conditions. As signals at any



**Fig. 4.18** Primary cellular targets of drugs: **a** Transmembrane drug-receptor complex  $DR$  affects intracellular molecular pathways resulting in a change of product  $P$ ; **b** Drug interferes with intracellular enzymes thereby affecting molecular pathways; **c** Drug-receptor complex opens transmembrane channels thereby changing ionic composition within the cell and subsequently the cell potential; **d** Drug-receptor complex blocks intracellular uptake of molecules; **e** Drug-receptor complex affects transmembrane channels via intracellular pathways; **f** Drug binds to intracellular receptors located in the cell nucleus thereby affecting protein biosynthesis. ( $A$ ,  $B$ ,  $C$ ,  $P$ : intracellular molecules,  $D$ : drug,  $E$ : enzyme,  $R$ : receptor)

level could impact on lower and higher levels, there is no single level controlling drug response. Our expectation is that the biological response to the drug will have therapeutic benefit, i.e., will improve the patient's signs and symptoms, either by acting on the causal chain of the disease or by enhancing endogenous defense or homeostatic mechanisms.

Pharmacologic drug response at the target generally increases with increasing receptor occupancy or target engagement. This is not necessarily the case for



**Fig. 4.19** Response integration of drug-induced effects. Drugs exert their molecular effects on receptors at the cell surface or within cells. Functionally related cells from tissues and organs (e.g., heart, lungs, liver) are grouped into organ systems (e.g., circulatory system, respiratory system). Organ systems are integrated at the whole body level to enable general functioning. Cellular drug effects are propagated through all organizational levels up the whole body. Each level could impact on lower or higher levels as indicated by the *reversible arrow*, i.e., there is no single level of control (D: drug)

responses at higher organizational levels. Engagement of several receptors (with different affinities, or in different tissues) might cause less pronounced effects at higher drug doses. Also, inherent whole body homeostasis and feedback mechanisms could produce non-monotonic dose–response relationships, such as a U-shape dose–response.

## 4.2.2 Receptor Binding

Receptor binding is necessary for producing any drug action [12]. We consider here the binding of one drug molecule,  $D$ , to one receptor,  $R$ , forming a drug–receptor complex,  $DR$ , according to the following reversible reaction:



What can be said about the amount of bound receptor molecules,  $DR$ , over time? If we consider a formation rate constant,  $k_{\text{on}}$ , and a dissociation rate constant,  $k_{\text{off}}$ , we can express the concentration of the drug–receptor complex,  $[DR]$ , as a differential equation:

$$\frac{d}{dt}[DR] = k_{\text{on}} \cdot [D] \cdot [R] - k_{\text{off}} \cdot [DR] \quad (4.46)$$

$[D]$ ,  $[R]$ ,  $[DR]$  are the molar concentrations of free and bound entities. Under dynamic equilibrium, the derivative in (4.46) is zero, yielding

$$K_D := \frac{k_{\text{off}}}{k_{\text{on}}} = [D_{eq}] \cdot \frac{[R_{eq}]}{[DR_{eq}]} \quad (4.47)$$

where  $K_D$  is the equilibrium dissociation rate for the interaction between drug  $D$  and receptor  $R$ . If we assume that the total receptor concentration is conserved at all times,

$$[R_T] = [R] + [DR] \quad (4.48)$$

we can solve (4.47) and (4.48) for  $[DR_{eq}]$ :

$$[DR_{eq}] = \frac{[D_{eq}] \cdot [R_T]}{K_D + [D_{eq}]} \quad (4.49)$$

If we assume that binding of drug to the receptor has a negligible effect on free drug concentrations, we can express (4.49) as follows:

$$[DR_{eq}] = \frac{[D] \cdot [R_T]}{K_D + [D]} \quad (4.50)$$

If drug concentrations match the value of  $K_D$ ,  $[DR_{eq}]$  is equal to half of the total receptor concentration.

The ratio of occupied to total receptors, called receptor occupancy ( $RO$ ), satisfies the following equation:

$$RO = \frac{[DR_{eq}]}{[R_T]} = \frac{[D]}{K_D + [D]} \quad (4.51)$$

The receptor occupancy is illustrated in Fig. 4.20 as a function of concentration  $[D]$ .

How fast does the concentration of the drug-receptor complex,  $[DR]$ , reach dynamic equilibrium  $[DR_{eq}]$ ? From (4.46) and (4.48) we get the following linear differential equation:

$$\left. \begin{aligned} \frac{d}{dt}[DR] &= k_{\text{on}} \cdot [D] \cdot [R_T] - (k_{\text{off}} + k_{\text{on}} \cdot [D]) \cdot [DR] \\ [DR](0) &= 0 \end{aligned} \right\} \quad (4.52)$$

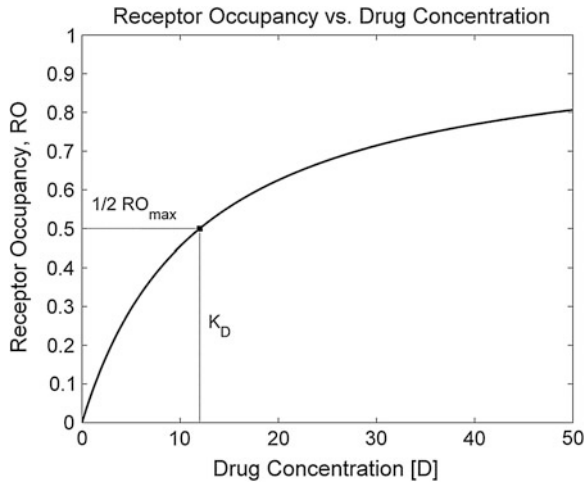
with solution:

$$[DR] = \frac{k_{\text{on}} \cdot [D] \cdot [R_T]}{k_{\text{off}} + k_{\text{on}} \cdot [D]} \cdot (1 - e^{-(k_{\text{off}} + k_{\text{on}} \cdot [D]) \cdot t}) \quad (4.53)$$

which is equivalent to

$$[DR] = [DR_{eq}] \cdot (1 - e^{-(k_{\text{off}} + k_{\text{on}} \cdot [D]) \cdot t}) \quad (4.54)$$

As time approaches infinity,  $DR$  approaches  $DR_{eq}$ , with half-life



**Fig. 4.20** Receptor occupancy as a function of drug concentration

$$t_{1/2} = \ln(2)/(k_{\text{off}} + k_{\text{on}} \cdot [D]) \quad (4.55)$$

The above derivation made no assumption about the conservation of total drug amounts, i.e.,

$$[D_T] = [D] + [DR] \quad (4.56)$$

See Exercise 4.8 for the calculation of the time courses of  $[D]$ ,  $[R]$ , and  $[DR]$  to steady state while conserving total receptor and total drug amounts.

### 4.2.3 Pharmacodynamic Models

In their basic form, PD models describe the relationship between receptor occupancy and drug effects at the cellular level. Receptor occupancy may be replaced by drug concentrations at the receptor site. Modifications of these relationships are to be made if plasma drug concentrations or drug effects at higher organizational levels are considered.

Under the assumption that bound receptor concentration,  $[DR]$ , is proportional to induced effect,  $E$

$$E = \alpha \cdot [DR] \quad (4.57)$$

and that maximum response,  $E_{\text{max}}$ , is achieved by total receptor occupancy, i.e.,  $[DR] = R_T$

$$E_{\text{max}} = \alpha \cdot [R_T] \quad (4.58)$$

we get using (4.51)

$$\frac{E}{E_{\max}} = \frac{[DR]}{[R_T]} = \frac{[D]}{K_D + [D]} \quad (4.59)$$

or

$$E = \frac{E_{\max} \cdot [D]}{K_D + [D]} \quad (4.60)$$

Under above assumptions, the relationship between drug concentration and drug effect is described by a fractional polynomial (4.60), noted as the  $E_{\max}$  model. Due to its algebraic form, it possesses some important properties:

- At zero drug concentration,  $[D] = 0$ , drug effect is zero,  $E = 0$
- Drug effect  $E$  increases monotonically with drug concentration  $[D]$
- Half maximum effect  $E_{\max}$  is produced at drug concentration  $[D]$  equal to  $K_D$
- Drug effect  $E$  has a limiting value,  $E_{\max}$ .

$E_{\max}$  is the most widely applied PD model in pharmacologic modeling. Its specific characteristics remain even if we weaken our assumption that  $E$  is proportional to receptor occupancy  $RO$ . Stephenson [13] suggested separating  $RO$  from  $E$  by introducing the concept that receptors need to be activated beyond binding (Robert Stephenson, 1925–2004). This activation was named stimulus  $S$ . According to Kenakin [12], stimulus  $S$  can be expressed as:

$$S = \frac{e \cdot [D]}{[D] + K_D} \quad (4.61)$$

where  $e$  is noted as efficacy, and

$$E = \frac{S}{S + S_{50}} \quad (4.62)$$

which together gives a typically  $E_{\max}$  relationship.

A similar relationship often holds between drug concentrations  $c$  in plasma and drug response  $E$ , even at the organ or whole body level [14].

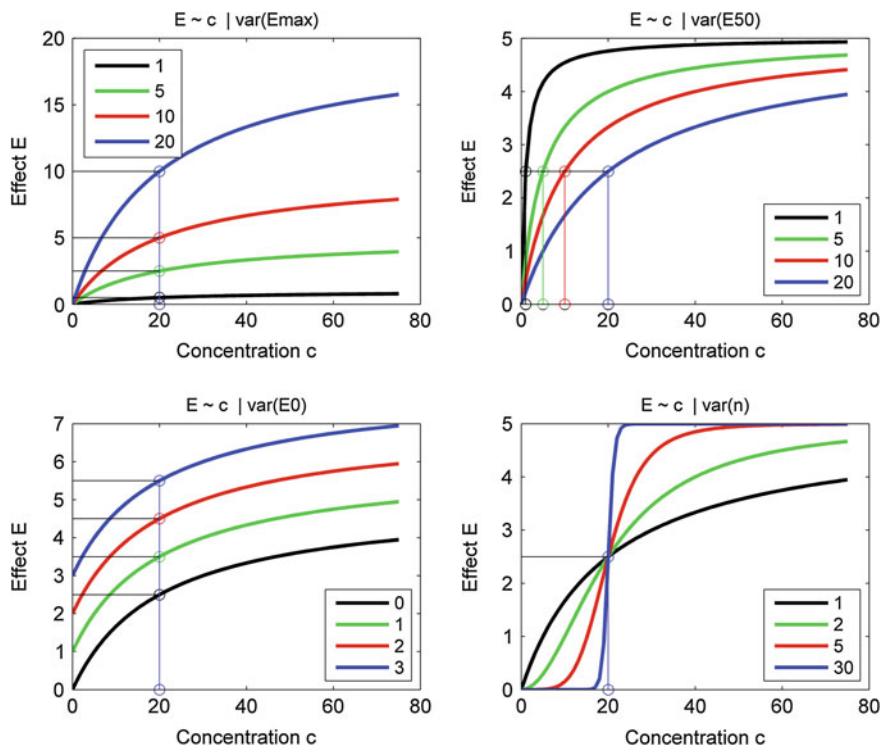
The basic form of the  $E_{\max}$  model is given by:

$$E = \frac{E_{\max} \cdot c}{c + EC_{50}} \quad (4.63)$$

A more general form introduces two other parameters, namely  $n$ , the Hill coefficient, and  $E_0$ , a baseline effect:

$$E = E_0 + \frac{E_{\max} \cdot c^n}{c^n + EC_{50}^n} \quad (4.64)$$





**Fig. 4.21** Parameter sensitivity for the  $E_{\max}$  model

Figure 4.21 illustrates the sensitivity of the  $E_{\max}$  model with regards to its parameters. Changes in  $EC_{50}$ , the concentration at half maximum effect, have no impact on the attained maximum effect. The parameter  $n$  determines the steepness of the curve at  $EC_{50}$ . Large values of  $n$  induce a switch for the effect.

The  $E_{\max}$  model directly couples receptor site concentrations with drug effects; it makes no provisions if concentrations are measured at a different site.

#### 4.2.4 Example: Saquinavir

PD models were applied to saquinavir (SQV), a protease inhibitor for treating HIV infection, to confirm the most appropriate dosage of a new soft gelatin capsule [15]. In this case, a PK model was not available.

The PD modeling data were generated in a randomized open-label parallel study where SQV was given TID as monotherapy in doses of 400, 800, and 1'200 mg to 74 HIV-infected patients for 8 weeks. SQV plasma concentrations were assessed, along with HIV RNA as a surrogate efficacy marker. Data were empirically fitted to six PD exposure-effect models (Table 4.1). SQV  $AUC_{24}$ ,

**Table 4.1** Mathematical models for exposure–response relationships.  $E_0$ —baseline effect in the absence of drug;  $EC_{50}$ —drug concentration at half maximum effect;  $E_{\max}$ —maximum drug effect;  $\gamma$  = slope coefficient

Model number	Model description	Parameter description	Model equation
1	Constant	$P_1$ : constant	$y = P_1$
2	Linear, 0	$P_1$ : slope	$y = P_1 \cdot x$
3	Linear	$P_1$ : intercept, $P_2$ : slope	$y = P_1 \cdot x + P_2$
4	$E_{\max}$	$P_1$ : $E_{\max}$ , $P_2$ : $EC_{50}$	$y = P_1 \cdot x/(x + P_2)$
5	$E_{\max}$ , $E_0$	$P_1$ : $E_{\max}$ , $P_2$ : $EC_{50}$ , $P_3$ : $E_0$	$y = P_3 + P_1 \cdot x/(x + P_2)$
6	Sigmoid $E_{\max}$	$P_1$ : $E_{\max}$ , $P_2$ : $EC_{50}$ , $P_3$ : $\gamma$	$y = P_1 \cdot x^{P_3}/(x^{P_3} + P_2^{P_3})$

calculated as the mean of the 8 h  $AUC$  values recorded at weeks 1 and 3 multiplied by 3 to give an estimate of 24 h exposure, was taken as an independent (exposure) variable, and peak reduction in viral load RNA over the 8 week treatment as a dependent variable.

Model selection was based on two quantitative model selection criteria, the Schwarz criterion ( $SC$ ) [16], and the Akaike information criterion ( $AIC$ ) [17] (Hirotugu Akaike, 1927–2009; Gideon Schwarz, 1933–2007). These balance the quality of data fit and the number of parameters required to obtain the fit. Their respective definitions are as follows:

$$SC = N \cdot \log(WRSS) + NP \cdot \log(N) \quad (4.65)$$

$$AIC = N \cdot \log(WRSS) + 2 \cdot NP$$

where  $N$  is the number of observations,  $WRSS$  the weighted residual sum of squares for deviations between model predictions and measurements, and  $NP$  the number of parameters. When comparing several models for a given set of data, the model associated with the smallest value of  $SC$  or  $AIC$  is regarded as being superior to the other models. Beside these model selection criteria, other common quality-of-fit criteria were also applied, such as plots of residuals and coefficient of variation ( $CV$ ) of parameter estimates.

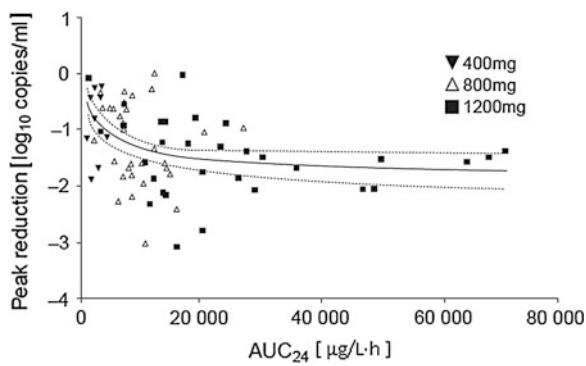
Table 4.2 summarizes the modeling results. The two-parameter maximum effect ( $E_{\max}$ ) model (#4) most appropriately described the relationship between  $AUC_{24}$  and peak reduction in plasma HIV RNA based on  $SC$  and  $AIC$  criteria. The respective model fit is illustrated in Fig. 4.22.

On the basis of results obtained with other antiviral agents at the time of this study, a  $\log_{10}$  decrease of 0.5 in viral load by week 8 was assumed to indicate a treatment response. According to Fig. 4.22, response rates were 45 % (5 of 11) in the 400 mg group, 84 % (27 of 32) in the 800 mg group, and 94 % (29 of 31) in the 1,200 mg group. These results clearly favored the 1,200 mg TID dosage. The median value for SQV  $AUC_{24}$  in patients following the 1,200 mg TID treatment was about 20,000  $\mu\text{g/L}\cdot\text{h}$ , which is nearly 6 times the  $EC_{50}$  value based on HIV RNA peak reduction. This exposure corresponds to 85 % of the maximum

**Table 4.2** Modeling results of SQV exposure ( $AUC_{24}$ ) versus response (peak reduction in plasma HIV RNA)

Model number	Model description	Parameter	Units	Parameter estimate	CV (%)	Akaike criterion	Schwartz criterion
1	Constant	Constant	$\log_{10}c$	-1.28	7	272.0	274.3
2	Linear, 0	Slope	$\log_{10}c/(\mu\text{g/L}\cdot\text{h})$	-0.00	12	325.0	327.3
3	Linear	Intercept	$\log_{10}c$	-1.08	11	267.8	272.4
		Slope	$\log_{10}c/(\mu\text{g/L}\cdot\text{h})$	-0.00	40		
4	$E_{\max}$	$E_{\max}$	$\log_{10}c$	-1.79	11	259.0	263.6
		$EC_{50}$	$\mu\text{g/L}\cdot\text{h}$	3226.16	45		
5	$E_{\max}, E_0$	$E_{\max}$	$\log_{10}c$	-1.67	38	260.9	267.8
		$EC_{50}$	$\mu\text{g/L}\cdot\text{h}$	3945.22	126		
		$E_0$	$\log_{10}c$	-0.15	520		
6	Sigmoid $E_{\max}$	$E_{\max}$	$\log_{10}c$	-1.77	21	261.0	267.9
		$EC_{50}$	$\mu\text{g/L}\cdot\text{h}$	3191.06	51		
		$\gamma$	-	1.03	59		

$\log_{10}c = \log_{10}$  of HIVRNA copies per mL



**Fig. 4.22** Relationship between peak reduction and  $AUC_{24}$  of SQV. A two-parameter  $E_{\max}$  model gave the best data fit. 74 HIV patients were assigned to three treatment arms: 400 mg ( $n = 11$ ), 800 mg ( $n = 32$ ), and 1,200 mg ( $n = 31$ )

achievable effect. These results support 1,200 mg TID as the appropriate dosing regimen, since only marginally higher effects can be expected if SQV 24-h exposure ( $AUC_{24}$ ) is made to exceed 20,000  $\mu\text{g/L}\cdot\text{h}$ .

## 4.3 Time Course of Drug Response

### 4.3.1 General Concepts

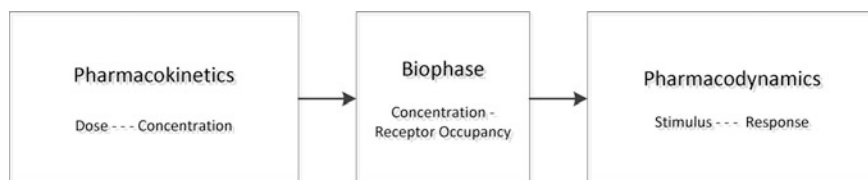
Several factors affect the time course of drug response (Fig. 4.23). Firstly PK, which determines how a dosage regimen translates into drug concentrations over time at different body locations; secondly, the delay needed to equilibrate drug concentrations at the biophase; thirdly, the dynamic processes involved in the integration of PD signals. Such processes are in general physiologically based and the description of these processes depends on the concrete mechanisms involved (see Chap. 5, *Drug–Disease Modeling*).

Studying the effects of drug response on clinical endpoints and deriving therapeutic drug regimens is the ultimate goal of combined PK-PD analyses.

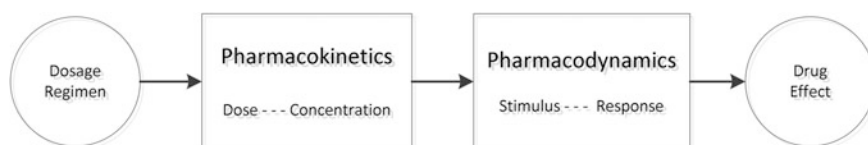
### 4.3.2 PK–PD Relationships

PK and PD models can be combined in different ways to describe the dynamics of drug effects. In the simplest case, a PK model is directly linked to a PD model without consideration of a biophase, as conceptually shown in Fig. 4.24.

The equations for a direct-link PK-PD model, combining a one-compartment PK model with first-order absorption and first-order elimination with a PD  $E_{\max}$  model, are given in Eq. 4.66.



**Fig. 4.23** Pharmacokinetics (PK) and pharmacodynamics (PD) determine drug response over time. Explicit consideration of biophase phenomena may be required to account for delays between drug concentrations and effects. Between stimulus and response there is a cascade of reactions. Response could be considered at different biological levels (cells, tissues, organs). To be most useful, response would be a clinical endpoint, such as blood pressure or survival



**Fig. 4.24** Direct-link PK-PD model

$$\begin{aligned}
 \text{PK: } & \left\{ \begin{aligned} \frac{da_0}{dt} &= -k_{01} \cdot a_0; & a_0(0) &= Dose \\ \frac{da_1}{dt} &= k_{01} \cdot a_0 - k_{10} \cdot a_1; & a_1(0) &= 0 \\ c &= \frac{a_1}{V} \end{aligned} \right\} \\
 \text{PD: } & E = \frac{E_{\max} \cdot c}{c + EC_{50}}
 \end{aligned} \tag{4.66}$$

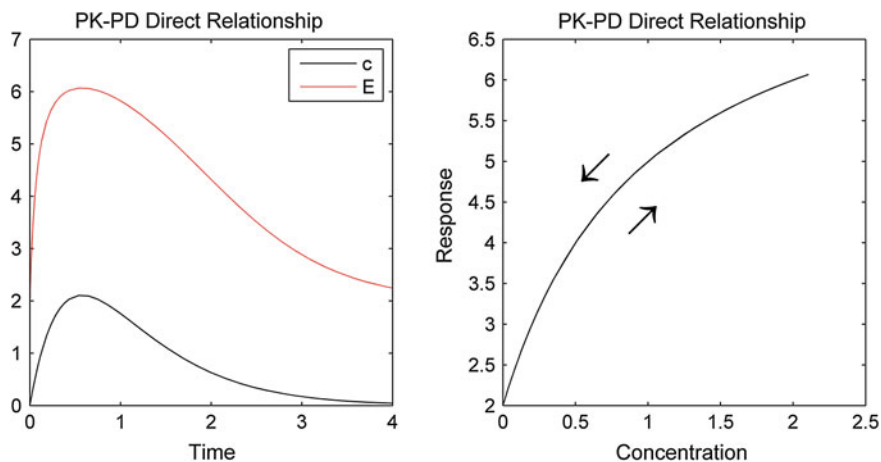
An illustration of the direct-link PK-PD model is given in Fig. 4.25.

Often, there is a delay between plasma drug concentrations and effect. This can be accounted for by considering an equilibration delay between plasma drug concentrations and drug concentrations at the biophase. To represent this delay, a compartment, called effect compartment, is dynamically linked to plasma concentrations without impacting PK. Figure 4.26 illustrates this situation conceptually.

The equations for an indirect-link PK-PD model are given in Eq. 4.67.

$$\begin{aligned}
 \text{PK: } & \left\{ \begin{aligned} \frac{da_0}{dt} &= k_{01} \cdot a_0; & a_0(0) &= Dose \\ \frac{da_1}{dt} &= k_{01} \cdot a_0 - k_{10} \cdot a_1; & a_1(0) &= 0 \\ c &= \frac{a_1}{V} \end{aligned} \right\} \\
 \text{EC: } & \frac{dc_e}{dt} = k_{e0} \cdot (c - c_e); & c_e(0) &= 0 \\
 \text{PD: } & E = \frac{E_{\max} \cdot c_e}{c_e + EC_{50}}
 \end{aligned} \tag{4.67}$$

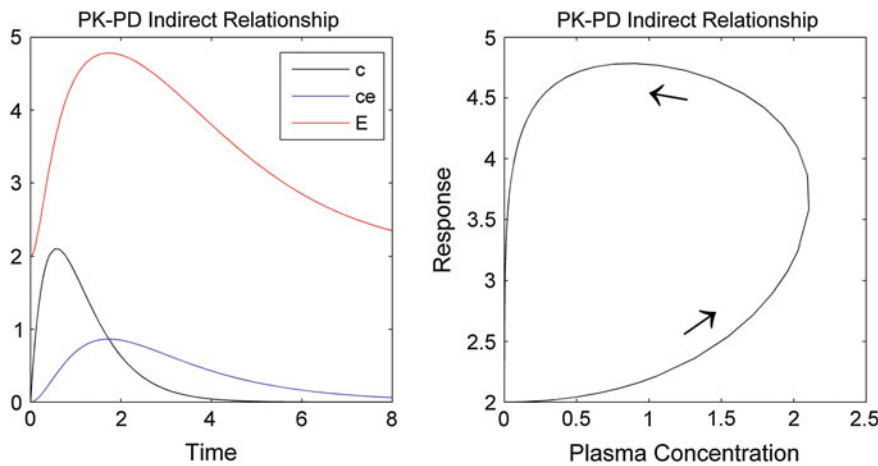
From these equations, the following behavior of concentrations and effects can be derived (Fig. 4.27).



**Fig. 4.25** Direct-link PK-PD model. *Left panel:* Time course of concentrations ( $c$ ) and response ( $E$ ). *Right panel:* Concentration-response relationship. Arrows indicate how concentration and response vary over time starting and ending at the bottom left



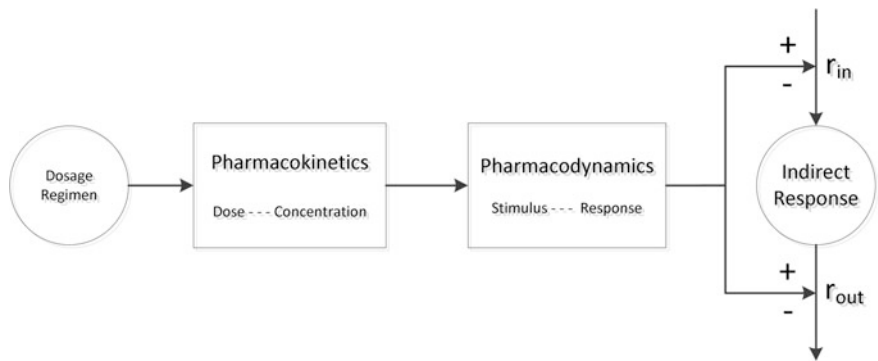
**Fig. 4.26** Effect compartment (indirect-link) PK-PD model



**Fig. 4.27** Indirect-link PK-PD model. *Left panel:* Time course of concentrations ( $c$ ), effect compartment concentrations ( $c_e$ ), and response ( $E$ ). *Right panel:* Concentration–response relationship. Arrows indicate how concentration and response vary over time starting and ending at the bottom left

Indirect response (IDR) models (Fig. 4.28) are another type of models reflecting delays between plasma concentrations and effects.

IDR models represent drug effects through the perturbation of a (physiologic) system. The state of the system is characterized by a response variable  $R$  which

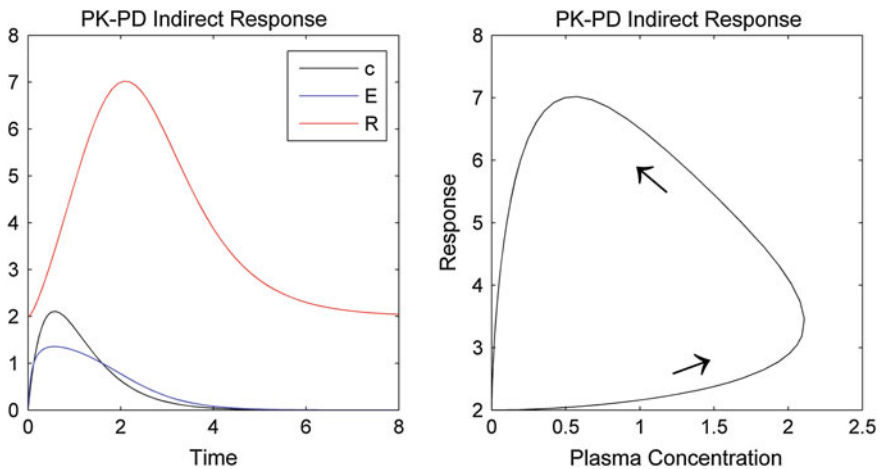


**Fig. 4.28** Indirect response (IDR) model. The primary PD response changes the state of an indirect response variable by increasing/decreasing its formation rate ( $r_{in}$ ) or its elimination rate ( $r_{out}$ )

undergoes formation at formation rate  $r_{in}$  and degradation at rate constant  $r_{out}$ . Often, degradation is taken as a first-order process with degradation rate constant  $k_{out}$ . Response variables could be an endogenous entity (e.g., an enzyme activity), a functional measurement (e.g., blood pressure), or a clinical rating scale (e.g., the Hamilton Depression Rating Scale). An indirect response could be coupled to another indirect response. The perturbation of the system is accomplished by adding a PD effect (via an  $E_{max}$  model) on the parameter  $r_{in}$  or  $r_{out}$ . This could either increase or decrease  $r_{in}$  or  $r_{out}$ . Equation (4.68) describes a situation where plasma drug concentrations increase the parameter  $k_{out}$  according to an  $E_{max}$  model.

$$\left. \begin{array}{l} \text{PK: } \left\{ \begin{array}{l} \frac{da_0}{dt} = -k_{01} \cdot a_0; \quad a_0(0) = Dose \\ \frac{da_1}{dt} = k_{01} \cdot a_0 - k_{10} \cdot a_1; \quad a_1(0) = 0 \\ c = \frac{a_1}{V} \end{array} \right. \\ \\ \text{PD: } E = \frac{E_{max} \cdot c}{c + EC_{50}} \\ \\ \text{IDR: } \frac{dR}{dt} = k_{in} - k_{out} \cdot (1 + E) \cdot R; \quad R(0) = k_{in}/k_{out} \end{array} \right\} \quad (4.68)$$

An illustration of the model described in (4.68) is given in Fig. 4.29 and the MATLAB implementation in Listing 4.9.



**Fig. 4.29** Indirect response model. *Left panel:* Time course of concentrations ( $c$ ), PD effect on  $k_{out}$  ( $E$ ), and response ( $R$ ). *Right panel:* Concentration–response relationship. *Arrows* indicate how concentration and response vary over time starting and ending at the bottom left

**Listing 4.9** Program **PKPDIndirectResponse.m**

```

function PKPDIndirectResponse()
%PKPDINDIRECTRESPONSE Solve indirect response model.
%   PKPDINDIRECTRESPONSE computes the concentration, drug
%   effect, and response variable in a PK-PD indirect response
%   model with impact on a physiologic mechanism by the drug effect.
%   The model definition contains the Mass option.

% parameters
p.ka = 2;
p.k = 1.5;
p.V = 2;
p.Emax = 2;
p.EC50 = 1;
p.rin = 2;
p.kout = 1;

% DGL and variable assignment
options = odeset;
options.Mass = [1 0 0 0 0; 0 1 0 0 0; 0 0 0 0 0; 0 0 0 0 0; ...
               0 0 0 0 1];
inits = [10,0,0,0,p.rin/p.kout];
[time,y] = ode15s(@derivalgeb,[0 8],inits,options,p);
conc = y(:,3);
E = y(:,4);
R = y(:,5);

figure('Units','characters','Position',[10 10 150 25], ...
       'PaperPositionMode','auto');
subplot(1,2,1)
plot(time,conc,'k');      % concentration
hold on
plot(time,E,'b');         % drug effect
plot(time,R,'r');         % response
xlabel('Time','FontSize',15);
legend('Location','Best','c','E','R');
title('PK-PD Indirect Response','FontSize',15);

subplot(1,2,2)
plot(conc,R,'k')
xlabel('Plasma Concentration','FontSize',15);
ylabel('Response','FontSize',15);
title('PK-PD Indirect Response','FontSize',15);
text(1.5,2.7,'\leftarrow','Rotation',200,'FontSize',22);
text(1.0,6.0,'\leftarrow','Rotation',320,'FontSize',22);
print('-dtiff','-r900','PKPDIndirectResponse.tif')

end

```



```

function dydt = derivalgeb(~,y,p)
%DERIVALGEB Compute the right-hand side of the DAE.
%   DYDT = ...

a0 = y(1);
a1 = y(2);
c  = y(3);           % concentration
E  = y(4);           % drug effect
R  = y(5);           % response
da0 = -p.ka*a0;
da1 = p.ka*a0 - p.k*a1;
dc  = c - a1/p.V;
dE  = E - p.Emax*c/(c+p.EC50);
dR  = p.rin - (p.kout-E)*R;
dydt = [da0; da1; dc; dE; dR];

end

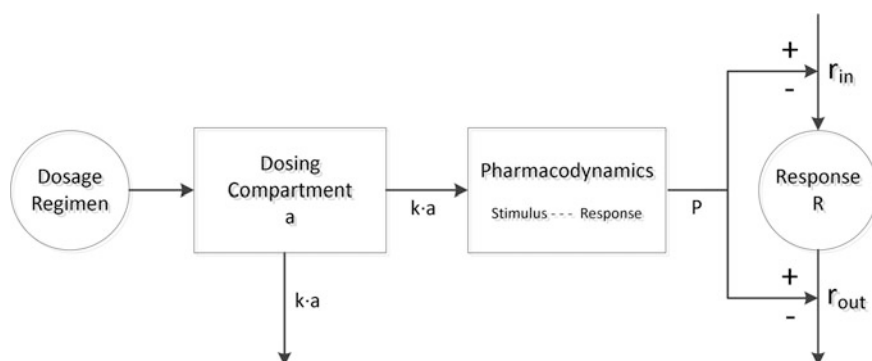
```

The three PK-PD models presented in (4.66), (4.67), and (4.68) form the cornerstone of nearly all empirical PK-PD modeling. Occasionally, a more elaborate equation than the single ODE describing drug response,  $R$ , might be needed to capture physiologic complexity. Examples are the Gompertz cell growth model (Chap. 2), the cell life span model for erythrocytes (Sect. 5.3), integrated glucose-insulin models (Sect. 5.4), viral dynamics models (Sect. 5.5), and a model for the dynamics of cardiovascular drug action [18].

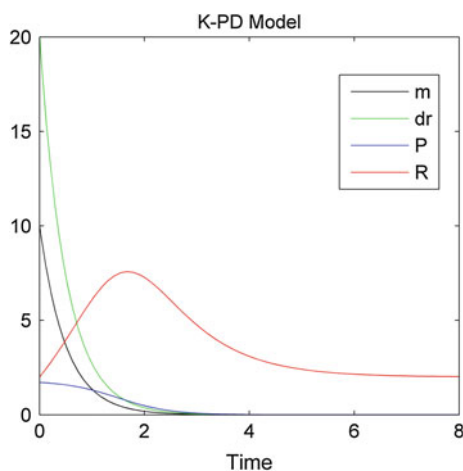
### 4.3.3 K-PD Analyses

A PK-PD model contains extensive quantitative information about a drug, its PK and PD. Development of such models is the gold standard of pharmacometric analysis. However, sometimes PK information is not available to develop a proper PK-PD model, or the PK-PD model is too complex to deliver results in time. Parameter estimation in PK-PD models is usually done using a mixed effects modeling approach (Chap. 6, *Population Analysis*) which is computationally intense, and a complex PK-PD model might turn out impractical. In such situations the K-PD (standing for kinetics of pharmacodynamic response) model might be applicable because it tries to model drug response profiles without using drug concentrations [19]. The idea is to create a time-dependent ‘stimulus’ that acts as a perturbation on an appropriate dynamic system by affecting system parameters according to some  $E_{\max}$ -type relationship (Fig. 4.30).

The equations for the K-PD model are shown below (4.69) and an illustration of potential outcome in Fig. 4.31.



**Fig. 4.30** K-PD model. Drug is dosed into the dosing compartment according to some dosage regimen. Drug is eliminated from the dosing compartment by a first-order rate,  $k \cdot a$ . The same first-order rate produces a primary PD response  $P$  (e.g., according to an  $E_{\max}$  model with  $k \cdot a$  as input). The response  $P$  acts on a dynamic system by either changing its input or output rate



**Fig. 4.31** Kinetics of pharmacodynamics (K-PD) response model. This model describes drug effects over time without drug concentrations.  $m$  is drug amounts in a virtual dosing compartment which is eliminated via a first-order rate.  $dr$  (dose rate) is a signal which is proportional to the drug amounts in the dosing compartment.  $P$  stands for the pharmacologic effect induced by the signal  $dr$ .  $P$  is related to  $dr$  via an  $E_{\max}$  model.  $R$  is the response variable which is the output of a dynamic system perturbed by  $P$

$$\left. \begin{aligned} \frac{da}{dt} &= -k \cdot a; & a(0) &= Dose \\ dr &= k \cdot a \\ P &= \frac{P_{\max} \cdot dr}{dr + Pdr_{50}} \\ r_{\text{out}} &= k_{\text{out}} \cdot R \cdot (1 + P) \\ \frac{dR}{dt} &= r_{\text{in}} - r_{\text{out}}; & R(0) &= \frac{r_{\text{in}}}{k_{\text{out}}} \end{aligned} \right\} \quad (4.69)$$

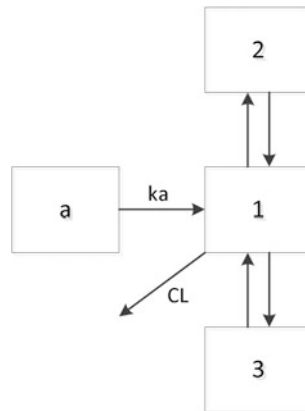
#### 4.3.4 Example: Ibandronate

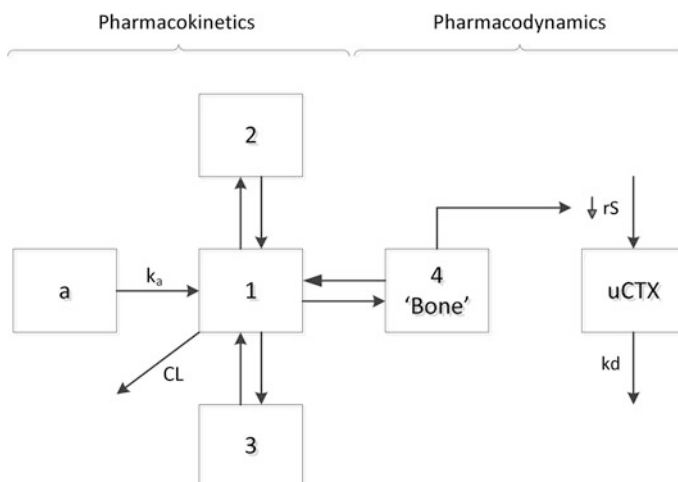
An extensive modeling and a simulation project was undertaken to develop and validate a pharmacologically realistic mathematical model for ibandronate, a potent, nitrogen-containing bisphosphonate for the treatment of postmenopausal osteoporosis, the aim being to identify practical dosing regimens for clinical evaluation. During the course of the project, three models were developed: A PK model, a PK-PD model, and finally a K-PD model [20, 21].

##### PK model

The PK model was developed using data from two studies; PK data from a study of oral and IV ibandronate in healthy postmenopausal Caucasian women ( $n = 34$ ) and PK data from a study of IV ibandronate in postmenopausal Japanese women with osteopenia ( $n = 50$ ). The model comprised a first-order absorption rate for oral dosing, and a three-compartment disposition model with linear drug elimination (Fig. 4.32).

**Fig. 4.32** PK model for oral ibandronate. The PK is represented by a first-order absorption model linked to a three-compartment disposition mode [22]





**Fig. 4.33** PK-PD model for ibandronate. PK is represented by a four-compartment disposition model. Drug concentrations in the 'Bone' compartment are linked to a decrease in urinary *CTX*, i.e., C-telopeptide of the  $\alpha$ -chain of type I collagen, expressing drug-induced inhibition of *uCTX* forming bone cells, osteoclasts

### PK-PD model

Through their action on osteoclasts, bisphosphonates reduce osteoclast-mediated bone resorption. Consequently, levels of markers of bone resorption, such as *CTX* derived from type I collagen and measureable in serum and urine, are also reduced. Bone marker data provide a rapid, easily obtainable, and readily measurable indication of the PD response to ibandronate treatment. Abundant serum and urinary *CTX* data (*sCTX* and *uCTX*) from several clinical studies were used to develop the PK-PD model for ibandronate. The magnitude of the pharmacologic effect of ibandronate and, therefore, the PD response, is determined by its concentration in bone, where it accumulates over time, rather than in serum. Initially, the three-compartment disposition PK model described above for ibandronate was extended by adding a fourth compartment, i.e., 'bone' (Fig. 4.33), which allowed a link to the PD response.

This fourth compartment was not identifiable using ibandronate PK data alone, due to an inadequate duration of PK sampling and/or serum drug concentrations during the terminal elimination phase that were below the lower limit of quantification of the assay. The PD of ibandronate was modeled using an indirect response model under the assumption that *uCTX* levels decreased as an effect of ibandronate, which inhibits osteoclast-mediated bone resorption. The multi-compartment PK-PD model for ibandronate adequately described the PK of IV ibandronate and the time course of *uCTX* in the Japanese study. The mathematical description of the model is shown below in (4.70).

$$\left. \begin{aligned}
 \frac{da_1}{dt} &= -\left(\frac{CL}{V_1} + k_{12} + k_{13} + k_{14}\right) \cdot a_1 + k_{21} \cdot a_2 + k_{31} \cdot a_3 + k_{41} \cdot a_4 \\
 \frac{da_2}{dt} &= k_{12} \cdot a_1 - k_{21} \cdot a_2 \\
 \frac{da_3}{dt} &= k_{13} \cdot a_1 - k_{31} \cdot a_3 \\
 \frac{da_4}{dt} &= k_{14} \cdot a_1 - k_{41} \cdot a_4 \\
 \frac{d}{dt}uCTX &= ks \cdot INH \cdot FUN - kd \cdot uCTX \\
 \frac{da_e}{dt} &= \frac{CL}{V_1} \cdot a_1
 \end{aligned} \right\} \quad (4.70)$$

$INH$  and  $FUN$  are defined as follows:

$$INH = 1 - \frac{\left(\frac{a_4}{V_4}\right)^{HILL}}{IC_{50}^{HILL} + \left(\frac{a_4}{V_4}\right)^{HILL}}, \quad FUN = 1 + \frac{R_{tar} - ks}{ks} \cdot (1 - e^{-k_{qq} \cdot t}) \quad (4.71)$$

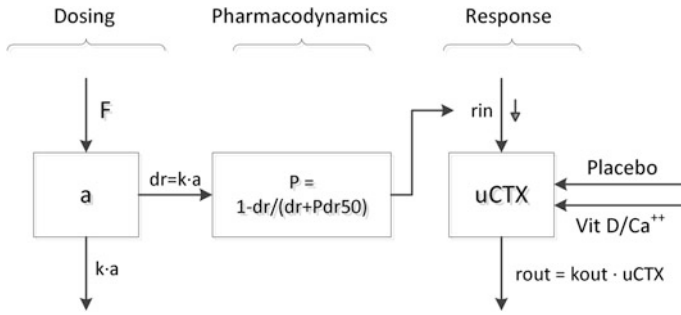
$INH$  is an inhibitory  $E_{max}$ -type relationship taking values between 0 and 1, and  $FUN$  defines a time-dependent function taking values between 1 (for  $t = 0$ ) and  $R_{tar}/ks$  (for  $t = \infty$ ).  $R_{tar}$  accounts for a possible change in  $ks$  with rate constant  $k_{qq}$ .

### K-PD model

Although successful, the ‘physiologic’ PK-PD model for ibandronate was complex, needed long computer run times, and presented the numerical difficulties associated with solving stiff systems, i.e., systems with present rate constants of widely different orders of magnitude. A simplified version of the PK-PD model would, therefore, be more practical.

A characteristic feature of ibandronate is the large difference in the time scales for evaluating PK, as measured by drug concentrations (hours), and PD, as measured by  $uCTX$  levels (weeks and months). The apparent dissociation between PK and PD justified using a K-PD approach, i.e., to ‘abstract’ the PK. The reduction of  $uCTX$  in response to IV ibandronate was modeled as a decrease in the production rate of  $uCTX$  via an inhibitory sigmoidal  $E_{max}$  model, where the independent variable was the dose-driving rate, which is a product of the elimination rate from the ‘virtual’ effect site (where the drug is delivered) and the amount of drug in this compartment (Fig. 4.34).

When the K-PD model was fitted to the data used to develop the more complex PK-PD model, the fit was virtually indistinguishable from the more complex PK-PD model. In addition, computer run times were reduced to just 2 h. The robustness of the K-PD model was investigated by evaluating the stability of the K-PD model parameters in conditions where ibandronate total-dose exposure was kept constant, but the administration schedule was varied. This exercise was prompted by the need for a model that would be capable of evaluating the PD response to different ibandronate dosing schedules.



**Fig. 4.34** K-PD model for ibandronate. Drug amounts,  $a$ , in a virtual dosing compartment induce an inhibitory signal  $k \cdot a$  on the production of  $uCTX$ .  $uCTX$  is regulated by zero-order production,  $r_{in}$ , and first-order elimination with rate constant  $k_{out}$ . In addition, the model assumes effects of placebo treatment and of supplementary therapy (Vit D,  $Ca^{++}$ ) on  $uCTX$

The equations for the K-PD model are shown in (4.72). The  $uCTX$  variable was amended to  $uCTX_{composite}$  to reflect the additional effects of placebo ( $PBO$ ) and vitamin supplementation ( $VIT$ ).

$$\left. \begin{aligned}
 \frac{da}{dt} &= -k \cdot a \\
 \frac{d}{dt} uCTX &= r_{in} \cdot P - r_{out} \\
 r_{out} &= k_{out} \cdot uCTX \\
 dr &= F \cdot k \cdot a \\
 P &= 1 - \frac{dr^{HILL}}{Pdr_{50}^{HILL} + dr^{HILL}} \\
 PBO &= 1 + slope \cdot t \\
 VITD &= 1 - VIT \cdot (1 - e^{KVIT \cdot t}) \\
 uCTX_{composite} &= uCTX \cdot PBO \cdot VITD
 \end{aligned} \right\} \quad (4.72)$$

Supplemental therapy (daily oral calcium/vitamin D) is indicated in clinical practice and has been shown to decrease the rate of bone metabolism [23]. Empirically, the effects of supplemental therapy were introduced into the K-PD model in addition to the effects of ibandronate treatment on  $uCTX$ . Supplemental therapy was estimated to eventually reduce  $uCTX$  by approximately 30 %. The relative effect of placebo on  $uCTX$  was expressed as a linear relationship representing disease progression (see Sect. 5.5). The parameters of the K-PD model are listed in Table 4.3.

### Computational Model

A MATLAB program implementing (4.72) is shown in Listing 4.10. As shown in Fig. 4.35, it allows running a number of prespecified dosage regimens by selecting them from a GUI.

**Listing 4.10** Program **ibandronate.m**: implementation of the ibandronate model

```
function ibandronate
%IBANDRONATE select regimens for simulations
%   IBANDRONATE creates a list-selection dialog box and lets the
%   user select from a list which regimens type are to be simulated.
%   No input parameters are needed. This function has to be called
%   first to start the simulation process.

% Create a list-selection dialog box
listAdmins = {'po25oad', 'po100qlm', 'po150qlm', 'placebo', ...
    'placebo(fast)', 'int', 'iv1000qlms', 'test'};
selectionNum = listdlg('Name', 'Dose Administration', ...
    'PromptString', 'Select dose administration(s):', ...
    'SelectionMode', 'multiple', 'ListSize', [180,300], ...
    'ListString', listAdmins);
selection = listAdmins(selectionNum); % vectorized selection names
allreg(selection);

% save graphs as TIFF files
uctxFigure = figure(1);
print(uctxFigure, '-r900', '-dtiff', 'uctx')
ctxAUCFigure = figure(2);
print(ctxAUCFigure, '-r900', '-dtiff', 'ctxAUC')

end

function allreg(selection)
%ALLREG call simulation for all regimens
%   ALLREG(SELECTION) initiates simulation for all drug
%   administrations specified in |SELECTION|.
%   Examples for calling:
%       allreg({'po150qlm'})
%       allreg({'po100qlm', 'po25oad', 'placebo'})
%       allreg({'po150qlm', 'po100qlm', 'po25oad', 'placebo'})

hpw = waitbar(0,'Please wait...');
uctxFigure = figure(1); set(uctxFigure, 'Visible', 'off')
uctxAxes = axes; set(uctxAxes, 'FontSize',15)
ctxAUCFigure = figure(2); set(ctxAUCFigure, 'Visible', 'off')
ctxAUCAxes = axes; set(ctxAUCAxes, 'FontSize',15)

numSimulations = length(selection);
uctxLegend = cell(numSimulations, 1);
ctxAUCLegend = cell(numSimulations, 1);
for i=1:numSimulations
    [rAUC, info] = solveiban1(selection{i}, hpw);
    display(info);
    uctxLegend{i} = num2str(selection{i});
    ctxAUCLegend{i} = num2str(rAUC);
end
close(hpw)
```

```

set(uctxFigure,'units','normalized','Position', ...
[0.1, 0.5, 0.3, 0.4] , 'Visible','on');
title('uCTX vs. Time');
legend(uctxLegend, 'Location','East');
set(ctxAUCFigure, 'units','normalized','Position', ...
[0.6, 0.5, 0.3, 0.4] , 'Visible','on');
title('CTX vs. Time / AUC (legend)');
legend(ctxAUCLegend, 'Location','East');

end

function [AUC, outText] = solveiban1(regimenType,hpw)
%SOLVEIBAN1 Solve ibandronate model.
% [AUC, OUTTEXT] = SOLVEIBAN1(REGIMENTYPE,HPW) solves the
% ibandronate model for 1 regimen, specified as |REGIMENTYPE|.
% |HPW| is the handle to the wait bar that has to display the
% progress of calculation. The function returns the solution in
% graphical form, area under the curve |AUC|, and a message,
% |OUTTEXT|, if successfully passed.

% model parameters / initialization
p.rin = 255; % formation rate
p.kout = 1.06; % degradation rate
p.slope = 2.32E-4; % slope contribution of placebo effect
p.VIT = 0.314; % vitamin/ca++ effect
p.KVIT = 0.0118; % rate constant
p.VITDCA = 1; % Vit.D Calc yes(1)/no(0)
p.HILL = 0.913; % shape factor
p.Pdr50 = 17.2; % potency for inhibition
p.k1 = 0.0138; % elimination rate constant +VITD/Ca++
p.k2 = 0.112; % elimination rate constant -VITD/Ca++
p.Y20 = p.rin/p.kout;
p.k = p.k1*p.VITDCA+p.k2*(1-p.VITDCA);

% dose time and dose quantity preparation
[p.doseTimes, p.doseAmounts, sCol] = doseSchedule(regimenType);
p.numDoses = length(p.doseTimes); % number of doses
i=1:p.numDoses-1;
doseIndexes = num2cell(i);
% preparation for time domain: [0; t(1)]
tspan = [0 1200];
options = odeset('Events',@events);
% Initial values (stage 1)
amount = p.doseAmounts(1); % 1-st dose = initial value;
y0 = [amount; p.Y20; 0];
[t,y,~,~,iE] = ode45(@derivatives, tspan, y0, options, p);
% Prepare table for graph
timePoints = t;
yPoints = y;

```



```

% Handle Events
while ~isempty(iE)
    waitbar(iE(end)/p.numDoses, hpw, ['Please wait...', ...
        regimenType]);
    switch iE(end)
        case doseIndexes % event is a dose time
            iE(end);
            % preparation for time domain: [t(n); t(n+1)]
            tspan = [t(end), t(end) + 1:1200];
            amount = p.doseAmounts(iE(end));
            % Initial values (stage after event)
            y0 = [y(end,1) + amount, y(end,2:end)];
            [t,y,~,~,iE] = ode45(@derivatives, tspan, y0, ...
                options, p);
            timePoints = [timePoints; t]; %ok
            yPoints = [yPoints; y]; %ok Prepares table for graphic
        case p.numDoses % end of time integration event
            break;
        otherwise
            messageID = 'Book:ibandronate:UnknownEvent';
            messageStr = 'Unknown event ''%s''.';
            error(messageID, messageStr, num2str(iE(end)));
    end
end

% Compute Area Under the Curve (AUC)
AUC = yPoints(end,3);
% Prepare graphs
numPoints = length(timePoints);
ctx = zeros(numPoints,1);
pctctx = zeros(numPoints,1);

i=1:numPoints;
PBO = 1+p.slope*timePoints(i,1); % baseline fraction due to disease
VITD = (1-p.VIT.*(1-exp(-p.KVIT.*timePoints(i,1)))).^p.VITDCA;
ctx(i,1) = yPoints(i,2).*PBO.*VITD;
pctctx(i,1) = (ctx(i,1)-p.Y20)./p.Y20.*100;

figure(1);
% set(uctxFigure, 'Visible', 'off')
plot(timePoints, pctctx, sCol,'LineWidth', 2);
xlabel('Time [d]');
ylabel('uCTX_{ PCTBS}');
grid on; hold on;

figure(2);
% set(ctxAUCFigure, 'Visible', 'off');
plot(timePoints, ctx, sCol, 'LineWidth', 2);
xlabel('Time [d]');
ylabel('CTX');
grid on; hold on;
outText = ['regimen: ', num2str(regimenType), ' passed'];

end

```

```

function [doseTimes, doseAmounts, sCol] = doseSchedule(regimenType)
%DOSCHEDULE Creates a vector of dosing times and amounts of drug.
% [DOSETIMES,DOSEAMOUNT,SCOL] = DOSCHEDULE(REGIMENTYPE) returns
% a vector |DOSETIMES| of dosing times and a scalar |DOSEAMOUNT|
% of the dosing amount. |REGIMENTYPE| must be either 'po25oad',
% 'po100q1m', 'po150q1m', 'placebo', 'placebo(fast)', 'int',
% 'iv1000q1ms', or 'test'. |SCOL| is a colour option the type of
% dosing regimen.
switch regimenType
case 'po25oad'
    sCol = 'b';           % blue
    dose = 2500;          % microgram
    F1 = 0.00757;         % bioavailability
    doseTimes = 0:1:1080;
    doseAmounts = ones(1,length(doseTimes))*dose*F1;
case 'po100q1m'
    sCol = 'g';           % green
    dose = 100000;        % microgram
    F1 = 0.00664;         % bioavailability
    doseTimes= 0:30:1080;
    doseAmounts = ones(1,length(doseTimes))*dose*F1;
case 'po150q1m'
    sCol = 'r';           % red
    dose = 150000;        % microgram
    F1 = 0.00664;         % bioavailability
    doseTimes = 0:30:1080;
    doseAmounts = ones(1,length(doseTimes))*dose*F1;
case 'placebo'
    sCol = 'k';           % black
    % no dose neither F1 as it is placebo (dose = 0)
    doseTimes = 0:1080;
    doseAmounts = zeros(1,length(doseTimes));
case 'placebo(fast)'
    sCol = 'c';           % cyan
    % no dose neither F1 as it is placebo (dose = 0)
    doseTimes = [0 1080];
    doseAmounts = zeros(1,length(doseTimes));
case 'int'
    sCol = 'y';           % yellow
    dose = 20000;         % microgram
    F1 = 0.00664;         % bioavailability
    % dose given on day:    0,    2, ... , 22
    % and then             90,  92, ... , 112
    % and then             180, 182, ... , etc.
    doseTimes = reshape(repmat((0:2:22)',1,length(0:90:990)) ...
        + repmat(0:90:990, length(0:2:22), 1), 1, ...
        length(0:90:990)*length(0:2:22));
    doseAmounts = ones(1,length(doseTimes))*dose*F1;
case 'iv1000q1ms'
    sCol = 'c';
    dose = 1000;          % ug !!!!!
    F1 = 1;
    doseTimes = 0:30:1080;
    doseAmounts = ones(1,length(doseTimes))*dose*F1;

```

```

    case 'test' % another test example
        sCol = 'm'; % magenta
        doseTimes = [ 0 125 500 600 620 1000 1080];
        doseAmounts = [664 664 664 664 664 664 664];
    otherwise
        messageID = 'Book:ibandronate:UnknownRegimen';
        messageStr = ['Unknown regimen '%s'. A regimen must ', ...
            'be: 'po25oad'', 'po100qlm'', 'po150qlm'',', ...
            ''placebo'', 'placebo(fast)', 'int'',', ...
            ''iv1000qlms'' or 'test''.'];
        error(messageID, messageStr, regimenType);
    end
    doseTimes = doseTimes';
    doseAmounts = doseAmounts';

end

function dydt = derivatives(t, y, p)
%DERIVATIVES Compute the right-hand side of the ODE.
% DYDT = ...

dydt(1) = -p.k*y(1);
dydt(2) = p.rin*(1 - (y(1)*p.k)^p.HILL/(p.Pdr50^p.HILL ...
    +(y(1)*p.k)^p.HILL)) - p.kout*y(2);
dydt(3) = y(2)*(1 + p.slope*t)*(1 - p.VIT*(1-exp(-p.KVIT*t))) ...
    ^p.VITDCA;
dydt = dydt'; % must be a column vector

end

function [value,isterminal,direction] = events(t,~,p)
%FEVENTS Specify events of the ODE model.
% [VALUE,ISTERMINAL,DIRECTION] = ...

% Locate the time when "t - <event-time>" passes 0
% dose time - events
value(:,1) = t - p.doseTimes(2:p.numDoses)'; % time event values
isterminal(:,1) = ones(1,p.numDoses - 1); % [1 1 ... ]
direction(:,1) = zeros(1,p.numDoses - 1); % [0 0 ... ]
% time integration - event
value(p.numDoses,1) = t - 1081; % end of time integration
isterminal(p.numDoses,1) = 1; % Stop the integration definitely
direction(p.numDoses,1) = 0; % both direction

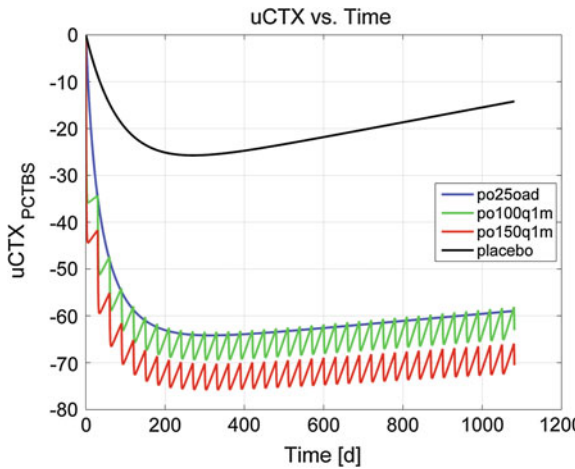
end

```

**Table 4.3** Model parameters for the K-PD model describing ibandronate, supplemental, and placebo treatment effects on  $uCTX$  in post-menopausal patients [22]

Model parameter	Unit	Parameter estimate
$r_{in}$	$\mu\text{g mmolCR}^{-1} \text{d}^{-1}$	255
$k_{out}$	$\text{d}^{-1}$	1.06
$Pdr_{50}$	$\mu\text{g d}^{-1}$	17.2
$KDE(-VitD/Ca^{++})$	$\text{d}^{-1}$	0.112
$KDE(+VitD/Ca^{++})$	$\text{d}^{-1}$	0.014
$HILL$ coefficient		0.913
$slope$ (placebo)	$\text{d}^{-1}$	$2.32 \text{ E-4}$
$VIT$	$\mu\text{g mmolCR}^{-1} \text{d}^{-1}$	0.314
$KVIT$	$\text{d}^{-1}$	0.012
$F$		0.007

**Fig. 4.35** Time course of  $uCTX$  (percent change from baseline, PCTBS), a PD biomarker of ibandronate response, following different oral dosage regimens of ibandronate (2.5 mg oad, 100, and 150 mg monthly) and placebo



4.4 Exercises

Exercise 4.1

Show that  $V_{app}$  is time-independent if ratios of drug concentrations in the body between any two tissues or fluids are constant over time.

Exercise 4.2

Show that first-order kinetics implies dose-proportionality and that the superposition principle holds.

Exercise 4.3

Show that dose-proportionality does not imply first-order kinetics.

**Exercise 4.4**

Describe the one-compartment model with IV bolus input and zero-order elimination.

**Exercise 4.5**

Write MATLAB code for multiple oral administrations into a one-compartment model with first-order elimination. Show numerically that the superposition principle holds.

**Exercise 4.6**

Construct a PBPK model comprising blood, muscle, liver, and gut. Develop equations and MATLAB code.

**Exercise 4.7**

Derive the equation for equilibrium concentrations for the reaction  $D + R \leftrightarrow DR$  under the assumption that total drug and total receptor are conserved.

**Exercise 4.8**

Calculate the time-dependent solution for  $[DR]$  in Exercise 4.7.

**Exercise 4.9**

Derive the Michaelis-Menten rate law from enzyme kinetics considerations.

**Exercise 4.10**

Show that multiple nested  $E_{\max}$  models yield a further  $E_{\max}$  model.

**Exercise 4.11**

Describe empirical criteria that allow differentiation between the direct link, the effect compartment, and indirect response PK-PD models.

**Exercise 4.12**

Develop the equations for competitive inhibition of an enzyme.

**References**

1. Rowland M, Tozer T (2011) Clinical pharmacokinetics and pharmacodynamics. Concepts and applications, 4th edn. Wolters Kluwer, Philadelphia
2. Gibaldi M, Perrier D (1982) Pharmacokinetics, 2nd edn. Marcel Dekker, New York
3. Darwich AS, Neuhoﬀ S, Jamei M, Rostami-Hodjegan A (2010) Interplay of metabolism and transport in determining oral drug absorption and gut wall metabolism: a simulation assessment using the “Advanced Dissolution, Absorption, Metabolism (ADAM)” model. *Curr Drug Metab* 11:716–729
4. Agoram B, Woltosz WS, Bolger MB (2001) Predicting the impact of physiological and biochemical processes on oral drug bioavailability. *Adv Drug Deliv Rev* 50:S41–S67
5. Rowland M, Carl Peck C, Geoffrey Tucker G (2011) Physiologically-based pharmacokinetics in drug development and regulatory science. *Annu Rev Pharmacol Toxicol* 51:45–73

6. Poulin P, Jones RDO, Jones H, Gibson C, Rowland M, Chien J, Ring BJ, Adkison KK, Ku S, He H, Vuppugalla R, Marathe P, Fischer V, Dutta S, Sinha VK, Bjoernsson T, Lave T, Yates JW (2011) PHRMA CPCDC initiative on predictive models of human pharmacokinetics. Part 5: Prediction of plasma concentration–time profiles in human by using the physiologically-based pharmacokinetic modeling approach. *J Pharm Sci* 100:4127–4157
7. <http://www.mathworks.ch/matlabcentral/fileexchange/index?term=id%3A37132>
8. <http://www.mathworks.ch/matlabcentral/fileexchange/index?term=id%3A37752>
9. He G, Massarella J, Ward P (1999) Clinical pharmacokinetics of the prodrug oseltamivir and its active metabolite Ro 64-0802. *Clin Pharmacokinet* 37:471–484
10. Rayner CR, Chanu P, Gieschke R, Boak LM, Jonsson EN (2008) Population pharmacokinetics of oseltamivir when coadministered with probenecid. *J Clin Pharmacol* 48:935–947
11. Brennan BJ, Davies B, Cirrincione-Dall G, Morcos PN, Beryozkina A, Chappey C, Aceves Baldó P, Lennon-Chrimes S, Rayner CR (2012) Safety, tolerability and pharmacokinetics of intravenous oseltamivir: single- and multiple-dose phase I studies in healthy volunteers. *Antimicrob Agents Chemother* 56:4729–4737
12. Kenakin TP (eds) (2009) *A pharmacology primer. Theory, applications, and methods*, 3rd edn. Elsevier, Amsterdam
13. Stephenson RP (1956) A modification of receptor theory. *Brit J Pharmacol* 11:379–393
14. Holford NHG, Sheiner LB (1982) Kinetics of pharmacological response. *Pharmacol Ther* 16:143–166
15. Gieschke R, Fotteler B, Buss N, Steimer JL (1999) Relationships between exposure to saquinavir monotherapy and antiviral response in HIV-positive patients. *Clin Pharmacokinet* 37:75–86
16. Schwarz G (1978) Estimating the dimensions of a model. *Ann Stat* 6:461–464
17. Akaike H (1974) A new look at the statistical model identification. *IEEE Trans Autom Control* 19:716–723
18. Francheteau P, Steimer JL, Merdjan A, Guerret M, Dubray C (1993) A mathematical model for the dynamics of cardiovascular drug action: application to intravenous dihydropyridines in healthy volunteers. *J Biopharm Pharmacokinet* 23:493–514
19. Jacqmin P, Snoeck E, van Schaick EA, Gieschke R, Pillai G, Steimer JL, Girard P (2007) Modelling response time profiles in the absence of drug concentrations: definition and performance evaluation of the K-PD model. *J Pharmacokinet Pharmacodyn* 34:57–85
20. Pillai G, Gieschke R, Goggin T, Jacqmin P, Schimmer RC, Steimer JL (2004) A semimechanistic and mechanistic population PK–PD model for biomarker response to ibandronate, a new bisphosphonate for the treatment of osteoporosis. *Brit J Clin Pharmacol* 58:618–631
21. Reginster JY, Gieschke R (2006) Clinical utility of a pharmacostatistical model for ibandronate in postmenopausal osteoporosis. *Curr Drug Metab* 7:827–836
22. Pillai G, Gieschke R, Goggin T, Steimer JL (2010) Population pharmacokinetics of ibandronate in Caucasian and Japanese post-menopausal women. 10th PAGE meeting, Basel
23. Reid IR, Ames RW, Evans MC, Gamble GD, Sharpe SJ (1995) Long-term effects of calcium supplementation on bone loss and fractures in postmenopausal women: a randomized controlled trial. *Am J Med* 98:331–335

## Chapter 5

# Drug-Disease Modeling

Disease modeling is applied for two different reasons. First, it attempts to characterize a disease completely enough so that the impact of potential treatments can be envisaged. Physiologic models of affected body organs or systems provide the basis for implementing disease mechanisms. The large amount of (patho)-physiologic information makes for potentially (over) elaborate physiologic/disease models. Drug-disease models combine pharmacologic models with disease models. Here, a compromise needs to be found between pharmacologic/biological plausibility and computational feasibility. Second, disease modeling tries to assess the progression of a disease with and without treatment. Due to the paucity of information determining the time course of a disease, disease progression models will mainly be built empirically. Drug-disease models help to identify and assess appropriate targets at an early state of drug development, as well as the planning of trials in later stages of development.

### 5.1 Terms and Concepts

Physiology investigates body function under normal conditions. It was established in its own right in 1628 by the work of William Harvey (1587–1657), who was the first to describe the systemic blood circulation in detail. However, it took another 200 years until physiology was accepted as an experimental-based science rather than a set of questionable rules dictated with high authority. Claude Bernard (1813–1878) became the first integrative physiologist who emphasized that experiments should be guided by hypotheses to enhance interpretation of the many experimental results. He also stressed the importance of mathematics providing the final form of all our natural laws. Claude Bernard recognized the importance of the constancy of the “milieu intérieur” for the human body [1] and Walter B. Cannon (1871–1945) coined the term “homeostasis” to designate the steady states resulting from coordinated physiological processes [2]. A disease can then be considered as a manifest disturbance in the homeostasis of some bodily systems.

**Table 5.1** Number of entities at different levels of the human body (average values)

Level	Number
Gene	25,000
Protein	250,000–1,000,000
Cell	$10^{13}$ – $10^{14}$
Cell types	300
Tissue <sup>a</sup>	4
Organ systems <sup>b</sup>	9

<sup>a</sup> see Table 5.2<sup>b</sup> see Table 5.3**Table 5.2** Main types of human tissues

Tissue	Function (examples)
Epithelial	Enclosure of organs keeping them separated from the environment
Connective	Formation and connection of bone, joints, tendon, cartilage
Muscle	Contraction
Nerve	Generation and propagation of electrical signals

Before introducing disease modeling approaches, some aspects of human physiology should be mentioned. The whole body can be regarded as composed of a number of functional subsystems which rely on the interplay of different organs. Organs (and tissues) are formed by cells with a common differentiation. Cells contain the intracellular fluid (cytosol) and subcellular structures which support the functioning of the cell and its tasks within an organ or tissue. Table 5.1 provides some quantitative information about different levels of the human body [3, 4].

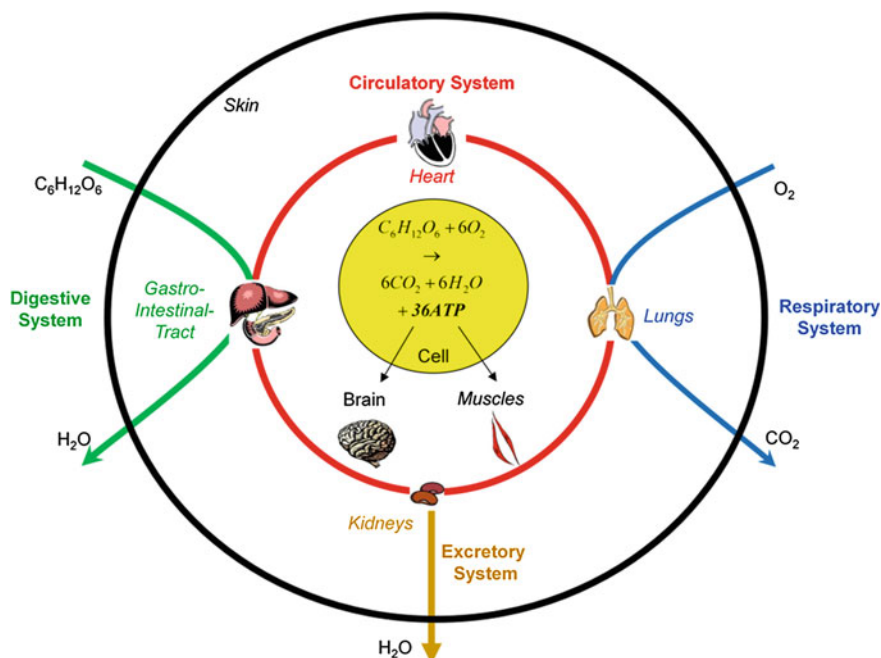
Tissues are defined as groups of similar cells that are specialized to perform a common function (Table 5.2).

From a functional viewpoint, the body can be categorized into subsystems as listed in Table 5.3 and illustrated in Fig. 5.1.

**Table 5.3** Body subsystems and organs involved

Subsystem	Organs
Circulatory system	Heart, vessels, blood, bone marrow
Respiratory system	Lungs, trachea, nose
Digestive system	Mouth, oesophagus, stomach, duodenum, intestine, liver, pancreas
Excretory system	Kidney, intestine, bladder
Movement system	Muscles, skeleton
Defence system	Skin; immune system: blood, lymph nodes
Information processing and control system	Nervous system: brain, spinal cord, peripheral nerves; endocrine system: hypothalamus, pancreas, thyroid gland, adrenal glands
Reproductive system	Ovary, uterus, vagina, prostate, testes, penis

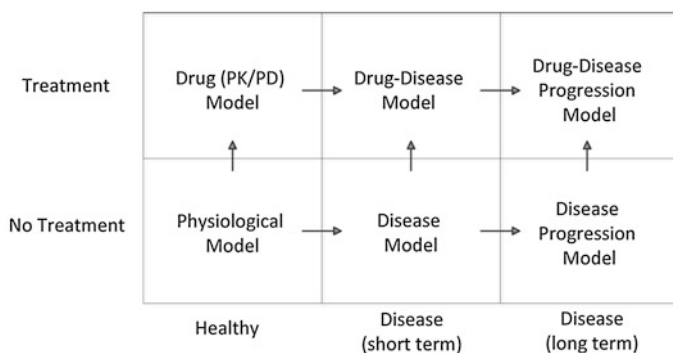




**Fig. 5.1** Key functional systems of the human body, powered by the production of the high-energy cell fuel adenosine triphosphate (ATP) [5]. Each cell has the equipment to produce ATP from glucose ( $C_6H_{12}O_6$ ) and oxygen ( $O_2$ ) as indicated by the chemical reaction (yellow circle). Glucose is produced by the liver or directly provided in food. Red cells collect ambient oxygen from the lungs and carry it to the cells; the circulation also provides glucose. Carbon dioxide ( $CO_2$ ) and water ( $H_2O$ ) leave the cell as waste products and are eliminated by the kidneys and gastrointestinal tract. The main consumers of ATP are brain and muscle (as part of the movement system). The brain is the key information processing and control system of the body; it shares control with the endocrine system. The defense system is indicated by the skin, the largest organ of the body. Not indicated in the scheme are the immune and reproductive systems

Disease models are developed to characterize disease mechanisms and/or to describe disease progression. Models that characterize disease mechanisms often start from a physiologic model of biological systems underlying the disease. Accounting for processes leading to the disease results in a disease model. These models may be extended to disease progression models if one is interested in the time course of the disease and how it can be changed by drug treatment. This is mainly the case for chronic diseases, such as Alzheimer's. Whereas disease models can often be built as mechanistic and biologically realistic models, disease progression modeling is mainly empirical as generally little is known about temporary changes in the pathophysiology of a disease over time.

To develop a drug-disease model one could start from a disease model and amend it with drug-specific PK and PD properties. As these properties are often learned first from studies in healthy volunteers, it is not uncommon to extend PK-PD models from the healthy to the disease situation. In doing so, it is important



**Fig. 5.2** Types of models related to disease and disease progression modeling. Physiological models implement the basic mechanisms that are supposed to be affected in a disease. Drug models represent the PK and PD of the drug with emphasis on the mode of action of the drug. Disease models are modifications of physiological models to reflect disease mechanisms. Drug-disease models are extensions of disease or drug models. Disease models may be used for target identification and validation. Drug-disease models are used in support of drug indication, dosage selection, and clinical trial design. Disease progression models, often empirically based, investigate the course of a disease with and without drug treatment. Drug-disease progression models assess the time course of disease under treatment and allow differentiation between symptomatic and disease modifying effects. All models may be created de novo without referring to other models

to consider the pathophysiology of the disease, i.e., the functioning of the body under diseased conditions. As will be shown later ([Sect. 5.3](#)), feedback mechanisms may be present in healthy volunteers but absent in patients with impact on the complexity of respective models. For chronic diseases, most valuable are drug-disease models that describe disease progression either as an extension of a drug-disease model or of a disease progression model ([Fig. 5.2](#)).

In [Sect. 5.2](#) we will introduce a biophysically realistic model for the action potential (AP) and its propagation. This is done quite elaborately to illustrate the interplay between biophysics, physiology, and mathematical modeling. A more extensive description of mathematical models in physiology is provided by Keener and Sneyd [[6](#), [7](#)]. In [Sects. 5.3](#), [5.4](#), and [5.5](#) we present own modeling work that supported clinical drug development programs from different therapeutic areas. [Section 5.6, Disease Progression Modeling](#), introduces basic disease progression models based on work by Post et al. [[8](#)].

## 5.2 Central Nervous System Disorders

In 1952, Alan Hodgkin and Andrew Huxley published their seminal paper on calculations on the squid giant axon [[9](#)]. It is still a miracle how they could achieve their computationally highly demanding results using only mechanical hand calculators. In 1963, they received, together with John Eccles (1903–1997), the

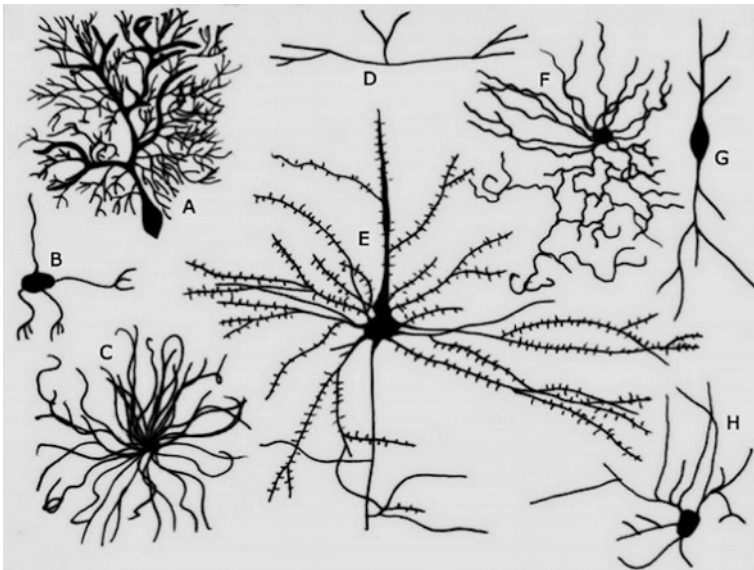
Nobel Prize in Medicine and Physiology for their ground-breaking work elucidating in a mathematical model the ionic mechanisms underlying nerve firing. This model became known as the Hodgkin-Huxley (HH) model. Their work gave also support to a new discipline, mathematical biology.

In the following, we will first present the electrophysiologic basis that underpins the HH model. This quite elaborate description will help to better understand those issues that are encountered when building biologically realistic brain models. Next, the HH model equations are derived, computationally implemented, and simulated. This will demonstrate the very nonlinear behavior of the HH model. And as a last topic in this chapter, we will describe how collections of virtual neuron can be used to mimic the biology of CNS disorders such as schizophrenia.

### 5.2.1 Physiologic Background

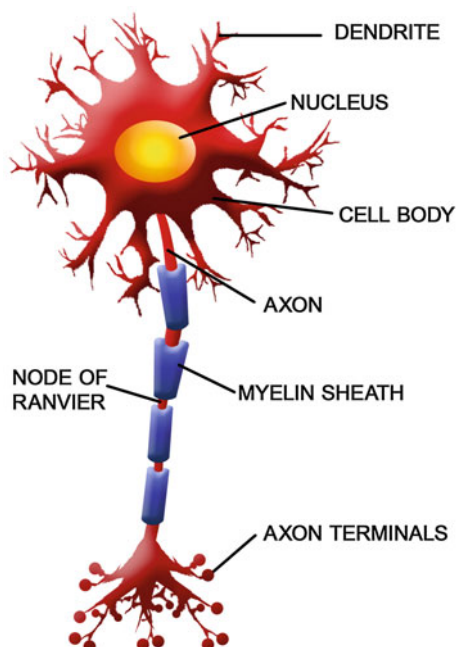
Excellent monographs are available giving a detailed overview on neuroscience topics [10–12]. Here we summarize some key features that should be known to put the modeling into context.

Neurons differ widely in size, shape, and structure (see Fig. 5.3) so that it is difficult to talk about a typical neuron. For building a mathematical model, however, we will consider a representative neuron as shown in Fig. 5.4.



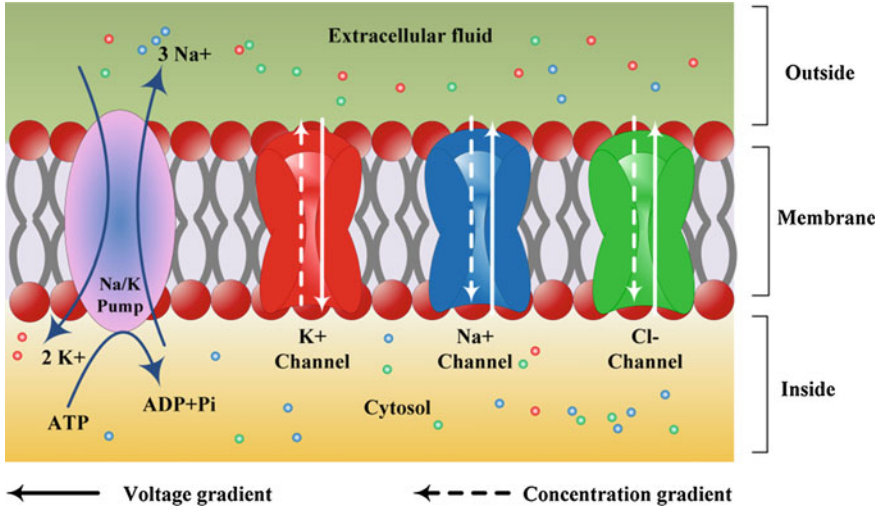
**Fig. 5.3** Selected types of neurons differing in shape and size found in both brain and spinal cord. Each also differs in function: *A* Purkinje cell, *B* Granule cell, *C* Motor neuron, *D* Tripolar neuron, *E* Pyramidal cell, *F* Chandelier cell, *G* Spindle neuron, *H* Stellate cell. (<http://boingboing.net/2012/05/17/the-beautiful-shapes-of-neuron.html>)

**Fig. 5.4** A single neuron shown in a simplified form. It consists of a cell body (soma) with the nucleus, dendrites and an axon that gives rise to axon terminals. Like all cells, the neuronal cell is enclosed by a membrane that separates the neuronal cell from the outside. The axon, besides the neuronal membrane, may be covered by a myelin sheath that consists of segments separated by short narrow intervals known as nodes of Ranvier. (from © Depositphotos/Balint Roxana #4087535)



Morphologically, the neuron consists of four main components, each with specialized functionality in signal generation and information exchange between neurons: Dendrites, cell body, axon, and axon terminals. The dendrites and cell body receive signals from other neurons. The cell body (also called soma) contains structures with organelles important for neuron vitality. One such structure is the centrally located nucleus that stores the genetic material [13]. Usually, neurons have one axon that originates from the cell body and gives rise to numerous spines and the axon terminals. Through the axon the signal is conducted to the axon terminals and further to other neurons or other organs (muscles, endocrine glands). The whole neuron is enclosed by the neuronal membrane. The composition of the membrane varies among dendrites, cell body, and axon. The axon, besides the neuron membrane, can have an additional sheath, the myelin sheath [12] that consists of segments separated by short narrow intervals known as nodes of Ranvier (Fig. 5.4).

The main task of each neuron is to ensure communication with other neurons within the CNS, i.e., receiving information at the dendrites and soma, processing it, and finally sending the information via axon and axon terminals to other neurons, or other types of cells. Due to their electrical properties, neurons can process, generate, and conduct electrical signals. The key electrical signal is the AP and the neuronal cell membrane plays the main role in processing electrical signals. *The function of neurons cannot be understood without understanding the structure and function of the membrane and its associated proteins* [13].



**Fig. 5.5** The ionic basis of the resting membrane potential is the voltage difference across the membrane. In our analysis, we consider the following three types of ions (and corresponding channels) determining the resting membrane potential: sodium (Na<sup>+</sup>), potassium (K<sup>+</sup>) and chloride (Cl<sup>-</sup>). The Na/K pump moves Na<sup>+</sup> ions from the cytosol to the extracellular fluid, and K<sup>+</sup> ions from the extracellular fluid to the cytosol. This concentrates Na<sup>+</sup> ions in the extracellular space and K<sup>+</sup> ions within the cell, a prerequisite for eliciting an AP. APs conduct signals (information) along the axons, and provide the basis for neural brain communication

The neuronal cell membrane separates the cell inside, containing intracellular fluid, cytosol, from the outside containing extracellular fluid that bathes the whole neuron (Fig. 5.5). Both the extracellular and intracellular fluids are salty and consist predominantly of water with dissolved ions. The membrane is built mainly of phospholipids and contains protein channels (also called leak channels) that are permeable to ions dissolved both in the intracellular and extracellular fluids. The channels allow ions to pass through the membrane.

Neuronal signaling depends crucially on the membrane potential,  $V$ , which is the difference between the electrical potential inside and outside the neuron (Fig. 5.5).

$$V = V_{\text{in}} - V_{\text{out}} \quad (5.1)$$

If the neuron (membrane) is at rest, i.e., no signals are processed, the membrane potential is equal to a value known as the resting potential. Several ions (Na<sup>+</sup>, K<sup>+</sup>, Cl<sup>-</sup>, and, others) contribute to the resting potential, and each contribution is characterized by a specific potential, the equilibrium potential.

In the following section, we will introduce these different types of electrical potentials and explain their meaning for neuronal functionality.

### Equilibrium Potential

The equilibrium potential results from two different forces existing across the neuronal membrane, an electrical driving force and a chemical driving force.

These forces affect all types of ions, but we will regard only  $K^+$ ,  $Na^+$ ,  $Cl^-$ , due to their dominant role in the process. Furthermore, the driving forces can be linked to each type of ions separately. The ions are present inside the cell (in the cytosol) as well as outside the cell (in the extracellular fluid) and can pass along a channel if a concentration gradient exists for the corresponding type of ions.

Let us first consider  $K^+$  ions which have a higher concentration within the cell than outside. Due to this concentration gradient,  $K^+$  ions are moved through the channel permeable to  $K^+$  to the outside of the cell (chemical driving force). The  $K^+$  ions are positively charged, and their accumulation outside the cell induces an electrical potential across the membrane which forces the  $K^+$  ions back into the cell (electrical driving force), against the concentration gradient. Equilibrium is achieved when there is no net movement of ions across the membrane. The electrical potential difference at which this happens is called the equilibrium potential of  $K^+$  ions. In a similar way,  $Na^+$  and  $Cl^-$  ions produce their own equilibrium potential.

Walter Nernst (1864–1941) worked out an equation for the membrane potential, also called the Nernst potential:

$$\left. \begin{aligned} E_{ion} &= \frac{R \cdot T}{z \cdot F} \cdot \ln \frac{[ion]_{out}}{[ion]_{in}} & \Leftrightarrow & E_{ion} = \frac{k \cdot T}{z \cdot q} \cdot \ln \frac{[ion]_{out}}{[ion]_{in}}; \\ ion &\equiv K^+, Na^+, Cl^- \dots \end{aligned} \right\} \quad (5.2)$$

where:

- $E_{ion}$   $\equiv$  Ionic equilibrium potential (in volts or mV);
- $[ion]_{out}$   $\equiv$  Extracellular concentration of that ion (in mol/m<sup>3</sup>);
- $[ion]_{in}$   $\equiv$  Intracellular concentration of that ion (in mol/m<sup>3</sup>);
- $R$   $\equiv$  Ideal gas constant, 8.314 J/(K·mol);
- $T$   $\equiv$  Temperature (in K);
- $z$   $\equiv$  Charge of the ion (dimensionless quantity);
- $F$   $\equiv$  Faraday's constant, 96485.3415 C/mol or J/(V·mol);
- $k$   $\equiv$  Boltzmann's constant,  $1.38 \cdot 10^{-23}$  J/K;
- $q$   $\equiv$  Magnitude of the electronic charge,  $1.602 \cdot 10^{-19}$  C.

The equilibrium potential for  $K^+$ ,  $Na^+$ , and  $Cl^-$  ions, calculated from the Nernst equation, are shown in Table 5.4.

The  $Na^+$  ions are more concentrated outside the cell, and the  $K^+$  ions more inside the cell due to the Na/K pump, shown in Fig. 5.5. The Na/K pump is an enzyme contained in the cell membrane. The ATP-dependent pump moves 3  $Na^+$  ions from the cytoplasm to the extracellular fluid and 2  $K^+$  ions into the opposite direction. The pumps move ions independently of the existing concentration gradient, meaning that even if the ion concentration is high the pumps will work against the prevailing ion concentration gradient. This is why the pumps need energy.

**Table 5.4** Ion concentration and membrane equilibrium potential for selected types of ions at a temperature of 20 °C (293.15 K), calculated according to the Nernst equation (for squid giant axon)

Ion	[ion] <sub>in</sub> (mM)	[ion] <sub>out</sub> (mM)	Charge(z)	<i>E</i> <sub>ion</sub> (mV)
K <sup>+</sup>	400	20	+1	−75.7
Na <sup>+</sup>	50	440	+1	54.9
Cl <sup>−</sup>	52	560	−1	−60.0

Resting Potential

The resting potential is the electrical potential across the membrane in the resting neuron, i.e., when it is neither receiving nor sending signals, but maintaining the membrane potential at a stable level. The resting potential ensures that the neuron is able to produce an AP.

If K<sup>+</sup> ions were the only ions traversing the membrane through their specific channel, then the resting potential would be equal to the potassium equilibrium potential *E*<sub>K</sub>. However, there are other ions with their own channels; all contribute to the resting potential. For multiple ions, the resting membrane potential *V* is expressed by the Goldman or constant field equation (David Goldman, 1910–1998). If we consider only the dominant ions (K<sup>+</sup>, Na<sup>+</sup>, and Cl<sup>−</sup>), the Goldman equation reads:

$$V = \frac{R \cdot T}{F} \cdot \ln \frac{P_K \cdot [K^+]_{out} + P_{Na} \cdot [Na^+]_{out} + P_{Cl} \cdot [Cl^-]_{in}}{P_K \cdot [K^+]_{in} + P_{Na} \cdot [Na^+]_{in} + P_{Cl} \cdot [Cl^-]_{out}}$$

(5.3)

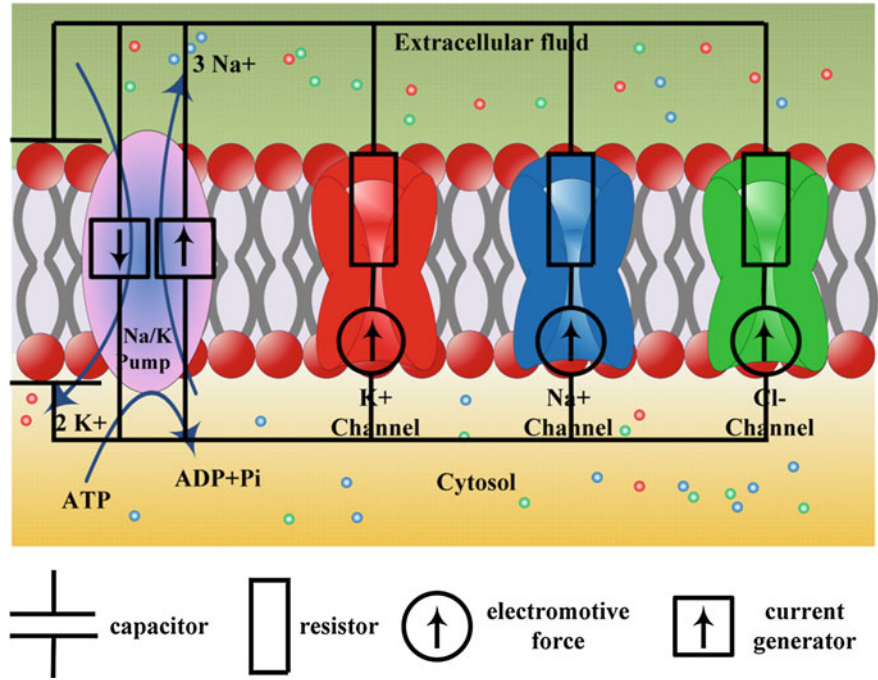
The variables *P*<sub>Na</sub>, *P*<sub>K</sub>, and *P*<sub>Cl</sub> are dimensionless permeability values, usually given as relative permeability, for example (at rest) *P*<sub>K</sub>:*P*<sub>Na</sub>:*P*<sub>Cl</sub> = 1:0.05:0.45, and refer to the potassium permeability, *P*<sub>K</sub> = 1. Thus, assuming concentrations and permeabilities as given in Table 5.5, the resting potential calculated according to Eq. (5.3) is *V* = −58.18 mV. In general, the resting membrane potential in neuronal cells is approximately between −70 and −55 mV.

Based on the resting potential (Fig. 5.5), we can construct an electrical representation for the neuronal membrane that distinguishes among the K<sup>+</sup> channel, Na<sup>+</sup> channel, and Cl<sup>−</sup> channel (Fig. 5.6). These channels are the main contributors to the resting membrane potential and other ion channels can be ignored. Additionally, the Na/K pump ensures that Na<sup>+</sup> and K<sup>+</sup> ions are moved properly to the inside and outside of the neuron. Eventually, this leads to a representation which abstracts from the concrete biological background (Fig. 5.7).

**Table 5.5** Membrane resting potential at 20 °C (293.15 K) calculated by the Goldman equation (squid giant axon)

Ion	[ion] <sub>in</sub> (mM)	[ion] <sub>out</sub> (mM)	<i>P</i> <sub>ion</sub> (dimensionless)	<i>V<sub>m</sub></i> (mV)
K <sup>+</sup>	400	20	1	−58.18
Na <sup>+</sup>	50	440	0.05	
Cl <sup>−</sup>	52	560	0.45	





**Fig. 5.6** The biological cell membrane (at rest) and its equivalent electrical circuit representation. From the *left to right*: capacitor to collect and separate electrical charges, current generators that move K<sup>+</sup> ions to the cytosol and Na<sup>+</sup> ions to the extracellular fluid (Na/K pump), resistors and electrical forces representing K<sup>+</sup>, Na<sup>+</sup>, and Cl<sup>-</sup> channels and equilibrium potentials, respectively

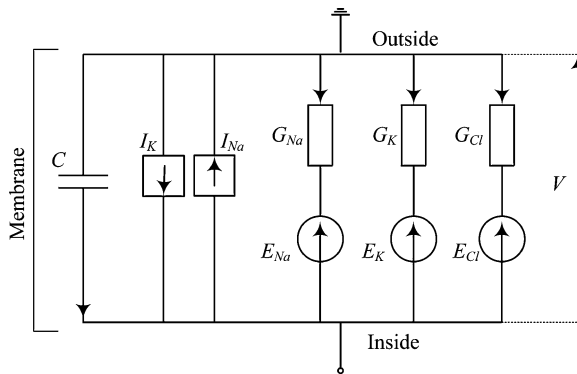
In order to calculate the equilibrium and resting potential, a simple calculator can be used that is available at MATLAB Central. In Appendix A, the reader will also find hints on how to create such tool using MATLAB's GUI.

### Action Potential

As long as the neuron is at rest the membrane potential is stable. "At rest" means that there is no stimulating signal. Now, assume an electrical signal or stimulus is delivered to the neuron, for example in form of an injected electrical current. It can be an electrical pulse caused from another neuron or from a microelectrode inserted into the cell. Then, three events could happen.

If the electrical current is negatively charged, it will make the potential in the cell more negative, thereby decreasing the membrane potential  $V$  ( $V < \text{resting potential}$ ). This change of the membrane potential is called hyperpolarization. Following hyperpolarization, the neuron returns to the resting potential. This reaction of the neuron is called "passive response".





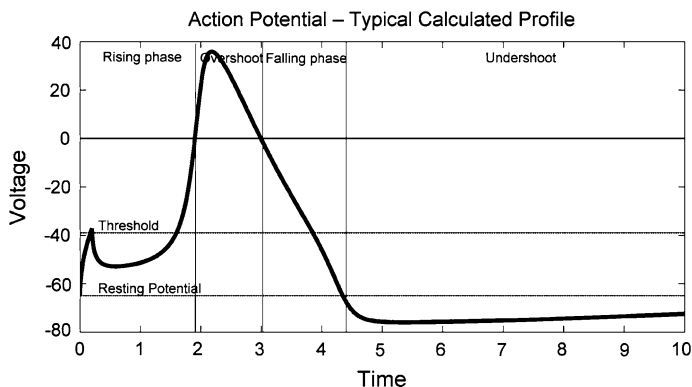
**Fig. 5.7** Electrical representation of the cell membrane at rest, derived from its biological structure and properties (see Fig. 5.6). For each type of ion ( $\text{Na}^+$ ,  $\text{K}^+$  or  $\text{Cl}^-$ ) passing through its channel, there is a branch containing an electromotive force,  $E$ , connected to a resistor with conductance  $G$  (inverse quantity of resistance). The electromotive force in each branch is equal to the corresponding equilibrium potential  $E_K$ ,  $E_{\text{Na}}$  or  $E_{\text{Cl}}$ . Additionally, there is a capacitor branch with capacitance  $C$ , as the cell membrane is able to store and separate electrical charges.  $V$  is the membrane potential that is equal to the resting potential. The Na/K pumps are present in this electrical circuit in form of the current generators  $I_{\text{Na}}$  and  $I_K$ .

If the electrical current is positive, it will increase the membrane potential ( $V > \text{resting potential}$ ) and its extent will depend on the stimulus amplitude and its duration. This is called depolarization. For small amplitudes or too short current durations, the neuron returns to the resting potential (passive response). If the stimulating current has large enough amplitude or is of sufficient duration, the depolarization reaches a *threshold* and triggers a further, steep increase in the membrane potential, called an *AP*. The typical time course of the AP is shown in Fig. 5.8. Extensive measurements have shown that the AP is accompanied by characteristic  $\text{Na}^+$  and  $\text{K}^+$  ion movements across the membrane as detailed in the legend to Fig. 5.8.

The first to measure membrane voltage during an AP were Hodgkin and Huxley in 1939 [14]. They observed the AP time course on an oscilloscope screen and recorded it. Their seminal work [9] consisted in mathematically modeling the time course of the AP in terms of dynamic ionic mechanisms. The permeability of  $\text{Na}^+$  and  $\text{K}^+$  channels was considered to be voltage-dependent (*voltage-gated channels*). Guided by the membrane potential, voltage-gated channels can be opened or closed, thereby generating the AP and propagating it along the axon.

### 5.2.2 Hodgkin and Huxley Mathematical Model

As described in the previous section, the AP is generated in a neuron by the electrical properties of the cell membrane and ion currents moving both across the membrane and along the neuron (axon). Based on these properties, Hodgkin and

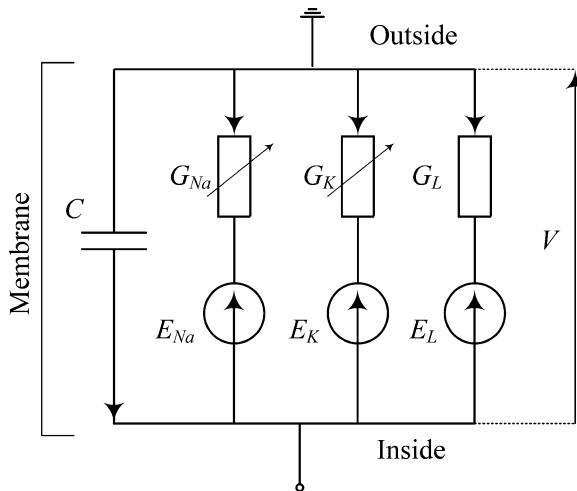


**Fig. 5.8** Typical profile of the AP in a neuron stimulated by a short input current; voltage in mV, time in ms. Mathematical analysis of the ion currents allows the following interpretation of the AP profile: 1. *Rising phase* (time: 0–1.8 ms)—due to stimulation,  $\text{Na}^+$  channels open, and a strong  $\text{Na}^+$  current enters the cell depolarizing the membrane and finally producing a voltage overshoot ( $V > 0$ ). 2. *Overshoot phase* (time: 1.8–3 ms)—the voltage is positive,  $\text{K}^+$  channels open and  $\text{K}^+$  ions leave the cell causing repolarization; the  $\text{Na}^+$  current is becoming weaker. 3. *Falling phase* (time: 3–4.4 ms)—the voltage reaches resting potential levels.  $\text{K}^+$  ions continue to flow from the inside to the outside of the cell. At the end of the falling phase, the  $\text{K}^+$  current becomes weaker. 4. *Undershoot phase* (time:  $> 4.4$  ms)—the voltage decreases below the resting potential (hyperpolarization).  $\text{Na}^+$  and, more gradually,  $\text{K}^+$  permeabilities reach their resting state values. Finally, the neuron is back at rest and ready to react to the next impulse. The given time intervals vary depending on the axon parameters, but the time profile is similar for each AP, except for the peak at the very beginning showing the impact of the stimulation current on the voltage

Huxley (HH) built a model for the AP and its propagation in a single neuron. Their fundamental idea for the model building was the same as already described for the resting potential, i.e., to use an electric circuit analog to represent the cell membrane and its properties. In this way, the ionic channels can be represented by resistors (or conductances), the electrical potential difference across the membrane by an electromotive force, and the ability to separate and store electrically charged particles (capacitance) by a capacitor.

The membrane has two main types of protein molecules functioning as channels: Potassium ( $\text{K}^+$ ) channels (permeable to potassium ions) and sodium ( $\text{Na}^+$ ) channels (permeable to sodium ions). Both channels are assumed to be voltage-gated; they can be closed (inactivated) and opened (activated) depending on the voltage prevailing in the channel vicinity. Additionally, one more channel type will be assumed, leak channels, for all other types of ion movements across the neuronal membrane. Leak channels are assumed to be non-gated channels, and take into account the natural leakage of the cell membrane to ions.

We will describe two models: A simplified model where electrical currents cross membranes only, and a more complete model where electrical currents move across membranes and along a neuron.



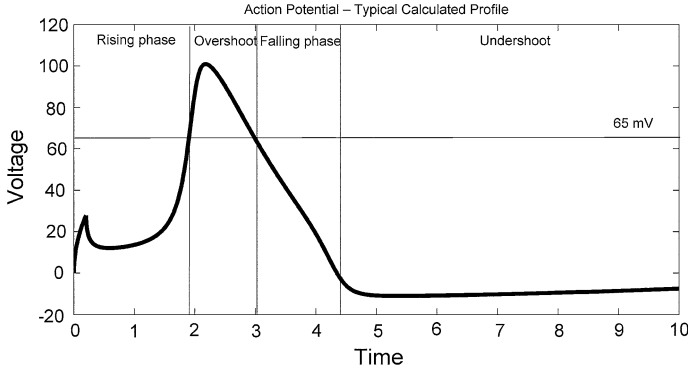
**Fig. 5.9** The electrical circuit representation of the HH model. The branches for the  $\text{Na}^+$  and  $\text{K}^+$  gated channels contain an electromotive force,  $E$ , connected to a resistor with conductance  $G$  (inverse quantity of resistance). The important point is that the conductance in both cases is not a constant value but depends on the membrane potential  $V$  indicated with an ‘extra’ arrow crossing the resistor. The third branch represents the leak channel and is similar to the others but the conductance is constant, mimicking the electrical circuit of the membrane at rest. Additionally, there is one branch with a capacitor (and capacitance  $C$ ) as the cell membrane is able to store and separate electrical charges. This ohmic model is a fundamental construct in the HH approach for building a mathematical model describing a single neuron

### Simplified Model

For the simplified model, the electrical circuit representing the physiological neuron and its biological behavior is shown in Fig. 5.9. Obviously, all  $\text{K}^+$  channels are represented by one branch; similarly all  $\text{Na}^+$  channels. As the  $\text{K}^+$  and  $\text{Na}^+$  channels are voltage-gated, the electrical conductance of the corresponding branches is a direct function of the number of open (closed) channels, and an indirect function of the membrane potential. The leak branch represents all leak channels (non-gated channels) and its conductance is constant. The driving force for the ionic currents is the membrane potential difference between the inside and outside of the cell (in fact, between the membrane surfaces on both sides).

In agreement with Hodgkin and Huxley [9], we use as reference for the voltage the resting potential of the squid neuron (equal to  $-65$  mV). Values of the membrane potential as well as the equilibrium potential are to be understood as differences from this resting potential. In order to implement this convention we have simply to move the baseline from  $-65$  mV to 0; in other words, any potential value must be increased by 65 ( $E_k + 65 \rightarrow E_k$ ,  $E_{\text{Na}} + 65 \rightarrow E_{\text{Na}}$ ,  $V + 65 \rightarrow V$ ). As an example, the typical AP, described in the previous sections and depicted in Fig. 5.8, would have the values shown in Fig. 5.10.

The following ODE system was suggested by Hodgkin and Huxley [9].



**Fig. 5.10** Typical calculated profile with reference to the new coordinate system (65 mV  $\rightarrow$  0 V), time in ms, voltage in mV

$$\left. \begin{aligned}
 C \cdot \frac{dV}{dt} &= -\overbrace{g_K \cdot n^4}^{G_K} \cdot (V - E_K) - \overbrace{g_{Na} \cdot h \cdot m^3}^{G_{Na}} \cdot (V - E_{Na}) - \overbrace{g_L}^{G_L} \cdot (V - E_L) + I_{inj} \\
 \frac{dn}{dt} &= \alpha_n - (\alpha_n + \beta_n) \cdot n \\
 \frac{dm}{dt} &= \alpha_m - (\alpha_m + \beta_m) \cdot m \\
 \frac{dh}{dt} &= \alpha_h - (\alpha_h + \beta_h) \cdot h
 \end{aligned} \right\} \quad (5.4)$$

The first equation in (5.4) is based on Kirchhoff's (Gustav Kirchhoff, 1824–1887) first law (junction rule) [15, 16] stating that current across the membrane is preserved. Thus, the injected current  $I_{inj}$  is the sum of ionic and capacitance currents. As expressed in Eq. (5.4), conductivities  $G_{Na}$  and  $G_K$  are proportional to maximal conductances  $g_{Na}$  and  $g_K$ , and entail time-dependent gating functions,  $m$ ,  $n$ , and  $h$  [17].  $G_L$  ( $= g_L$ ) stands for the conductance of the leak channel. The values  $g_{Na}$ ,  $g_K$ , and  $g_L$  were obtained from the experimental data.

Initial values for (5.4) are

$$V(0) = 0, \quad n(0) = n_\infty, \quad m(0) = m_\infty, \quad h(0) = h_\infty \quad (5.5)$$

$n_\infty$ ,  $m_\infty$ , and  $h_\infty$  are the values of  $n$ ,  $m$ , and  $h$  at steady state.

The parameters in functions  $n$ ,  $m$ , and  $h$ , i.e.,  $\alpha_{(\cdot)}$  and  $\beta_{(\cdot)}$ , depend on  $V$ , as in the following equation:

$$\left. \begin{aligned} \alpha_n &= \frac{0.01 \cdot (10 - V)}{e^{\frac{10-V}{10}} - 1} \cdot \phi; & \beta_n &= 0.125 \cdot e^{-\frac{V}{80}} \cdot \phi \\ \alpha_m &= \frac{0.1 \cdot (25 - V)}{e^{\frac{25-V}{10}} - 1} \cdot \phi; & \beta_m &= 4 \cdot e^{-\frac{V}{18}} \cdot \phi \\ \alpha_h &= 0.07 \cdot e^{-\frac{V}{20}} \cdot \phi; & \beta_h &= \frac{1}{e^{\frac{30-V}{10}} + 1} \cdot \phi \end{aligned} \right\}; \quad \phi = \phi(T) \quad (5.6)$$

The formulas in Eq. (5.6) were experimentally found by Hodgkin and Huxley [9], and depend on temperature  $T$  according to:

$$\phi = 3^{\frac{T-6.3}{10}} \quad (5.7)$$

Thus, the temperature factor  $\phi$  in (5.6) cancels out for temperature  $T = 6.3$  °C (i.e., under squid conditions).

There are many publications presenting the numerical analysis as well as computational methods including MATLAB implementations of the HH model (5.4). The reader is recommended to look, for example, at [18] and [19] where also various examples and exercises are available.

### The Complete Model: Cable Equation

The complete model describes the propagation of an AP along the axon. This requires consideration of both the electrical current across the membrane and along the neuron (axon). Thus, the electrical potential between neuron inside and outside traveling along the axon depends on space ( $x$ ) and time ( $t$ ), indicated as  $V(x, t)$ . Based on these considerations Hodgkin and Huxley extended the basic model (Eq. 5.4) and introduced the cable, or telegraph, equation. This is motivated by Fig. 5.11.

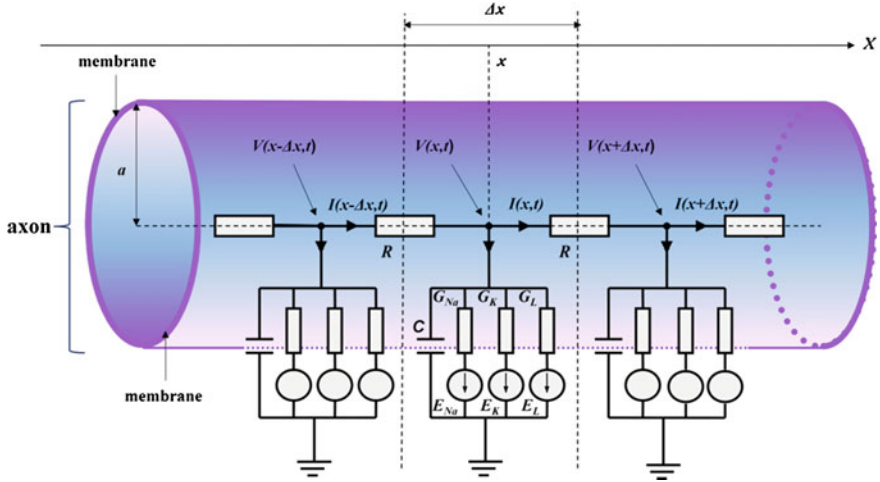
Each compartment of length  $\Delta x$  in Fig. 5.11 is represented by the electrical circuit with common parameters, as in the HH basic model, and expressed as  $R \cdot \Delta x$ ,  $G \cdot \Delta x$ ,  $C \cdot \Delta x$ , where  $R$  stands for resistance per unit length,  $G$  for conductance per unit length, and  $C$  for capacitance per unit length. Assuming that  $V(x, t)$ , and  $I(x, t)$  indicate membrane voltage and internal longitudinal ion current, respectively, the following equation can be derived from Kirchhoff's second law (loop rule) [15, 16]:

$$V(x + \Delta x, t) - V(x, t) = -R \cdot \Delta x \cdot I \quad (5.8)$$

Dividing both sides by  $\Delta x$ , and approaching  $\Delta x$  toward 0, we can write (5.8) as:

$$\left. \frac{V(x + \Delta x, t) - V(x, t)}{\Delta x} = -R \cdot I \right\} \Rightarrow \frac{\partial V}{\partial x} = -R \cdot I \quad (5.9)$$

$\Delta x \rightarrow 0$



**Fig. 5.11** Axon as cable conductor. A neuron's axon (or dendrite) can be considered as a long cable of radius  $a$ . Our approach assumes that the axon can be cut into a number of segments, also called compartments, each one of length  $\Delta x$ , and specific resistance  $R$ . The cell membrane in each compartment can be represented by the electrical circuit in the same way as it is shown in the HH basic model (Fig. 5.9), with electromotive forces  $E_{Na}$ ,  $E_K$ ,  $E_L$ , conductances  $G_{Na}$ ,  $G_K$ ,  $G_L$ , and capacitance  $C$

According to Kirchhoff's first law (junction rule) we can also derive another equation for the ionic membrane current. As illustrated in Fig. 5.11 we can write:

$$I(x + \Delta x, t) - I(x, t) = -G_K \cdot \Delta x \cdot [V(x, t) - E_K] - G_{Na} \cdot \Delta x \cdot [V(x, t) - E_{Na}] - G_L \cdot \Delta x \cdot [V(x, t) - E_L] - C \cdot \Delta x \cdot \frac{\partial V(x, t)}{\partial t} \quad (5.10)$$

After dividing of both sides by  $\Delta x$ :

$$\frac{I(x + \Delta x, t) - I(x, t)}{\Delta x} = -G_K \cdot (V - E_K) - G_{Na} \cdot (V - E_{Na}) - G_L \cdot (V - E_L) - C \cdot \frac{\partial V}{\partial t} \quad (5.11)$$

Again, if  $\Delta x$  approaches 0, then

$$\frac{\partial I}{\partial x} = -G_K \cdot (V - E_K) - G_{Na} \cdot (V - E_{Na}) - G_L \cdot (V - E_L) - C \cdot \frac{\partial V}{\partial t} \quad (5.12)$$

Regarding (5.9) and (5.12), the derivatives of  $V$  and  $I$  can be expressed as:

$$\left. \begin{aligned} \frac{\partial V}{\partial x} &= -R \cdot I \quad \Rightarrow \quad \frac{1}{R} \cdot \frac{\partial^2 V}{\partial x^2} = -\frac{\partial I}{\partial x} \\ \frac{\partial I}{\partial x} &= -G_K \cdot (V - E_K) - G_{Na} \cdot (V - E_{Na}) - G_L \cdot (V - E_L) - C \cdot \frac{\partial V}{\partial t} \end{aligned} \right\} \quad (5.13)$$

and after eliminating the partial derivative of  $I$  with respect to  $x$ , we arrive at the following form of the cable equation:

$$\frac{1}{R} \cdot \frac{\partial^2 V}{\partial x^2} = C \cdot \frac{\partial V}{\partial t} + G_K \cdot (V - E_K) + G_{Na} \cdot (V - E_{Na}) + G_L \cdot (V - E_L) \quad (5.14)$$

The left-hand side of this equation expresses the spatial change of the current in the cytoplasm, and the right-hand side the ionic current across the cell membrane per unit length of the cable (axon).

By introducing gating functions  $n$ ,  $m$ ,  $h$  from (5.4) into (5.14) and rewriting the membrane current as current per area of membrane, instead per length, we obtain the original HH equations:

$$\left. \begin{aligned} \frac{a}{2 \cdot R} \cdot \frac{\partial^2 V}{\partial x^2} &= C \cdot \frac{\partial V}{\partial t} + g_K \cdot n^4 \cdot (V - E_K) + g_{Na} \cdot h \cdot m^3 \cdot (V - E_{Na}) \\ &\quad + g_L \cdot (V - E_L) \\ \frac{\partial n}{\partial t} &= \alpha_n - (\alpha_n + \beta_n) \cdot n \\ \frac{\partial m}{\partial t} &= \alpha_m - (\alpha_m + \beta_m) \cdot m \\ \frac{\partial h}{\partial t} &= \alpha_h - (\alpha_h + \beta_h) \cdot h \end{aligned} \right\} \quad (5.15)$$

The parameters  $\alpha_n$ ,  $\beta_n$ ,  $\alpha_m$ ,  $\beta_m$ ,  $\alpha_h$  and  $\beta_h$  have the same meaning as in the basic model (5.4) and  $a$  is the diameter of the neuron.

It is important to note that parameters  $C$  and  $R$  in (5.15), other than in (5.14), are measured correspondingly as capacitance per area unit, and resistance times length unit (we recommend an exercise at the end of this chapter).

If time,  $t$ , and distance,  $x$ , domains are set to:

$$\left. \begin{aligned} t &\in [0; t_{\max}] \\ x &\in [0; L] \end{aligned} \right\} \quad (5.16)$$

then the initial conditions are

$$\left. \begin{aligned} V(x, t) &= 0 \\ n(x, t) &= n_{\infty} \\ m(x, t) &= m_{\infty} \\ h(x, t) &= h_{\infty} \end{aligned} \right\} ; \quad t = 0, x \in [0; L] \quad (5.17)$$

and boundary conditions are given as

**Table 5.6** Original parameter values of the Hodgkin-Huxley model for the squid giant axon

Parameter	Value	Units
$E_K$	-12	mV
$E_{Na}$	150	mV
$E_L$	10.598	mV
$V_0$	0	mV
$n_0$	0.3177	
$m_0$	0.0530	
$h_0$	0.5960	
$g_K$	36	mS/cm <sup>2</sup>
$g_{Na}$	120	mS/cm <sup>2</sup>
$g_L$	0.3	mS/cm <sup>2</sup>
$C$	1	uF/cm <sup>2</sup>
$R$	34.5	$\Omega \cdot \text{cm}$
$a$	0.0238	cm

$$\left. \begin{aligned} \frac{\partial V}{\partial x}(x, t) &= -\frac{R}{2 \cdot \pi \cdot a^2} \cdot I_{\text{inj}} ; & t \in [0 ; t_{\max}] , \ x = 0 \\ V(x, t) &= 0 ; & t \in [0 ; t_{\max}] , \ x = L \end{aligned} \right\} \quad (5.18)$$

Usually, the stimulus  $I_{\text{inj}}$  is an impulse of amplitude  $I_{\text{injA}}$  and duration  $t_d$  that can be expressed by the Heaviside function:

$$I_{\text{inj}} = I_{\text{injA}} \cdot [H(t) - H(t - t_d)] \quad (5.19)$$

The original parameter values of the complete Hodgkin-Huxley model are shown in Table 5.6, based on [9] and [20].

There are a number of numerical methods to solve the cable equation [21]. We can classify them into four groups: 1 Indirect methods; 2 Finite elements methods; 3 Finite difference procedures [22] (Forward-Euler, Backward-Euler, Crank–Nicholson methods); and 4 Compartmental procedures [22, 23].

One of the well-known indirect numerical methods is the traveling wave method which introduces a new variable,  $\xi$ , defined as follows:

$$\xi = x \pm \theta \cdot t \quad \Rightarrow \quad V(x, t) = V(x \pm \theta \cdot t) \quad (5.20)$$

The parameter  $\theta$  is interpreted as the velocity of an impulse (conduction velocity) traveling through the axon. This method allows transforming the PDE system (5.15) into an ODE system:



$$\left. \begin{aligned} \frac{\partial^2 V}{\partial x^2} &= \frac{\partial}{\partial x} \left( \frac{dV}{d\xi} \cdot \frac{\partial \xi}{\partial x} \right) = \frac{d^2 V}{d\xi^2} \\ \frac{\partial V}{\partial t} &= \frac{dV}{d\xi} \cdot \frac{\partial \xi}{\partial t} = \frac{dV}{d\xi} \cdot (\pm\theta) \\ \frac{\partial n}{\partial t} &= \dots = \frac{dn}{d\xi} \cdot (\pm\theta) \\ \frac{\partial m}{\partial t} &= \dots = \frac{dm}{d\xi} \cdot (\pm\theta) \\ \frac{\partial h}{\partial t} &= \dots = \frac{dh}{dt} \cdot (\pm\theta) \end{aligned} \right\} \quad (5.21)$$

finally leading to the following ODE system:

$$\left. \begin{aligned} \frac{a}{2R} \cdot \frac{dU}{d\xi} &= C \cdot U \cdot (\pm\theta) + g_K \cdot n^4 \cdot (V - E_K) + g_{Na} \cdot h \cdot m^3 \cdot (V - E_K) \\ &\quad + g_L \cdot (V - E_L) - I_{inj} \\ \frac{dV}{d\xi} &= U \\ \frac{dn}{d\xi} \cdot (\pm\theta) &= \alpha_n - (\alpha_n + \beta_n) \cdot n \\ \frac{dm}{d\xi} \cdot (\pm\theta) &= \alpha_m - (\alpha_m + \beta_m) \cdot m \\ \frac{dh}{dt} \cdot (\pm\theta) &= \alpha_h - (\alpha_h + \beta_h) \cdot h \end{aligned} \right\} \quad (5.22)$$

The price to be paid for this simplification is that one additional equation is to be solved. Furthermore, there is an additional parameter,  $\theta$ , and its value is to be found experimentally. This method was used by Hodgkin and Huxley [9] and solved with a trial and error procedure regarding the velocity  $\theta$ . Additional proposals for solution and discussions concerning Eq. (5.22) can be found in [17, 24] and [25].

### Computational Implementation

A MATLAB implementation of the compartmental method to solve the cable equation is shown in Listing 5.1. The program **hoghux.m** launches a simulation designed for solving (5.15). It illustrates the activity of a nerve cell expressed as a propagated AP, and visualizes the neuron responses to several stimulations (electrical currents injected into a neuron), under several specific circumstances, described by neuron- (axon-) specific parameters, such as capacitance, conductance, temperature, etc. The implementation can simulate various stimuli scenarios and enable the simulation results to be assessed.

**Listing 5.1** Program **hoghux.m**

```

function hoghux()
%HOGHUX Simulate the action potential in a single neuron.
% HOGHUX() simulates a single neuron under deterministic stimulation
% using the full Hodgkin-Huxley model (in 2 dimensional domain,
% i.e. time and distance). The model is described by a system of
% partial differential equations. To solve them, the compartmental
% method for second derivatives is applied.

% get axon parameters
p = axonparams;

% Initial values to dependent variables
i = 1:p.lx;
V0 = 0;      y0((1-1)*(p.lx) + i) = V0; % membrane potential
n0 = 0.3177; y0((2-1)*(p.lx) + i) = n0; % activation in K channel
m0 = 0.0529; y0((3-1)*(p.lx) + i) = m0; % activation in Na channel
h0 = 0.5961; y0((4-1)*(p.lx) + i) = h0; % inactivation in Na channel

% specifying option: use the default odeplot function for control
% purpose, to see progress of the running process. This control
% graph shows the time profile of membrane potential V(t,x) at x=0
absTolerance = ones(1, 4*p.lx)*1e-6;
options = odeset('Events', @fevents, 'OutputFcn', @odeplot, ...
    'OutputSel', 1, 'RelTol', 1e-6, 'AbsTol', absTolerance);
p.tmax = 5;      % simulation time [ms]
%p.tmax = 60;    % simulation time, long simuli [ms]
tspan = [0 p.tmax]; % integration interval [ms]
% prepare data for graphs
timePoints = [];
VnmhPoints = [];
Istim = [];
ie = -1;          % any non-empty value to avoid 't=0' as event
while ~isempty(ie)
    [t, y, te, ye, ie] = ode15s(@derivatives, tspan, y0, options, p);
    timePoints = [timePoints; t];      %#ok<AGROW>
    VnmhPoints = [VnmhPoints; y];      %#ok<AGROW>
    Istim = [Istim; stimulus(t,p)];    %#ok<AGROW>
    tspan = [te(end) p.tmax];
    y0 = ye(end,:);
    % check if an event is detected
    ie = ie(te == te(end)); % eliminate already handled events
    for iEvent = ie
        switch iEvent
            case 1 % event -> stop stimulation
                p.Iinj = 0;
            case 2 % event -> end of observation
                ie = [];
            otherwise
                % no action needed - placeholder for other events
        end;
    end;
end
end

```

```

end
% extract values for creating graphs
V(:,i) = VnmhPoints(:,(1-1)*(p.lx) + i);
n(:,i) = VnmhPoints(:,(2-1)*(p.lx) + i);
m(:,i) = VnmhPoints(:,(3-1)*(p.lx) + i);
h(:,i) = VnmhPoints(:,(4-1)*(p.lx) + i);

% Save state of action potential at selected locations x in axon
xLocations = 1:round(p.lx/20):p.lx-1; % points along axon
for ix = xLocations
    ind = find(ix==xLocations);
    figSave.stateV(ind) = figure;
    set(figSave.stateV(ind), 'Position',[100+ix 50 560 420]);
    subplot(2, 1, 1);
    plot(timePoints, V(:,ix), 'LineWidth', 2);
    xlabel('Time [ms]');
    ylabel('V [mV]');
    x = num2str((ix - 1)*p.dx);
    title(['x=', num2str(x), 'cm'])
    subplot(2,1,2);
    plot(timePoints,[n(:,ix) m(:,ix) h(:,ix)], 'LineWidth', 2);
    xlabel('Time [ms]');
    ylabel('Probability of n,m,h');
    legend('n','m','h');
end
% Save state of action potential at all locations x in axon
figSave.allstateV = figure;
set(figSave.allstateV, 'Position',[100+ix+1 50 560 600]);
for ix = xLocations
    subplot(7, 1, 1:3);
    plot(timePoints, V(:,ix), 'b', 'LineWidth', 2);
    xlabel('Time [ms]');
    ylabel('V [mV]');
    hold on;
    subplot(7,1, 4:6);
    plot(timePoints,[n(:,ix) m(:,ix) h(:,ix)], 'LineWidth', 2);
    xlabel('Time [ms]');
    ylabel('Probability of n,m,h');
    legend('n','m','h');
    hold on;
end
subplot(7, 1, 7);
plot(timePoints, Istim, 'b', 'LineWidth', 2);
xlabel('Time [ms]');
ylabel('I [\mu A]');
ylim([0 max(Istim)+1]);

% Plot membrane potential V(t,x) as surface over time and distance
figSave.solutionV = figure; set(axes, 'FontSize',15)
set(figSave.solutionV, 'Position',[190+p.lx 50 560 420]);
surf(p.x,timePoints,V, 'FaceColor','interp',...
'EdgeColor','none',...
'FaceLighting','phong');
colorbar
axis tight
title('Cable Equation: Solution V(x,t)');
xlabel('Distance [cm]');
ylabel('Time [ms]');

```

```

xlabel('Voltage [mV]');
% print all figures in TIFF format
printFig(figSave)

end

function p = axonparams
%AXONPARAMS setup axon parameters
% P = AXONPARAMS creates a structure |P| with the physiological
% parameters of the neuron/axon

% Axon parameters
p.a = 0.0238; % axon radius [cm]
p.L = 6; % axon length [cm]
p.dx = 0.05; % distance integration step [cm]
p.x = 0:p.dx:p.L; % steps along axon [cm]
p.lx = length(p.x); % number of steps within axon
p.R = 34.5; % resistance (axoplasm) [ohm*cm]
p.C = 1; % membrane capacitance [uF/cm^2]
p.gK = 36; % max. potassium (K) conductance [mS/cm^2]
p.gNa = 120; % max. sodium (Na) conductance [mS/cm^2]
p.gL = 0.3; % leak (L) conductance [mS/cm^2]
p.EK = -12; % K equilibrium potential [mV]
p.ENa = 115; % N equilibrium potential [mV]
p.EL = 10.598; % L equilibrium potential [mV]
p.Iinj = 10; % injected stimulation current [uA]
p.td = 0.2; % duration of injected current [ms]
% p.td = 20; % duration for long stimulus [ms]
p.T = 18.5; % temperature [Celsius]
% p.T = 6.3; % temperature [Celsius]
p.phi = tempfactor(p.T); % temperature factor

end

function dydt = derivatives(t, y, p)
%DERIVATIVES Compute the right-hand side of the ODE.
% DYDT = ...

i=1:p.lx;
V = y((1-1)*(p.lx) + i);
n = y((2-1)*(p.lx) + i);
m = y((3-1)*(p.lx) + i);
h = y((4-1)*(p.lx) + i);
% boundary condition at the begin of axon (x = 0)
Istim = stimulus(t, p);
Im1 = (Istim/(2*pi*p.a) - p.a/p.R/p.dx*(V(1)-V(2))*1000)/p.dx;
dVdt(1) = (1/p.C)*(Im1 - p.gK*n(1)^4*(V(1)-p.EK) ...
- p.gNa*m(1)^3*h(1)*(V(1)-p.ENa) - p.gL*(V(1)-p.EL));
dndt(1) = nderiv(n(1), V(1), p.phi);
dmdt(1) = mderiv(m(1), V(1), p.phi);
dhdt(1) = hderiv(h(1), V(1), p.phi);
% computation between the begin and end of axon (0 < x < L)
i=2:p.lx-1;
Imi = p.a./(p.R*2).*(V(i-1)-2*V(i)+V(i+1))*1000/(p.dx.^2);
dVdt(i) = (1/p.C)*(Imi - p.gK*n(i).^4*(V(i)-p.EK) ...
- p.gNa.*m(i).^3.*h(i).*(V(i)-p.ENa) - p.gL.*(V(i)-p.EL));

```

```

dndt(i) = nderiv(n(i), V(i), p.phi);
dmdt(i) = mderiv(m(i), V(i), p.phi);
dhdt(i) = hderiv(h(i), V(i), p.phi);
% boundary condition at the end of axon (x = L)
Iml = p.a/(p.R*2)*(V(p.lx-1) - 2*V(p.lx))*1000/(p.dx^2);
dVdt(p.lx) = (1/p.C)*(Iml - p.gK*n(p.lx)^4*(V(p.lx)-p.EK)...
    - p.gNa*m(p.lx)^3*h(p.lx)*(V(p.lx)-p.ENa) ...
    - p.gL*(V(p.lx)-p.EL));
dndt(p.lx) = nderiv(n(p.lx), V(p.lx), p.phi);
dmdt(p.lx) = mderiv(m(p.lx), V(p.lx), p.phi);
dhdt(p.lx) = hderiv(h(p.lx), V(p.lx), p.phi);

dydt = [dVdt(1)           % (x = 0)
        dVdt(i)'          % (0 < x < L)
        dVdt(p.lx)        % (x = L)
        dndt(1)           % (x = 0)
        dndt(i)'          % (0 < x < L)
        dndt(p.lx)        % (x = L)
        dmdt(1)           % (x = 0)
        dmdt(i)'          % (0 < x < L)
        dmdt(p.lx)        % (x = L)
        dhdt(1)           % (x = 0)
        dhdt(i)'          % (0 < x < L)
        dhdt(p.lx)];      % (x = L)
end

function dndt = nderiv(n, V, phi)
%NDERIV Time derivative of |N|.
%   DNDT = NDERIV(N, V, PHI) takes 2 arguments: |V| - membrane
%   potential, |N| - concentration of proteins activating (opening)
%   K channel and |PHI| - temperature factor. It returns the
%   derivative value of |N|.

% Calculate values of the forward and backward rates: alpha, beta
alpha = (0.01.*(-V + 10))./(exp((-V + 10)/10) - 1)*phi;
beta = 0.125*exp(-V/80)*phi;
dndt = alpha - (alpha + beta).*n;
end

function dmdt = mderiv(m, V, phi)
%DMDT Time derivative of |M|.
%   DMDT = MDERIV(M, V, PHI) takes 2 arguments: |V| - membrane
%   potential, |M| - concentration of proteins activating (opening)
%   Na channel and |PHI| - temperature factor. It returns the
%   derivative value of |M|.

% Calculate values of the forward and backward rates: alpha, beta
alpha = 0.1.*(-V + 25)./(exp((-V + 25)/10) - 1)*phi;
beta = 4*exp(-V/18)*phi;
dmdt = alpha - (alpha + beta).*m;
end

```

```

function dhdt = hderiv(h, V, phi)
%HDERIV Time derivative of |H|.
%   DHDT = HDERIV(H, V, PHI) takes 2 arguments: |V| - membrane
%   potential, |H| - concentration of proteins inactivating (closing)
%   Na channel and |PHI| - temperature factor. It returns the
%   derivative value of |H|.

% Calculate values of the forward and backward rates: alpha, beta
alpha = 0.07*exp(-V/20)*phi;
beta = 1./(exp((-V + 30)/10) + 1)*phi;
dhdt = alpha - (alpha + beta).*h;

end

function phi = tempfactor(T)
%TEMPFACTOR Compute the temperature factor
%   PHI = TEMPFACTOR(T) calculates the temperature factor |PHI| to
%   be considered in HH model if the temperature |T| is different
%   than 6.3 Celsius.

phi = 3^((T-6.3)/10);

end

function [value, isterminal, direction] = fevents(t, ~, p)
%FEVENTS Specify events for the ODE model.
%   [VALUE,ISTERMINAL,DIRECTION] = ...

% 1 Event: Locate time to stop current injection (stimuli)
value(1,1) = t - p.td; % detect p.td (time to stop current)
isterminal(1,1) = 1; % interrupt integration
direction(1,1) = 1; % positive direction only
% 2 Event: Locate time to stop observation/integration
value(2,1) = t - p.tmax; % detected p.tmax (end of observation)
isterminal(2,1) = 1; % interrupt integration
direction(2,1) = 1; % positive direction only

end

function I = stimulus(t, p)
%STIMULUS Stimulus initialization
%   I = STIMULUS(T,P) calculates the stimulus at time |T|, and with
%   parameters |P|; here, |P.Iinj| - is the current amplitude

I = p.Iinj*(ones(length(t),1));

end

function printFig(fig)
%PRINTFIG Print figures to files
%   PRINTFIG(FIG) saves in TIFF format graphs created during
%   simulation in external files. |FIG| is a structure containing
%   names of all figures to be saved/printed

for f = fig.stateV
    x = find(f == fig.stateV);

```

```

    fileName = ['stateVx_', num2str(x), '.tif'];
    print(f, '-r600', '-dtiff', fileName);
end
print(fig.allstateV, '-r600', '-dtiff', 'allstateV.tif');
print(fig.solutionV, '-r600', '-dtiff', 'solutionVxt.tif');

end

```

The compartmental method solves the PDE form of the HH complete model by utilizing an ODE solver and regarding spatial relations between compartments. This method gives sufficient accuracy in the calculation of the AP propagating along the axon, assuming that the local geometry does not change along the axon (constant radius of the axon).

If the geometry along the axon varies, the PDE solving function `pdepe` described in [Chap. 3, \*Differential Equations in MATLAB\*](#), can be used. It requires a special form of the PDE, as indicated below:

$$\left. \begin{aligned} \mathbf{c} \left( x, t, \mathbf{u}, \frac{\partial \mathbf{u}}{\partial x} \right) \circ \frac{\partial \mathbf{u}}{\partial t} &= x^{-\mu} \cdot \frac{\partial}{\partial x} \left[ x^{\mu} \cdot \mathbf{f} \left( x, t, \mathbf{u}, \frac{\partial \mathbf{u}}{\partial x} \right) \right] + \mathbf{s} \left( x, t, \mathbf{u}, \frac{\partial \mathbf{u}}{\partial x} \right); \\ t &\in [t_0; t_f], \quad x \in [a; b] \end{aligned} \right\} \quad (5.23)$$

The initial conditions must have the form

$$\mathbf{u}(x, t_0) = \mathbf{u}_0(x) \quad (5.24)$$

and the boundary conditions must be rearranged to

$$\left. \begin{aligned} \mathbf{p}(x, t, \mathbf{u}) + \mathbf{q}(x, t, \mathbf{u}) \circ \mathbf{f} \left( x, t, \mathbf{u}, \frac{\partial \mathbf{u}}{\partial x} \right) &= \mathbf{0} \\ t &\in [t_0; t_f], \quad x = a \quad \text{or} \quad x = b \end{aligned} \right\} \quad (5.25)$$

Our task is now to restructure the AP model given in [\(5.15–5.19\)](#) to the form [\(5.23–5.25\)](#). We notice that [\(5.15\)](#) is equivalent to

$$\left. \begin{aligned} C \cdot \frac{\partial V}{\partial t} &= \frac{\partial}{\partial x} \left( \frac{a}{2 \cdot R} \cdot \frac{\partial V}{\partial x} \right) - g_K \cdot n^4 \cdot (V - E_K) - g_{Na} \cdot h \cdot m^3 \cdot (V - E_{Na}) \\ &\quad - g_L \cdot (V - E_L) \\ \frac{\partial n}{\partial t} &= \alpha_n - (\alpha_n + \beta_n) \cdot n \\ \frac{\partial m}{\partial t} &= \alpha_m - (\alpha_m + \beta_m) \cdot m \\ \frac{\partial h}{\partial t} &= \alpha_h - (\alpha_h + \beta_h) \cdot h \end{aligned} \right\} \quad (5.26)$$

which can also be modified to the following vector form:

$$\left\{ \begin{array}{l} \left[ \begin{array}{c} C \\ 1 \\ 1 \\ 1 \end{array} \right] \circ \left[ \begin{array}{c} \frac{\partial V}{\partial t} \\ \frac{\partial n}{\partial t} \\ \frac{\partial m}{\partial t} \\ \frac{\partial h}{\partial t} \end{array} \right] = \frac{\partial}{\partial x} \left[ \begin{array}{c} \frac{a}{2 \cdot R} \cdot \frac{\partial V}{\partial x} \\ 0 \\ 0 \\ 0 \end{array} \right] + \left[ \begin{array}{c} - \sum_{i \equiv K, Na, L} G_i \cdot (V - E_i) \\ \alpha_n - (\alpha_n + \beta_n) \cdot n \\ \alpha_m - (\alpha_m + \beta_m) \cdot m \\ \alpha_h - (\alpha_h + \beta_h) \cdot h \end{array} \right] ; \\ G_K \equiv g_K \cdot n^4, \quad G_{Na} \equiv g_{Na} \cdot h \cdot m^3, \quad g_K \equiv g_L \end{array} \right\} \quad (5.27)$$

Comparing (5.23) to (5.27), we observe the following relations:

$$\begin{array}{l} \mathbf{c}(\cdot) \equiv \left[ \begin{array}{c} C \\ 1 \\ 1 \\ 1 \end{array} \right], \quad \mathbf{f}(\cdot) \equiv \left[ \begin{array}{c} \frac{a}{2 \cdot R} \cdot \frac{\partial V}{\partial x} \\ 0 \\ 0 \\ 0 \end{array} \right], \quad \mathbf{s}(\cdot) \equiv \left[ \begin{array}{c} - \sum_{i \equiv K, Na, L} G_i \cdot (V - E_i) \\ \alpha_n - (\alpha_n + \beta_n) \cdot n \\ \alpha_m - (\alpha_m + \beta_m) \cdot m \\ \alpha_h - (\alpha_h + \beta_h) \cdot h \end{array} \right], \\ \mu = 0 \end{array} \quad (5.28)$$

Comparing (5.17) to (5.24), we can directly specify the initial conditions as:

$$\mathbf{u}(x, t_0) = \left[ \begin{array}{c} V(x, t_0) \\ n(x, t_0) \\ m(x, t_0) \\ h(x, t_0) \end{array} \right] \equiv \mathbf{u}_0(x) = \left[ \begin{array}{c} 0 \\ n_\infty \\ m_\infty \\ h_\infty \end{array} \right]; \quad t = 0, \quad x \in [0; L] \quad (5.29)$$

Both boundary conditions in (5.18) can be properly restructured to:

$$\left\{ \left[ \begin{array}{c} \frac{I_{inj}}{4 \cdot \pi \cdot a} \\ 0 \\ 0 \\ 0 \end{array} \right] + \left[ \begin{array}{c} 1 \\ 1 \\ 1 \\ 1 \end{array} \right] \circ \left[ \begin{array}{c} \frac{a}{2 \cdot R} \cdot \frac{\partial V}{\partial x} \\ 0 \\ 0 \\ 0 \end{array} \right] = \mathbf{0}, \quad \left[ \begin{array}{c} V(L, t) \\ 0 \\ 0 \\ 0 \end{array} \right] + \left[ \begin{array}{c} 0 \\ 1 \\ 1 \\ 1 \end{array} \right] \circ \left[ \begin{array}{c} \frac{a}{2 \cdot R} \cdot \frac{\partial V}{\partial x} \\ 0 \\ 0 \\ 0 \end{array} \right] = \mathbf{0} \right\} \quad (5.30)$$

and having compared (5.25) to (5.30), we can specify the vectors  $\mathbf{p}$  and  $\mathbf{q}$  linked to the boundary conditions (left,  $x = 0$ , and right,  $x = L$ ):



$$\mathbf{p}_{\text{left}} = \begin{bmatrix} \frac{I_{\text{inj}}}{4 \cdot \pi \cdot a} \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad \mathbf{q}_{\text{left}} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \quad \mathbf{p}_{\text{right}} = \begin{bmatrix} V(L, t) \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad \mathbf{q}_{\text{right}} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} \quad (5.31)$$

Based on the derived specifications for the functions  $\mathbf{c}(\cdot)$ ,  $\mathbf{f}(\cdot)$ , and  $\mathbf{s}(\cdot)$  in (5.28) as well as the initial (5.29) and boundary conditions (5.31), the MATLAB implementation, **pdepeHH.m**, with the PDE solver, **pdepe**, is shown in Listing 5.2.

**Listing 5.2** Program **pdepeHH.m**

```
function pdepeHH
%PDEPEHH Simulate the action potential in a single neuron.
% %PDEPEHH simulates a single neuron under deterministic
% stimulation using the full Hodgkin-Huxley model (in 2 dimensional
% domain, i.e., time and distance). The model is described by a
% system of partial differential equations. To solve them, the
% MATLAB solver PDEPE is applied.

clear all

p = axonparams;

% steady state value
p.V0 = 0;
p.n0 = 0.3177;
p.m0 = 0.0529;
p.h0 = 0.5961;
p.tmax = 5;
p.dt = 0.01;
t = 0:p.dt:p.tmax;
x = p.x;
msym = 0; %symmetry parameter (slab)
p.h = waitbar(0, 'Please wait...');
sol = pdepe(msym, @pdehh, @pdeinic, @pdebouc, x, t, [], p);
close(p.h)
% Extract the first solution component as V(x,t)
V = sol(:,:,1);
% Plot membrane potential V(t,x) as surface over time and distance
solutionV = figure; set(axes, 'FontSize',15)
set(solutionV, 'Position',[190 50 560 420]);
surf(x, t, V, ...
    'FaceColor','interp', 'EdgeColor','none','FaceLighting','phong');
colorbar;
axis tight
title('HH Model: PDE Solution V(x,t)');
xlabel('Distance [cm]');
ylabel('Time [ms]');
zlabel('Voltage [mV]');
print(solutionV, '-r600', '-dtiff', 'PDEsolutionVxt');

end
```

```

function p = axonparams
% .....
% .....
% .....

end

function [c,f,s] = pdehh(~, t, u, DuDx, p)
%PDEHH Evaluate the quantities defining the ODE system (HH Model).
% [C,F,S] = PDEHH(X, T, U, DUDX, P) defines the form of the PDE
% (Hodgkin-Huxley model) according to the general transient
% equation required by MATLAB's pdepe. The input arguments are:
% space variable |X|, time |T|, solution vector |U|, and its
% partial derivative vector, |DUDX|, with respect to |X|.
% The structure |P| is used to pass additional parameters and
% other values required for the evaluation of |C|, |F| and |S|.
% See also pdepe.

V = u(1);
n = u(2);
m = u(3);
h = u(4);

f = [p.a/(2*p.R)*1000; 0; 0; 0].*DuDx;
c = [p.C; 1; 1; 1];
s = [-(V-p.EK)*p.gK*n^4 - (V-p.ENa)*p.gNa*m^3*h - (V-p.EL)*p.gL ;
      nderiv(n, V, p.phi)
      mderiv(m, V, p.phi)
      hderiv(h, V, p.phi)];

% computations take place here
waitbar(t/p.tmax, p.h)

end

function u0 = pdeinic(~, p)
%PDEINIC Evaluate the initial conditions.
% U0 = PDEINIC (X, P) returns the initial conditions vector |U0|,
% evaluated at point |X| (if not used here, indicated as |~|),
% and with consideration of additional parameters |P|.

u0(1)= p.V0;
u0(2)= p.n0;
u0(3)= p.m0;
u0(4)= p.h0;

end

```

```

function [pl,ql,pr,qr] = pdebouc(~, ~, ~, ur, t, p)
%PDEBOUC Evaluate the boundary conditions.
% [PL,QL,PR,QR] = PDEBOUC(XL, UL, XR, UR, T, P) returns quantities
% |PL|, |QL|, |PR| and |QR| defining the boundary conditions
% evaluated at time |T|. |PL| and |QL| correspond to the left
% boundary point |XL| with the solution |UL|; |PR| and |QR|
% correspond to the right boundary points |XR| with the solution
% |UR|. Additional parameters are passed using the structure |P|.

Istim = p.Iinj*(heaviside(t)- heaviside(t - p.td));
pl = [Istim/(4*pi*p.a); 0; 0; 0];
ql = [1; 1; 1; 1];
pr = [ur(1); 0; 0; 0];
qr = [0; 1; 1; 1];

end

function dndt = nderiv(n, V, phi)
% .....
% .....
% .....

end

function dmdt = mderiv(m, V, phi)
% .....
% .....
% .....

end

function dhdt = hderiv(h, V, phi)
% .....
% .....
% .....

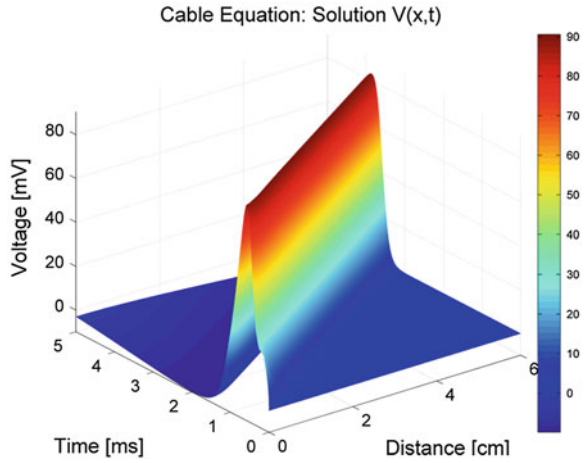
end

function phi = tempfactor(T)
% .....
% .....
% .....

end

```

**Fig. 5.12** Solution  $V(x,t)$  of the cable equation as function of distance,  $x$ , and time,  $t$ . The solution shows the response of the HH axon model to a stimulus of  $10 \mu\text{A}$  and  $0.2 \text{ ms}$  duration at temperature,  $T = 18.5^\circ\text{C}$



In the following we will use the MATLAB program **hoghux.m** to investigate the properties of the AP, and to examine some phenomena connected to AP generation. The axon parameters are the same as the original values in the HH model [9], and are shown in Table 5.6.

### Time Course and Profile of the Action Potential

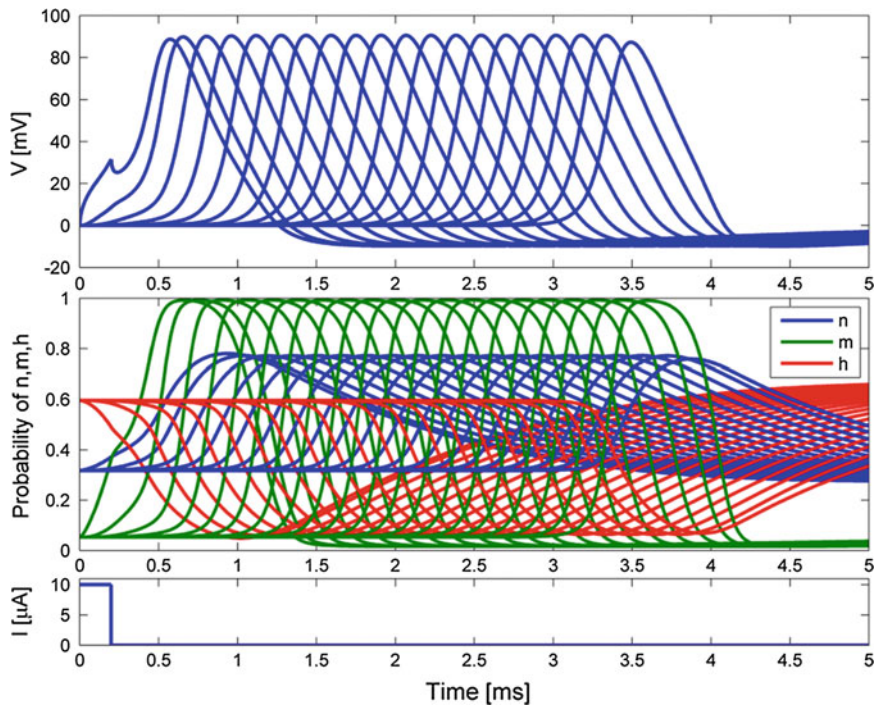
Figure 5.12 shows the solution of (5.15), i.e., the membrane potential as a function of distance and time,  $V(x,t)$ . A more detailed view is shown in Fig. 5.13 presenting APs over time at various distances within the axon,  $x = 0, \Delta x, 2 \cdot \Delta x, \dots, L$ , where  $L$  is the length of the axon.

### Conduction Velocity

The simulation program **hoghux.m** creates AP time courses at various distances  $x$  (four examples are shown in Fig. 5.14). We can use these graphs to estimate the conduction velocity of the AP,  $\theta$ . For two selected time courses, say at  $x_1 = 2.1 \text{ cm}$  and  $x_2 = 4.2 \text{ cm}$ , we can graphically derive the times,  $t_1 \sim 1.6 \text{ ms}$  and  $t_2 \sim 2.7 \text{ ms}$ , where the voltages attain their maximum value. The velocity  $\theta$  is then easily obtained:

$$\theta = \frac{x_2 - x_1}{t_2 - t_1} = \frac{4.2 - 2.1}{2.7 - 1.6} = 1.9 \text{ cm/ms} = 19 \text{ m/s} \quad (5.32)$$

This velocity of the AP propagation along the axon is calculated for the temperature of  $18.5^\circ\text{C}$ . There is a strong dependency on temperature; the AP propagates slower if the temperature is lower, as indicated in Fig. 5.15, created for the

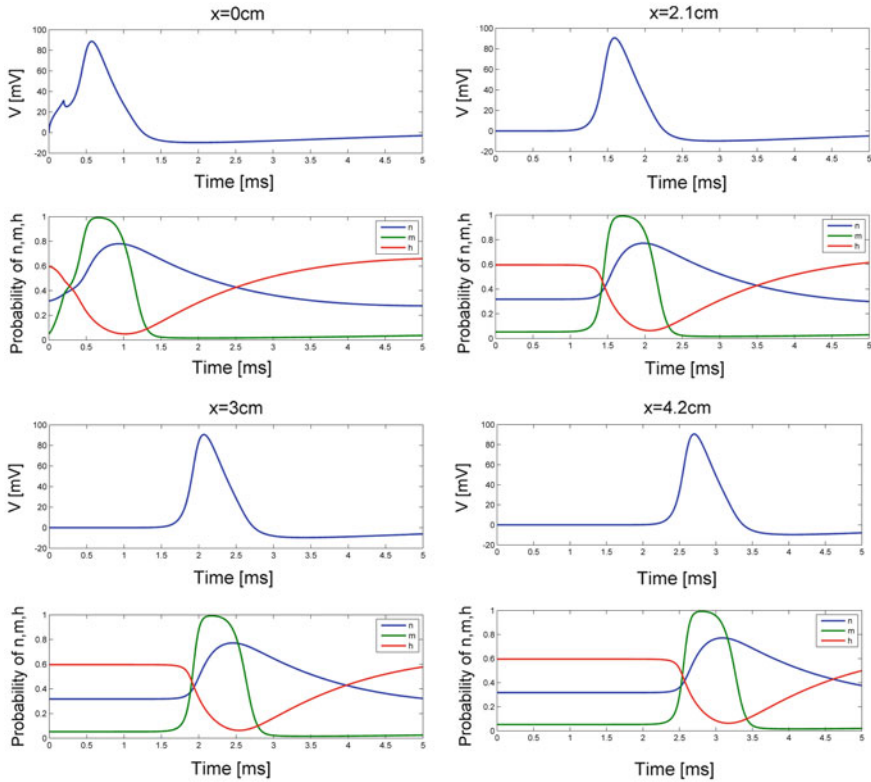


**Fig. 5.13** Response of HH axon model to a stimulus of  $10\ \mu\text{A}$  and  $0.2\ \text{ms}$  duration and at temperature  $T = 18.5\ ^\circ\text{C}$ . The *upper graph* contains time courses of the membrane potential  $V$  for various distances,  $x = 0, 0.3, 0.6 \dots 5.7, 6\ \text{cm}$ . The *lower graph* shows the time courses of quantities  $n$ ,  $m$  and  $h$

temperature  $T = 6.3\ ^\circ\text{C}$ . However, we leave it to the user to verify this dependency, using the program `hognux.m`, and calculating the conduction velocity for a couple of other temperature values.

### Long Stimulus

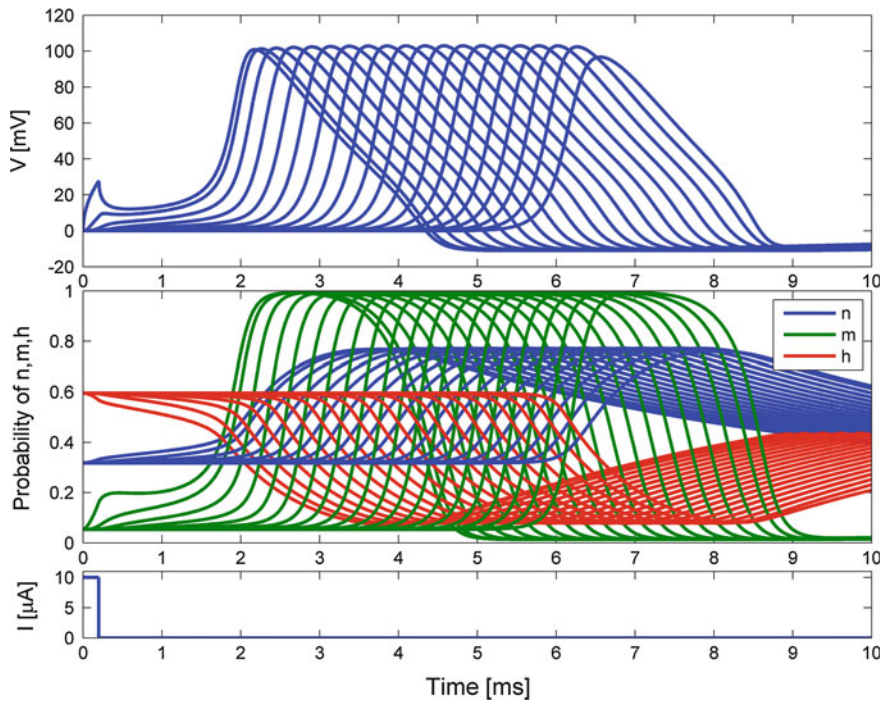
If the stimulus is long enough, repetitive firing can be observed along the axon as shown in Figs. 5.16 (at all analyzed distances) and 5.17 (at a selected distance,  $x = 3\ \text{cm}$ ). A detailed analysis of the time courses in Fig. 5.17 shows one more interesting property of the neuronal axon. Once an AP starts, a new AP at the same location can only be elicited after a certain interval, called *absolute refractory period*. Initialization of the next AP is possible in the hyperpolarization phase after the absolute refractory period, but requires a proper stimulus and stimulus duration. The period where it can happen is called *relative refractory period*. Normally, the next AP can be generated at the end of the hyperpolarization phase when the membrane is very close to the resting potential as also confirmed in Fig. 5.17.



**Fig. 5.14** AP generated at distances  $x = 0, 2.1, 3, 4.2$  cm, at temp.  $18.5^\circ\text{C}$ . Each graph shows the time course of voltage  $V$  and channel activation and inactivation quantities  $n$ ,  $m$  and  $h$ . The first AP, at  $x = 0$ , is fired as a reaction to a stimulus of  $10\ \mu\text{A}$  and  $0.2$  ms duration. A typical peak can be observed at  $t = 0.2$  ms

### Initial Values of the Quantities $n$ , $m$ , and $h$

All generated graphs also show the model-predicted time courses of gating variables  $m$ ,  $n$ , and  $h$ . If the simulation time is long enough, as shown in Fig. 5.18, we notice that time courses of these variables converge to steady state values,  $n_\infty$ ,  $m_\infty$ , and  $h_\infty$ , and are independent of the initial values given in the HH model. To start computations with the neuron at rest, we should use these as initial values. The graph in Fig. 5.18 confirms that initial and steady state values of  $n$ ,  $m$ , and  $h$  are equal. However, it is recommended to run the **hognux.m** program with other initial values and verify that this convergence indeed takes place. The reader will find an exercise on this topic at the end of this chapter.



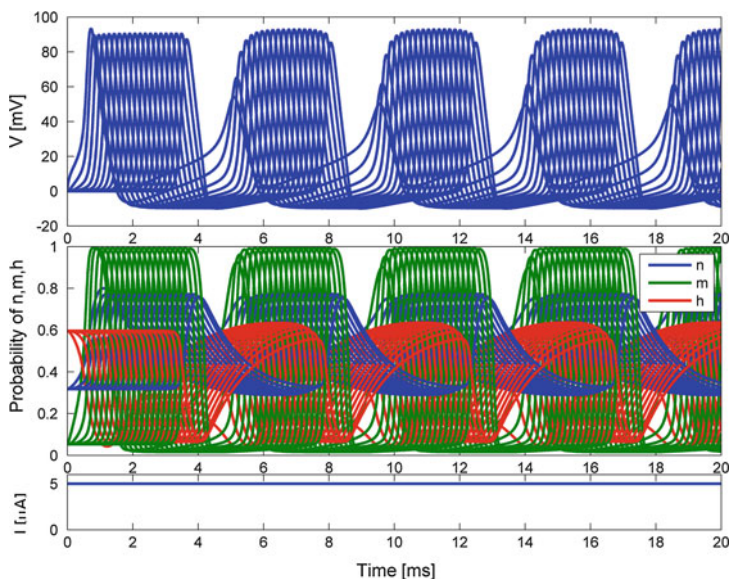
**Fig. 5.15** HH axon model response to a stimulus and assumed temperature,  $T = 6.3^\circ\text{C}$  (standard temperature for the HH model). *Upper panel*: time courses of the membrane potential  $V$  for various distances,  $x = 0, 0.3, 0.6, \dots, 5.7, 6$  cm; *middle panel*: time courses of quantities  $n$ ,  $m$  and  $h$ ; *lower panel*: a stimulus of  $10\ \mu\text{A}$  and  $0.2$  ms duration

### Threshold

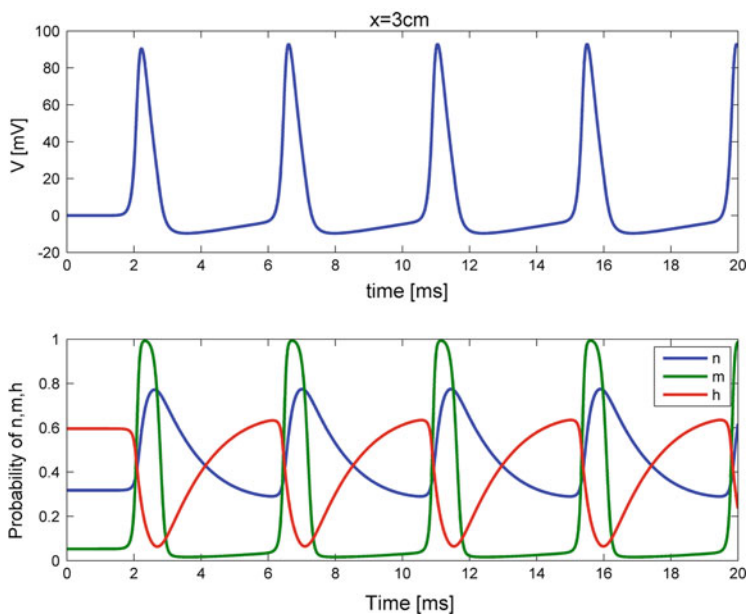
The threshold is the minimal value the membrane potential must achieve to create an AP. Experimentally, it can be found by trial and error through manipulating stimulating currents and their duration until an AP is fired. We leave it to the reader to carry out the experiment computationally using the MATLAB program **hoghux.m**.

All the above simulation scenarios were created with the **hoghux.m** program, however, it would have been possible using **pdepeHH.m** with proper modifications. As an example, the **pdepeHH.m** program generates a solution for the HH PDE equivalent to the **hoghux.m** program (Fig. 5.19).



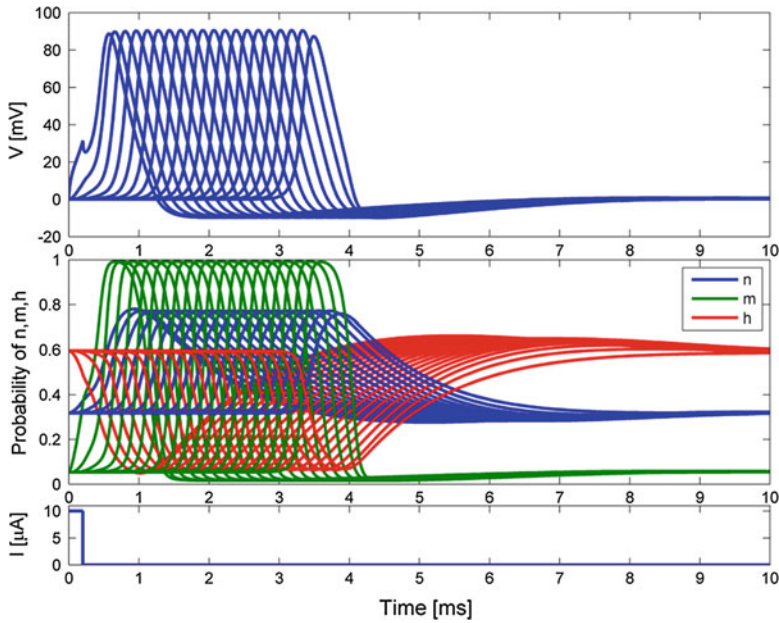


**Fig. 5.16** HH model response to a stimulus of  $10 \mu\text{A}$  and  $20 \text{ ms}$  duration ( $T = 18.5 \text{ }^\circ\text{C}$ ); *upper panel*: time courses of membrane potential  $V$  at various distances,  $x = 0, 0.3, 0.6, \dots, 5.7, 6 \text{ cm}$ ; *lower panel*: time courses of quantities  $n$ ,  $m$ , and  $h$

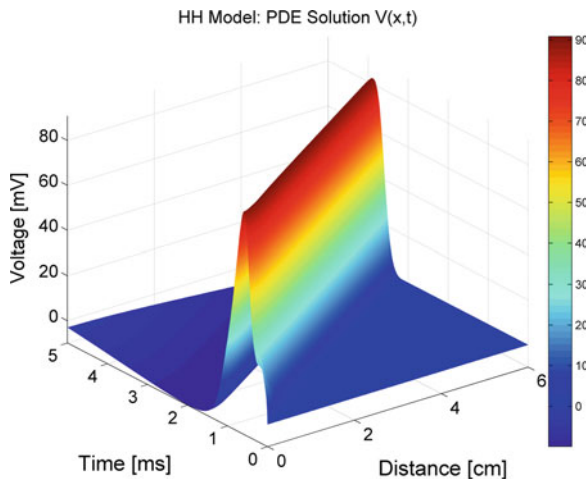


**Fig. 5.17** HH model response to a stimulus of  $10 \mu\text{A}$  and  $20 \text{ ms}$  duration ( $T = 18.5 \text{ }^\circ\text{C}$ ); *upper panel*: time course of the membrane potential  $V$  at distance,  $x = 3 \text{ cm}$ ; *lower panel*: time courses of quantities  $n$ ,  $m$  and  $h$





**Fig. 5.18** HH model response to a stimulus of  $10 \mu\text{A}$  and  $0.2 \text{ ms}$  duration ( $T = 18.5^\circ\text{C}$ ) with a long simulation time. *Upper panel:* Time course of the membrane potential  $V$  at various distances,  $x = 0, 0.3, 0.6, \dots, 5.7, 6 \text{ cm}$ . *Lower panel:* Time courses of quantities  $n$ ,  $m$  and  $h$



**Fig. 5.19** HH model implemented with the PDE solver: Solution  $V(x,t)$  of the cable equation as function of distance  $x$ , and time,  $t$ . The solution shows the response of the HH axon model to a stimulus of  $10 \mu\text{A}$  and  $0.2 \text{ ms}$  duration at temperature  $T = 18.5^\circ\text{C}$ . This graph, generated by the **pdepeHH.m** program, is equivalent to the graph in Fig. 5.12 (created by the **hoghux.m** program based on the compartmental method)

### 5.2.3 Application to Neuronal Networks and CNS Disorders

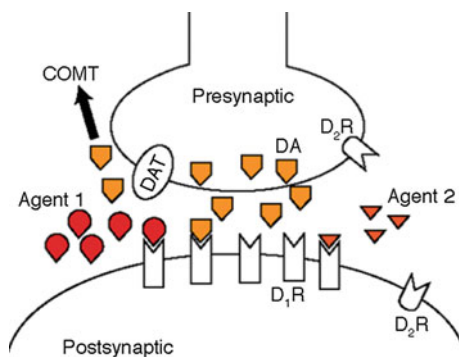
In the previous section, we described a biologically realistic mathematical model of a single neuron. This can be extended to multiple neurons to capture micro- and macro-neurocircuitry. In the following we describe a fascinating approach pioneered by In Silico Biosciences (ISB) who patented a biologically realistic platform to forecast treatment effects of antipsychotics and antidementia drugs.

Neurons intercommunicate by releasing neurotransmitters into the synaptic cleft, a narrow space between an axon terminal of the presynaptic neuron and a dendritic spine of the postsynaptic neuron. Neurotransmitters engage receptors located on the pre- and/or postsynaptic membrane and thereby induce specific effector mechanisms. Up to now, synaptic receptors have been the predominant target of drugs developed against mental disorders. As an illustration, Fig. 5.20 presents events at a dopaminergic synapse where dopamine is released from the presynaptic neuron and finds its dopamine  $D_1$  and  $D_2$  receptors in pre- and postsynaptic membranes.

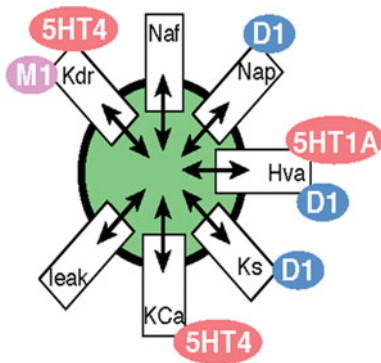
Drug receptor occupancy is the result of the competition between endogenous neurotransmitters and drug. The level of occupancy can be calculated if the human receptor affinity of the drug is known.

Regularly and often via a cascade of intracellular events, occupied and activated receptors change the conductance of a number of ion channels, resulting in membrane and synaptic currents and APs (Fig. 5.21).

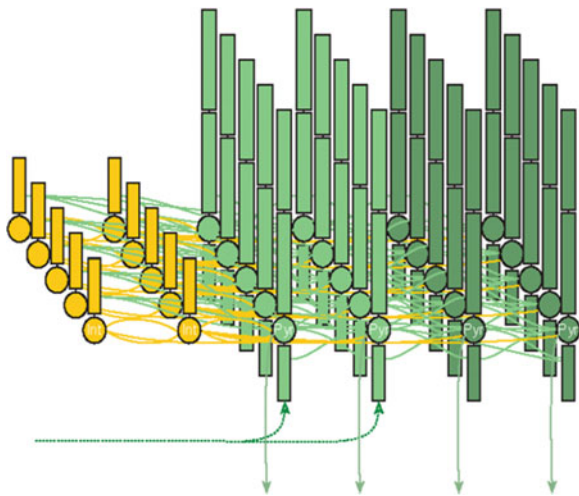
APs propagate through a network of neurons organized within brain nodes (cortex, hippocampus, striatum, ...) and connecting tracts. Brain nodes have their specific microcircuitry [27] which needs to be considered when modeling



**Fig. 5.20** Synaptic receptors and drugs. The neurotransmitter dopamine (DA), released into the synaptic cleft, competes with Agent 1 and Agent 2 for  $D_1$  and  $D_2$  receptors ( $D_1R$ ,  $D_2R$ ), respectively. DA is inactivated either by transport into the presynaptic cell via a dopamine transporter (DAT), or via metabolism by catechol-O-methyltransferase (COMT) [26]

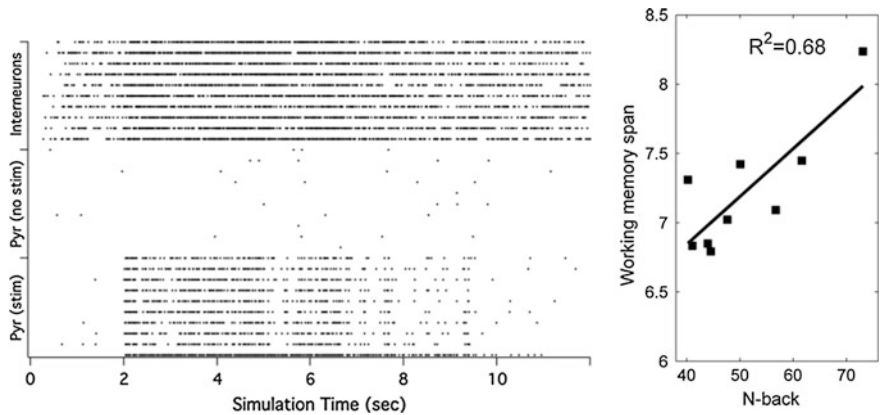


**Fig. 5.21** Receptor effects on membrane currents. The *green circle* represents the soma of a neuron. Changes in receptor activation are linearly connected to channel conductances. For example:  $g'_{Kdr} = g_{Kdr} \cdot (1 - \theta_{5HT4}^{Kdr} \cdot 5HT4^{eff})$  describes the conductance of the Kdr channel following a change in 5HT4 activation according to  $5HT4^{eff} = (5HT4^a - 5HT4^c)/5HT4^c$ . 5HT4<sup>a</sup> is the actual, 5HT4<sup>c</sup> the control activation. 5HTx—serotonin receptors, M1—muscarinic receptor, D1—dopamine receptor, Kdr—direct rectifier K<sup>+</sup> channel, Naf—fast Na<sup>+</sup> channel, Nap—persistent sodium current, Hva—high voltage activated calcium channel, Ks—slow activating potassium channel, KCa—calcium-activated potassium channel [26]



**Fig. 5.22** Representation of the prefrontal cortex (PFC). Pyramidal cells (Pyr, in green) and interneurons (in yellow) are connected according to published data. Stimulatory input (*vertical green dashed line*) arrives from other brain regions and four output signals leave the PFC. Receptors and currents are not indicated [26]

respective functional output. For instance, if we are interested in modeling a memory trace, we should aim to build a computational model of the cortex that mimics its respective biological properties (Figs. 5.22, 5.23).



**Fig. 5.23** Calibration of model output. *Left panel* (model output): At 2 s a current is injected leading to 10 activated pyramidal neurons firing APs for a period (memory span) of about 7 s. *Right panel*: Correlation between model output (memory span) and clinical results (N-back test). Each *square* refers to a treatment group of patients [26]

Beyond modeling a single brain node, the cortex, the approach described was applied to model a network of brain nodes implicated in schizophrenia (cortical/subcortical and striatal circuits). In a blinded prospective evaluation, the model correctly predicted the lower performance of JNJ37822681 on the positive and negative syndrome scale (PANSS) and the higher extra-pyramidal symptom (EPS) liability compared to a positive control (olanzapine) [28].

### 5.3 Renal Anemia

#### 5.3.1 Disease Definition

Characteristic symptoms of anemia are fatigue and weakness. Laboratory blood analysis reveals a low red blood cell (RBC) count. This can be due, among other causes, to insufficient erythropoiesis (RBC production in the bone marrow) or increased RBC loss. In renal anemia, erythropoiesis is impaired by inadequate availability of erythropoietin, an erythropoiesis-stimulating hormone produced in the kidneys.

#### 5.3.2 Physiologic Model

RBCs carry the protein hemoglobin (Hb) which to reversibly binds oxygen. Oxygen is attached to Hb in the lungs and removed from Hb in tissues and organs where it is broadly used in cell metabolism. RBCs are produced in the bone

marrow and eliminated e.g., by the spleen which detects aged RBCs. RBCs have a life span of 100–120 days. Each RBC carries about 30 pg Hb. The normal Hb concentration in blood is 16 g/dL for men and 14 g/dL for women. Assuming a total blood volume of 6 L, our body contains about 900 g Hb at any given time. Taking into account the life span of 120 days, we arrive at an Hb production rate of 7.5 g/d under physiologic conditions.

Before we consider the pathologic condition of decreased Hb, we develop a model for Hb dynamics under normal conditions by assuming that erythropoiesis is regulated as indicated in Fig. 5.24.

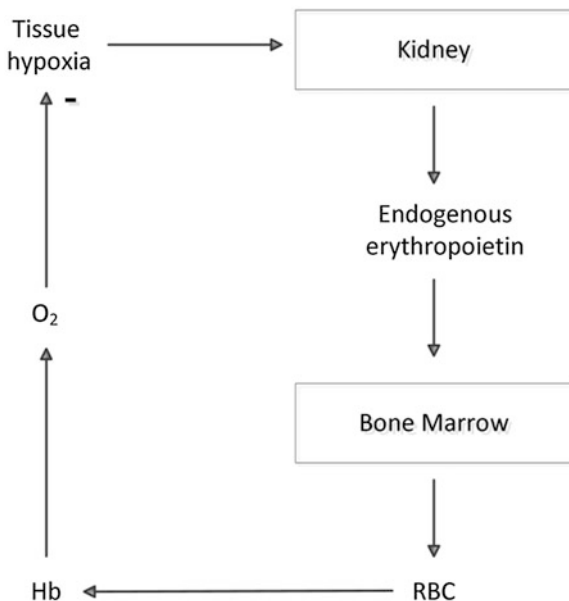
The following DDE model describes Hb concentrations according to Fig. 5.24 by implementing a feedback mechanism and the RBC life span.

$$\left. \begin{aligned} \frac{d}{dt}Hb &= r_{in}(t) - r_{out}(t), \quad Hb(0) = Hb_0 \\ r_{out}(t) &= r_{in}(t - LS) \\ r_{in}(t) &= e^{-\beta \cdot (Hb(t) - Hb_{tar})} \end{aligned} \right\} \quad (5.33)$$

where,  $r_{in}(t)$  denotes the Hb-dependent production rate of Hb at time  $t$ , and  $r_{out}(t)$  Hb elimination (or loss) rate at time  $t$ . Assuming a constant RBC life span ( $LS = 100$  d),  $r_{out}(t)$  equals  $r_{in}$  at time  $t - LS$ . Target Hb,  $Hb_{tar}$  was set to 150 g/L;  $r_{in} = 7.5$  g/d, and  $Hb_0 = 150$  g/L may be good approximations of physiologic conditions.

The MATLAB implementation of the physiologic model, (5.33), with a single disturbance is shown in Listing 5.3, and results in Fig. 5.25.

**Fig. 5.24** Regulation of red blood cells (RBCs). Upon erythropoietin stimulation, the bone marrow releases RBCs into the blood. RBCs contain hemoglobin (Hb) which binds oxygen ( $O_2$ ) and carries it in the blood to peripheral tissues. If the tissue oxygen supply is insufficient, cells in the kidney increase production of erythropoietin. The concentration of RBCs in blood is about  $5 \cdot 10^{12}/L$  and their life span about 120 d. This indicates a production rate of about three million RBCs per second!



**Listing 5.3** Program **epoDDE01.m**

```

function epoDDE01
%EPoDDE01 Dynamic of Hb in a physiologic model.
%   EPoDDE01 shows how to implement a patho-physiologic model as
%   a DDE system. It produces a graph showing the Hb time profile
%   in renal anemia.

options = ddeset('RelTol',1e-6,'AbsTol',1e-6,'InitialY',10);
p.LS = 100;
p.hb0 = 11;
p.hb00 = 14;
p.tstop = 250;
p.start = 20;
p.beta = 0.02;
p.rin0 = 7.5;
p.stim = 0;
sol = dde23(@derivatives, p.LS, p.hb0, [0 p.tstop], options, p);
set(axes,'FontSize', 15)
plot(sol.x,sol.y,'LineWidth',2)
hold on
rin = prod(sol.x, sol.y, 0, p);
tlin = linspace(p.LS, p.tstop, 1001);
rout = prod(tlin, deval(sol,tlin-p.LS), p.LS,p);
plot(sol.x,rin,'-g','LineWidth',2)
plot(tlin,rout,'-r','LineWidth',2)
plot([0 100],[p.rin0 p.rin0],'-r','LineWidth',2)
legend('Hb [g/dL]','Hb production [g/d]','Hb elimination [g/d]', ...
       'Location', 'East')
xlabel('Time [d]')
print('-dtiff', '-r900', 'Eporegulation')

end

function dydt = derivatives(t, y, Z, p)
%DERIVATIVES Compute the right-hand side of the DDE.
%   DYDT = ...

Hb = y(1);
lagHb = Z(1,1);
dHB = prod(t, Hb, 0, p) - prod(t, lagHb, p.LS, p);
dydt = dHB;

end

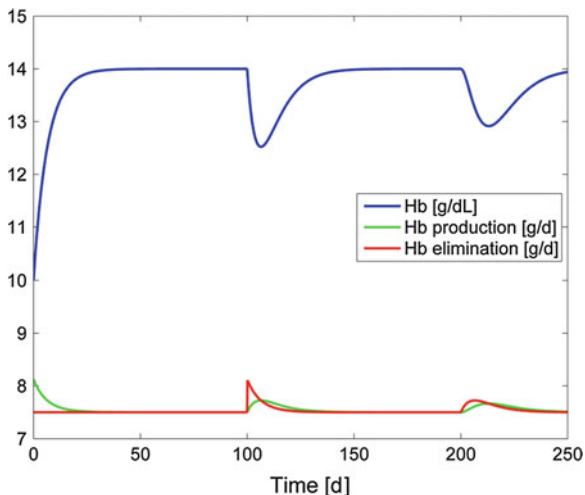
function prod = prod(t,Hb,delay,p)
%PROD Evaluate Hb production.
%   PROD = PROD(T,HB,DELAY,P) computes the hemoglobin production.
%   Input parameters: |T| - time, |HB| - current value of hemoglobin,
%   |DELAY| - life span of erythrocytes, |P| - structure for
%   additional parameters.

tt = t-delay;
prod = (p.rin0 + p.stim*heaviside(tt-p.start)) ...
       .*exp(p.beta*(p.hb00-Hb).*(delay==0|tt>0));

end

```

**Fig. 5.25** Time course of Hb and its production and elimination rate following a change in Hb from 14 g/dL to 10 g/dL at day 0. Hb was set to 14 g/dL during the DDE history period (before day 0) and to 10 g/dL at day 0. The ‘baseline’ Hb value of 14 g/dL is readily achieved, but a shortlasting decrease in Hb occurs at each multiple of life span, LS (100 d)



### 5.3.3 Pathophysiologic Model

Chronic kidney disease (CKD) may impair the kidneys’ ability to produce erythropoietin. Furthermore, it may impair renal function causing uremia followed by a decrease in bone marrow RBC production and a decrease in RBC life span. Treatment options comprise the administration of an erythropoietin-stimulating agent (ESA), dialysis (to remove uremic substances from the body), and kidney transplantation. Figure 5.26 is a modification of Fig. 5.24 indicating the pathophysiology and ESA treatment option for renal anemia. Treatment with an ESA should maintain Hb within ‘physiologic’ ranges. Too low Hb levels will fail to correct the anemia; too high Hb levels carry the risk of vascular events.

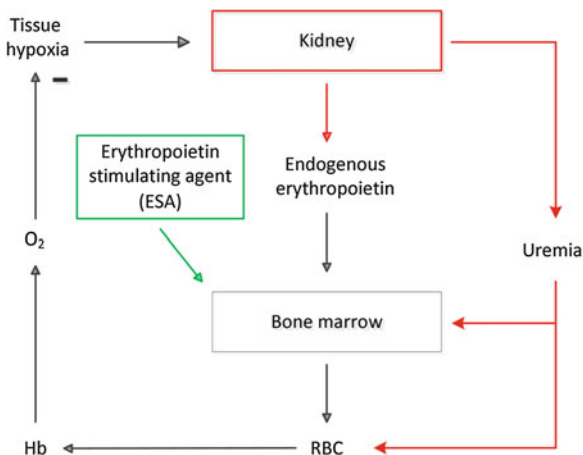
In the following we will introduce a modification of the model presented in (5.33). We make the simplifying assumption that Hb feedback regulation can be ignored when considering renal anemia, i.e., the kidneys are not able to react to hypoxia-induced factors and produce erythropoietin. The relevant model is as follows:

$$\frac{d}{dt}Hb = r_{in}(t) - r_{in}(t - LS) ; \quad Hb(0) = Hb_0 \quad (5.34)$$

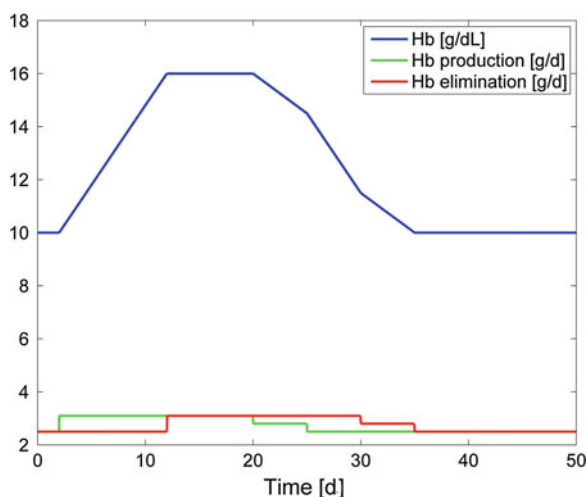
As before,  $r_{in}(t)$  is the production rate of Hb at time  $t$ ,  $r_{in}(t - LS)$  the Hb loss rate at time  $t$ ,  $LS$  the RBC life span and  $Hb_0$  a baseline Hb value. Both  $LS$  and  $r_{in}$  may decrease with the severity of the disease. Before the model described in Eq. (5.34) is applied to drug treatment, some consequences of this model should be demonstrated (Fig. 5.27).

Figure 5.27 illustrates some important features of the dynamics of Hb. If the production rate of Hb is kept constant over a period longer than life span, resulting Hb values finally reach steady state. This is readily acknowledged as under this condition Hb elimination eventually equals Hb production. To keep Hb values at

**Fig. 5.26** Pathophysiology of renal anaemia and chronic kidney disease. Impaired kidney function causes a decrease in endogenous erythropoietin production and an increase in uremic substances resulting in low RBC (and Hb) levels and a shortened RBC life span. One treatment option is to administer an erythropoietin stimulating agent (ESA)



**Fig. 5.27** Hemoglobin (Hb) time course following time-dependent changes in Hb production. Hb production is piecewise constant with varying amplitude. Hb elimination equals delayed Hb production. The delay corresponds to the life span of Hb and was set to 10 days in this example



an elevated steady state, stimulation must occur continuously. As  $r_{in}(t)$  will become the production rate of Hb due to bone marrow stimulation following administration of an ESA, the treatment schedule must be evaluated carefully to avoid excessive Hb values as there is no longer internal feedback control of Hb.

### 5.3.4 Drug-Disease Model

In the previous chapter we showed how Hb reacts if its production is changed under renal anemia conditions where Hb has no feedback to its production. This knowledge can be used when extending the disease model to a drug-disease model for a drug acting as an ESA. Drug concentrations at the target (bone marrow) are determined by the PK of the drug, and effects on RBC (or Hb) production by its



PD. In Eq. (5.35) we linked drug concentrations to Hb production via an  $E_{\max}$  model with parameters  $S_{\max}$  and  $SC_{50}$ . As there is no feedback control of Hb, any drug dosage regimen needs to be carefully evaluated with respect to avoiding excessive Hb values.

$$\left. \begin{aligned} \frac{d}{dt}Hb &= r_{\text{in}}(t) - r_{\text{in}}(t - LS) ; & Hb(0) &= Hb_0 \\ r_{\text{in}} &= r_{\text{in}0} \cdot DE \\ DE &= 1 + \frac{S_{\max} \cdot c(t)}{SC_{50} + c(t)} \\ c(t) &= PK(DR, PK_{\text{par}}, t) \end{aligned} \right\} \quad (5.35)$$

Abbreviations, parameter values (units):

$Hb_0$	$\equiv$ Hb at start of treatment, 110 (g/L)
$LS$	$\equiv$ Life span of erythrocytes, 40 (d)
$r_{\text{in}0}$	$\equiv$ Hb production rate at start of treatment, $Hb_0/LS$ (g/d)
$DE$	$\equiv$ Drug effect
$S_{\max}$	$\equiv$ Maximum relative increase in erythropoiesis stimulation, 0.5
$SC_{50}$	$\equiv$ Drug concentration producing half maximum stimulation, 0.05 (mg/L)
$DR$	$\equiv$ Dosage regimen
$PK_{\text{par}}$	$\equiv$ PK parameters
$t$	$\equiv$ Time after treatment start (d)
$c(t)$	$\equiv$ Drug concentration.

We will apply this model to a fictive ESA drug which is given either intravenously or subcutaneously. We assume a one-compartment model and the following PK parameters:

$V$	$\equiv$ Volume of distribution, 20 (L)
$ka$	$\equiv$ First-order absorption rate constant, 0.1 (1/d)
$ke$	$\equiv$ First-order elimination rate constant, 0.5 (1/d)
$F$	$\equiv$ Subcutaneous bioavailability, 0.6.

The dosage regimen is a single dose of 5 mg.

The MATLAB program **epodosing.m** (Listing 5.4) uses the delay differential equation solver **dde23** with events. This allows the investigation of multiple dosing regimens as well as the assessment of dosing rules that keep Hb values within a target range.

#### Listing 5.4 Program **epodosing.m**

```
function epodosing
%EPDOSING investigate the multiple dosing of an ESA.
%   EPDOSING uses a DDE model applied to an ESA.
%   It allows investigation of dosing regimens, and evaluation of Hb
%   based adaptive dosing rules.

p.hb0 = 110;
```

```

p.LS      = 40;
p.hb00    = 110; % history
p.tstop   = 80;
p.start   = 20;
p.beta    = 0.;
p.rin0    = 8;
p.s0      = 8;
p.smax    = 0.5;
p.sc50    = 0.05;
p.ka      = 0.1;
p.ke      = 0.5;
p.V       = 20;
p.F       = 0.6;
p.dosetime = 0.01; % or 10:10:60;
p.dose     = 8*(ones(1,length(p.dosetime)));
p.route   = 'IV';    % 'IV' or 'SC'
p.HbM     = 140;
p.HbMax   = p.HbM;
p.HbMin   = p.HbM;
figure;
% prepare options for the DDE model
options = ddeset('RelTol',1e-5,'AbsTol',1e-5,'Events',@fevents);
sol = dde23(@derivatives,p.LS,[p.hb00 0 0],[0 p.tstop],options,p);
while sol.x(end) < p.tstop
    if length(sol.ie) > 1 && sol.ie(end-1)==8
        sol.ie
    end
    if sol.ie(end)<=length(p.dosetime)
        dd = p.dose(sol.ie(end));
    end
    route = strcmp(p.route,'SC');
    options = ddeset('RelTol',1e-7,'AbsTol',1e-7, ...
        'InitialY',[sol.y(1,end) sol.y(2,end) + dd*route*p.F ...
        sol.y(3,end) + dd*(1-route)],'Events',@fevents);
    sol = dde23(@derivatives, p.LS,sol, [sol.x(end) p.tstop], ...
        options, p);
    subplot(3,1,1)
    h311Hb = plot(sol.x, sol.y(1,:), 'b', 'LineWidth',2);
    hold on
    h311Max = plot([0,p.tstop], [p.HbMax,p.HbMax], 'b');
    h311Min = plot([0,p.tstop], [p.HbMin,p.HbMin], 'b'); %ok<NASGU>
    axis([0 p.tstop 100 150])
    ylabel('Hemoglobin [g/L]')

    subplot(3,1,2)
    yy = prod(sol.x,sol.y(3,:),p);
    h312Pr = plot(sol.x', yy, 'g');
    hold on
    tt=p.LS+sol.x;
    index = tt <= p.tstop;
    h312R = plot(tt(index), yy(index), 'r'); %ok<NASGU>
    ylabel('Hb Rates [g/d]')

    subplot(3,1,3)
    h313SC50 = plot([0 p.tstop], [p.sc50 p.sc50], '--b');
    xlabel('Time [d]')
    hold on
    if strcmp(p.route,'SC')

```

```

        [AX,h313DC,h313DA]=plotyy(sol.x, 1/p.V*sol.y(3,:), ...
            sol.x, 1/p.V*sol.y(2,:));
        set(h313DC, 'LineWidth',2)
        set(get(AX(1),'Ylabel'),'String', ...
            'Drug Concentration [ng/mL]')
        set(get(AX(2),'Ylabel'),'String', 'Drug Amount [mg]')
    else
        h313DC = plot(sol.x, 1/p.V*sol.y(3,:), 'LineWidth',2);
        ylabel('Drug Concentration [ng/mL]')
    end
end

function dydt = derivatives(t, y, Z, p)
%DERIVATIVES Compute the right-hand side of the DDE.
% DYDT = ...

tlag = t-p.LS;
m0 = y(2);
dm0 = -p.ka*m0;
m = y(3);
dm = p.ka*m0-p.ke*m;
c = m/p.V;
mlag = Z(3,1);
clag = mlag/p.V;
rin = prod(t, c, p);
rout = prod(tlag,clag,p);
dydt = [rin - rout ; dm0; dm];

end

function prod = prod(t, c, p)
%PROD evaluate Hb production
% PROD = PROD(T,C,P) computes the hemoglobin production based on
% the drug effect formula (Hill equation). Input parameters:
% |T| - time, |C| - concentration, |P| - additional parameters
% (required by the Hill equation).

E = hilleffect(c, 1, p.smax.*(t>=0), p.sc50, 1);
prod = p.s0.*E;

end

function E = hilleffect(c, E0, Emax, EC50, n)
.....
.....
.....

end

function [value,isterminal,direction] = events(t, y, ~, p)
%FEVENTS Specify events for the DDE model.
% [VALUE,ISTERMINAL,DIRECTION] = FEVENTS(T,Y,Z,P) detects the exact
% time points of events specified as a function of time |T|, vector
% of dependent variables |Y|, and a structure of some (specific)

```

```
% other parameters |P|. Matrix |Z| corresponds to all possible
% delays and contains solutions |Y| at the delayed argument.
%
% Output parameters: |VALUE| - value investigated as event;
% |ISTERMINAL| = 0|1 where 0 stands for 'don't stop', 1 for 'stop
% integration'; |DIRECTION| = -1|0|1 where (-1) means 'negative
% direction only', 0 - 'both direction', 1 - 'positive direction
% only'

% define events:
% - dose times (time event)
% - Hb level for dose changes

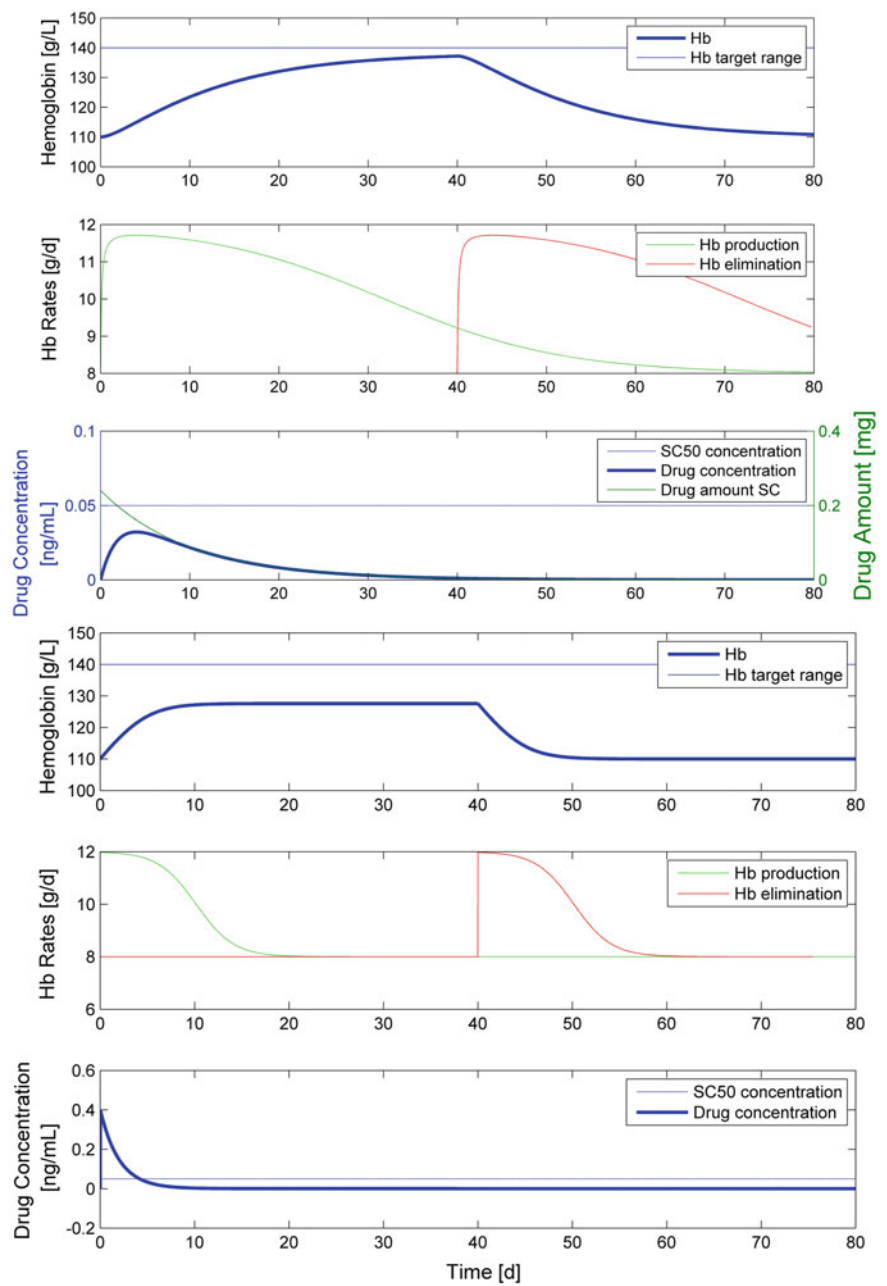
value(:,1) = [t - p.dosetime'; y(1)-p.HbM];
isterminal(:,1) = [ones(1,length(p.dosetime))'; 0];
direction(:,1) = [ones(1,length(p.dosetime))'; 1];

end
```

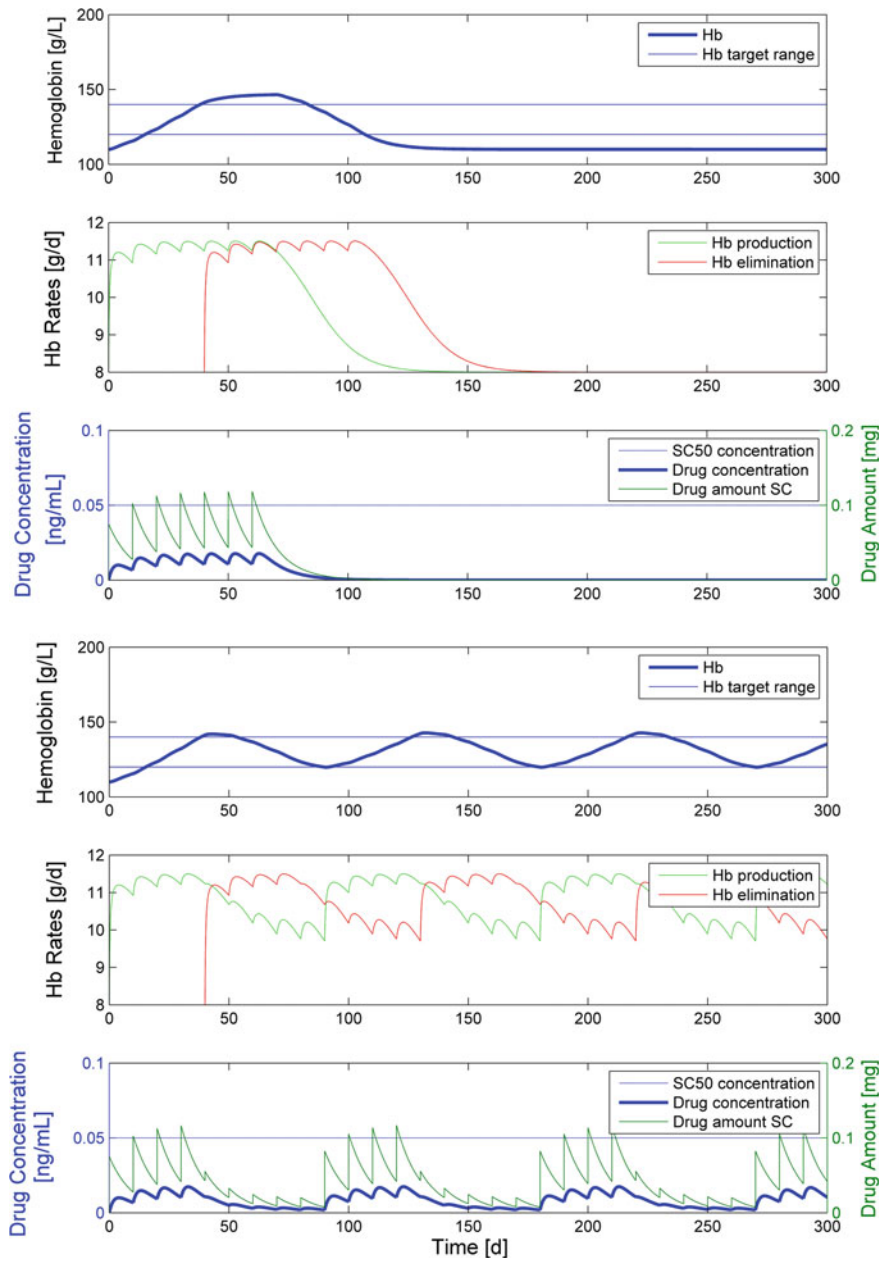
As proteins, ESAs cannot be given orally. Figure 5.28 (generated by the **epodosing.m** program) shows how the route of administration (SC versus IV) of an ESA could impact on the Hb time course. The same subcutaneous dose provides a larger effect on Hb than an IV dose, despite an assumed bioavailability of  $F = 0.6$ . This is explained by the longer Hb stimulation following SC administration and the low value of SC50.

Effects of multiple ESA dosing on Hb with and without dose adaptation are shown in Fig. 5.29. A constant regimen given over a period longer than RBC life span produces relatively stable Hb values, but these may be too high and occur too late. One simple dose adaptation rule is as follows: Reduce (increase) dose when upper (lower) target range is hit. A potential drawback of this dosing strategy is the occurrence of large Hb fluctuations, illustrated in Fig. 5.29. This phenomenon was documented in 1991 by Garred and Pretlac [29] who proposed using mathematical modeling for erythropoietin therapy to achieve smoother Hb profiles instead of pushing the dose up and down based on observed Hb values. The MATLAB program implementing the effects of multiple ESA dosing, **epoDosingControl.m**, is available at MATLAB Central.

In Chap. 7, we will apply our model to the situation where patients switch from a short to a long-acting ESA. The exploration of dose adjusting rules will then be a part of design optimization.



**Fig. 5.28** Impact of dosing with 8 mg SC (upper three panels) and IV (lower three panels) on Hb time course (life span model without feedback). SC dosing produces a delayed and larger increase in Hb than IV dosing. SC bioavailability was set to  $F = 0.6$



**Fig. 5.29** Impact of multiple SC doses on Hb time course. Without dose adjustment excessive Hb values ( $> 140$  g/L) are reached (*upper three panels*). A dose adjustment rule based on the Hb target range (120–160 g/L) causes large fluctuations in Hb (*lower three panels*)

## 5.4 Diabetes Mellitus

### 5.4.1 Disease Definition

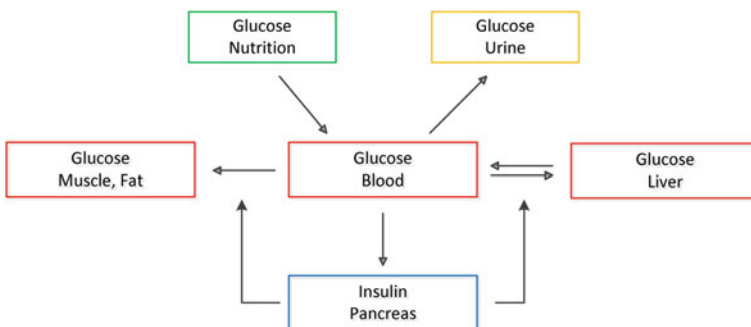
The main symptoms of diabetes are increased thirst, frequent urination, fatigue, and weakness. Laboratory analysis detect elevated blood sugar levels, either under fasting conditions and/or following a glucose load in a glucose tolerance test (i.e., a drink of 75 mg glucose).

### 5.4.2 Physiologic Model

The disposition of glucose in muscle, fat, and liver is regulated by insulin, a hormone produced in the pancreas. Importantly, glucose triggers the release of insulin. There are more hormones involved in the regulation of glucose, but for the sake of simplicity we will concentrate on glucose and insulin in this chapter.

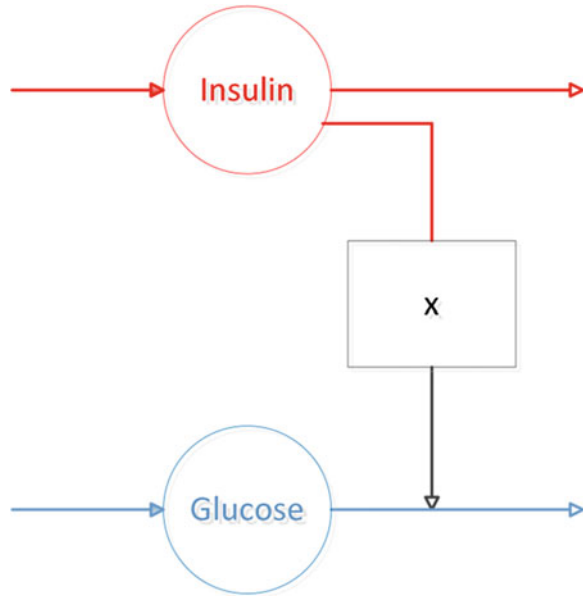
The basic scheme below shows the interrelationship between insulin and glucose where glucose promotes its own insulin-dependent disposal. Triggered by glucose, insulin is released from pancreatic islet cells. If the pancreatic production becomes insufficient, exogenous administration of insulin is required (Fig. 5.30).

Mathematical models for the interplay between glucose and insulin were pioneered by Richard Bergman et al. [30]. They investigated the feasibility of using mathematical models to determine insulin sensitivity in individual patients. Insulin sensitivity is defined as the ability of insulin to enhance glucose disposition and is usually assessed after a glucose challenge (intravenously or orally). The concept of the basic glucose–insulin model (or ‘glucose minimal model’ [30]) is depicted in Fig. 5.31.



**Fig. 5.30** Relationship between glucose and insulin. Glucose triggers its own insulin-dependent disposal

**Fig. 5.31** Basic model for the assessment of insulin sensitivity. Insulin concentrations are used as a forcing function for X representing the insulin-induced effect on glucose disposal



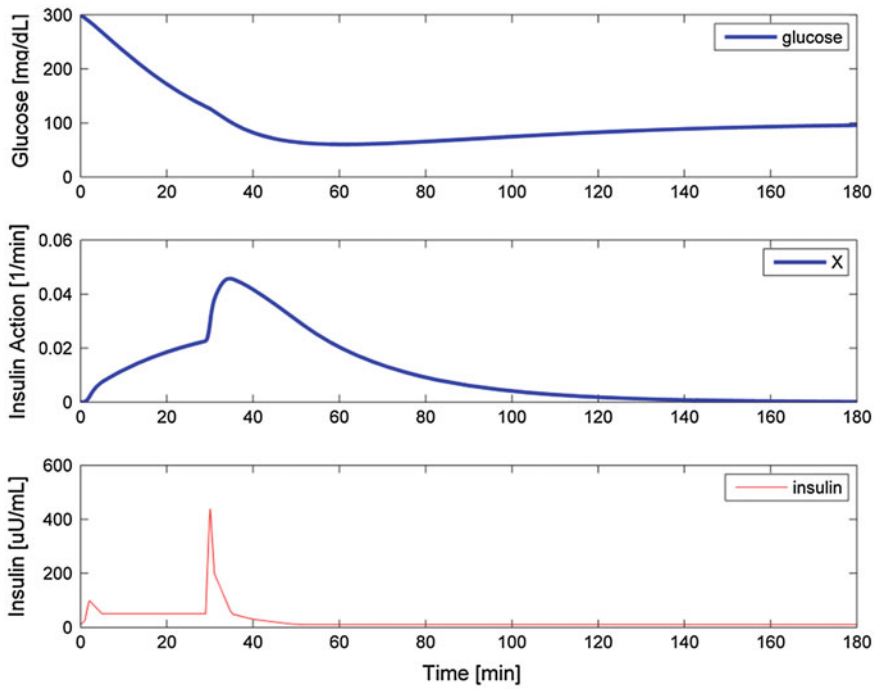
Equations of the glucose minimal model are presented below:

$$\left. \begin{aligned} \frac{dg}{dt} &= S_G \cdot [g_b - g(t)] - x(t) \cdot g(t) + \frac{r_{in}(t)}{V} ; \quad g(0) = g_b \\ \frac{dx}{dt} &= p_2 \cdot \{S_I \cdot [i(t) - i_b] - x(t)\}; \quad x(0) = 0 \end{aligned} \right\} \quad (5.36)$$

Abbreviations and parameter values (units):

- $t$          $\equiv$  Time (min)
- $g(t)$      $\equiv$  Glucose concentrations at time  $t$  (mg/dL)
- $g_b$        $\equiv$  Glucose concentrations at baseline, 120 (mg/dL)
- $i(t)$      $\equiv$  Insulin concentration at time  $t$  ( $\mu$ U/mL)
- $i_b$        $\equiv$  Insulin concentration at baseline, 10 ( $\mu$ U/mL)
- $x(t)$      $\equiv$  Insulin action on glucose disposal (1/min)
- $r_{in}(t)$   $\equiv$  Input rate of external glucose (mg/min)
- $V$          $\equiv$  Distribution volume of glucose, 70 (dL)
- $S_G$        $\equiv$  Glucose effectiveness, 0.03 (1/min)
- $S_I$        $\equiv$  Insulin sensitivity, 0.0008 (mL/( $\mu$ U·min))
- $p_2$        $\equiv$  Rate constant for change in insulin action, 0.04 (1/min).





**Fig. 5.32** Minimal model applied to intravenous glucose tolerance test (IVGTT): Simulation of glucose levels (*upper panel*) following a glucose load of 300 mg/kg at time 0 in a 70 kg adult. Glucose is disposed by insulin action (*middle panel*), a virtual entity created from measured insulin levels (*lower panel*). At 30 min, a rapid insulin infusion is administered

$S_G$ , glucose effectiveness, is the ability of glucose to promote its own disposal. This model was developed to determine the parameters  $p_2$ ,  $S_G$ , and  $S_I$  from experimental data (glucose challenge) in healthy volunteers and patients. Measured insulin concentrations, linearly interpolated, were used as a forcing function.

An illustration of the minimal model is given in Fig. 5.32 and the corresponding program, **minmod.m**, is listed thereafter (Listing 5.5).

**Listing 5.5** Program **minmod.m**

```

function minmod
%MINMOD Implement the glucose-insulin minimal model
% MINMOD solves the glucose-insulin minimal model that assesses
% insulin sensitivity based on observed insulin concentrations
% and using a forcing function.

p.ib    = 10;      % uU/mL
p.gb    = 100;     % mg/dL
p.SG    = 0.03;    % 1/min
p.SI    = 0.0008;  % 1/min * mL/uU
p.p2    = 0.04;    % 1/min
p.doseIV = 21000;  % mg
p.V      = 70;     % Vd glucose dL
[t,y] = ode15s(@derivatives,[0 180],[p.doseIV/p.V 0],[1,p];
subplot(3,1,1)
plot(t,y(:,1),'LineWidth',2);
legend('glucose')
ylabel('Glucose [mg/dL]');      % xlabel('Time [min]');
subplot(3,1,2)
plot(t,y(:,2),'LineWidth',2);
legend('X')
ylabel('Insulin Action [1/min]'); % xlabel('Time [min]');
subplot(3,1,3,'LineWidth',2)
plot(t,i(t),'r')
legend('insulin')
ylabel('Insulin [uU/mL]'); xlabel('Time [min]');
print('-dtiff','-r900','minmod')

end

function dydt = derivatives(t, y, p)
%DERIVATIVES Compute the right-hand side of the ODE.
% DYDT = ...

glu = y(1);      % glucose
x   = y(2);      % insulin action as forcing function
dglu = -(p.SG+x)*glu + p.SG*p.gb;
dx   = - p.p2*(x - p.SI*(i(t)-p.ib));
dydt = [dglu; dx];

end

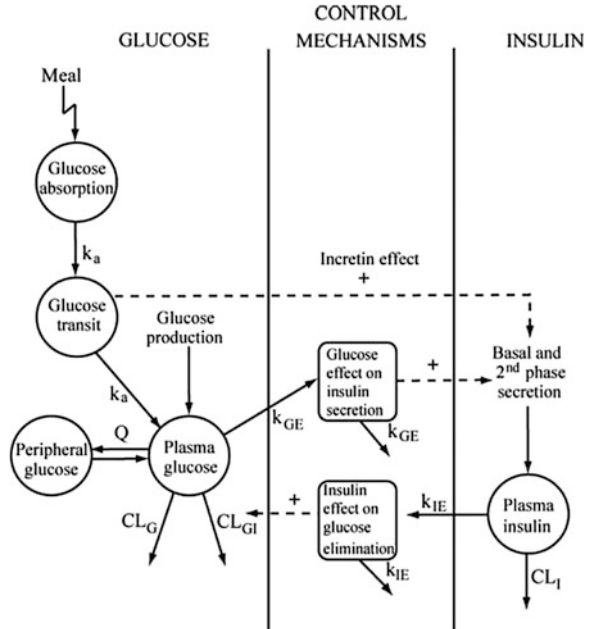
function ins = i(t)
%I interpolate an insulin value
% INS = I(T) computes an insulin value |INS| at time |T|,
% using the linear interpolation given observation times and
% values.

tObs  = [ 0  1  2  5 10 20 29 30 31 35 40 50 100 180]';
insObs = [10 25 100 50 50 50 50 450 200 50 30 10 10 10]';
interpolation = fit(tObs, insObs, 'linearinterp');
ins = feval(interpolation, t);

end

```

**Fig. 5.33** Empirical model of glucose and insulin allowing the simulation of glucose and insulin profiles following multiple meals. The incretin effect is due to glucose gut absorption. *Solid lines* connected to *circles* indicate flows, *dashed lines* control mechanisms



The minimal model described above was extended in many ways. Assessment of model parameters following meal glucose tolerance tests made it applicable to a more familiar route of glucose administration. A further extension was the development of an integrated model for the regulation of glucose and insulin concentrations following intravenous and oral glucose challenge in healthy volunteers and type 2 diabetic patients [31, 32]. This enabled the creation of a model that could simultaneously characterize and simulate 24 h glucose and insulin profiles following multiple meal tests [33] (Fig. 5.33). These models were obtained through simultaneous analysis of data from many individuals (see Chap. 6).

## 5.5 Viral Infection

### 5.5.1 Definition

An infection is the invasion of microorganisms, such as bacteria or viruses, into the human body (host) and their multiplication therein. Pathogenic virus infections cause symptoms such as fever and headache. Laboratory analyses may detect viral titers in specimens such as nasopharyngeal swabs (influenza) or blood (hepatitis C). Viruses can be classified according to their tropism, i.e., the specificity with which they infect body tissues. Thus, influenza viruses find their “receptors” on

the epithelial cells of the respiratory tract, and hepatitis viruses on liver cells. A special feature of viruses is that they can only replicate within the cells of the host as they have no replication apparatus of their own. Viruses replicate in the host cells until no further virus progeny can be created and the host cell dies. New viruses are then released and infect other cells.

### 5.5.2 Viral Kinetics

Figure 5.34 shows a generic conceptual model for viral kinetics (VK). It assumes that before infection occurs, target cells  $T$  are in a dynamic steady state,  $T = r/d$ .

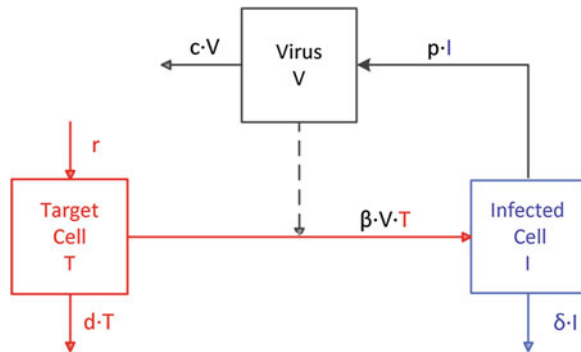
The mathematical description is given below.

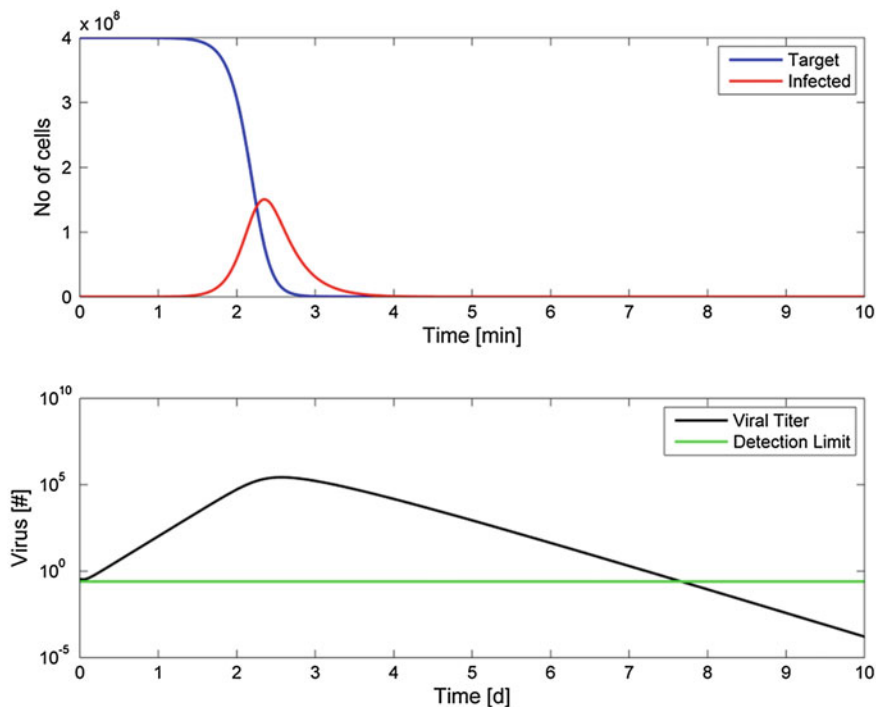
$$\left. \begin{aligned} \frac{dT}{dt} &= r(t) - d \cdot T(t) - \beta \cdot T(t) \cdot V(t) ; & T(0) &= T_0 \\ \frac{dI}{dt} &= \beta \cdot T(t) \cdot V(t) - \delta \cdot I(t) ; & I(0) &= I_0 \\ \frac{dV}{dt} &= p \cdot I(t) - c \cdot V(t) ; & V(0) &= V_0 \end{aligned} \right\} \quad (5.37)$$

Abbreviations, parameter values (units):

- $T(t)$   $\equiv$  Number of tissue cells at time  $t$  (#)
- $I(t)$   $\equiv$  Number of infected cells at time  $t$  (#)
- $V(t)$   $\equiv$  Number of viruses at time  $t$  (#)
- $r(t)$   $\equiv$  Production rate of tissue and infected cells at time  $t$  (#/d)
- $d$   $\equiv$  Elimination rate constant of tissue cells (1/d)
- $\delta$   $\equiv$  Elimination rate constant of infected cells (1/d)
- $c$   $\equiv$  Elimination rate constant of virus (1/d)
- $\beta$   $\equiv$  Infection rate constant (1/d)
- $p$   $\equiv$  Rate of virus increase per infected cell (1/d).

**Fig. 5.34** Generic VK model. *Solid lines* indicate flows of cells and virus, the *dashed line* a control mechanism



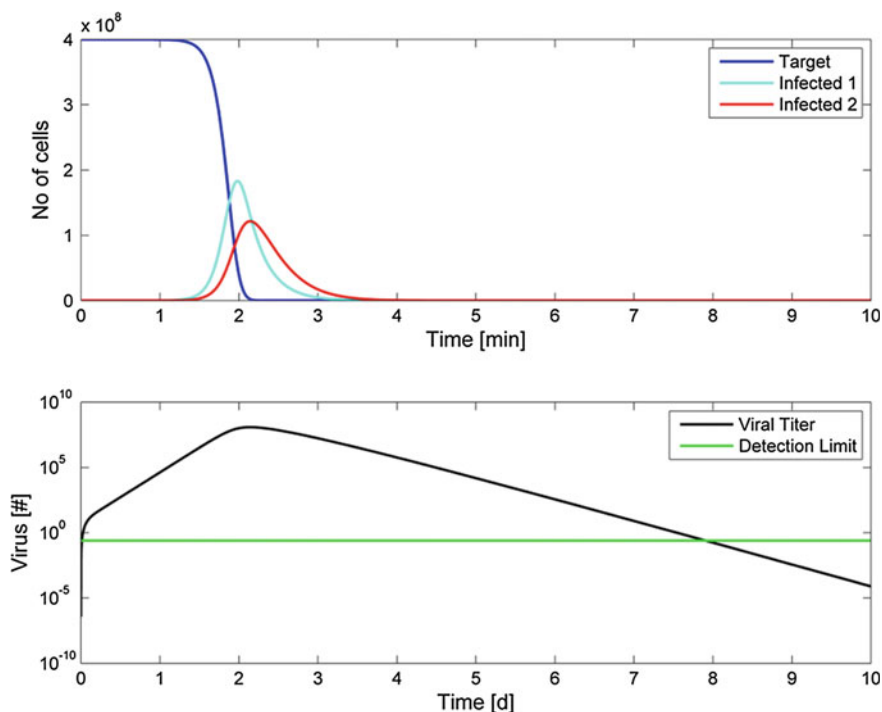


**Fig. 5.35** Influenza virus infection: Time course of target and infected cells (*upper panel*) and viral titer (*lower panel*). Target cell depletion model (Eq. (5.37),  $r = 0$ ,  $d = 0$ )

This model (or slight variations thereof) was applied to describe the viral kinetics of the AIDS-causing human immunodeficiency virus (HIV) [34], the hepatitis-causing hepatitis virus B (HBV) [35], and hepatitis virus C (HCV) [36]. It was also shown that it can be applied to influenza virus A infection [37]. Below, we will describe the application of the generic model to influenza A infections as presented in [37].

The model described in (5.37) and shown in Fig. 5.35 can be made more flexible by adding a second infected cell compartment allowing for a delay in the occurrence of virus-producing infected cells (Fig. 5.36).

$$\left. \begin{aligned} \frac{dT}{dt} &= r(t) - d \cdot T(t) - \beta \cdot T(t) \cdot V(t) ; & T(0) &= T_0 \\ \frac{dI_1}{dt} &= \beta \cdot T(t) \cdot V(t) - k \cdot I_1(t) ; & I_1(0) &= I_{10} \\ \frac{dI_2}{dt} &= k \cdot I_1(t) - \delta \cdot I_2(t) ; & I_2(0) &= I_0 \\ \frac{dV}{dt} &= p \cdot I_2(t) - c(t) ; & V(0) &= V_0 \end{aligned} \right\} \quad (5.38)$$

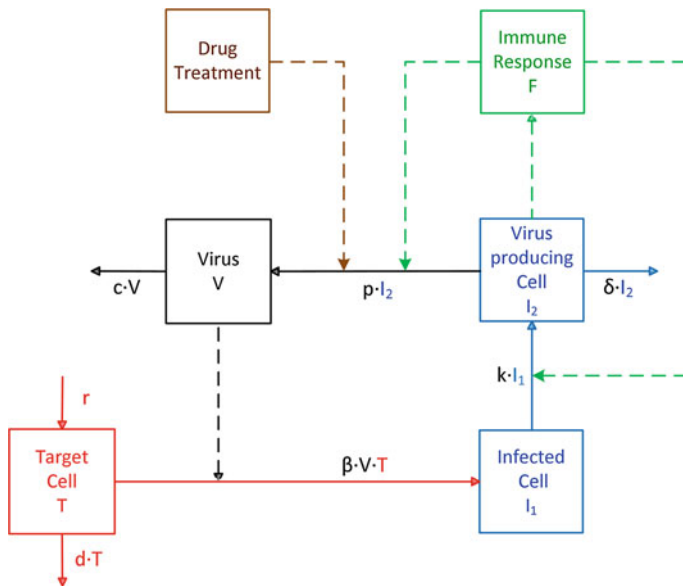


**Fig. 5.36** Influenza virus infection: Time course of target and infected cells (*upper panel*) and viral titer (*lower panel*). Target cell depletion model with delay compartment (Eq. (5.38),  $r = 0$ ,  $d = 0$ )

The viral kinetics describes the time course of the virus without any reaction from the body. In all probability, the immune system will act on the virus either by eliminating infected cells more quickly or by suppressing the release of virus from virus producing cells, etc. Drug treatment could effectively prevent the release of virus from infected cells as is the case for neuraminidase inhibitors.

Figure 5.37 shows how the human immune response or drug treatment could impact on the viral kinetics.

Past influenza infections have shown that it would be highly desirable to have reliable treatment options at hand for the outbreak of a pandemic. Viral kinetics modeling offers an opportunity if it can succeed in (a) linking in vitro viral characteristics (such as initial growth rate) to VK model parameters and (b) finding a relationships between drug dose, drug concentrations at the target site, and drug effect on VK model parameters.



**Fig. 5.37** Viral kinetics, pathophysiology and treatment options. *Solid lines* indicate flows of cells and virus, *dashed lines* control mechanisms

## 5.6 Disease Progression Modeling

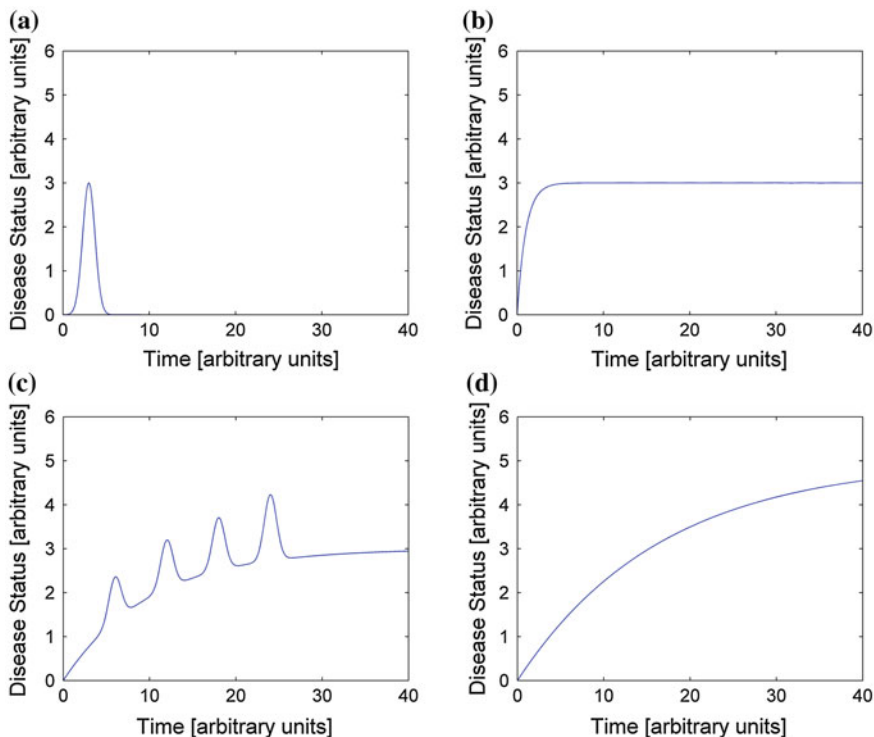
### 5.6.1 General Idea

Disease progression modeling seeks to capture the time course of a disease and hopefully its modification due to drug treatment. Diseases may evolve in different ways. Some are rapidly cured by bodily systems such as the immune system, others repeatedly relapse, such as depression, and again others steadily deteriorate, such as Alzheimer's disease (Fig. 5.38).

Disease progression modeling in the context of PK-PD modeling was first mentioned by Chan [38] and later revisited by Post [8]. Its main application is with chronic disease. If we treat a patient with a chronic disease, we aim to change the untreated disease status trajectory for the better. A simple disease status trajectory follows a linear equation with time:

$$S = S_0 + \alpha \cdot t \quad (5.39)$$

where  $S_0$  is the disease status at baseline and  $\alpha$  the rate of disease progression. Drug treatment may impact on  $S$ , by shifting  $S$  up or down at a given point in time, or on  $\alpha$ , or on both. Treatment effects on  $S$  alone without affecting  $\alpha$  are called symptomatic effects to indicate that the disease progression rate  $\alpha$  did not change. During or by the end of treatment the original disease status may be reached again.



**Fig. 5.38** Examples of time courses of untreated diseases; **a** acute disease with spontaneous remission (e.g. seasonal influenza), **b** stable chronic disease (e.g. cataract), **c** progressive disease with acute episodic attacks (e.g. multiple sclerosis), **d** steadily progressive disease (e.g., Alzheimer's disease)

Disease-modifying treatment effects would change the disease progression rate  $\alpha$  which eventually may result in a true treatment effect. Figure 5.39 shows the treatment effect in case of linear disease progression.

A more general description of disease progression builds on the IDR model (Sect. 4.3.2). A marker of disease status,  $S$ , may be described by the following equation:

$$\frac{dS}{dt} = k_{\text{inp}} - k_{\text{outp}} \cdot S; \quad S(0) = \frac{k_{\text{inp}}}{k_{\text{outp}}} \quad (5.40)$$

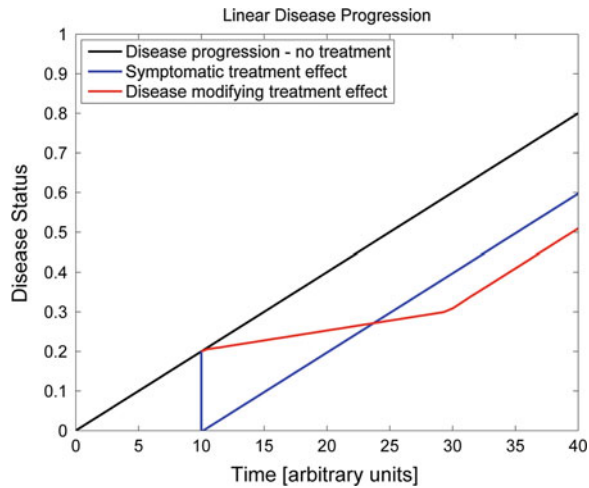
$k_{\text{inp}}$  and  $k_{\text{outp}}$  are time-independent and keep the marker  $S$  at a dynamic steady state around the initial physiologic value  $S(0)$ .

In disease,  $k_{\text{inp}}$  and/or  $k_{\text{outp}}$  become time-dependent. For example:

$$\frac{d}{dt}k_{\text{in}} = -b \cdot k_{\text{in}}; \quad k_{\text{in}}(0) = k_{\text{inp}} \quad (5.41)$$



**Fig. 5.39** Linear disease progression with and without treatment. Treatment may be symptomatic, producing a sudden change in disease status but leaving the disease status trajectory unchanged. Disease-modifying treatment changes the disease status trajectory. In the short-term symptomatic treatment may look more favorable



or

$$\frac{d}{dt}k_{in} = a - b \cdot k_{in} ; \quad k_{in}(0) = k_{inp} \quad (5.42)$$

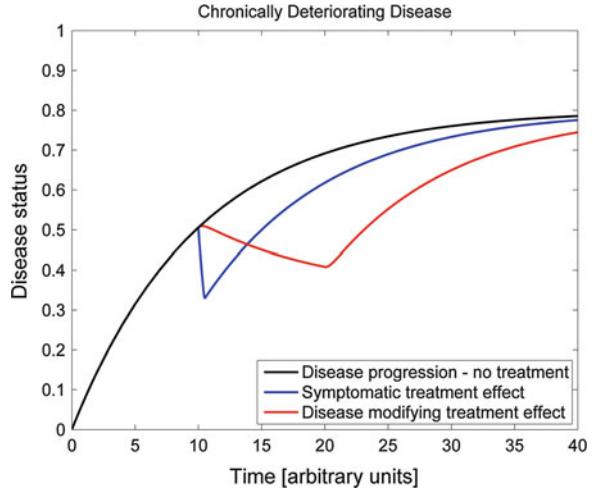
with rate constants  $a$  and  $b$ . Symptomatic treatment effects act on the state variable  $k_{in}$ , whereas protective treatment changes the derivative of  $k_{in}$ . Similar considerations apply for  $k_{out}$ .

Assessment of the disease status requires a quantifiable variable or biomarker that represents the patients' clinical condition. The variable may be continuous (e.g., hemoglobin, insulin sensitivity, viral load, bone mineral density) or categorical (e.g., psychiatric rating scales). Figure 5.40 shows an example of symptomatic and disease-modifying treatment effects for a chronically deteriorating disease.

### 5.6.2 Alzheimer's Disease

Disease progression models for Alzheimer's disease (Alois Alzheimer, 1864–1915) have been published by different authors [39–41]. All rely on a database provided by the Alzheimer's Disease and Neuroimaging Initiative (ADNI). The ADNI database comprises several hundred patients with different degree of cognitive impairment and age-matched controls. Subjects were examined at yearly intervals with respect to *ADAScog*, a clinical measure of cognitive disease status, and other measures over up to 3 years, which is a relatively short time with respect to the duration of the disease.

**Fig. 5.40** Symptomatic and disease-modifying treatment effects for a chronically deteriorating disease. Treatment was administered from time = 10 to time = 20



Ito et al. [39] developed a linear disease progression model

$$ADAScog(t) = ADAScog_0 + \alpha \cdot t \quad (5.43)$$

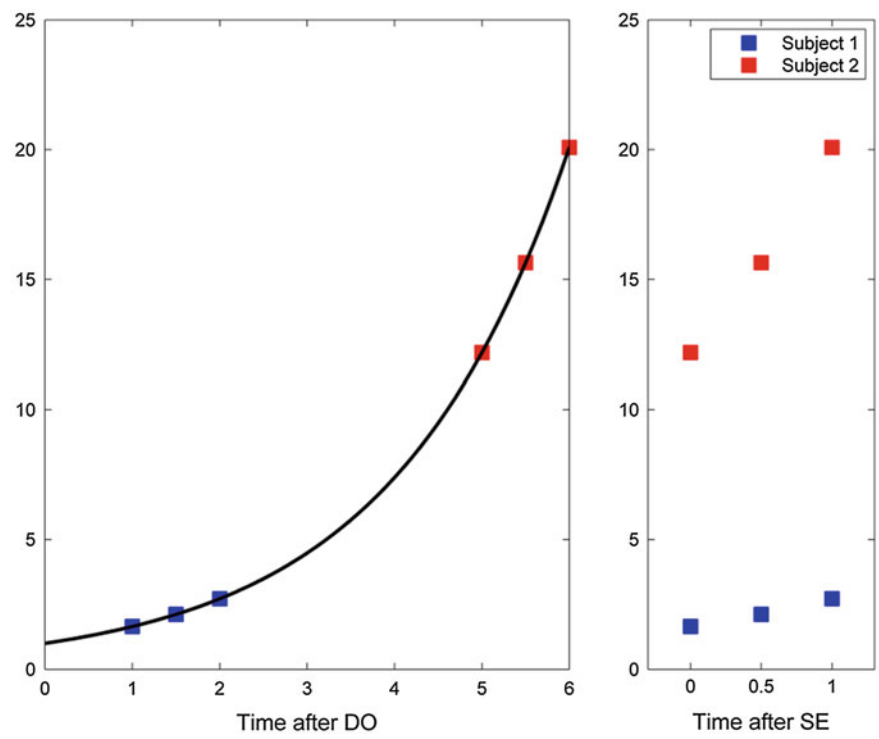
where  $ADAScog_0$  is the value of  $ADAScog$  at baseline (= study entry) and  $\alpha$  the rate of untreated disease progression. Equation (5.43) was fitted to ADNI data using a mixed-effects modeling approach assessing the impact of covariates (sex, gender, age, ...) on disease progression. It turned out that  $ADAScog_0$  was an important covariate on  $\alpha$ .

Based on their review of the ADNI database, Samtani et al. [40] suggested using a nonlinear model to describe disease progression. They proposed the following ODE:

$$\frac{d}{dt}ADAScog = r \cdot ADAScog^\alpha \cdot \left(1 - \frac{ADAScog}{70}\right) \quad (5.44)$$

where parameters  $r$  and  $\alpha$  control the steepness of the  $ADAScog$  time course. Equation (5.44) defines a logistic growth curve with a ceiling at  $ADAScog = 70$ . Given the limited observation period, parameter estimation in (5.44) is not without problems.

A completely different approach was chosen by Yang et al. [41]. Their idea was to synchronize the observed  $ADAScog$  data by introducing a ‘time after disease onset’ which is different from the ‘time after study entry’ (Fig. 5.41). In principle, this would better characterize disease progression.



**Fig. 5.41** Synchronization of ADAScog data observed at time after study entry (SE) (*right panel*) to a new time scale referenced to disease onset (DO) (*left panel*)

## 5.7 Exercises

### Exercise 5.1

Show how to obtain the transformation (5.14)  $\rightarrow$  (5.15), and then check units for correctness, assuming that model parameters are given as in Table 5.6, and voltage  $V$  in mV.

### Exercise 5.2

Assuming that the initial values of gating variables  $n$ ,  $m$ , and  $h$  in (5.15) are unknown, how we can find these values with help of the `hodhux.m` program?

## References

1. Gross CG (1998) Claude Bernard and the constancy of the internal environment. *Neuroscientist* 4:380–385
2. Cannon WB (1963) *The wisdom of the body*. Norton & Company, New York

3. Kunsch K, Kunsch S (2007) *Der Mensch in Zahlen*, 3rd edn. Elsevier, Munich
4. National Cancer Institute. <http://proteomics.cancer.gov/whatisproteomics>. Accessed 15 Mar 2013
5. Alberts B, Johnson A, Lewis J, Raff M, Roberts K, Walter P (2008) *Molecular biology of the cell*, 5th edn. Taylor & Francis, New York
6. Keener J, Sneyd J (2009) *Mathematical physiology I: cellular physiology*, 2nd edn. Springer, New York
7. Keener J, Sneyd J (2009) *Mathematical physiology II: systems physiology*, 2nd edn. Springer, New York
8. Post TM, Freijer JJ, DeJongh J, Danhof M (2005) Disease system analysis: basic disease progression models in degenerative disease. *Pharmaceut Res* 22:1038–1049
9. Hodgkin AL, Huxley AF (1952) A quantitative description of membrane current and its application to conduction and excitation in nerve. *J Physiol* 117:500–544
10. Kandel ER, Schwartz JH, Jessell TM, Siegelbaum SA, Hudspeth AJ (eds) (2012) *Principles of neuronal science*, 5th edn. McGraw-Hill, New York
11. Martin JH (2003) *Neuroanatomy. Text and Atlas*, 3rd edn. McGraw-Hill, New York
12. Purves D, Augustine GJ, Fitzpatrick D, Hall WC, LaMantia AS, McNamara JO, White LE (eds) (2008) *Neuroscience*, 4th edn. Sinauer Associates, Sunderland
13. Bear MF, Connors BW, Paradiso MA (2007) *Neuroscience. Exploring the brain*, 3rd edn. Lippincott William & Wilkins, Philadelphia
14. Hodgkin AL, Huxley AF (1939) Action potential recorded from inside a nerve fibre. *Nature* 144:710–711
15. Clayton PR (2000) *Fundamentals of electric circuit analysis*. Wiley, New York
16. Küpfmüller K, Kohn G (1993) *Theoretische Elektrotechnik und Elektronik*, 14th edn. Springer, Heidelberg
17. Cronin J (2008) *Mathematical aspects of Hodgkin-Huxley neural theory*. Cambridge University Press, Cambridge
18. Gabbiani F, Cox SJ (2010) *Mathematics for neuroscientists*. Academic Press, San Diego
19. Wallisch P, Lusignan M, Benayoun M, Baker TI, Dickey AS, Hatsopoulos NG (2009) *MATLAB for Neuroscientists: an introduction to scientific computing in MATLAB*. Academic Press, San Diego
20. Cooley JW, Dodge FA (1952) Digital computer solutions for excitation and propagation of the nerve impulse. *Biophys J* 6:583–599
21. Larsson S, Thomee V (2005) *Partial differential equations with numerical methods*. Springer, Berlin
22. Carnevale T, Hines M (2009) *The neuron book*. Cambridge University Press, Cambridge
23. Genesis Reference Manual <http://www.genesis-sim.org/GENESIS/Hyperdoc/Manual.html>. Accessed 15 Mar 2013
24. Muratov CB (2000) Quantitative approximation scheme for the traveling wave solutions in the Hodgkin–Huxley model. *Biophys J* 79:2893–2901
25. Scott A (2002) *Neuroscience: a mathematical primer*. Springer, New York
26. Geerts H, Spiros A, Roberts P, Carr R (2013) Quantitative systems pharmacology as an extension of PK/PD modeling in CNS research and development. *J Pharmacokinetic Pharmacodyn*. doi: 10.1007/s10928-013-9297-1. Accessed on 15 Mar 2013
27. Shepherd GM, Grillner S (2010) *Handbook of brain microcircuitry*. Oxford University Press, New York
28. Geerts H, Spiros A, Roberts P, Twyman R, Alphs L, Grace A (2012) Blinded prospective evaluation of computer-based mechanistic schizophrenia disease model for predicting drug response. *PLOS ONE* 7:e49732
29. Garred LJ, Pretlac R (1991) Mathematical modeling of erythropoietin therapy. *ASAIO J* 37:M457–M459
30. Bergman RN, Ider YZ, Bowden CR, Cobelli C (1979) Quantitative estimation of insulin sensitivity. *Am J Physiol* 236:E667–E677

31. Silber HE, Jauslin PM, Frey N, Gieschke R, Simonsson USH, Karlsson MO (2007) An integrated model for glucose and insulin regulation in healthy volunteers and type 2 diabetic patients following intravenous glucose provocations. *J Clin Pharmacol* 47:1159–1171
32. Jauslin PM, Silber HE, Frey N, Gieschke R, Simonsson USH, Jorga K, Karlsson MO (2007) An integrated glucose-insulin model to describe oral glucose tolerance test data in type 2 diabetics. *J Clin Pharmacol* 47:1244–1255
33. Jauslin PM, Frey N, Karlsson MO (2011) Modeling of 24-hour glucose and insulin profiles of patients with type 2 diabetes. *J Clin Pharmacol* 51:153–164
34. Perelson AS, Neumann AU, Markowitz M, Leonard JM, Ho DD (1996) HIV-1 dynamics in vivo: virion clearance rate, infected cell life-span, and viral generation time. *Science* 271:1582–1586
35. Nowak MA, Bonhoeffer S, Hill AM, Boehme R, Thomas HC, McDade H (1996) Viral dynamics in hepatitis B virus infection. *Proc Natl Acad Sci USA* 93:4398–4402
36. Snoeck E, Chanu P, Lavielle M, Jacqmin P, Jonsson EN, Jorga K, Goggin T, Grippo J, Jumbe NL, Frey N (2010) A comprehensive hepatitis C viral kinetic model explaining cure. *Clin Pharmacol Ther* 87:706–713
37. Baccam P, Beauchemin C, Macken CA, Hayden FG, Perelson AS (2006) Kinetics of influenza A virus infection in humans. *J Virol* 80:7590–7599
38. Chan PLS, Holford NHG (2001) Drug treatment effects on disease progression. *Annu Rev Pharmacol Toxicol* 41:625–659
39. Ito K, Corrigan B, Zhao Q, French J, Miller R, Soares H, Katz E, Nicholas T, Billing B, Anziano R, Fullerton T and the Alzheimer's disease neuroimaging initiative (2011) Disease progression model for cognitive deterioration from Alzheimer's Disease. *Alzheimer's Dement* 7:151–160
40. Samtani MN, Farnum M, Lobanov V, Yang E, Raghavan N, DiBernardo A, Narayan V and the Alzheimer's disease neuroimaging initiative (2012) An improved model for disease progression in patients from the Alzheimer's disease neuroimaging initiative. *J Clin Pharmacol* 52:629–644
41. Yang E, Farnum M, Lobanov V, Schultz T, Raghavan N, Samtani MN, Gerald Novak G, Vaibhav Narayan V, DiBernardo A, and the Alzheimer's Disease Neuroimaging Initiative (2011) Quantifying the pathophysiological timeline of Alzheimer's disease. *J Alzheimer's Dis* 26:745–753

## Chapter 6

# Population Analyses

Population analysis is a model-based methodology for assessing parameter variability among individuals in a study population without calculating individual parameter values. It was already established for statistically linear models before it was applied to nonlinear PK-PD models. One of its main goals is to assess how observable patient characteristics (covariates) such as gender and body weight, impact on drug effects, thereby providing an argument (or counter-argument) for “one dose fits all”.

Population analyses or mixed-effects modeling for nonlinear models became practical at the end of the 1970s and have been continuously refined thereafter. Parameters can now be estimated using stochastic algorithms that avoid previous likelihood approximations. The search for covariates became automated and model diagnostic procedures much more extended, as exemplified by visual predictive checks (VPCs).

### 6.1 Terms and Concepts

For the approval of a drug by a regulatory agency, the applicant has to state an indication and a dosage regimen, i.e., how the drug is to be administered (how much, how often, and how long). If applicable, this also includes how the dosage regimen is to be adapted to special patient groups.

For cilazapril (Inhibace<sup>®</sup>), a drug against hypertension, the dosage recommendations are [1]:

The recommended initial dose of INHIBACE is 2.5 mg once daily. Dosage should be adjusted according to blood pressure response, generally, at intervals of at least 2 weeks. The usual dose range for INHIBACE is 2.5–5 mg once daily. A dose of 10 mg should not be exceeded.

Dosage in Elderly Patients (Over 65 years): INHIBACE treatment should be initiated with 1.25 mg (half of a 2.5 mg tablet) once daily or less, depending on the patient's volume status and general condition.

Dosage Adjustment in Renal Impairment: The following dose schedules are recommended in patients with hypertension: Creatinine clearance (CrCl) > 40 mL/min: Initial dose of Inhibace 1 mg once daily, CrCl 10–40 mL/min: 0.5 mg once daily, CrCl < 10 mL/min: not recommended....

For oseltamivir (Tamiflu<sup>®</sup>), a drug against influenza virus infections, the dosage recommendations are [2]:

Adults: Treatment: Begin within 2 days of onset of symptoms. Usual: 75 mg bid for 5 days. CrCl 10–30 mL/min: Usual: 75 mg qd for 5 days....

Thus, for a certain range of CrCl values, the dosing frequency is to be halved.

The information on the usage of a drug under certain conditions, provided in package insert, can be gained through dedicated small-sized studies, e.g., comparing renally healthy with renally impaired subjects, or from large-scale Phase II/III studies, which are done in the population the drug is being developed for (the target population). Population analysis methodology was created to especially address the latter situation. Population PK-PD analyses are done during drug development to quantitatively determine how drug effects are related to drug concentrations and drug administration, and to find those covariates, such as CrCl, that impact on a drug dosage regimen. A related issue is quantifying the sources of between-subject (or interindividual) variability in PK and PD.

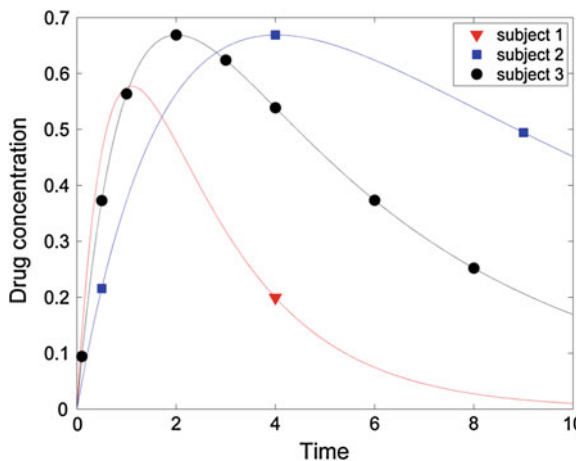
## 6.2 Model-based Analysis of Clinical Data

Sparseness and imbalance are often encountered features of data collected from clinical trials. Sparseness refers to the situation that only very few measurements are available per individual. For example, PK sampling in a Phase III trial is often limited to a few assessments due to practical reasons. Imbalance means that the number of assessments could differ between individuals, some being extensively studied and others sparsely. Another feature is that observations across subjects are not taken at the same (and optimal) times, making statistical summaries of the data more cumbersome. These issues are clearly existent when we try to analyze clinical data from more than one study, as is often the case for a population PK-PD analysis. Model-based population analysis methodology is suitable for combining data from Phase I and Phase II studies with a relatively few subjects and many assessments per subject as well as from Phase III studies with many subjects and few assessments per subject. Thus, it makes no assumption that all subjects have the same rich number of assessments; rather it allows for sparse and unequal observations per subject (Fig. 6.1).

There are different model-based approaches for analyzing of PK-PD data from a population of subjects (see Fig. 6.2).

A first approach, the naive pooled data (NPD) approach, handles the data as if they were from one individual thereby neglecting inter-individual variability. Occasionally, this approach might be suitable for providing initial parameter

**Fig. 6.1** Unbalanced and sparse drug concentration data for three individuals: subject 1 has only one assessment, whereas subject 3 has assessments allowing detailed characterization of drug absorption and elimination



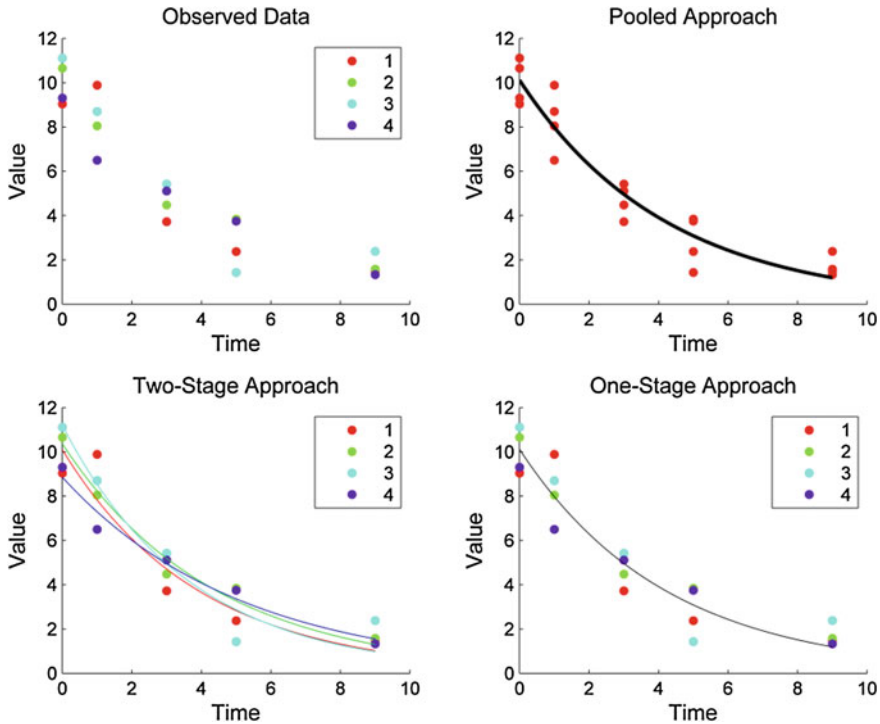
values for more refined approaches. However, it has the potential to generate misleading results, especially under sparse and unbalanced data conditions. A second approach, known as the two-stage (TS) approach, analyzes the data from each subject separately and then calculates statistical measures for the individually estimated parameters, e.g., means and standard deviations. This approach respects the fact that the data are grouped by individual, but relies on enough data per subject to perform respective estimations. In Phase III studies, this might not often be the case. It has also been shown that intersubject variability estimates from the TS approach are biased upward [3]. A third approach, known as the one-stage (OS) approach, uses all data simultaneously but respects the fact that data are grouped by subjects. Estimates from this approach are population parameters, i.e., population “typical” values for model parameters, intersubject variability in model parameters, and intrasubject variability for the residual error(s). A special feature of the OS approach is that it can be applied to observational data which often show different degrees of imbalance and sparseness. This approach has become the gold standard in population PK-PD analysis and will be presented in more detail.

## 6.3 Population Approach

### 6.3.1 Pharmacostatistical Model

The population OS approach, or nonlinear mixed-effects modeling (NLMEM), was introduced in the 1970s by Lewis Sheiner (1940–2004) and Stuart Beal (1941–2006) together with the NONMEM software program [4]. We will present it in the context of PK and PD, although it is more generally applicable.





**Fig. 6.2** Simultaneous analysis of data from several subjects. *Upper left*: observed data in four subjects. *Upper right*: pooled approach—one model is fitted to all data without considering that data are grouped by subjects. *Lower left*: two-stage approach—one model is fitted to each individual data set separately and parameters are summarized. *Lower right*: one-stage approach (population approach)—one model is fitted to all data grouped by subjects

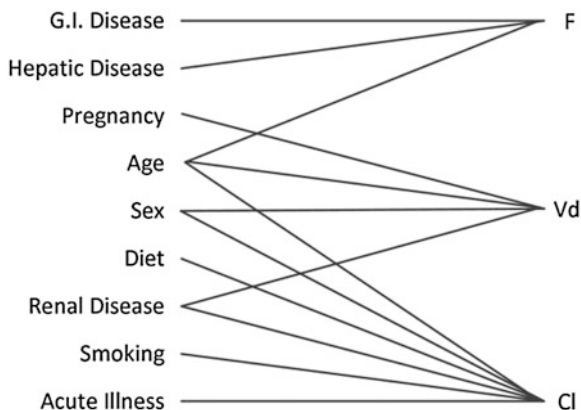
At the heart of a population analysis is a two-level pharmacostatistical model. The first-level or structural model is a parametric PK-PD model which captures the data (dosing, observations) of each individual subject. The second-level or covariate model links the population covariates (e.g., gender and body weight) to the PK-PD model parameters.

The PK-PD model is built according to the explanations in [Chap. 4](#). It describes the data for each single individual according to

$$\mathbf{y}_i = f(\mathbf{x}_i, \boldsymbol{\varphi}_i) + \boldsymbol{\varepsilon}_i \quad (6.1)$$

where  $y_i$  are observed values,  $f$  is a function that calculates model predictions given model predictors  $\mathbf{x}_i$  and model parameters  $\boldsymbol{\varphi}_i$ , and  $\boldsymbol{\varepsilon}_i$  is a random variable with expectation 0 and variance  $\boldsymbol{\Sigma}$  accounting for the differences between observed and predicted values. Subscript  $i$  refers to individual  $i$  and runs from 1 to  $n$ , which is the number of subjects in the population. Common model predictors are drug dose and time after dosing.  $\mathbf{y}_i$ ,  $\mathbf{x}_i$ , and  $\boldsymbol{\varepsilon}_i$  are vectors of length  $n_i$  ( $n_i \geq 1$ ), the number of

**Fig. 6.3** Relationships between covariates and PK parameters [5]



observations for subject  $i$ , which could differ between subjects. Thus, population analyses do not require the same sampling times across subjects.

The covariate model of a population model describes the distribution of the model parameters  $\varphi_i$  in the population. A covariate is an identifiable and (easily) measurable quantity, which accounts for a patient's feature and which may have systematic influence on the PK and/or PD. There are continuous-type covariates (e.g., age, body weight, body surface area, and CrCl) and categorical-type covariates (gender, race, smoker, disease state, metabolic state: fasted vs. fed). Some relationships between covariates and PK parameters have been documented (Fig. 6.3).

The general covariate model has the form

$$\varphi_i = g(\mathbf{z}_i, \boldsymbol{\beta}, \mathbf{b}_i) \quad (6.2)$$

where  $\mathbf{z}_i$  is a subject-specific covariate vector,  $\boldsymbol{\beta}$  a parameter vector, and  $\mathbf{b}_i$  a random vector with covariance  $\boldsymbol{\Psi}$ . The elements of  $\mathbf{z}_i$  are called fixed effects,  $\boldsymbol{\beta}$  is the vector of fixed effects parameters,  $\mathbf{b}_i$  are the random effects, and the elements of  $\boldsymbol{\Psi}$  are the random effects parameters. Often, a more specific form of the function  $g$  is assumed:

$$\varphi_i = \mathbf{A}_i \cdot \boldsymbol{\beta} + \mathbf{B} \cdot \mathbf{b}_i \quad (6.3)$$

or

$$\log(\varphi_i) = \mathbf{A}_i \cdot \boldsymbol{\beta} + \mathbf{B} \cdot \mathbf{b}_i \quad (6.4)$$

which ensures positive parameters for  $\varphi_i$ . The matrices  $\mathbf{A}_i$  and  $\mathbf{B}$  are called design matrices for the fixed and random effects. If there are no covariates included in the population model,  $\mathbf{A}_i$  is the  $n \times n$  identity matrix where  $n$  is the number of PK-PD parameters. The result of a population analysis is described by the triple  $(\boldsymbol{\beta}, \boldsymbol{\Psi}, \boldsymbol{\Sigma})$ .

Consider the following example: a drug's IV pharmacokinetics may be described by a one-compartment model with PK parameters volume of

distribution,  $V$  and clearance,  $CL$ . There may be no covariates affecting the PK parameters. The pharmacostatistical model then has the form:

$$\left. \begin{aligned} \mathbf{y}_i &= Dose \cdot \exp\left(-\frac{CL_i}{V_i} \cdot \mathbf{t}_i\right) + \boldsymbol{\varepsilon}_i \\ \begin{bmatrix} CL_i \\ V_i \end{bmatrix} &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \beta_1 \\ \beta_2 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} b_{CL,i} \\ b_{V,i} \end{bmatrix} \end{aligned} \right\} \quad (6.5)$$

$\mathbf{A}_i$  and  $\mathbf{B}$  are the  $2 \times 2$  identity matrices.

To model the potential effects of body weight on  $CL$  and sex on  $V$ , the pharmacostatistical model is updated as follows:

$$\begin{bmatrix} CL_i \\ V_i \end{bmatrix} = \begin{bmatrix} 1 & 0 & WT_i & 0 \\ 0 & 1 & 0 & SEX_i \end{bmatrix} \cdot \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} b_{CL,i} \\ b_{V,i} \end{bmatrix} \quad (6.6)$$

The fixed effects design matrix contains the individual body weight,  $WT$ , and sex,  $SEX$  (0 for males, 1 for females). The elements of the vector  $\mathbf{A}_i \boldsymbol{\beta}$  are named (population) typical values for the corresponding PK-PD parameters. Each combination of covariate values determines therefore its own set of typical parameter values. Given a set of data including covariates, population analysis means finding functions  $f$ ,  $g$ , and a set of parameters  $(\boldsymbol{\beta}, \boldsymbol{\Psi}, \boldsymbol{\Sigma})$  that optimize a given criterion. Maximum likelihood is a widely accepted criterion for mixed-effects model parameter estimation [6].

### 6.3.2 Estimation Issues in Pharmacostatistical Models

To estimate the parameters of a pharmacostatistical model, the likelihood function for each subject,  $L_i$ , is computed, where  $L_i$  is expressed in the following form:

$$L_i(\boldsymbol{\beta}, \sigma^2, \boldsymbol{\Psi} | \mathbf{y}_i) = p(\mathbf{y}_i | \boldsymbol{\beta}, \sigma^2, \boldsymbol{\Psi}) = \int p(\mathbf{y}_i | \boldsymbol{\beta}, \mathbf{b}_i, \sigma^2) \cdot p(\mathbf{b}_i | \boldsymbol{\Psi}) d\mathbf{b}_i \quad (6.7)$$

The likelihood function over all subjects is then

$$L(\boldsymbol{\beta}, \sigma^2, \boldsymbol{\Psi} | \mathbf{y}) = \prod_{i=1}^m \int p(\mathbf{y}_i | \boldsymbol{\beta}, \mathbf{b}_i, \sigma^2) \cdot p(\mathbf{b}_i | \boldsymbol{\Psi}) d\mathbf{b}_i \quad (6.8)$$

The likelihood function has to be maximized with regard to the parameters  $\boldsymbol{\beta}$ , variance  $\sigma^2$ , and matrix  $\boldsymbol{\Psi}$ . Instead of maximizing  $L$ , it is convenient to minimize the negative logarithm of the likelihood function  $L$ :

$$\min_{\{\boldsymbol{\beta}, \sigma^2, \boldsymbol{\Psi}\}} -\log L = \min_{\{\boldsymbol{\beta}, \sigma^2, \boldsymbol{\Psi}\}} -\sum_{i=1}^m \log \left[ \int p(\mathbf{y}_i | \boldsymbol{\beta}, \mathbf{b}_i, \sigma^2) \cdot p(\mathbf{b}_i | \boldsymbol{\Psi}) d\mathbf{b}_i \right] \quad (6.9)$$

The integral in Eq. (6.7) has in general no closed form expression. To make optimization feasible, different approximations to likelihood as such or to the integral were suggested and implemented in different software. Another approach is to use a Markov (Andrey Markov, 1856–1922) chain based on a stochastic algorithm that provides the maximum likelihood parameters [7].

In the following, we will illustrate step by step how a typical population analysis is done in MATLAB.

## 6.4 Population Analyses in MATLAB

Population analyses in MATLAB require the Statistics Toolbox or the System Biology Toolbox. Here we demonstrate the use of the Statistics Toolbox. The basic functions for NLMEM are **nlmefit** and **nlmefitsa**. The main difference between the two is how they deal with the likelihood calculation. **nlmefit** implements different approximations for calculating of likelihood, and **nlmefitsa** uses a stochastic approximation algorithm based on Markov chains. In the following, we will illustrate some features of **nlmefit** and **nlmefitsa** using simulated data.

### 6.4.1 Data

The data were generated from the pharmacostatistical model described in (6.10). This model is the same as described above in (6.6), with a few modifications: the structural model uses parameters  $V$  and  $ke$ , and a proportional error term, while the covariate model is linked to log-transformed PK parameters.

$$\left. \begin{aligned} y_i &= \frac{Dose}{V_i} \cdot \exp(-ke_i \cdot t_i) \circ (1 + \varepsilon_i) \\ \log \begin{bmatrix} V_i \\ ke_i \end{bmatrix} &= \begin{bmatrix} 1 & 0 & WT_i & 0 \\ 0 & 1 & 0 & SEX_i \end{bmatrix} \cdot \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} b_{V,i} \\ b_{ke,i} \end{bmatrix} \end{aligned} \right\} \quad (6.10)$$

The symbol  $\circ$  means the element-wise multiplication of two vectors. The second equation in (6.10) can be expressed more conveniently as

$$\left. \begin{aligned} V_i &= \exp(\beta_1 + \beta_3 \cdot WT_i + b_{V,i}) \\ ke_i &= \exp(\beta_2 + \beta_4 \cdot SEX_i + b_{ke,i}) \end{aligned} \right\} \quad (6.11)$$

Table 6.1 contains the parameters which were chosen to generate the data.

*WT* was drawn from a uniform distribution between 40 and 60 kg and then centered around 50, and *SEX* was drawn from a binomial distribution with  $p = 0.5$ ; 0 coding for male, and 1 for female.

Listing 6.1 shows how the data are generated in MATLAB.

**Listing 6.1 Program `nlmeData.m`**

```
function nlmeData
%NLMEData Create a dataset with simulated data
%   NLMEData generates for a NLMEM analysis. The body weight
%   distribution wtc is assumed to be uniform between 40 and 60,
%   centered around 50. sex has a binomial distribution with p=0.5;
%   0 coding for male, and 1 for female. The variability of the model
%   parameters id log-normal distributed, and the residuals - normal
%   distributed.

rng(0,'twister') % for reproducibility
n      = 40;
times  = 0.5*[0; 1; 4; 8; ];
beta0  = log([10 0.8]); % V ke
psi     = [0.3 0; 0 0.1];
% random effects
xb      = mvnrnd(beta0,psi,n);
wtc     = 40+20*rand(n,1)-50; %centered weight
sex     = binornd(1,0.5,n,1);
xb(:,1) = xb(:,1)+0.2*wtc;
xb(:,2) = xb(:,2)+0.9*sex;
phi     = exp(xb);
reserr  = 0.1;
groups  = repmat(1:n,numel(times),1);
groups  = vertcat(groups(:));
y       = zeros(n*numel(times),1);
x       = zeros(n*numel(times),1);
for i=1:n
    idx  = groups == i;
    f     = model(phi(i,:),times);
    % For ODE model form use the function: modelODE(...)
    y(idx) = f + f*reserr.*randn(numel(times),1);
    x(idx) = times;
end
xx1 = repmat(wtc,1,length(times))' %#ok<NOPRT>
xx2 = repmat(sex,1,length(times))'  %#ok<NOPRT>
Data = [groups x y vertcat(xx1(:)) vertcat(xx2(:))] %#ok<NOPRT>
dlmwrite('nm.dat',Data,'delimiter','\t', 'precision',3)
save('nmdata.mat','Data')

end

function pred = model(phi,t)
%MODEL compute model prediction values from the analytical form
%   PRED = MODEL(PHI,T) calculates the predicted values |PRED| at
%   observation points |T|, given model parameters |PHI|

pred = 100./phi(1)*exp(-phi(2).*t);

end
```

```
function pred = modelODE(phi,t)    %#ok<DEFNU>
%MODELODE compute model prediction values from the ODE model form
%   PRED = MODELODE(PHI,T) calculates the predicted values
%   |PRED| at observation points |T| and given model parameters |PHI|

options = odeset('RelTol',10-9,'AbsTol',10-9);
sol = ode45(@ode,[0,10],100./phi(1),options);
pred = deval(sol,t)';

    % Call the ODE model
    function dydt=ode(~,y)

        dydt = -phi(2)*y;
    end
end
```

**Table 6.1** Parameters to generate data for a population analysis

Item	Symbol	Value
Number of subjects	$n$	40
Observation times	$t$	[0; 0.5; 2; 4]
Dose	$Dose$	100
Clearance	$\beta_1$	0.8
Volume of distribution	$\beta_2$	10
Effect of <i>WT</i> on $V$	$\beta_3$	0.2
Effect of <i>SEX</i> on $ke$	$\beta_4$	0.9
Intersubject variability = $\text{var}([b_1,b_2])$	$\Psi$	[0.3 0; 0 0.1]
Residual variability = $\text{var}(\varepsilon)$	$\sigma^2$	0.05

The tabulated output of the `nlmeData.m` program is shown for three subjects in Table 6.2.

6.4.2 Exploratory Analysis

It is good practice to graphically explore the data before investing more resources. Data incompatibilities may be detected requiring resolution before a proper NLMEM analysis is started. Listing 6.2 contains the MATLAB implementation for the exploratory analysis of the simulated data.

Figure 6.4 suggests an effect of sex on drug elimination. More refined exploration shows this very clearly (Fig. 6.5).

To check a potential effect of body weight on volume of distribution, maximum drug concentrations were plotted against body weight (Fig. 6.6).

**Listing 6.2** Program **explore.m**

```

function explore()
%EXPLORE Conduct exploratory data analysis
% FUNCTION EXPLORE() reads raw data from a population and creates
% graphs for exploratory data analysis

close all;
clear; clc;
nmData = load('nmData');

id = nmData.Data(:,1);
time = nmData.Data(:,2);
conc = nmData.Data(:,3);
wt = nmData.Data(:,4);
sex = nmData.Data(:,5);
[id time conc wt sex]          %#ok<NOPRT>    % show data
unique(id) '                   %#ok<NOPRT>    % show subjects

% individual profiles
fig1 = figure;
for i = 1:20;                  %length(unique(id));
    subplot(4,5,i)
    idx = id == i;
    sexi = sex(idx);
    if sexi == 1
        psb = '-ro';
    else
        psb = '-b*';
    end
    semilogy(time(idx),conc(idx),psb)
    axis([0 4 10^-5 10^5])
    title(['ID=',num2str(i)])
end
[~,hx]=suplabel('Time');      set(hx,'FontSize',14)
[~,hy]=suplabel('Drug Concentration','y'); set(hy,'FontSize',14)
[~,ht]=suplabel(['Individual Observations',10,10], 't');
set(ht,'FontSize',16)
print(fig1, '-r900', '-dtiff', 'explore1');

fig2 = figure;
for i = 1:length(unique(id));
    idx = id == i;
    sexi = sex(idx);
    if sexi == 1
        psb = '-ro';
    else
        psb = '-b*';
    end
    ax(i) = subplot(1,2,sexi+1); %#ok<AGROW>
    semilogy(time(idx),conc(idx),psb)
    axis([0 4 10^-5 10^5])
    title(['SEX=',num2str(sexi(1))], 'FontSize',15)
    xlabel('Time', 'FontSize',15)
    hold on
end

```

```
set(ax(1), 'FontSize', 14); set(ax(2), 'FontSize', 15)
print(fig2, '-r900', '-dtiff', 'explore2');

maxc = grpstats([conc sex], id, @max);
wt = grpstats(wt, id, @max);
fig3 = figure; set(axes, 'FontSize', 15)
plot(wt(maxc(:,2)==0)+50, maxc(maxc(:,2)==0), '*b', 'MarkerSize', 7)
hold on
plot(wt(maxc(:,2)==1)+50, maxc(maxc(:,2)==1), 'or', 'MarkerSize', 7)
xlabel('Body Weight [kg]')
ylabel('Cmax [\mug/L]')
legend('male', 'female')
print(fig3, '-r900', '-dtiff', 'explore3');

end
```

6.4.3 Pharmacostatistical Model Development

When developing a pharmacostatistical model, we generally start from a model without covariates. According to Fig. 6.4, there is only one disposition phase discernible. Thus, we start with a one-compartment model (Sect. 4.1) and do not consider covariates for the moment.

The call to the **nlmefit** function is

```
[<beta>, <PSI>, <stats>, <b>] = nlmefit(<X>, <y>,
    <group>, <G>, @<fun>, <beta0>);
```

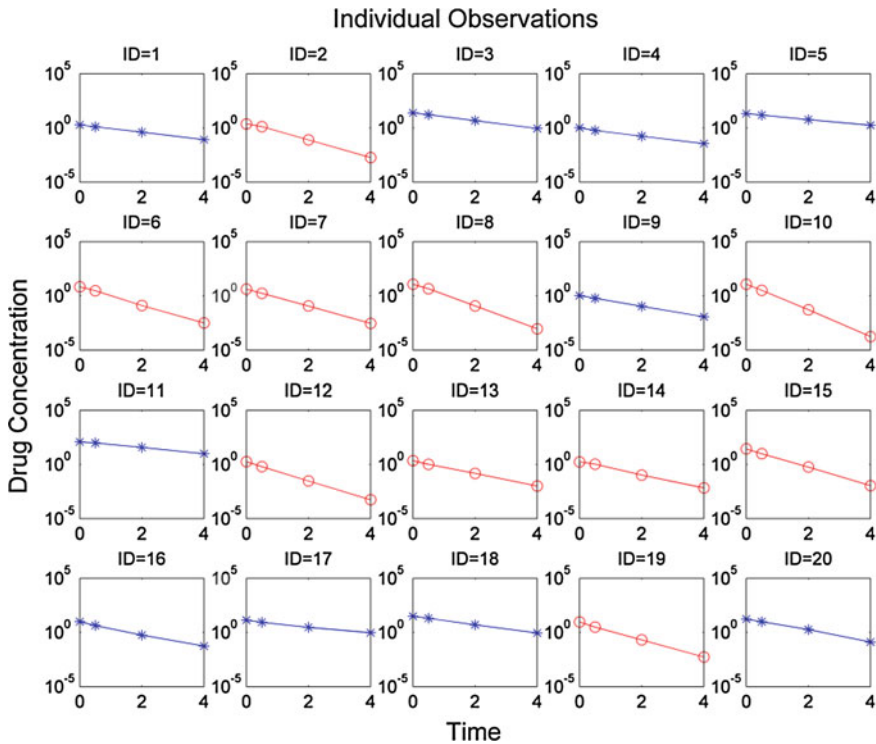
<X> is a matrix of predictors for calculating of model-predicted responses, <y> are the observed responses, <group> is a vector that determines the partitioning of observations into groups, <G> is a matrix of group-specific predictors, <fun> is a function handle returning model predictions from model parameters

Table 6.2 Common data structure for an NLMEM analysis

ID	TIME	DOSE	CONC	WTc	SEX
1	0	100	1.94	6.62	0
1	0.5		1.27	6.62	0
1	2		0.409	6.62	0
1	4		0.0819	6.62	0
2	0	100	2.31	1.71	1
2	0.5		1.31	1.71	1
2	2		0.0792	1.71	1
2	4		0.00183	1.71	1
3	0	100	24.7	0.994	0
3	0.5		16.6	0.994	0
3	2		4.73	0.994	0
3	4		0.884	0.994	0

TIME, time after dosing (h); DOSE, drug amount (mg); CONC, drug concentrations (mg/L); WTc, body weight centered around 50 (kg); SEX: 0-male, 1-female





**Fig. 6.4** Exploratory analysis (I): drug concentrations versus time profiles, sex indicated by color, *blue* = male, *red* = female (20 out of 40 profiles)

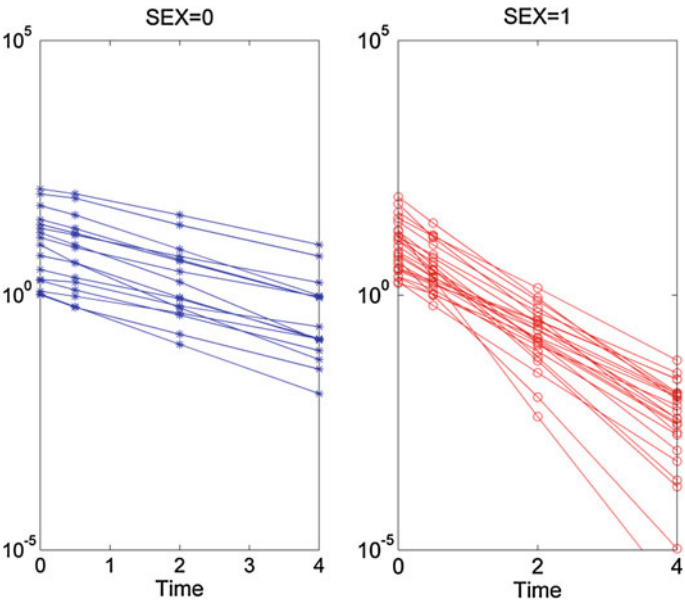
and predictor values, and  $\langle \text{beta0} \rangle$  is a vector with initial estimates for the fixed effects. Without the consideration of covariates,  $\langle \text{beta0} \rangle$  has the same number of elements as there are model parameters. The function returns:  $\langle \text{beta} \rangle$ —fixed effects parameter estimates;  $\langle \text{PSI} \rangle$ —covariance matrix for random effects;  $\langle \text{stats} \rangle$ —a structure containing information about the maximized log-likelihood, error variance, Akaike and Bayesian information criteria, covariance matrix and errors for the estimates, and residuals;  $\langle \text{b} \rangle$ —fixed effects estimates.

The function handle  $\text{@}\langle \text{fun} \rangle$  has the form

```
 $\langle \text{ypred} \rangle = \langle \text{fun} \rangle(\langle \text{PHI} \rangle, \langle \text{XFUN} \rangle, \langle \text{VFUN} \rangle);$ 
```

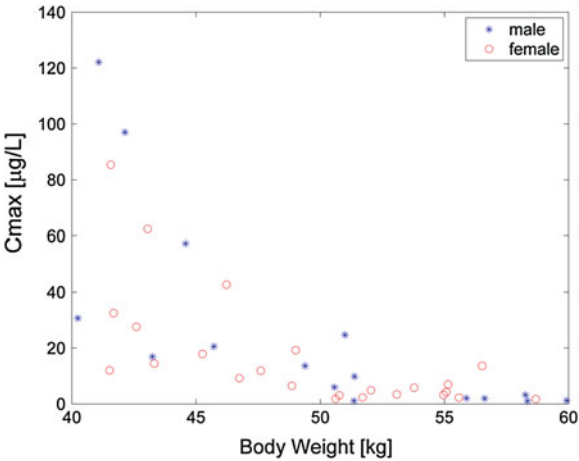
The arguments are:  $\langle \text{PHI} \rangle$ , a column vector of model parameters;  $\langle \text{XFUN} \rangle$ , an array of predictors; and  $\langle \text{VFUN} \rangle$ , an array of group-specific predictors.  $\langle \text{ypred} \rangle$  is a vector of predicted values.

In our example,  $\langle \text{X} \rangle := \text{time}$  is the column vector of all observation times,  $\langle \text{y} \rangle := \text{conc}$  the column vector of all observations, and  $\langle \text{group} \rangle := \text{id}$  the column vector of subject identifiers,  $\langle \text{fun} \rangle := \text{modelana}$  is our function that return model predictions. All these vectors have the same length.  $\langle \text{G} \rangle := \text{dose}$  is a  $1 \times 40$  column vector containing the administered dose for each subject, in this



**Fig. 6.5** Exploratory analysis (II): logarithmic drug concentrations versus time profiles; impact of sex on drug elimination (half-life)

**Fig. 6.6** Exploratory analysis (III): body weight versus observed maximum drug concentration



example 100 mg. The function `<fun>` can be described in two ways, either algebraically or as an ODE. We will use both methods. It will turn out that the algebraic solution runs much faster than the ODE version. However, for more complex problems, there might be no algebraic solution or it might be too cumbersome to develop one.

The first function we define handles the calculation of the predictions given individual parameters and subject-specific information, such as the applied dose. There are two functions defined, the first, function **modelana**, for the analytical solution of the problem, and the second, function **modelODE**, for the numerical solution.

### Listing 6.3 Function **modelana**

```
function pred = modelana(phi,t,G)
%MODELANA Compute prediction values from the analytical model form.
%   PRED = MODELANA(PHI,T,G) calculates the predicted values values
%   |PRED| at observation points |T|, given model parameters |PHI|
%   and group specific predictor |G|. The prediction uses the
%   analytical model form.

dose = G;
pred = dose./phi(1)*exp(-phi(2).*t);

end
```

### Listing 6.4 Function **modelODE**

```
function pred = modelODE(phi,t,G)          %#ok<DEFNU>
%MODELODE Compute model prediction values from the ODE model form
%   PRED = MODELODE(PHI,T,G) calculates the predicted values |PRED|
%   at observation points |T|, given model parameters |PHI| and group
%   and group specific predictor |G|. The prediction uses the
%   ODE model form.

dose = G;
options = odeset('RelTol',10^-3,'AbsTol',10^-3);
sol = ode113(@ode, [0,10], dose./phi(1), options);
pred = deval(sol,t);

% Call the ODE model
function dydt=ode(~,y)
    dydt = -phi(2)*y;
end
end
```

The model predictions are then calculated by the **nlmeBaseAnalysis.m** program which prepares the data for the call of **nlmefit**, and conducts a basic population analysis (covariates are not included).

**Listing 6.5** Program **nlmeBaseAnalysis.m**

```

function nlmeBaseAnalysis
%NLMEBASEANALYSIS Conduct a basic population analysis.
%   NLMEBASEANALYSIS computes the model estimates without the
%   consideration of covariates affecting the PK parameters.
%   It also produces some basic graphs of goodness of fit and
%   visual predictive checks.

clear; clc; close all;
rng(0,'twister')          % for reproducibility
tic
nmData = load('nmData');
Data   = nmData.Data;
id      = Data(:,1);
time    = Data(:,2);
conc    = Data(:,3);
wt      = Data(:,4);
sex     = Data(:,5);
n       = length(unique(id));
dose    = 100*ones(n,1);
wt      = grpstats(wt,id,@min);
sex     = grpstats(sex,id,@min);

%prepare input for nlmefit
nphi = 2;                %number of PK parameters
P = [ 1 0; 0 1];         %covariance pattern for random effects
%fixed effects matrix
A = repmat(eye(nphi,nphi),[1,1,n]);          % NO COVARIATES
%                                           % NO COVARIATES
%                                           % NO COVARIATES
b0 = [1; 1];              % NO COVARIATES

options = statset('OutputFcn',@nlmefitoutputfcn);

% Calling Nonlinear Mixed-Effects (NLME) regression mode
[beta,PSI,stats,b] = nlmefit(time,conc,id,dose,@modelana,b0, ...
    'Options', options, ...
    'REParamsSelect',[1 2], ...
    'ApproximationType', 'LME', ...
    'ErrorModel','Proportional', ...
    'CovPattern',P, ...
    'OptimFun','fminunc', ...
    'ParamTransform',[1 1], ...
    'FeGroupDesign',A)    %#ok<NOPRT>

% Calling Stochastic Approx. Expectation-Maximization mode (SAEM)
[beta,PSI,stats,b] = nlmefitsa(time,conc,id,dose,@modelana,b0, ...
%   'Options', options, ...
%   'REParamsSelect',[1 2], ...
%   'LogLikMethod', 'is', ...
%   'ErrorModel','Proportional', ...
%   'CovPattern',P, ...
%   'OptimFun','fminunc', ...
%   'ParamTransform',[1 1], ...
%   'FeGroupDesign',A)    %#ok<NOPRT>

```

```

beta          %#ok<NOPRT>
exp(beta)
PSI           %#ok<NOPRT>
stats         %#ok<NOPRT>
stats.covb
%print(figure(1), '-r600', '-dtiff', 'baseiterations')
toc
tic
predTime(id,time,conc,beta,b,wt,sex)
predObs(id,time,conc,beta,b,wt,sex)

nrep = 1000;
vpc(nrep,n,time,conc,beta,PSI,stats.errorparam,wt,sex)
toc

end

function pred = modelvpc(phi,t,G,error)
%MODELVPC Compute model prediction values for the VPC
%   PRED = MODELVPC(PHI,T,G,ERROR) calculates predicted values
%   |pred| at observation points |T|, given model parameters |phi|,
%   and group specific predictor |G|. Residual error |error| is
%   assumed. The values are needed for the visual prediction check

%   (VPC)

dose = G;
pred = dose./phi(1)*exp(-phi(2).*t) + ...
       dose./phi(1)*exp(-phi(2).*t)*error;
end

function predTime(id,time,conc,beta,b,~,~)
%PREDTIME Create individual graphs for the goodness of fit.
%   PREDTIME(ID,TIME,CONC,BETA,B,WT,SEX) generates individual graphs
%   with time courses (observations, population and individual
%   predictions) for the goodness of fit purpose. The input:
%   |ID| - subject numbers; |TIME| - observation times;
%   |CONC| - observed drug concentrations; |BETA| - model estimates;
%   |B| - random effect values; |WT|,|SEX| - weight and sex as
%   covariates.
%
%   This function uses suplabel from MATLAB Central:
%   http://www.mathworks.ch/matlabcentral/fileexchange/7772-suplabel
%   Copyright (c) 2004, Ben Barrowes
%   All rights reserved.
%   Code covered by the BSD License: http://www.mathworks.com/
%   matlabcentral/fileexchange/view_license?file_info_id=7772

display 'predTime'
n = length(unique(id));
d = repmat(beta(1:2), 1, n);
qq1 = d + [b(1,:); b(2,:)];
qq2 = qq1*0;                                % NO COVARIATES
qq = qq1 + qq2;
pkpar = exp(qq);
solu = exp(d+qq2);

```

```

fig1 = figure;
for i=1:20
    subplot(5,4,i)
    semilogy(time, modelana(pkpar(:,i),time, 100),'-g')
    hold on
    semilogy(time(id==i),conc(id==i),'o');
    semilogy(time,modelana(solu(:,i),time,100),'-r')
    axis([0 6 10^-3 10^2])
end
[~,hx]=suplabel('Time'); set(hx,'FontSize',14)
[~,hy]=suplabel('Drug Concentration','y'); set(hy,'FontSize',14)
[~,ht]=suplabel('Individual Time Courses','t'); set(ht,'FontSize',18)

print(fig1,'-dtiff','-r900','basefig1')

end

function predObs(id,time,conc,beta,b,~,~)
%PREDOBS Create a basic goodness of fit
% PREDOBS(ID,TIME,CONC,BETA,B,WT,SEX) generates a graph, observed
% vs. predicted values) for the goodness of fit purpose.
% The input: |ID| - subjects numbers; |TIME| - observation times;
% |CONC| - observed drug concentrations; |BETA| - model estimates;
% |B| - random effect values; |WT|,|SEX| - weight and sex as
% covariates.

display 'predObs'
fig2 = figure;
n = length(unique(id));
d = repmat(beta(1:2), 1, n);
qq1 = d + [b(1,:); b(2,:)];
qq2 = qq1*0; % NO COVARIATES
qq = qq1 + qq2;
pkpar = exp(qq);
solu = exp(d+qq2);
set(axes,'FontSize',15);
for i=1:n
    hp = loglog(modelana(solu(:,i),time(id==i),100), ...
        conc(id==i),'r*');
    hold on
    hi = loglog(modelana(pkpar(:,i),time(id==i),100), ...
        conc(id==i),'g*');
end
legend([hp hi], 'population prediction','individual prediction')
xlabel('Predicted')
ylabel('Observed')
print(fig2,'-dtiff','-r900','basefig2')

end

```

Table 6.3 summarizes the results generated by the **nlmeBaseAnalysis.m** program.

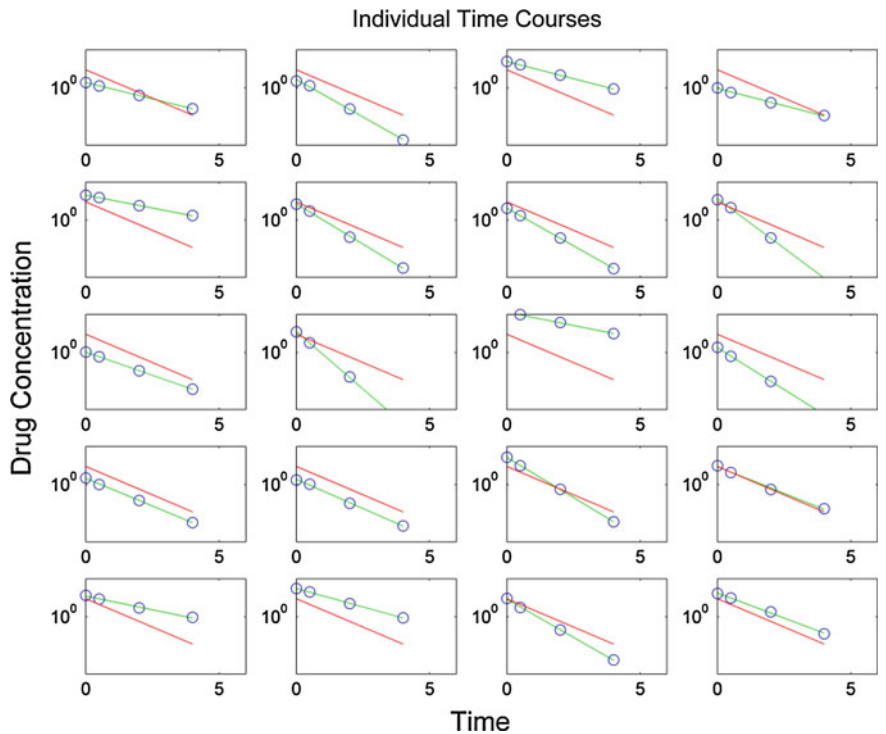
A graphical representation of the results shows that individual predictions nicely matched observations, but that population predictions did not (Figs. 6.7, 6.8).

The red stripes in Fig. 6.8 refer to population predictions at times of measurement (0, 0.5, 2, 4 h). Without the inclusion of covariates, each subject has the same population PK parameters yielding the same population predictions.

**Table 6.3** Summary of results using the `nlmeBaseAnalysis.m` program in 40 subjects

Fixed effects parameters	Value	SE	BSV
$\beta_1$	2.4015	0.206	1.6883
$V [=exp(\beta_1)]$	11.04		
$\beta_2$	0.3174	0.081	0.2609
$ke [=exp(\beta_2)]$	1.37		
$\sigma^2$	0.099		
$\log L$	-58.7948		

SE—standard error of estimate, BSV—between-subject variability (variance of random effects), Log  $L$ —logarithm of likelihood (no covariates were considered)



**Fig. 6.7** Base analysis (no covariates included): population prediction (*solid red line*), individual prediction (*solid green line*) and observations (*circles*) against time (20 out of 40 profiles)

**6.4.4 Covariate Inclusion**

The next step in the analysis is the inclusion of covariates in the pharmacostatistical model (see Listing 6.6). According to our exploratory analysis, we will add covariate effects of sex on the elimination rate constant and of body weight on the volume of distribution. A more systematic approach in the light of empirical data

analysis would be to test all covariates on all model parameters. In order to make this practical, some automation would be required.

**Listing 6.6** Program `nlmeFinalAnalysis.m`. The changes needed to include covariates are indicated as **% INCLUDE COVARIATES**

```
function nlmeFinalAnalysis
%NLMEFINALANALYSIS Conduct a final population analysis.
% NLMEFINALANALYSIS computes the model estimates taking into
% account the potential effects of the population covariates
% (gender, body weight) It also produces some basic graphs of
% goodness of fit and visual predictive check.

clear; clc; close all;
rng(0,'twister')           % for reproducibility
tic
nmData = load('nmData');
Data = nmData.Data;
id = Data(:,1);
time = Data(:,2);
conc = Data(:,3);
wt = Data(:,4);
sex = Data(:,5);
n = length(unique(id));
dose = 100*ones(n,1);
wt = grpstats(wt,id,@min);
sex = grpstats(sex,id,@min);

%prepare input for nlmeFit
nphi = 2;                  %number of PK parameters
P = [ 1 0; 0 1];           %covariance pattern for random effects
%fixed effects matrix
A = repmat(eye(nphi,nphi+2),[1,1,n]); % INCLUDE COVARIATES
A(1,3,:) = wt(:);          % INCLUDE COVARIATES
A(2,4,:) = sex(:);         % INCLUDE COVARIATES
b0 = [1; 1; 1; 1];        % INCLUDE COVARIATES

options = statset('OutputFcn',@nlmeFitOutputFcn);

% Calling Nonlinear Mixed-Effects (NLME) regression mode
[beta,PSI,stats,b] = nlmeFit(time,conc,id,dose,@modelana,b0, ...
    'Options', options, ...
    'REParamsSelect',[1 2], ...
    'ApproximationType', 'LME', ...
    'ErrorModel','Proportional', ...
    'CovPattern',P, ...
    'OptimFun','fminunc', ...
    'ParamTransform',[1 1], ...
    'FeGroupDesign',A) %#ok<NOPRT>

% Calling Stochastic Approx. Expectation-Maximization mode (SAEM)
% [beta,PSI,stats,b] = nlmeFitSA(time,conc,id,dose,@modelana,b0, ...
% 'Options', options, ...
% 'REParamsSelect',[1 2], ...
% 'LogLikMethod','is', ...
% 'ErrorModel','Proportional', ...
% 'CovPattern',P, ...
% 'OptimFun','fminunc', ...
% 'ParamTransform',[1 1], ...
% 'FeGroupDesign',A) %#ok<NOPRT>
```



```

beta          %#ok<NOPRT>
exp(beta)
PSI           %#ok<NOPRT>
stats         %#ok<NOPRT>
stats.covb
print(figure(1), '-r900', '-dtiff', 'iterations')
toc
tic
predTime(id,time,conc,beta,b,wt,sex)
predObs(id,time,conc,beta,b,wt,sex)

nrep = 1000;
vpc(nrep,n,time,conc,beta,PSI,stats.errorparam,wt,sex)
toc
end

function pred = modelvpc(phi,t,G,error)
%MODELVPC Compute model prediction values for the VPC
%   PRED = MODELVPC(PHI,T,G,ERROR) calculates predicted values
%   |pred| at observation points |T|, given model parameters |phi|,
%   and group specific predictor |G|. Residual error |error| is
%   assumed. The values are needed for the visual prediction check
%   (VPC)

dose = G;
pred = dose./phi(1)*exp(-phi(2).*t) + ...
      dose./phi(1)*exp(-phi(2).*t)*error;

end

function predTime(id,time,conc,beta,b,wt,sex)
%PREDTIME create individual graphs for the goodness of fit.
%   PREDTIME(ID,TIME,CONC,BETA,B,WT,SEX) generates individual graphs
%   with time courses (observations, population and individual
%   predictions) for the goodness of fit purpose. The input:
%   |ID| - subject numbers; |TIME| - observation times;
%   |CONC| - observed drug concentrations; |BETA| - model estimates;
%   |B| - random effect values; |WT|,|SEX| - weight and sex as
%   covariates.
%
%   This function uses suplabel from MATLAB Central:
%   http://www.mathworks.ch/matlabcentral/fileexchange/7772-suplabel
%   Copyright (c) 2004, Ben Barrowes
%   All rights reserved.
%   Code covered by the BSD License: http://www.mathworks.com/
%   matlabcentral/fileexchange/view_license?file_info_id=7772

display 'predTime'
n = length(unique(id));
d = repmat(beta(1:2), 1, n);
qq1 = d + [b(1,:); b(2,:)];
qq2 = [beta(3)*wt, beta(4)*sex]';           % INCLUDE COVARIATES
qq = qq1 + qq2;
pkpar = exp(qq);
solu = exp(d+qq2);

```

```

figTimeCourse = figure;
for i=1:20
    subplot(5,4,i)
    semilogy(time, modelana(pkpar(:,i),time,100),'-g')
    hold on
    semilogy(time(id==i),conc(id==i),'o');
    semilogy(time,modelana(solu(:,i),time,100),'-r')
    axis([0 6 10^-3 10^2])
end
[~,hx]=suplabel('Time'); set(hx,'FontSize',14)

[~,hy]=suplabel('Drug Concentration','y'); set(hy,'FontSize',14)
[~,ht]=suplabel('Individual Time Courses','t'); set(ht,'FontSize',18)

print(figTimeCourse,'-dtiff','-r900','finalTimeCourse')
end

function predObs(id,time,conc,beta,b,wt,sex)
%PREDOBS create a basic goodness of fit
% PREDOBS(ID,TIME,CONC,BETA,B,WT,SEX) generates a graph, observed
% vs. predicted values) for the goodness of fit purpose.
% The input: |ID| - subjects numbers; |TIME| - observation times;
% |CONC| - observed drug concentrations; |BETA| - model estimates;
% |B| - random effect values; |WT|,|SEX| - weight and sex as
% covariates.
%
% This function uses format ticks from MATLAB Central:
% http://www.mathworks.com/matlabcentral/fileexchange/
% 15986-format-tick-labels/content/format_ticks.m
% Copyright (c) 2007,
% Origin Version Created by Alex Hayes

display 'predObs'
n = length(unique(id));
d = repmat(beta(1:2), 1, n);
qq1 = d + [b(1,:); b(2,:)];
qq2 = [beta(3)*wt, beta(4)*sex]'; % INCLUDE COVARIATES
qq = qq1 + qq2;
pkpar = exp(qq);
solu = exp(d+qq2);
popp = [];
indp = [];
obsv = [];
for i=1:n
    popp = [popp; modelana(solu(:,i),time(id==i),100)]; %#ok<AGROW>
    indp = [indp; modelana(pkpar(:,i),time(id==i),100)]; %#ok<AGROW>
    obsv = [obsv; conc(id==i)]; %#ok<AGROW>
end
figObsPopInd = figure;
set(axes,'FontSize',15);
loglog(popp,obsv,'r*')
hold on
loglog(indp,obsv,'g*')
xlabel('Predicted')
ylabel('Observed')
legend('population prediction','individual prediction')
print(figObsPopInd,'-dtiff','-r900','finalObsPopInd')

```

```

% Graph Observation vs.Population Prediction
figObsPop = figure;
h = scatterhist(log10(popp), log10(obsv), 'Direction', 'out');
set(h(1), 'FontSize', 15);
XTickL = get(h(1), 'XTickLabel');
lx = length(XTickL);
XTickLmod = cell(1, lx);
for i=1:lx
    XTickLmod{i} = ['10^', '{', num2str(XTickL(i,1:2), '%2d'), '}'];
end
YTickL = get(h(1), 'YTickLabel');
ly = length(YTickL);
YTickLmod = cell(1, ly);
for i=1:ly
    YTickLmod{i} = ['10^', '{', num2str(YTickL(i,1:2), '%2d'), '}'];
end
hl = get(h(1));
format_ticks(h(1), XTickLmod, YTickLmod);
set(hl.Children, 'Marker', '*', 'Color', 'Red')
xlabel({'', '', 'Predicted'})
ylabel({'Observed', '', ''})
title('Observation vs.Population Prediction ');

% Graph Observation vs.Individual Prediction
figObsInd = figure;
h = scatterhist(log10(indp), log10(obsv), 'Direction', 'out');
set(h(1), 'FontSize', 15);
XTickL = get(h(1), 'XTickLabel');
lx = length(XTickL);
XTickLmod = cell(1, lx);
for i=1:lx
    XTickLmod{i} = ['10^', '{', num2str(XTickL(i,1:2), '%2d'), '}'];
end
YTickL = get(h(1), 'YTickLabel');
ly = length(YTickL);
YTickLmod = cell(1, ly);
for i=1:ly
    YTickLmod{i} = ['10^', '{', num2str(YTickL(i,1:2), '%2d'), '}'];
end
hl = get(h(1));
format_ticks(h(1), XTickLmod, YTickLmod);
set(hl.Children, 'Marker', '*', 'Color', 'Green')
xlabel({'', '', 'Predicted'})
ylabel({'Observed', '', ''})
title('Observation vs.Individual Prediction ');

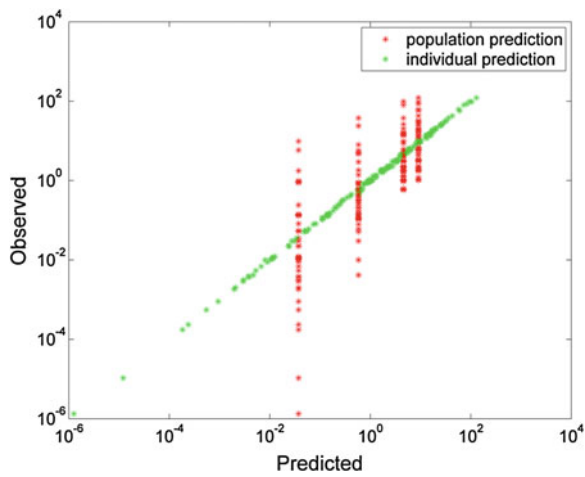
print(figObsPop, '-dtiff', '-r900', 'finalObsPop')
print(figObsInd, '-dtiff', '-r900', 'finalObsInd')

end

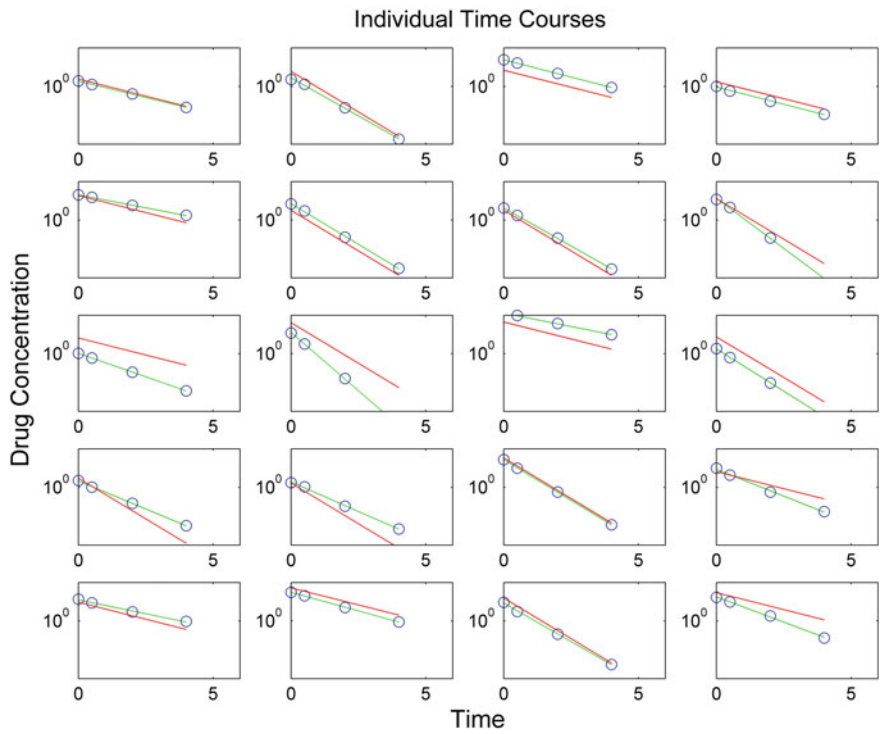
```

A graphical representation of the results shows that both individual and population predictions match observations (Figs. 6.9, 6.10).

Another visualization of the scatterplots in Fig. 6.10 (population and individual prediction vs. observations) is provided by the **scatterplothist** function that

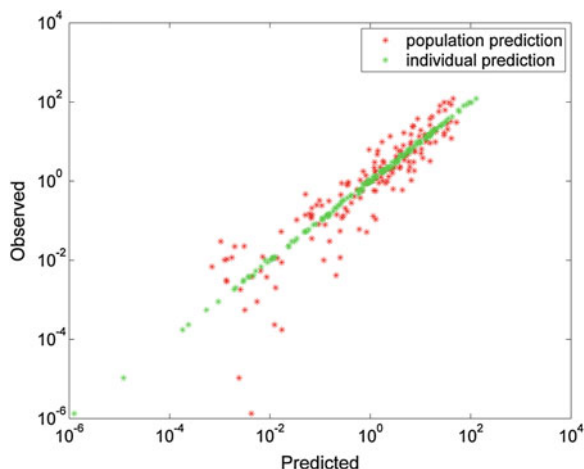


**Fig. 6.8** Base analysis (no covariates included): predicted (population and individual) versus observed concentrations



**Fig. 6.9** Final population analysis: population predictions (solid red line), individual predictions (solid green line), and observations (circles) against time (20 out of 40 profiles)

**Fig. 6.10** Final population analysis: predicted (population, individual) versus observed drug concentrations



generates a scatterplot and adds two histograms close to each axis for the dependent and independent variable. The histograms show the marginal distribution hidden in the scatterplot (Fig. 6.11).

Information about parameter values is given for each iteration of the likelihood optimization algorithm (Fig. 6.12). Stable values indicate that a necessary condition for an optimum has been achieved.

The results generated by the **nlmeFinalAnalysis.m** program are summarized in Table 6.4. The first observation is that in comparison to the base model the value of  $-\log L$  decreased by about 46 units, a statistically significant change [8], favoring the covariate model. The effect of body weight on volume of distribution,  $V$ , was estimated as  $\beta_3 = 0.18$ , and the effect of sex on drug elimination rate constant,  $ke$ , as  $\beta_4 = 0.87$ . For an interpretation of these effects in terms of PK parameters reference is to be made to the model equations (6.11). Quite reassuringly, the covariate analysis found parameter values very similar to those used to create the data.

Model diagnostics (such as comparing predicted with observed data) should find out whether there are deficits in the current model [9]. Structural misspecification, for instance, may pose a real problem in later model applications. A diagnostics procedure that may guard to some extent against structural and covariate model misspecification is the VPC [10]. The basic idea of a VPC is to simulate predictions for a number of subjects (replicates) from the final model while including all random effects and to compare a confidence range of predictions with observed data. The **nlmeFinalAnalysis.m** program calls **vpc** to perform a VPC. The full MATLAB implementation of the VPC is shown in Listing 6.7, and the results in Fig. 6.13.

**Listing 6.7** Function **vpc**: model diagnostics based on the VPC

```

function vpc(nrep,nsub,time,conc,beta,PSI,error,wt,sex)
%VPC conduct Visual Predictive Checks
% VPC(NREP,NSUB,TIME,CONC,BETA,PSI,ERROR,WT,SEX) is a simulation
% method to evaluate if the model correctly describes the observed
% data, and if it can predict responses. The evaluation is based on
% the following input: |NREP| - replications(simulations) number;
% |NSUB| - number of subjects; |TIME| - observation times;
% |CONC| - observed drug concentrations; |BETA|- model estimates;
% |PSI| - covariance matrix; |ERROR| - standard error of |BETA|;
% |WT|,|SEX| - weight, sex as covariates
%
% This function uses plot_c from MATLAB Central:
% http://www.mathworks.com/matlabcentral/fileexchange/31752-plotci
% Copyright (c) 2011, Zbigniew
% All rights reserved.
% Code covered by the BSD License: http://www.mathworks.com/matlabcentral/fileexchange/view\_license?file\_info\_id=31752

display 'vpc'

times = unique(time);
ntimes = length(times);
B1 = repmat(1:nrep,nsub*ntimes,1);
B2 = repmat((1:nsub),ntimes,1);
Bx = repmat(vertcat(B2(:)),nrep,1);
B3 = repmat(times,nrep*nsub,1);
BB = [B1(:) Bx B3(:)];
B4 = zeros(length(B3),1);

for k = 1:nrep
    vpc1 = mvnrnd(zeros(2,1),PSI,nsub);
    vpc2 = bsxfun(@plus,vpc1,[beta(1) beta(2)]);
    vpc3 = vpc2 + [beta(3)*wt beta(4)*sex]; % INCLUDE COVARIATES
%     vpc3 = vpc2; % NO COVARIATES
    for i = 1:nsub;
        pred = modelvpc(exp(vpc3(i,:)),times,100,error)';
        B4((k-1)*nsub*ntimes+(i-1)*ntimes+1):(k-1)*nsub*ntimes+ ...
            (i-1)*ntimes+4) = pred(:);
    end
end
A = [BB B4];
p5 = @(x) prctile(x,5);
p95 = @(x) prctile(x,95);
figVPC = figure;
set(axes,'FontSize',15)
semilogy(times,grpstats(conc,time,@median), 'sr', ...
    'MarkerFaceColor','r')
hold on
semilogy(times,grpstats(conc,time,p5), 'sb', 'MarkerFaceColor','b')
semilogy(times,grpstats(conc,time,p95), 'sk', 'MarkerFaceColor','k')
legend('median', 'perc. 0.05', 'perc. 0.95')

E5x = zeros(nrep*ntimes,3);
E50x = zeros(nrep*ntimes,3);
E95x = zeros(nrep*ntimes,3);
for i=1:nrep

```

```

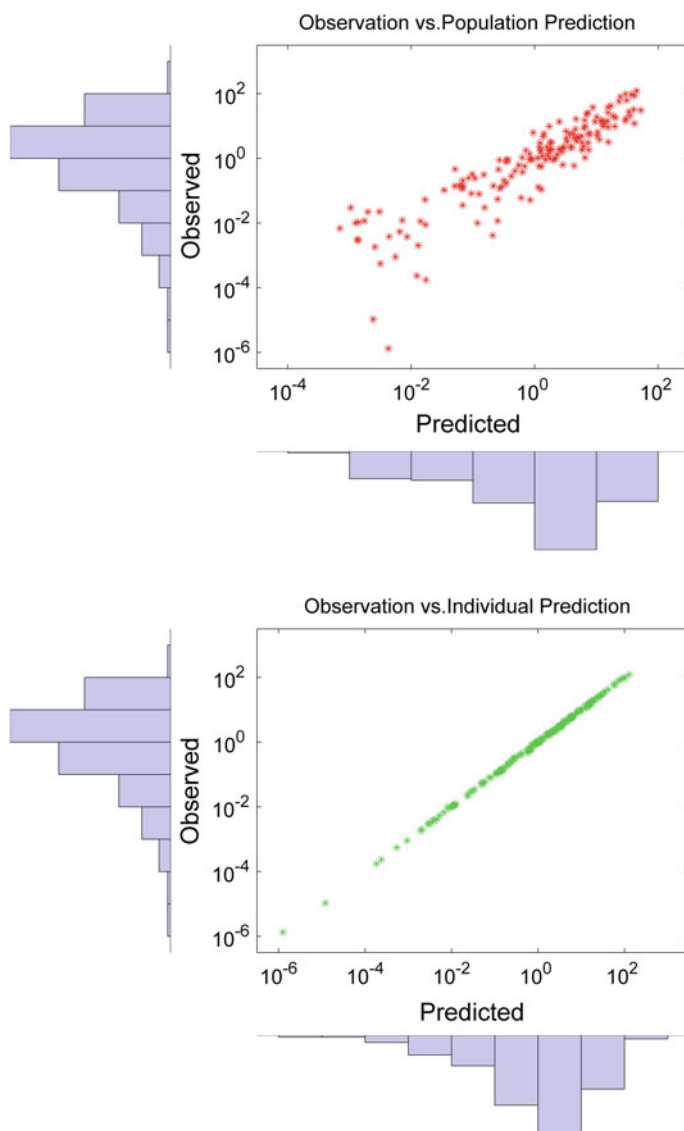
[E5,E50,E95] = grpstats(A(A(:,1)==i,[1,3,4]),A(A(:,1)==i,3), ...
    {p5 @median p95});
E5x((i-1)*ntimes+1:(i-1)*ntimes+4,[1 2 3]) = E5;
E50x((i-1)*ntimes+1:(i-1)*ntimes+4,[1 2 3]) = E50;
E95x((i-1)*ntimes+1:(i-1)*ntimes+4,[1 2 3]) = E95;
end

% get 5% and 95% percentiles
p5all = grpstats([E5x(:, [2 3]) E50x(:,3) E95x(:,3)],E5x(:,2),p5);
p95all = grpstats([E5x(:,3) E50x(:,3) E95x(:,3)],E5x(:,2),p95);
vpc = [p5all p95all];
semilogy(vpc(:,1),vpc(:,2),'-b')
semilogy(vpc(:,1),vpc(:,5),'-b')
predCI = [vpc(:,2) vpc(:,5)];
rgbfill = [171 205 255]/255;
% call plot_ci; the function is available on MATLAB Central:
plot_ci(times, predCI, 'PatchColor', rgbfill, 'PatchAlpha', 0.5, ...
    'LineWidth', 1, 'LineStyle','-', 'LineColor', 'b');
semilogy(vpc(:,1),vpc(:,3),'-r')
semilogy(vpc(:,1),vpc(:,6),'-r')
predCI = [vpc(:,3) vpc(:,6)];
rgbfill = [255 180 180]/255;
plot_ci(times, predCI, 'PatchColor', rgbfill, 'PatchAlpha', 0.5, ...
    'LineWidth', 1, 'LineStyle','-', 'LineColor', 'r');
semilogy(vpc(:,1),vpc(:,4),'-k')
semilogy(vpc(:,1),vpc(:,7),'-k')
predCI = [vpc(:,4) vpc(:,7)];
rgbfill = [200 200 200]/255;
plot_ci(times, predCI, 'PatchColor', rgbfill, 'PatchAlpha', 0.5, ...
    'LineWidth', 1, 'LineStyle','-', 'LineColor', 'k');
xlabel('Time')
ylabel('Concentration')
print(figVPC,'-dtiff','-r900','finalVPC')

end

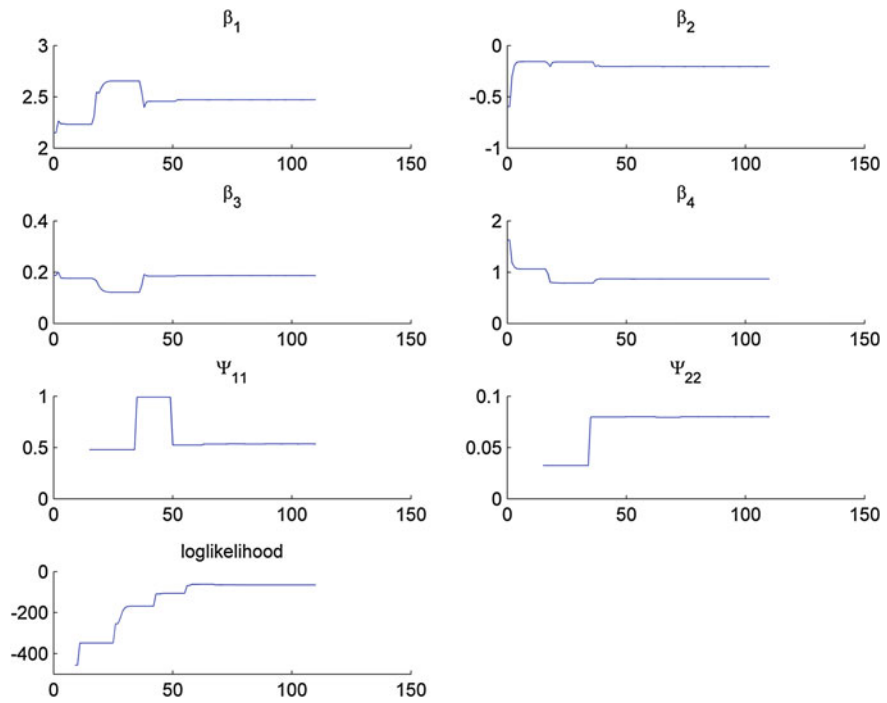
```

MATLAB offers also numerous functions and procedures for further explorations such as residual error distribution (see [11]).



**Fig. 6.11** Predicted versus observed concentrations and marginal histograms. Population predictions—*upper panel*, individual predictions—*lower panel*





**Fig. 6.12** Final population analysis: the *graph* shows the history of parameter estimates over iteration number. All parameters are converged to stable values

**Table 6.4** Summary of the results from the `nlmeFinalAnalysis.m` program in 40 subjects

Fixed effects parameters	Value	SE	BSV
$\beta_1$	2.4713	0.1164	0.5349
$V [=exp(\beta_1)]$	11.84		
$\beta_2$	-0.2049	0.0715	0.0800
$ke [=exp(\beta_2)]$	0.815		
$\sigma^2$	0.0989		
$\beta_3$ (BW on $V$ )	0.1869	0.0202	
$\beta_4$ (SEX on $ke$ )	0.8698	0.0919	
$\log L$	-12.3705		

SE—standard error of estimate, BSV—between-subject variability (variance of random effects),  $\log L$ —logarithm of likelihood (inclusion of covariates)

### 6.4.5 The Stochastic Approximation Algorithm

A more robust implementation for the estimation of population model parameters is given by the Markov chain-based stochastic approximation expectation–maximization (SAEM) algorithm [12], which is called in MATLAB using the **nlmefitsa** function. Compared to **nlmefit**, a slightly different coding is needed, as shown in Listing 6.8.

**Listing 6.8** Population analysis using stochastic approximation (SAEM algorithm)

```
function nlmeFinalAnalysis
% .....
% .....
% .....

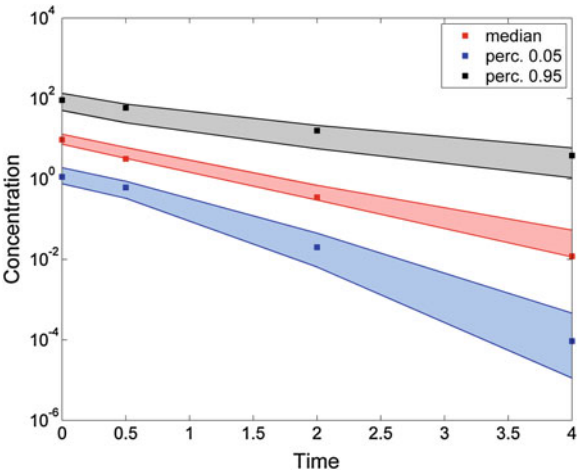
% Calling Stochastic Approx. Expectation-Maximization mode(SAEM)
[beta,PSI,stats,b] = nlmefitsa(time,conc,id,dose,@modelana,b0, ...
    'Options', options, ...
    'REParamsSelect',[ 1 2], ...
    'LogLikMethod', 'is', ...
    'ErrorModel','Proportional', ...
    'CovPattern',P, ...
    'OptimFun','fminunc', ...
    'ParamTransform',[1 1], ...
    'FeGroupDesign',A)      %#ok<NOPRT>

% .....
% .....
% .....

end
```

A summary of the population analysis results with SAEM is shown in Table 6.5 and the respective parameter Markov chains in Fig. 6.14. As an exercise, the reader may conduct a full analysis using the **nlmefitsa** function within **nlmeFinalAnalysis.m**.

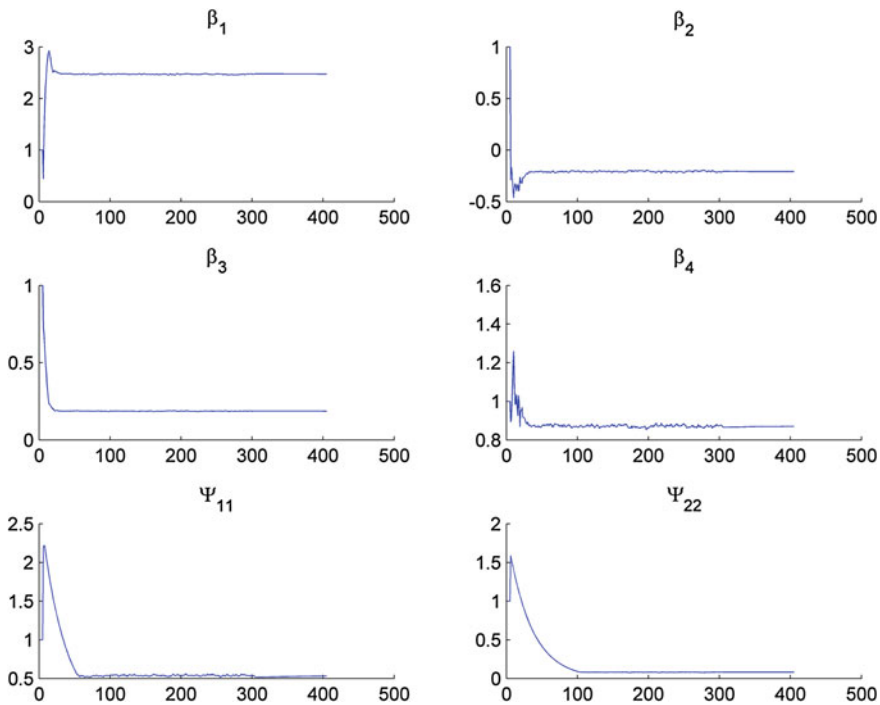
**Fig. 6.13** Visual predictive check (VPC) for the final population model (1000 replicates). Median and 90 % confidence boundaries are shown for the observed (*squares*) and simulated data (*colored regions*). *Colored regions* refer to the distribution of 5 % percentiles (*blue*), medians, 95 % percentiles



**Table 6.5** Summary of the results from the `nlmeFinalAnalysis.m` program calling SAEM in 40 subjects

Fixed effects parameters	Value	SE	BSV
$\beta_1$	2.4745		0.5308
$V [= \exp(\beta_1)]$	11.87		
$\beta_2$	-0.2083		0.0810
$ke [= \exp(\beta_2)]$	0.812		
$\sigma^2$	0.0989		
$\beta_3$ (BW on $V$ )	0.1867		
$\beta_4$ (SEX on $ke$ )	0.8706		
$\log L$	-12.1133		

SE—standard error of estimate, BSV—between-subject variability (variance of random effects),  $\log L$ —logarithm of likelihood (inclusion of covariates)



**Fig. 6.14** Final population analysis: population analysis based on SAEM. Markov chains for model parameters are shown. Stable values are achieved after 100 iterations. Number of burn-in iterations: 5

## 6.5 Exercises

### Exercise 6.1

Write a program for the NLMEM analysis of a two-compartment model with infusion as described in Chap. 3, in (3.17)–(3.19). Include the covariates (weight and sex). The simulated dataset for the analysis, **drugCData.mat**, is available at MATLAB Central. Compare the model with covariates to the model without covariates.

## References

1. Hoffmann-La Roche (2013) Product monograph: inhibace. [http://www.rochecanada.com/fmfiles/re7234008/Research/ClinicalTrialsForms/Products/ConsumerInformation/MonographsandPublicAdvisories/Inhibace/Inhibace\\_PM\\_18Jan2013HPE.pdf](http://www.rochecanada.com/fmfiles/re7234008/Research/ClinicalTrialsForms/Products/ConsumerInformation/MonographsandPublicAdvisories/Inhibace/Inhibace_PM_18Jan2013HPE.pdf). Accessed 14 Mar 2013

2. Physicians' Desk Reference (2013) Drug summary: Tamiflu. <http://www.pdr.net/drug-summary/tamiflu?druglabelid=2099&id=1256>. Accessed 14 Mar 2013
3. Sheiner LB, Beal SL (1981) Evaluation methods for estimating population pharmacokinetic parameters. II. Biexponential model and experimental data. *J Pharmacokinet Biopharm* 9:635–651
4. Sheiner LB, Beal SL (1980) Evaluation methods for estimating population pharmacokinetic parameters. I. Michaelis-Menten model: routine clinical data. *J Pharmacokinet Biopharm* 8:553–571
5. Sheiner LB, Grasela TH (1991) An introduction to mixed effect modeling. *J Pharmacokinet Biopharm* 19:11S–24S
6. Lavielle M, Mentre F (2007) Estimation of population pharmacokinetic parameters of saquinavir in HIV patients with the MONOLIX software. *J Pharmacokinet Pharmacodyn* 34:229–249
7. Kuhn E, Lavielle M (2005) Maximum likelihood estimation in nonlinear mixed effects models. *Comput Stat Data Anal* 49:1020–1038
8. Boeckmann AJ, Sheiner LB, Beal SL (1994) NONMEM users guide, part V. NONMEM Project Group, University of California, San Francisco
9. Karlsson MO, Savic RM (2007) Diagnosing model diagnostics. *Clin Pharmacol Ther* 82:17–20
10. Karlsson MO, Holford N (2008) A tutorial on visual predictive checks. In: Population Approach Group Europe, 17th meeting, Marseille
11. MATLAB<sup>®</sup> Programming Fundamentals, R2012b (2012) The MathWorks, Natick
12. Delyon B, Lavielle M, Moulines E (1999) Convergence of a stochastic approximation version of the EM algorithm. *Ann Stat* 27:94–128

## Chapter 7

# Clinical Trial Simulation

Clinical trial simulation offers the opportunity to run a clinical trial under a given design and assess the probability of a successful outcome. For this purpose, the trial is simulated many times, taking into account fixed and random effects that were assessed in a previous population analysis. Also, the impact of ad hoc assumptions regarding the population to be treated, the dosage regimen including dosage adjustment rules, and the pharmacology of the drug, etc., can be tested. Clinical trial simulation is applicable at all stages of drug development. Essentially, it is a forecast based on current knowledge codified by a PK-PD/disease model.

### 7.1 Terms and Concepts

During the development of a drug a large number of clinical trials have to be performed. Most important are Phase III trials in patients that are performed to confirm hypotheses on the indication and dosage regimen of a drug. Statistically, significant results in primary outcome variables have to be demonstrated in two Phase III trials to get regulatory drug approval. Phase III hypotheses have been generated in preceding studies, such as a dose finding study in Phase II. The scope of a Phase III trial (e.g., number of patients, duration of treatment, and comparative treatment) is regulated for some indications, but there is still room for alternatives.

Clinical trial simulation was suggested in the last decade of the twentieth century in response to the occurrence of late-phase clinical trial failures. Failures were attributed partly to the selection of poor study design. For instance, a previously successful trial design might have been uncritically used in upcoming studies. It was argued that a more appropriate design of such trials, to be evaluated by simulations, could have rescued a drug and saved large investments [1].

The basic rationale for computer simulation has existed for many years and the technique has been successfully used in several scientific and industrial application areas [2]. As of 1995, serious attempts were made developing software for the simulation of clinical trials. This was supported by the growing acceptance of population analyses providing us with reliable estimates for intersubject variability.

Since 2000, the clinical trial simulation methodology is available and broadly applied. FDA reviewers make use of it when providing answers to sponsors [3].

## 7.2 Execution of Trial Protocols

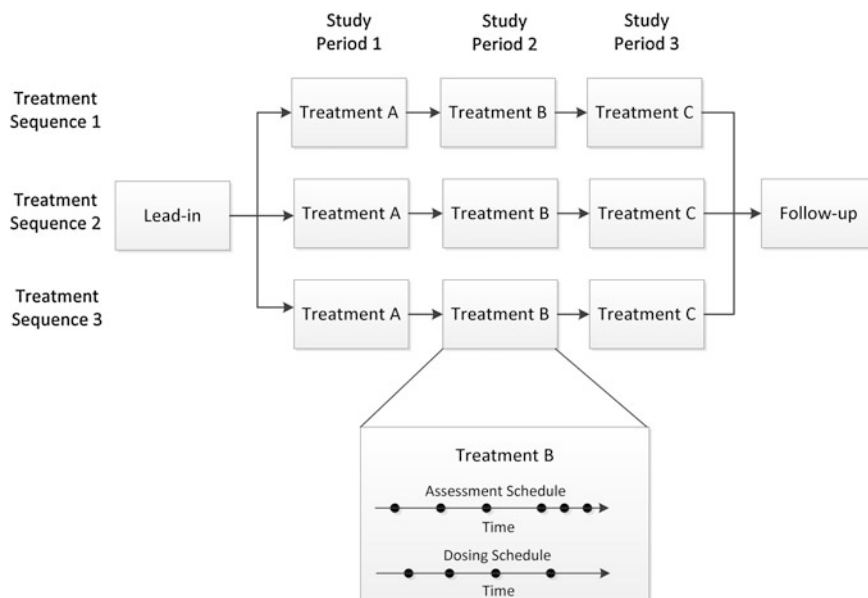
Trial protocols are fundamental to clinical drug development and need to be perfectly tailored to the objective of a study. They are part of a development strategy for a project that consumes many 100 million dollars and needs between 5 and 10 years to achieve a new drug application (NDA). A clinical trial protocol has a standardized structure determining the input (subjects, medication), the output (assessments), and the evaluation of the results (statistical analysis, mathematical modeling). Key elements of a trial protocol are shown in Table 7.1.

The specification of the primary study objective should be without any ambiguity and as quantitative as possible to allow appropriate statistical analysis. If the primary objective failed, but secondary objectives succeeded, it is tempting to declare secondary objectives as primary. Strictly speaking, the confirmation of secondary objectives needs another study.

The design of a study defines how subjects are allocated to treatment sequences (for an illustration of treatment sequences and periods see Fig. 7.1). In a parallel design, subjects are divided in as many groups as there are treatments. In a sequential design, subjects are reallocated to another treatment during the course of the study. One of the most often applied sequential designs is the 2-way crossover design, where subjects are assigned to two sequential treatments. They switch over to the other treatment after a prespecified treatment duration, including time to wash out drug amounts and, if applicable, treatment effects. In crossover trials, subjects serve as their own control, reducing the number of subjects to demonstrate statistically significant effects. Another design element refers to whether subjects and/or physicians administering the treatments are blinded to the treatment. If only subjects are blinded, we speak of a single-blind trial, if both subjects and physicians are blinded, of a double-blind trial. Double-blind trials

**Table 7.1** Trial protocol elements

Protocol term	Explanation
Trial objectives	Primary, secondary
Design	Parallel, sequential, blindness
Population	Demographics, genetics, disease severity inclusion, exclusion criteria
Treatments	Drugs, dosage regimen, route of administration
Assessments	Type of data, time schedule
Procedures	Lead-in phase, follow-up, event handling, criteria for withdrawal, and replacement of withdrawn subjects
Analysis	Specification of statistical evaluation



**Fig. 7.1** Clinical trial building blocks. Each clinical trial is composed of treatments that can be organized in sequence and/or in parallel (study design). Subjects eligible for the trial are assigned to treatment sequences which comprise one or more treatments. Each subject is assigned to only one sequence. From a time perspective, a trial is divided into a lead-in phase (optional), study periods, and a follow-up period (optional). Each treatment comprises a dosing and an assessment schedule

provide the best protection against bias, and are generally required for Phase III confirmatory trials.

The population to be studied must be characterized in detail. This refers not only to demographic variables such as gender, age, body weight, but could include tests of body function (blood pressure, glucose tolerance test) or protein and genetic markers in body fluids (blood, urine, cerebrospinal fluid). Inclusion and exclusion criteria are usually quite extensive lists specifying how to deal with previous diseases, concomitant treatments, pregnancy, etc.

The treatment information applies to both active compound and placebo. Dose strength, dosing frequency, and duration of treatment have to be specified in advance, together with the handling of noncompliance, e.g., substituting for a missed dose.

Assessments in a clinical study refer to both efficacy and safety. It is of great importance to be very precise how measurements are to be done, both in the laboratory and clinically, and to ensure that procedures are followed. A minimal requirement is that deviations from procedures are documented in a consistent way so that subsequent data analyses can cope with them, if at all possible.

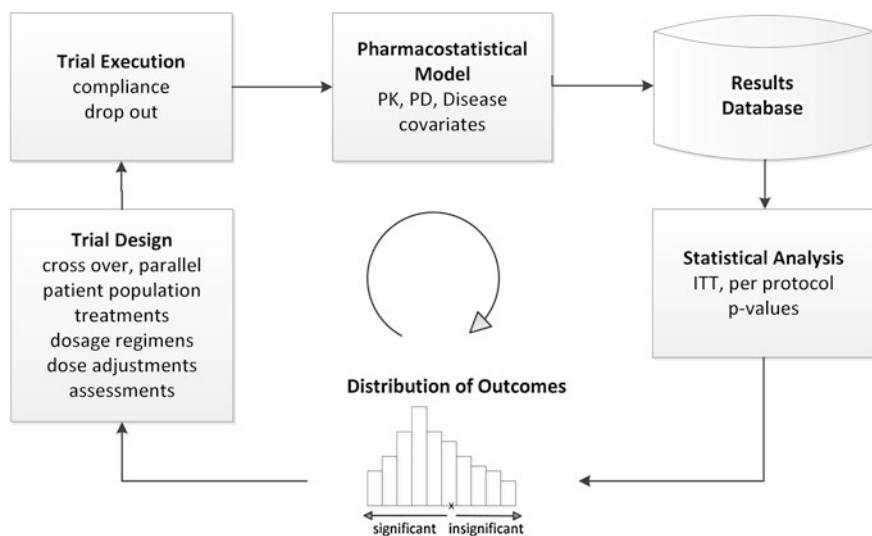
Study procedures are designed to handle events that have no fixed schedule. Patients that experience adverse events may need to be withdrawn or receive reduced dosages of treatment. As usual, these events require careful documentation and monitoring.



The statistical analysis describes how the assessments are turned into statistical statements about the trial. It usually distinguishes between different populations of subjects: all randomized (intent-to-treat), all dosed, and all protocol-adherent. Intent-to-treat analyses are based on the intended, not actual, doses. Interim analyses are a special case: these are prespecified looks into the data, e.g., to assess futility and whether to continue or stop the trial.

Simulation protocols consider all of the topics described. Based on a pharmacostatistical model, a clinical trial simulation executes a study with a suggested design many hundred times, including its statistical analysis, thereby creating a distribution of study outcomes. Depending on the complexity of the structural model, simulation including analysis takes from minutes to hours compared to the months or few years required by a real trial.

To mimic a clinical trial *in silico*, different types of models are needed (Fig. 7.2): a structural, or input–output model, to capture the PK-PD and disease aspects; a covariate model to capture subject characteristics; a trial execution model for the assignment to treatment groups, and for capturing events during the trial such as dose adaptation, drop-outs; and a stochastic model to reflect measurement error in assessments. The primary purpose of this new approach is to improve clinical development by generating better insights into the consequences of the choices made when designing a human trial.

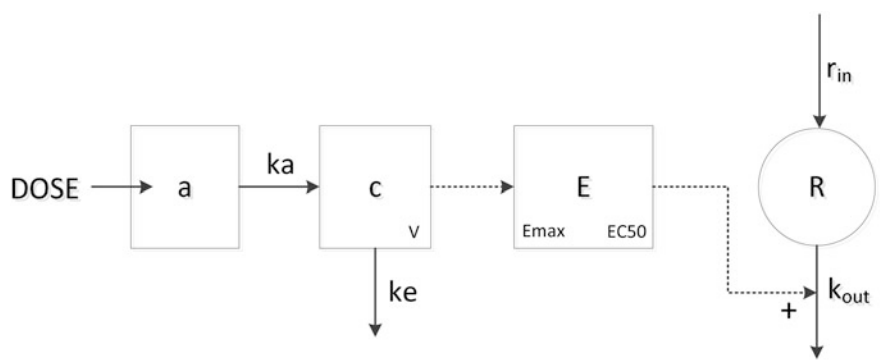


**Fig. 7.2** Clinical trial simulation. Controllable design factors (trial design, trial execution) are explored for the greatest chance of success. The pharmacostatistical model, incorporating disease and drug properties, provides a result database that is evaluated with statistical methods. The distribution of outcomes shows the power of each design

### 7.3 A Basic Clinical Trial Simulator

Below, we present a basic version of a clinical trial simulator. It was created to deal with the following situation. A single-dose placebo-controlled trial is to be conducted for an oral drug which acts on the production of a biomarker according to an IDR model (Fig. 7.3). Twenty subjects are planned for each treatment. The effects of the active drug (4 mg) should be significantly different from placebo at 72 h after dosing as assessed by a t-test without assuming equal variances at an  $\alpha$ -level of 0.05. What is the power of the trial? The power of a trial is defined as the probability of rejecting the null hypothesis under the alternative hypothesis. It can be easily calculated by counting how many of the replicated trials yielded a significant t-test result ( $p < 0.05$ ).

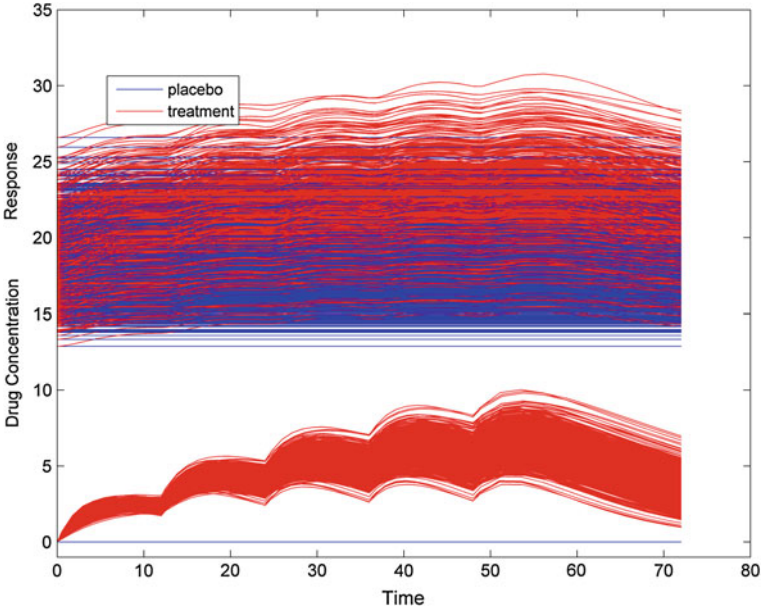
The PK and PD parameters are tabulated below (Table 7.2).



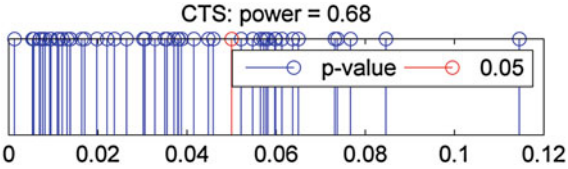
**Fig. 7.3** Basic clinical trial simulation: conceptual model. *Solid lines* indicate flows and are realized via ODEs, *dashed lines* indicate algebraic equations

**Table 7.2** Population PK-PD parameters for basic clinical trial simulation

Parameter	Unit	Value	Parameter	Unit	Value
Fixed effects			Random effects		
$ka$	1/h	0.2	$\omega ka$	%CV	0.2
$V$	L	25	$\omega V$	%CV	0.25
$ke$	1/h	0.05	$\omega ke$	%CV	0.16
$E_{\max}$	1	−0.9	$\omega E_{\max}$	%CV	0.04
$EC50$	mg/L	1.5	$\omega EC50$	%CV	0.16
$r_{in}$	1/h	10	$\omega r_{in}$	%CV	0.09
$k_{out}$	1/h	0.5	$\omega k_{out}$	%CV	−
Covariate effects					
$demo1ke$	1	0.85	$tvke = ke \cdot (demo1/40)^{demo1ke}$		
$demo2r_{in}$	1	0.85	$tvr_{in} = r_{in} \cdot demo2^{demo2r_{in}}$		



**Fig. 7.4** Basic Clinical Trial Simulation: two groups of 20 subjects each received either placebo or active treatment. A one-compartment PK model was linked to an IDR model and respective drug concentrations (*lower panel*) and drug responses (*upper panel*) were simulated over 72 h. This scenario was repeated 50 times



**Fig. 7.5** Distribution of trial outcomes ( $p$ -values). Each replicate in Fig. 7.4 yields a  $p$ -value based on a t-test comparing responses between placebo and active treatment at 72 h. The power is calculated as the ratio of the number of  $p$ -values  $< 0.05$  (34) to the number of replicates (50)

The program **CTSbasics.m** (Listing 7.1) was applied to the situation described above with 50 replicates and yielded results shown in Figs. 7.4 and 7.5.

**Listing 7.1** Program **CTSbasics** (the **hilleEffect** function, mentioned in this listing, contains only a header as the complete function is presented in [Chap. 2](#). *First Example of a Computational Model*)

```
function CTSbasics
%CTSBASICS A basic version of a clinical trial simulator.
% CTSBASICS conducts a single-dose placebo-controlled trial for
% a drug that acts on the production of a biomarker according to
% an indirect response model.

clc; clear; close all;
tic
rng(0,'twister')    % for reproducibility
nrep = 5;
hfresp = figure;
set(hfresp,'units','normalized','Position',[0.1, 0.4, 0.3, 0.4])
p(1:nrep) = 0; % to initialize vector p
for rep = 1:nrep
    rep      %#ok<NOPRT>
    nsubjects = 20;
    [del1,delr,de2] = fixedEffects(nsubjects);
    par = parameters(del1,delr,de2);
    [dosage,dosingTimes] = treatments();
    obs = observations();
    results = PSmodel(par,dosage,obs,dosingTimes);
    p(rep) = analysis(dosage,results);
end
yl = ylim;
ylim([yl(1)-1 yl(2)])
xlabel('Time')
ylabel('Drug Concentration Response')
legend('placebo','treatment','Location','Best')
print('-dtiff','-r600','CTSbasics')
outcomes(p)
display(['p = ', num2str(p)])
toc

function results = PSmodel(par, dosage, obs, dosingTimes)
%PSMODEL Conduct simulations.
% RESULTS = PSMODEL(PAR, DOSAGE, OBS, DOSINGTIMES) conducts
% simulations for all subjects based on the following input
% variables: |PAR| - model parameters, |DOSAGE| - dosing
% regimen, |OBS| - observations and |DOSINGTIMES| - dose times.
% It returns the matrix |RESULTS|.
% The function is nested due to |nsubjects|.

tlast = max([obs.var1, obs.var2, dosingTimes]);
intInt = [dosingTimes tlast]; % integration intervals
results = [];
for treatment = 1:length(dosage)
    dose = dosage(treatment);
    erg = [];
    for i = 1:nsubjects
        ti = [];
        yi = [];
        ergt = [];
        ergy = [];
```

```

ys = [dosage(treatment); 0; par.rin(i)/par.kout(i)];
for ni = 2:length(intInt)
    t1 = intInt(ni-1);
    t2 = intInt(ni);
    sol = ode45(@derivatives, [t1,t2], ys, [], par);
    graphics(treatment, sol)
    % sampling times in integration interval
    idx1 = obs.var1 >= t1 & obs.var1 <= t2;
    idx2 = obs.var2 >= t1 & obs.var2 <= t2;
    idx = [idx1 idx2];
    if sum(idx) > 0
        t = unique([obs.var1(idx1) obs.var2(idx2)]);
        y = deval(sol,t)';
        ti = [ti; t];          %#ok<AGROW>
        yi = [yi; y];          %#ok<AGROW>
    end
    ys = sol.y(:,end)' + [dosage(treatment) 0 0];
end
ergt = [ergt; ti];            %#ok<AGROW>
ergy = [ergy; yi];            %#ok<AGROW>
ergn = [ones(length(ergt),1)*dose, ...
        ones(length(ergt),1)*i, ergt, ergy];
erg = [erg; ergn];            %#ok<AGROW>
end
results = [results; erg];      %#ok<AGROW>
end

function dydt = derivatives(~ , y, par)
%DERIVATIVES Compute the right-hand side of the ODE.
% DYDT = DERIVATIVES(T, Y, PAR) calculates |DYDT|, the
% right-hand side of the ODE model, at points defined by
% the vector of dependent variables |Y|, time |T|, and
% with parameters |PAR|.
% The function is nested due to the index |i|.

ka    = par.ka(i);
ke    = par.ke(i);
V      = par.V(i);
rin    = par.rin(i);
kout   = par.kout(i);
emax   = par.emax(i);
ec50   = par.ec50(i);
a0     = y(1);
a1     = y(2);
r       = y(3);
c       = a1/V;
de     = hillEffect(c, 1, emax, ec50, 1);
da0dt  = -ka*a0;
da1dt  = ka*a0 - ke*a1;
drdt   = rin - kout*de*r;
dydt   = [da0dt; da1dt; drdt];

end

```

```

function graphics(treatment, sol)
%GRAPHICS Graphs with the drug concentration and response.
%   GRAPHICS(TREATMENT, SOL) produces a graph which shows the
%   time courses of the drug concentration and drug response
%   for the selected |TREATMENT| and values given in the
%   model solution structure |SOL|.

%   The function is nested with regard to |i| and |ni|

if treatment == 1
    hp1 = plot(sol.x, sol.y(2,:), '-b'); drawnow;
    hold on
    hp2 = plot(sol.x, sol.y(3,:), '-b'); drawnow;
end
if treatment == 2
    hp1 = plot(sol.x,sol.y(2,:), '-r'); drawnow;
    hold on
    hp2 = plot(sol.x,sol.y(3,:), '-r'); drawnow;
end
if i ~= 1 || ni ~= 2
    set(get(get(hp1,'Annotation'),'LegendInformation'),...
        'IconDisplayStyle','off'); % Exclude line from legend
end
set(get(get(hp2,'Annotation'),'LegendInformation'),...
    'IconDisplayStyle','off'); % Exclude line from legend
end

end
end

function obs = observations
%OBSERVATIONS Specify observation times
%   OBS = OBSERVATIONS creates a stucture, |OBS|, with observation
%   times for all dependent variables regarded in the analysis.

obs.var1 = 0:4:24;
obs.var2 = 0:24:72;

end

function [dosage, dosingTimes] = treatments
%TREATMENTS Dosing specification.
%   [DOSAGE, DOSINGTIMES] = TREATMENTS specifies the dose times
%   |DOSINGTIMES| and dose amounts |DOSAGE| for considered regimens.

% 2 regimens assumed
dosage(1) = 0 ; % placebo
dosage(2) = 4 ; % active drug treatment with 4 mg
dosingTimes = 0:12:48;

end

```

```

function par = parameters(demo1, demo1ref, demo2)
%PARAMETERS Create initial values for model parameters.
%   PAR = PARAMETERS(DEMO1, DEMO1REF, DEMO2) returns a structure
%   |PAR| containing initial values for model parameters, including
%   variability. The values are created for all simulated subjects.
%   |DEMO1|, |DEMO1REF|, |DEMO2| - covariates 1, reference for
%   covariate 1 and covariate 2, respectively.

n          = length(demo1);
tvka       = 0.2; % 1/h
omka       = 0.2;
par.ka     = tvka*exp(normrnd(0,omka,n,1)); % individual value of ka
tvke       = 0.05;
omke       = 0.16;
demo1ke    = 0.85; % covariate effect of demo1 on ke
par.ke     = tvke.*(demo1./demo1ref).^(demo1ke) ...
            .*exp(normrnd(0,omke,n,1));
tvV        = 25;
omV        = 0.25;
par.V      = tvV*exp(normrnd(0,omV,n,1));
tvrin      = 10;
omrin      = 0.09;
demo2rin   = 0.85;
par.rin    = tvrin*demo2rin.^demo2.*exp(normrnd(0,omrin,n,1));
tvkout     = 0.5;
omkout     = 0.0;
par.kout   = tvkout*exp(normrnd(0,omkout,n,1));
tvemax     = -0.90;
omemax     = 0.04;
par.emax   = tvemax*exp(normrnd(0,omemax,n,1));
tvec50     = 1.5;
omec50     = 0.16;
par.ec50   = tvec50*exp(normrnd(0,omec50,n,1));

end

function [de1, delr, de2] = fixedEffects(n)
%FIXEDEFFECTS Generate demographics.
%   [DE1, DE1R, DE2] = FIXEDEFFECTS(N) generates the demographic data
%   for all subjects included in the trial simulation. The number of
%   subjects is |N|. Two types of the data are generated: 1 - normal,
%   |DE1| with the reference value |DE1R|, and 2 - binary, |DE2|.
%   The normal type values represents such covariates as body weight,
%   height etc.; binary stands for covariates such as sex (F/M),
%   smokers (YES/NO), etc.

% type: 'truncated normal'
delr       = 40;
delmin     = 20;
delmax     = 60;
delsd      = 5;
de1        = truncnorm(n,delr,delsd,delmin,delmax);
% type: 'binary';
de2        = binornd(1,0.5,n,1);

end

```

```

function p = analysis(dosage, results)
%ANALYSIS Compare effects of the active drug and placebo
% P = ANALYSIS(DOSAGE,RESULTS) compares a group treated with the
% active drug to the placebo group using the statistical t-test to
% verify the null-hypothesis (no statistically significant effect
% of the drug) against the alternative hypothesis (the drug effect
% is significant). Input parameters: |DOSAGE| is a vector with dose
% values for both groups, |RESULTS| - drug effect values needed for
% the comparison. The returned value |P| is a probability of the
% drug effect value under the null hypothesis (the p-value).

placebo = results(results(:,1)==dosage(1) & results(:,3)==72,6);
verum    = results(results(:,1)==dosage(2) & results(:,3)==72,6);
[~, p, ~] = ttest2(verum,placebo,[],[],'unequal');

end

function outcomes(p)
%OUTCOMES Show the results of the CTS power calculation.
% OUTCOMES(P) produces a graph which shows how the p-values from
% all, |P|, are distributed against p = 0.05 (significance level
% for acceptance or rejection of the hypothesis H0)

power = sum(p<0.05)/length(p);
hfpower = figure;
hapower = axes;
set(hfpower,'units','normalized', ...
    'Position',[0.1, 0.1, 0.3, 0.15], 'PaperPositionMode','auto')
stem(p,ones(length(p),1))
hold on
stem(0.05,1,'-r')
set(hapower, 'YTickLabel',{' ',' ',' '}, 'YTick', [])
xlabel(' ')
legend('p-value','0.05', 'Orientation','horizontal')
title(['CTS: power = ', num2str(power)])
display(['CTS: power = ', num2str(power)])
print('-dtiff','-r600','CTSpower')

end

function out = truncnorm(n, mean, sd, min, max)
%TRUNCNORM Random vector from a truncated normal distribution.
% OUT = TRUNCNORM(N, MEAN, SD, MIN, MAX) returns a vector of normal
% distributed random numbers |OUT|, with the mean value |MEAN| and
% standard deviation |SD|, and truncated to the interval with
% lower cutoff |MIN| and upper cutoff |MAX|. |N| is the length of
% |OUT|.

```



```

res = normrnd(mean, sd, n, 1);
fi = find(res < min | res > max);
lf = length(fi);
while lf > 0
    res1 = normrnd(mean, sd, lf, 1);
    res(fi) = res1;
    fi = find(res < min | res > max);
    lf = length(fi);
end
out = res;

end

function E = hillEffect(c, E0, Emax, EC50, n)
% .....
% .....
% .....
end

```

## 7.4 Example: ESA Replacement

The following example makes reference to the drug–disease model we introduced in [Sect. 5.3, Renal Anemia](#). We showed the treatment effects on Hb following administration of a single erythropoiesis stimulating agent (ESA). Now, we consider the situation where one ESA (ESA1) is replaced by another (ESA2) with a longer half-life. It is assumed that patients had stable Hb values between 10.5 and 12.5 g/dL when being treated with a given IV dose of ESA1. The task is to find SC dosing rules for ESA2 that keep patients within the range of predefined Hb values. A conceptual model for the replacement scenario is shown in [Fig. 7.6](#).

### 7.4.1 PK Model

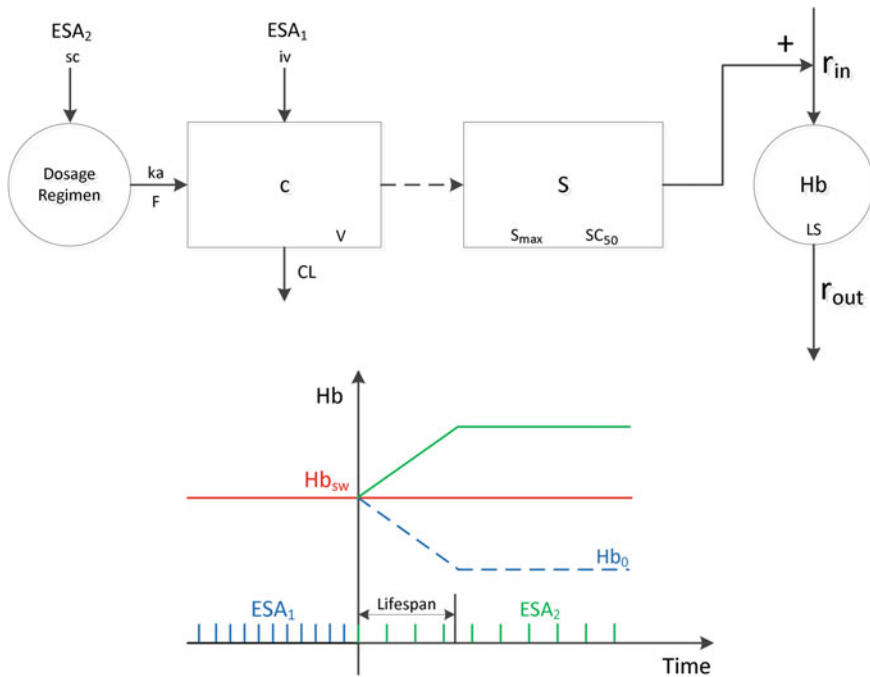
In accordance with the known PK of many ESAs [\[4\]](#) we assume a one-compartment model for the replacement ESA2 with first-order absorption and first-order elimination. Drug concentrations,  $c(t)$ , are described by the following equations:

$$\left. \begin{aligned} \frac{da}{dt} &= -k_a \cdot a \\ \frac{dc}{dt} &= \frac{k_a}{V} \cdot a - \frac{CL}{V} \cdot c \end{aligned} \right\}; \quad t \in [0; T] \quad (7.1)$$

with initial values

$$a(0) = F \cdot Dose, \quad c(0) = 0 \quad (7.2)$$

where  $Dose$  represents the dose of the drug,  $k_a$ —the first-order absorption rate constant,  $F$ —the absolute bioavailability following extravascular (SC) dosing,



**Fig. 7.6** Conceptual model to describe the Hb changes following replacement of one ESA (ESA1) by another (ESA2). *Upper panel:* a one-compartment PK model is coupled to an IDR model for hemoglobin by stimulating the production rate of hemoglobin according to an  $E_{\max}$  model with maximum effect  $S_{\max}$  and potency  $SC_{50}$ . The output rate of hemoglobin is the input rate delayed by erythrocyte lifespan ( $LS$ ). *Lower panel:* Hb values are assumed to be stable under treatment with ESA1. If ESA1 were dropped without replacement, Hb values would decline and reach a new steady state ( $Hb_0$ ) after a life span period. ESA2 treatment has to compensate for the missing Hb stimulation of ESA1. Note that ESA2 is given less frequently than ESA1

$CL$ —the drug clearance, and  $V$ —the volume of distribution. Population PK parameters for  $k_a$ ,  $F$ ,  $CL$ , and  $V$ , are listed in Table 7.3.

### 7.4.2 PD Model

The PD model was already described in Sect. 5.3.1. In short, ESA2 induces the production of red blood cells which have a life span of about 100 days. Thus, increase in Hb is mirrored by a respective decrease when the newly formed cells have reached their life span. This phenomenon makes it difficult to achieve stable Hb values in clinical practice as one is tempted to react to increasing Hb values with dose reductions and vice versa [5]. In this chapter, the model presented in Sect. 5.3.1 is amended to the situation when patients switch from one ESA to another. Equations (7.3)–(7.6) describe the resulting effect on Hb:

**Table 7.3** Population PK parameters for ESA2 [4]

Parameter	Unit	Value	Remark
Fixed effects			
$CL$	L/d	0.749	
$V$	L	4.72	
$ka$	1/d	0.825	
$F$	1	0.394	
Random effects			
$CL$	%CV	28	
$V$	%CV	27	
$ka$	%CV	82	
Covariate effects			
$bwCL$	1	0.571	$CL = (bw/50)^{bwCL}$ Effect of body weight on $CL$
$bwV$	1	0.443	$V = (bw/50)^{bwV}$ Effect of body weight on $V$
$ageV$	1	0.267	$V = (age/50)^{ageV}$ Effect of age on $V$

$$\frac{d}{dt}Hb = S(t) - S(t - LS) + S_{ESA1}(t); \quad t \geq 0 \quad (7.3)$$

$$S_{ESA1}(t) = \begin{cases} \frac{Hb_0 - Hb_{sw}}{LS} & ; \quad t \in [0 ; LS] \\ 0 & ; \quad t > LS \end{cases} \quad (7.4)$$

$$S(t) = \frac{Hb_0}{LS} \cdot [1 + E(c(t))] \quad (7.5)$$

$$E(c(t)) = \frac{S_{max} \cdot c(t)}{SC_{50} + c(t)} \quad (7.6)$$

$Hb$ —hemoglobin,  $t$ —time after switch from ESA1 to ESA2,  $S$ —production rate of hemoglobin,  $LS$ —life span of red blood cells (RBC),  $S_{ESA1}$ —residual effect of ESA1 on hemoglobin,  $Hb_0$ —hemoglobin at baseline (before ESA1 treatment),  $E$ —relative drug effect on hemoglobin production,  $Hb_{sw}$ —hemoglobin at time of switch,  $S_{max}$ —maximum relative increase from baseline in hemoglobin production,  $SC_{50}$ —ESA2 drug concentrations producing half-maximum drug effect.

At the time of switch (i.e., time zero), the ESA1 will no longer stimulate RBC production and Hb values will decline from values at the time of switch ( $Hb_{sw}$ ) to pretreatment levels ( $Hb_0$ ) after another life span. This rate of decline is described by the  $S_{ESA1}$  term (7.4). The  $S_{ESA1}$  term is only effective during one life span after the time of switch. Thereafter, previous ESA treatment has no impact on Hb which then can only be affected by ESA2 treatment. Note that no use is made of ESA1 PK.

Population PD parameters for these equations are listed below (Table 7.4). We use an empirical covariate relationship of ESA1 dose on  $SC_{50}$  of ESA2.

ESA2 treatment starts with drug amount dose which is assumed to be the final dose of ESA1 treatment.  $Dose$  is calculated as  $Dose = 10^u$  with  $u \sim U(1,4)$ , i.e., uniformly distributed over the interval [1, 4]. According to the covariate relationship, dose also impacts on the  $SC_{50}$  of ESA2.

**Table 7.4** Population PD parameters for ESA2 [4]

Parameter	Unit	Value	Remark
Fixed effects			
$LS$	d	61.3	
$Hb_0$	g/dL	9.3	
$S_{\max}$	1	0.425	
$SC_{50}$	ng/mL	0.898	
Random effects			
$LS$	%CV	32	
$Hb_0$	%CV	25	
$S_{\max}$	%CV	142	
$SC_{50}$	%CV	559	
Covariate effects			
$doseSC_{50}$	1	0.3	$SC_{50} \cdot (Dose/ref)^{doseSC_{50}}$ Effect of ESA1 dose on $SC_{50}$

**Table 7.5** Dose adaptation table (% change from previous dose)

Hb (g/dL)	Hb change from baseline (g/dL)				
	< -2	[-2, -1]	[-1, 1]	[1, 2]	>2
<8	100	100	100	100	100
8–9	50	50	50	50	50
9–10.5	25	25	25	25	25
10.5–13.5	50	25	0	–25	–50
13.5–14	–25	–25	–25	–25	–25
>14	–100	–100	–100	–100	–100

The dose adaptation rules, specified in Table 7.5 describe dose changes as a function of absolute Hb values and Hb differences relative to baseline. Hb values are observed weekly. Dose changes could start at each dosing time (1x/2 weeks), but any two dose adaptations had to be separated by 4 weeks.

### 7.4.3 Computational Model

To build the computational model for the situation described above, the following issues were taken into account:

- Delay differential equation: due to the life span-based PD model a DDE solver is needed.
- Events: dose adaptation has to be considered in the simulation, therefore the DDE solver must run with the **Events** option activated.
- In order to decrease simulation run time, a code was implemented that executes more than one subject at a time (vectorization).

The MATLAB implementation is shown in Listing 7.2.

**Listing 7.2 Program `esaCTS.m`**

```

function esaCTS(regimenType, numSub, runs)
%ESACTS Launch the Clinical Trial Simulation (CTS) for ESA model.
%   ESACTS(REGIMENTYPE,NUMSUB,RUNS) initializes and controls the
%   CTS based on the PK-PD ESA model. The simulation is performed
%   simultaneously for |NUMSUB| subjects, and consists of |RUNS|
%   repetitions. Graphs with plasma concentration, hemoglobin time
%   courses, and dose adaptation over time are initialized for the
%   simulation time. |REGIMENTYPE| is linked to the dose regimen and
%   must be here 'protocol' but the applied dose schedule function
%   can be easily extended for other regimens.
%
%   Example for calling:   esaCTS('protocol', 2, 10)
%   This will simulate the drug PK-PD 2 times, each time with 10
%   subjects
%
%   (see also esaCTS>DOSESCHEDULE).

% Check input arguments and provide default values if necessary
error(nargchk(0,3,nargin)); %#ok<NCHKN>
if nargin < 3
    runs = 10;
end
if nargin < 2
    numSub = 2;
end
if nargin < 1
    regimenType = 'protocol';
end

% create a data file for statistics (results database). To avoid
% that the file will be overwritten, the file name is linked to the
% current date and time.
c = fix(clock);
dt = [num2str(c(1)) num2str(c(2)) num2str(c(3)) '_' ...
      num2str(c(4)) num2str(c(5)) num2str(c(6))];
filename = [dt '.STA'];
fclose(fopen(filename, 'w'));
for nr=1:runs
    figs = createFigs(1);
    esaCTSmmain(regimenType, numSub, nr, figs, filename);
    if nr == 1      % save only figures from the first replication
        print(figs.consFigure, '-r300', '-dtiff', ['cons', '_'])
        print(figs.HbFigure, '-r300', '-dtiff', ['Hb', '_'])
        print(figs.doseFigure, '-r300', '-dtiff', ['dose', '_'])
        close([figs.consFigure, figs.HbFigure, figs.doseFigure])
    else
        close([figs.consFigure, figs.HbFigure, figs.doseFigure])
    end;
end
end

function esaCTSmmain(regimenType, numSub, numRun, graph, filename)
%ESACTSMMAIN Conduct a trial simulation for the ESA model.

```

```

% ESACTSMAIN(REGIMENTYPE,NUMSUB,NUMRUN,GRAPH,FILENAME) conducts
% one trial simulation for |NUMSUB| subjects simultaneously.
% |NUMRUN| denotes the current repetition number, |REGIMENTYPE| -
% dose regimen and |FILENAME| - a file for saving the simulation
% results. The structure |GRAPH| contains handles for figure
% windows and axes used for graphical presentation.
% See also esaCTS.

% Reset the random number generator to the default
if numRun == 1
    rng default
end
% Calculate dosing times and amount based on regimen
p = initializeParams(numSub);
[doseTimes, ~] = doseSchedule(regimentType);
% Correlate ESA2 dose and ES1 dose
doseAmount0 = p.doseESAcov;

consAxes = graph.consAxes;
HbAxes = graph.HbAxes;
doseAxes = graph.doseAxes;

% Simulate system for all subjects simultaneously

% Initialize parameters and set initial values for all subjects
% p = initializeParams(numSub);
% complete structure p (used in several functions)
p.numSub = numSub;
p.doseAmount(1:numSub) = doseAmount0;
p.doseTimes = doseTimes;
p.doseTrue(1:numSub) = 1;
p.lastAdapt(1:numSub) = -28;
p.switcher = 1;
p.tmax = doseTimes(end);
p.inity = zeros(1, numSub);
p.numRun = numRun;
p.doseNext = 2;

sol = [p.doseAmount p.c0 p.Hbsw]';
delay = p.LS;

% Simulate system for each treatment period
options = ddeset('Events',@fevents, 'RelTol',1e-5, 'AbsTol', 1e-5);
% Set time interval for this treatment period
tspan = [doseTimes(1), doseTimes(end)];

t0 = doseTimes(1);
tx = t0;
ie = numSub + 1;
p.dosex = p.doseAmount;
while t0 < p.tmax
    if ie > numSub
        % decrease accuracy to detect the next event
        % (after a 'dose time' event)
        options.AbsTol = 1e-4;
        options.RelTol = 1e-4;
        showDosing(tx, t0, p.dosex, p.doseAmount, numSub, doseAxes)
        % Save the current dose and time
    end
    [t0, tx, ie] = fevents(t0, tx, ie, options);
end

```

```

        tx = t0;
        p.dosex = p.doseAmount;
    else
        % increase accuracy to detect the next event
        % (after an 'end life span' event)
        options.AbsTol = 1e-6;
        options.RelTol = 1e-6;
    end
    sol = dde23(@derivatives, delay, sol, tspan, options, p);
    te = sol.xe';
    %ye = sol.ye';
    ie = sol.ie;
    % Plotting time courses
    tPoints = sol.x;
    concPoints = sol.y( numSub+1 : 2*numSub, :);
    HbPoints = sol.y(2*numSub+1 : 3*numSub, :);
    plot(concAxes, tPoints, concPoints)
    plot(HbAxes, tPoints, HbPoints)
    drawnow
    ie = ie(te == te(end)); % eliminate already handled events
    p.te = te(end);
    p = todoevents(sol.y', ie, p);
    % Prepare next period (next dose)
    t0 = sol.x(end);
    tspan = [t0 p.tmax];
    sol.ie = [];
    sol.xe = [];
    sol.ye = [];
    sol.discont = [];
    sol.y(1:numSub, end) = p.inity;
end
% save data for the statistical test (Hb end values)
savedata(filename, numRun, sol.y(:, end))

end

function [doseTimes, doseAmount] = doseSchedule(regimenType)
%DOSESCHEDULE Create a vector of dosing times and amounts of drug.
% [DOSETIMES, DOSEAMOUNT] = DOSESCHEDULE(REGIMENTYPE) returns
% a vector |DOSETIMES| of dosing times and a scalar |DOSEAMOUNT|
% of dosing amount. Currently, |REGIMENTYPE| can only be
% 'protocol' but there placeholder for other options.

treatmentWeeks = 56;
cycleWeeks = 2;
daysInWeek = 7;
hoursInDay = 24;
initialTime = 0;

% treatment time [hours]
endTime = treatmentWeeks * daysInWeek * hoursInDay;
% cycle time [hours]
cycleTime = cycleWeeks * daysInWeek * hoursInDay;

switch regimenType
    case 'protocol'

```

```

        dailyDoses = 1;
        doseInitial = 20;    % dose in [ng]
        timeOnDrug = 0;      % (first dose time) = (last dose time)
    % case 'other' % placeholder for other regimen(s)
    otherwise
        messageID = 'Book:esaCTS:UnknownRegimen';
        messageStr = ['Unknown regimen '%s'. Regimen must be ',...
            ''protocol''.'];
        error(messageID, messageStr, regimenType);
end

% create vector of treatment times
initialCycleTimes = ...
    initialTime : cycleTime : (endTime - cycleTime);
doseTimesWithinCycle = ...
    initialTime : hoursInDay/dailyDoses : timeOnDrug;
doseTimes = reshape(...
    repmat(doseTimesWithinCycle', 1, length(initialCycleTimes))+ ...
    repmat(initialCycleTimes, length(doseTimesWithinCycle), 1), ...
    1, length(initialCycleTimes)*length(doseTimesWithinCycle));
doseTimes = [doseTimes, endTime]; % adding end time of treatment
doseTimes = doseTimes/24;
doseAmount = doseInitial;

end

function params = initializeParams(numSub)
%INITIALIZEPARAMS Create initial values for model parameters.
% PARAMS = INITIALIZEPARAMS(NUMSUB) returns a structure
% |PARAMS| containing initial values for model parameters,
% including variability when necessary. The values are created
% for all simulated subjects; number of subjects is |NUMSUB|.

% Scope of variability
params = [];
pkCV = 0.32;
% PK parameters
ka = 0.825;                % 1-st order absorption rate [1/day]
kaCV = 0.82;
params.ka = variability('varnorm', ka, ka*kaCV, numSub, 0.5);
F = 0.394;                 % bioavailability
params.F = F*ones(1,numSub); % no variability
CL = 0.749;               % drug clearances [L/day]
CLv = variability('varnorm', CL, CL*pkCV, numSub, 0.3);
% weight covariate effect on CL
bwref = 66;               % reference value for the body weight covariate
bwCL = 0.571;
bwcov = covData(75, 10, numSub, 50, 90, 'normal'); % body weight
params.CL = CLv.*(bwcov/bwref).^bwCL;
V = 4.72;                 % volumes of distribution [L]
Vv = variability('varnorm', V, V*pkCV, numSub, 0.3);
% weight covariate effect on V
bwref = 66;               % reference value for body weight covariate
bwV = 0.443;
% age covariate effect on V
aref = 54;                % reference value for the age covariate
aV = 0.443;

```



```

acov = covData(75, 10, numSub, 20, 90, 'normal');
params.V = Vv.*(bwcov/bwref).^bwV.*(acov/aref).^aV;
params.c0 = zeros(1,numSub);      % initial concentrations
% PD parameters
LS = 61.3;                        % life span of RBC [day]
LSCV = 0.32;
params.LS = variability('varnorm', LS, LS*LSCV, numSub,30);
Hb0 = 9.3;                        % Hb conc. at baseline [ng/mL]
Hb0CV = 0.25;
H1 = covData(Hb0, Hb0*Hb0CV, numSub, 7, 10.5, 'normal');
H2 = covData(Hb0, Hb0*Hb0CV, numSub, 10.5, 12.5, 'normal');
params.Hb0 = min(H1,H2);
params.Hbsw = max(H1,H2);        % Hb conc. at switch    [ng/mL]
Smax = 0.425;                    % max effect
SmaxCV = 1.42;
params.Smax = variability('varnorm', Smax, Smax*SmaxCV, numSub, 0.5);
SC50 = 0.898;                    % concentration linked to 50% of max effect
SC50CV = 5.59;
SC50v = variability('varnorm', SC50, SC50*SC50CV, numSub, SC50);
% ESA dose covariate effect on SC50
doseESAref = 10^2.5;
doseESASC50 = 0.3;
a = 1; b = 4;
doseESAcov = rand(1, numSub)*(b - a) + a;
params.doseESAcov = 10.^doseESAcov;
params.SC50 = SC50v.*(params.doseESAcov./doseESAref).^doseESASC50;

end

function x = variability(distribution, m, s, num, lim)
% .....
% .....
% .....

end

function cx = covData(varargin)
%COVADATA Generate demographics.
% CX = COVADATA(VARARGIN) generates an 1-by-n array with random
% demographic data, for incorporation of a covariate effect in the
% trial simulation. Calling types:
% CX = COVADATA(MU,SIGMA,N,CXMIN,CXMAX,TYPE) for generation of |N|
% normally distributed values, with the mean |MU| and standard
% deviation |SIGMA| that represent such covariates as body weights,
% heights, etc.; |TYPE| must be 'normal'. The values are truncated
% to an interval with lower cutoff |MIN| and upper cut off |MAX|.
% CX = COVADATA(N,TYPE) for binary distributed data that represent
% covariates like sex, smokers ('yes','no'), etc. |TYPE| must be
% 'binary'.

na = nargin;
switch na
case 6
    % normal distribution
    mu = varargin{1};          % mean
    sigma = varargin{2};       % standard deviation

```

```

    n = varargin{3};          % length of the data array
    cxmin = varargin{4};      %
    cxmax = varargin{5};
    type = varargin{6};
    if ~strcmp(type, 'normal'); errormessage(1); end
    cx(1:n) = 0;
    for i = 1:n
        x = normrnd(mu, sigma, 1, 1);
        in = cxmin <= x & x <= cxmax;
        while ~in
            x = normrnd(mu, sigma, 1, 1);
            in = cxmin <= x & x <= cxmax;
        end
        cx(i) = x;
    end
case 2
    % binary distribution
    n = varargin{1};
    type = varargin{2};
    if ~strcmp(type, 'binary'); errormessage(1); end
    cx = binornd(1, 0.5, 1, n);
otherwise
    errormessage(3)
end

function errormessage(em)
%ERRORMESSAGE Error message.
%   ERRORMESSAGE(EM) displays an error message if the arguments,
%   varargin, passed into covarEffect are not correct. The
%   parameter |EM| is a message identifier number.
%   The function is nested due to |type| and |na|.

switch em
case 1
    messageID = 'Book:covarEffect:type';
    messageStr = 'Verify the distribution type ''%s''';
    error(messageID, messageStr, type);
case 2
    messageID = 'Book:covarEffect:type';
    messageStr = 'Verify the distribution type ''%s''';
    error(messageID, messageStr, type);
case 3
    messageID = 'Book:covarEffect:nargin';
    messageStr = 'Verify the number of arguments ''%d''';
    error(messageID, messageStr, na);

end

end

end

function E = hillEffect(c, E0, Emax, EC50, n)
% .....
% .....
% .....

end

```

```

function dydt = derivatives(~, Y, Z, p)
%DERIVATIVES Compute the right-hand side of the DDE.
%   DYDT = DERIVATIVES(T, Y, Z, P) ...

ns = p.numSub;
% set dependent variables for all subjects simultaneously
% (vectorization)
a = y(      1 : ns)';      % drug amount
c = y( ns+1 : 2*ns)';      % concentration
Hb = y(2*ns+1 : 3*ns)';    %#ok<NASGU> % hemoglobin
cdelay = diag(Z(ns+1 : 2*ns, 1:ns))'; % delay matrix linked to |c|

% PK Model (SC administration)
dAdt = -p.ka.*a;
dcdt = (p.ka./p.V./p.F.*a - p.CL./p.V.*c);

% PD Model
SESA = (p.Hb0 - p.Hbsw)./p.LS .* p.switcher;
EC    = hillEffect(c, 0, p.Smax, p.SC50, 1);
ECdelay = hillEffect(cdelay, 0, p.Smax, p.SC50, 1);
S      = p.Hb0./p.LS.*(1 + EC);
Sdelay = p.Hb0./p.LS.*(1 + ECdelay);
dHbdt = S - Sdelay + SESA;
% Derivatives vector of ODE system
dydt = [ dAdt'
         dcdt'
         dHbdt'];

end

function [value,isterminal,direction] = fevents(t, ~, ~, p)
%FEVENTS Specify events for the DDE model.
%   [VALUE,ISTERMINAL,DIRECTION] = ...

ns = p.numSub;

% Event: end of life-span (LS)
direction1(1:ns,1) = 1;
isterminal1(1:ns,1) = 1;
value1(1:ns,1) = t - p.LS;

% Event: Dosing
direction2(1,1) = 1;
isterminal2(1,1) = 1;
value2(1,1) = t - p.doseTimes(p.doseNext);

% Event 1 + 2
direction = [direction1; direction2];
isterminal = [isterminal1; isterminal2];
value = [value1; value2];

end

function pMod = todoevents(y, numEvents, p)
%TODOEVENTS Handle detected events.
%   PMOD = TODOEVENTS(Y, NUMEVENTS, P) conducts a proper action if

```

```

% events happen. The function handles events detected at 1 time
% point. In typical cases 1 event is linked to 1 time point.
% However, in general it may happen that simultaneous events take
% place, and the function then handles more than 1 event detected
% at the same time. |NUMEVENTS| is a vector with event numbers,
% |P| - structure with parameters. |Y| - current solution values
% (dependent variables). Output parameters: |PMod| is a changed
% |P|, according to action undertaken.

pMod = p;
ns = pMod.numSub;
% use 'for' as more than 1 event can happen at the same time
for i = numEvents
    % identify subject number
    switch i
        case num2cell(1:ns)
            % end of life-span (LS)
            subject = (p.numRun-1)*ns + i;
            display(['time:', num2str(p.te), ...
                ', event:end of life-span, ', 'subject:', ...
                num2str(subject)])
            pMod.switcher = 0;
            pMod.inity(1:ns) = y(end, 1:ns);
        case num2cell(ns+1)
            % dosing time
            display(['time:', num2str(p.te), ', event:dosing time'])
            [pMod.doseTrue, pMod.lastAdapt] = doseAdaptation(p, y);
            pMod.doseAmount = pMod.doseAmount.*pMod.doseTrue;
            pMod.doseAmount(pMod.doseAmount == 0 & ...
                pMod.doseTrue > 3/4) = 1;
            pMod.inity(1:ns) = y(end, 1:ns) + pMod.doseAmount;
            pMod.doseNext = pMod.doseNext + 1;
        otherwise
            messageID = 'Book:CTS:UnknownEvent';
            messageStr = 'Unknown event ''%s''.';
            error(messageID, messageStr, num2str(i));
    end
end

end

function [doseAdapt, lastAdapt] = doseAdaptation(p, y)
%DOSEADAPTATION Perform dose adaptation.
% [DOSEADAPT LASTADAPT] = DOSEADAPTATION(P, Y) specifies dose
% changes in terms of absolute Hb values, Hb values relative
% to baseline, and Hb values observed within any 4-weeks period.
% Dose changes could start at each dosing time (1x/2 weeks), but
% once the first dose adaptation has been performed the next dose
% adaptations can't occur more often than 1x/4 weeks. |Y| stands
% for independent variables in the model, and |P| - a structure
% with parameters needed for validation of the adaptation rules.
% This function returns |DOSEADAPT|- vector with values showing
% how the doses were changed, and |LASTADAPT| - vector with times
% when the doses were changed; both regarding all subjects.

adaptRule = (p.doseTimes(p.doseNext) - p.lastAdapt) >= 28; % 4-weeks
doseAdapt = p.doseTrue;

```

```

Hb = y(end, 2*p.numSub+1 : 3*p.numSub);
% Hemoglobin difference from baseline - levels
deltaHb = Hb - p.Hb0;
a = deltaHb < -2;
b = -2 <= deltaHb & deltaHb < -1;
c = -1 <= deltaHb & deltaHb < 1;
d = 1 <= deltaHb & deltaHb < 2;
e = 2 <= deltaHb ;
% dose adaption to hemoglobin concentration - levels
doseAdapt( Hb < 8 ) = 2;
doseAdapt( 8 <= Hb & Hb < 9 ) = 3/2;
doseAdapt( 9 <= Hb & Hb < 10.5 ) = 5/4;
doseAdapt(10.5 <= Hb & Hb <= 13.5 & a) = 3/2;
doseAdapt(10.5 <= Hb & Hb <= 13.5 & b) = 5/4;
doseAdapt(10.5 <= Hb & Hb <= 13.5 & c) = 1;
doseAdapt(10.5 <= Hb & Hb <= 13.5 & d) = 3/4;
doseAdapt(10.5 <= Hb & Hb <= 13.5 & e) = 1/2;
doseAdapt(13.5 < Hb & Hb <= 14 ) = 3/4;
doseAdapt(14 < Hb ) = 0;
% save adaptation flag and adaptation time for each subject
doseAdapt = doseAdapt.*adaptRule + ~adaptRule;
lastAdapt = p.doseTimes(p.doseNext)*adaptRule + ...
    p.lastAdapt.*(doseAdapt == 1);
end

function fg = createFigs(selgraph)
%CREATEFIGS Prepare figures.
% FG = CREATEFIGS(SELGRAPH) creates figure objects, indicated by
% the vector |SELGRAPH|, for several graphs used during the trial
% simulation. The handles of created objects (figure windows and
% axes) are stored in the structure |FG|.

% 1 - Set up figures for time courses
if ismember(1, selgraph)
    % Concentration
    fg.consFigure = figure;
    fg.consAxes = axes;
    set(fg.consAxes, 'FontSize', 15, 'YScale', 'log')
    xlabel('Time [d]')
    ylabel('Concentration [ng/mL]')
    title('Concentration Time Course')
    hold on; grid on;

    % hemoglobin time courses
    fg.HbFigure = figure;
    fg.HbAxes = axes;
    set(fg.HbAxes, 'FontSize', 15)
    xlabel('Time [d]')
    ylabel('Hb [g/dL]')
    title('Hb Time Course')
    hold on; grid on;

    % dosing
    fg.doseFigure = figure;
    fg.doseAxes = axes;
    set(fg.doseAxes, 'FontSize', 15, 'YScale', 'log')
    xlabel('Time [d]')

```

```

    ylabel('Dose [ng]')
    title('Dose Adaptation')
%     title('No Dose Adaptation')
    hold on; grid on;
    set([fg.consFigure; fg.HbFigure; fg.doseFigure], 'units', ...
        'normalized', {'Position'}, {[0.0, 0.4, 0.3, 0.4]; ...
        [0.35, 0.4, 0.3, 0.4]; [0.7, 0.4, 0.3, 0.4]});
end

% 2 - Set up a figure for CTS power values
if ismember(2, selgraph)
    fg.powerFigure = figure;
    fg.powerAxes = axes;
    set(fg.powerAxes, 'FontSize', 15, 'XScale', 'log')
    xlabel(' ')
    set(fg.powerFigure, 'units', 'normalized', 'Position', ...
        [0.1, 0.1, 0.6, 0.15], 'PaperPositionMode','auto')
    set(fg.powerAxes, 'YTickLabel',{'';'';''}, 'YTick', [], 'Box', 'on')
    hold on
end

end

function showDosing(t1, t2, val1, val2, ns, haxes)
%SHOWDOSING Show current and previos dose values.
%   SHOWDOSING(T1, T2, VAL1, VAL1, NS, HAXES) plots values of two
%   vectors, |VAL1| and |VAL2|, each containing |NS| elements, at
%   time points |T1| and |T2|. The graph axes are pointed by |HAXES|.
%
%   This function if used to show the difference between current and
%   previous dose.

% use 'xr' if jitter is needed to visualize multiple values
xr = rand(1,ns); %#ok<NASGU
plot(haxes, [t1, t2]', [val1; val2], '-+')

end

function savedata(filename, n, y)
%SAVEDATA Save simulation results.
%   SAVEDATA(FILENAME,N,Y) saves in the text file |FILENAME| final
%   values of the hemoglobin level, obtained in the repetition |N|.
%   The hemoglobin values are stored in the ODE model solution |Y|.

fid = fopen(filename, 'a');
lc = length(y);          % number of columns in the solution
nHb = lc/3*2+1 : lc; % select only Hb columns
fprintf(fid, '%s \n', num2str([n y(nHb)]));
fclose(fid);

end

```

### 7.4.4 Trial Simulations

To demonstrate the impact of the dose adaptation on the attainment of stable Hb values within a target range, we will use the **esACTS.m** program (Listing 7.2) to conduct trial simulations with and without dose adaptation. Both cases use the same numbers of subjects, **nsub**, and replications, **runs**. Listing 7.3 shows how to exclude the dose adaptation rules from trial simulations. Setting the elements of dose adaptation vector **doseAdapt** to **1** switches adaptation rules off (value **1** means that the current dose does not change). It is possible to use the same subjects' characteristics with and without dose adaptation by initializing the random number generator with the same initial value for both scenarios.

**Listing 7.3** The modified **doseAdaptation** function where **1** is assigned to all elements of vector **doseAdapt**. This excludes the dose adaptation rules from the trial simulations

```
function [doseAdapt, lastAdapt] = doseAdaptation(p, y)
%DOSEADAPTATION Perform dose adaptation.
%   [DOSEADAPT LASTADAPT] = ...

% .....
% .....
% .....

% dose adaptation to hemoglobin concentration - levels
doseAdapt(      Hb < 8      ) = 1;
doseAdapt( 8  <= Hb & Hb < 9      ) = 1;
doseAdapt( 9  <= Hb & Hb < 10.5    ) = 1;
doseAdapt(10.5 <= Hb & Hb <= 13.5 & a) = 1;
doseAdapt(10.5 <= Hb & Hb <= 13.5 & b) = 1;
doseAdapt(10.5 <= Hb & Hb <= 13.5 & c) = 1;
doseAdapt(10.5 <= Hb & Hb <= 13.5 & d) = 1;
doseAdapt(10.5 <= Hb & Hb <= 13.5 & e) = 1;
doseAdapt(13.5 <  Hb & Hb <= 14      ) = 1;
doseAdapt(14   <  Hb      ) = 1;

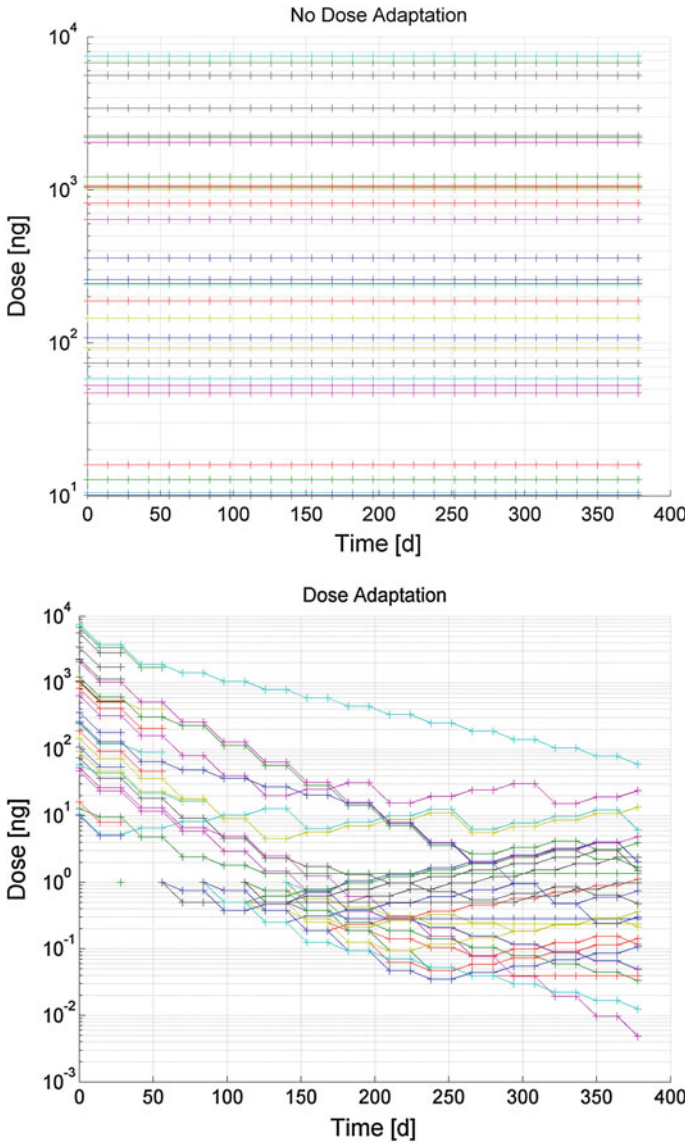
% .....
% .....
% .....

end
```

The results produced by **esACTS.m** (Listing 7.2, with **nsub** = 30, **runs** = 50) are shown in Figs. 7.7, 7.8, 7.9 and 7.10.

To evaluate the output, we compare 50 pairwise replicates of 30 subjects each produced by both scenarios. Given the success criterion “Hb is within 10.5–12.5 g/dL” fourfold tables are specified as shown below (Tables 7.6 and 7.7) and statistically evaluated, as described in Listing 7.4.

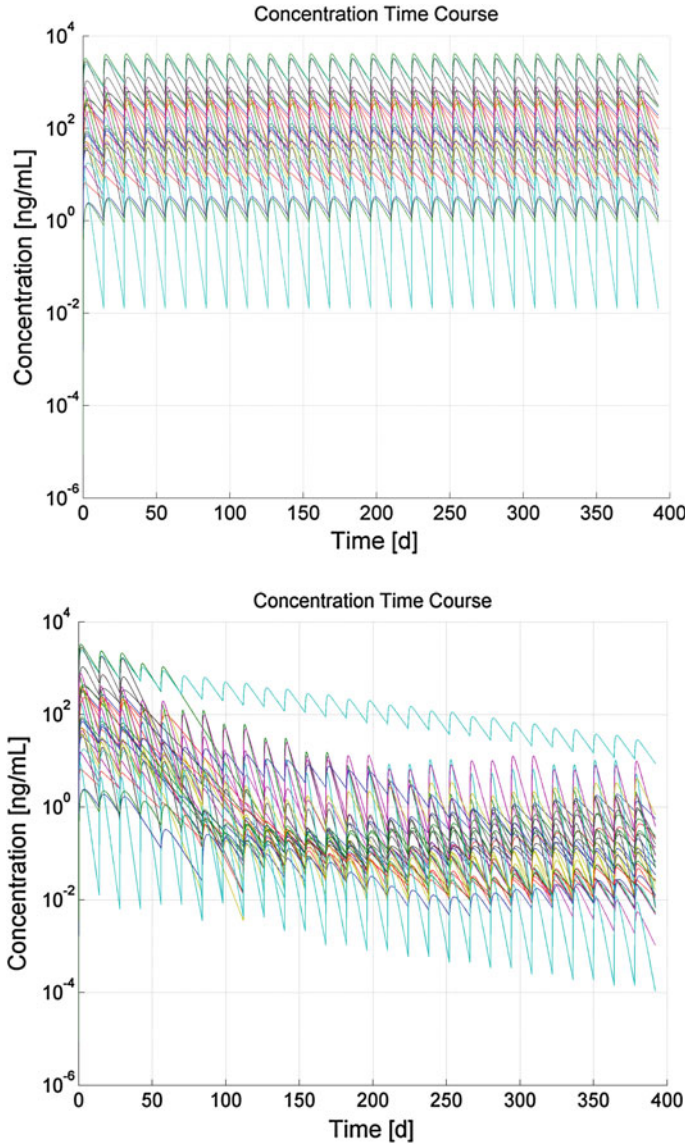
Listing 7.4 shows the **esACTSpower.m** program that can evaluate trial simulations for both dependent and independent distributions. It first calculates a



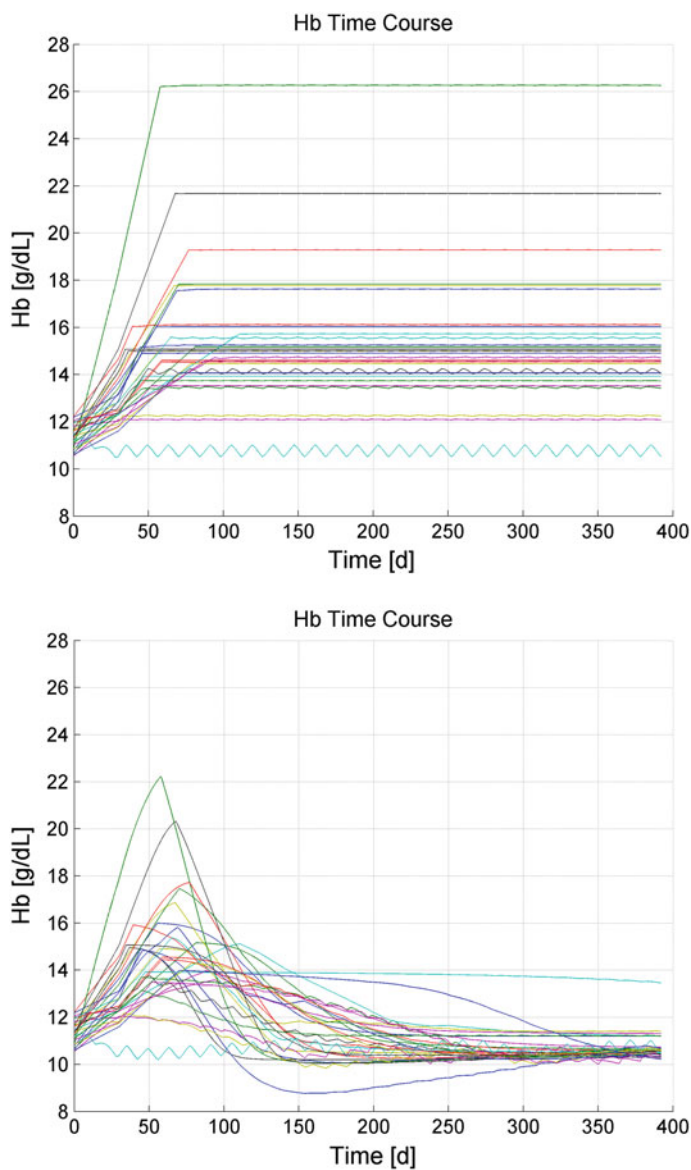
**Fig. 7.7** Example of a dosing history in 30 subjects receiving a longer-acting ESA at time zero; *upper panel*: without dose adaptation, *lower panel*: with dose adaptation (drug doses are adapted to absolute Hb values and changes from baseline). Doses set to zero are not indicated

vector of  $p$ -values (one  $p$ -value per pairwise replicates) and then the CTS power defined as the ratio of the number of  $p$ -values  $< \alpha$  to the number of replicates. Figures 7.10 and 7.11 show the results and graphs for simulations with dependent and independent samples, respectively.

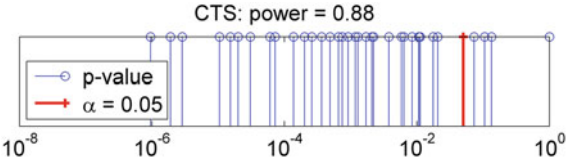




**Fig. 7.8** Drug concentration profiles of a new ESA in 30 subjects treated q2w over 56 weeks; *upper panel*: without dose adaptation, *lower panel*: with dose adaptation



**Fig. 7.9** Hb concentration profiles in 30 subjects who switched at time zero from a short-acting to a longer-acting ESA; *upper panel*: without dose adaptation; *lower panel*: with dose adaptation (drug doses are adapted to absolute Hb values and changes from baseline)



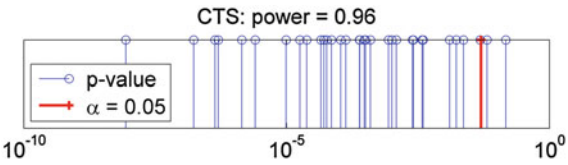
**Fig. 7.10** Distribution of trial outcomes ( $p$ -values) for dependent samples. The CTS power of 0.88 is obtained as ratio of  $p$ -values  $<0.05$  to the number of replicates (50) with 30 subjects per replicate

**Table 7.6** Fourfold table for two dependent distributions:  $a$ ,  $b$ ,  $c$ , and  $d$  are frequencies obtained from two different scenarios (with and without dose adaptation) for the same sample

	Without adaptation	
	Inside	Outside
With adaptation		
Inside	$a$	$b$
Outside	$c$	$d$

**Table 7.7** Fourfold table for two independent distributions:  $a$ ,  $b$ ,  $c$ , and  $d$  are frequencies obtained from two different scenarios (with and without dose adaptation) on two different samples

	Subjects	
	Inside	Outside
Without adaptation	$a$	$b$
With adaptation	$c$	$d$



**Fig. 7.11** Distribution of trial outcomes ( $p$ -values) for independent samples. The CTS power of 0.96 is obtained as the ratio of the number of  $p$ -values  $<0.05$  to the number of replications (50) with 30 subjects per replicate

**Listing 7.4** Program **esaCTSpower.m**

```

function esaCTSpower
%ESACTSPOWER Calculate the power of a trial simulation
%   ESACTSPOWER reads from external files the results of two ESA
%   trial simulations and calculates the power.
%
%   See also esaCTS.

fileNoAdapt = '2013519_141618_000.STA';
fileAdaptDip = '2013519_17927_DIP.STA';
fileAdaptInd = '2013519_194336_IND.STA';
% find the number of lines = the number of replications
fid = fopen(fileNoAdapt, 'r');
nLines = 0;
while (fgets(fid) ~= -1); nLines = nLines+1; end
reps= nLines;
alpha = 0.05;

% trial simulation with 2 dependent samples
p = analysis(fileNoAdapt, fileAdaptDip, reps, 'dependent');
figs = createFigs(2);
outcomes(p, figs.powerAxes, alpha);
print(figs.powerFigure, '-dtiff','-r600','esaCTSpowerDIP')

% trial simulation with 2 independent samples
p = analysis(fileNoAdapt, fileAdaptInd, reps, 'independent');
figs = createFigs(2);
outcomes(p, figs.powerAxes, alpha);
print(figs.powerFigure, '-dtiff','-r600','esaCTSpowerIND')

end

function pval = analysis(filename1, filename2, reps, testcase)
%ANALYSIS Verify effects of the active drug adaptation.
%   PVAL = ANALYSIS(FILENAME1, FILENAME1, REPS, TESTCASE) compares
%   two groups, the first treated without dose adaptation the
%   without. The results databases stored in files |FILENAME1| and
%   |FILENAME2| for the first and second group respectively (produced
%   in trial simulations by the esaCTS.m program). The statistical
%   z-test is used to verify the null-hypothesis H0 (there is no
%   statistically significant effect of the drug adaptation) against
%   the alternative hypothesis (the drug adaptation is significant).
%   The verification is carried out pairwise taking into account
%   results from the respective replication in both scenarios (with
%   and without adaptation), i.e. it is repeated |REPS| times. For
%   each replication pair, a p-value is computed, and finally a
%   vector |PVAL| is created that contains all p-values. The
%   comparison is possible for dependent samples: |TESTCASE| =
%   'dependent', or independent: |TESTCASE| = 'independent'.
%   See also esaCTS.

```

```

fid1 = fopen(filename1,'r'); X1 = fscanf(fid1,'%f'); fclose(fid1);
fid2 = fopen(filename2,'r'); X2 = fscanf(fid2,'%f'); fclose(fid2);
numSub = length(X1)/reps; % or numSub = length(X2)/reps;
% Ignore first column
X1 = reshape(X1, numSub, reps)'; X1r = X1(:,2:end);
X2 = reshape(X2, numSub, reps)'; X2r = X2(:,2:end);
%predifined range of Hb values
X1in = 10.5 <= X1r & X1r <= 12.5;
X2in = 10.5 <= X2r & X2r <= 12.5;

switch testcase
    case 'dependent'
        % One-tailed McNemar test (one-tailed)
        xYesYes = sum( X1in & X2in, 2); a = xYesYes;
        xYesNo = sum( X1in & ~X2in, 2); b = xYesNo;
        xNoYes = sum(~X1in & X2in, 2); c = xNoYes;
        xNoNo = sum(~X1in & ~X2in, 2); d = xNoNo;
        pval = zeros(numSub,1);
        for nr = 1:reps
            n = b(nr) + c(nr);
            mbc = min([b(nr), c(nr)]);
            pval(nr) = 0;
            for m = 0:mbc
                biC = factorial(n)/(factorial(m)*factorial(n-m));
                pval(nr) = pval(nr) + biC*(1/2)^n;
            end
        end
    case 'independent'
        % z test (if samples are independent)
        yNaYes = sum( X1in,2); a = yNaYes; % no adapt. - 'success'
        yNaNo = sum(~X1in,2); b = yNaNo; % no adapt. - 'failure'
        yAdYes = sum( X2in,2); c = yAdYes; % adapt. - 'success'
        yAdNo = sum(~X2in,2); d = yAdNo; % adapt. - 'failure'
        p1 = a./numSub; p2 = c./numSub;
        p = (a+c)./(2*numSub); q = 1-p;
        z = (p1-p2)./sqrt(p.*q./numSub+p.*q./numSub);
        pval = normcdf(z);
    otherwise
end
end

function outcomes(p, haxes, alpha)
%OUTCOMES Show the results of the CTS power calculation.
% OUTCOMES(P, HAXES, ALPHA) produces a graph in the axis box
% indicated by the handle |HAXES|. It shows how p-values, |P|,
% are distributed against |ALPHA| (the significance level for
% acceptance or rejection of the hypothesis H0).

```

```

power = sum(p < alpha)/length(p);
stem(haxes, p, ones(length(p),1))
hold on
stem(haxes, alpha, 1,'+r','LineWidth',2)
legend('p-value', ['{\alpha} = ', num2str(alpha)], 'Location', ...
      'SouthWest')
title(['CTS: power = ', num2str(power)])
display(['CTS: power = ', num2str(power)])

end

function fg = createFigs(selgraph)
% .....
% .....
% .....

end

```

Figure 7.10 shows the output of the **esaCTSpower.m** program after a simulation of dependent samples (the same subjects simulated twice, with and without dose adaptation).

For the simulation with independent samples (two different groups) the distribution of the  $p$ -values and the CTS power are illustrated in Fig. 7.11.

## 7.5 Exercises

### Exercise 7.1

Modify the **ceraCTS.m** program adding one event that will enable you to interrupt the running CTS process after 168 days and then decide, by entering (**Y** or **N**), if the procedure has to be continued or stopped.

## References

1. Hale M, Gillespie WR, Gupta S, Tuk B, Holford NH (1995) Clinical trial simulation. Streamlining your drug development process. *Appl Clin Trials* 5:35–40
2. Johnson SCD (1998) The role of simulation in the management of research: what can the pharmaceutical industry learn from the aerospace industry? *Drug Inf J* 32:961–969
3. Lee JY, Garnett CE, Gobburu JVS, Bhattaram VA, Brar S, Earp JC, Jadhav PR, Krudys K, Lesko LJ, Li F, Liu J, Madabushi R, Marathe A, Mehrotra N, Tornøe C, Wang Y, Zhu H (2000) Impact of pharmacometric analyses on new drug approval and labelling decisions. A review of 198 submissions between, and 2008. *Clin Pharmacokinet* 50:627–635
4. Chanu P, Gieschke R, Charoin JE, Pannier A, Reigner B (2010) Population pharmacokinetic/pharmacodynamic model for C.E.R.A. in both ESA-naïve and ESA-treated chronic kidney disease patients with renal anemia. *J Clin Pharmacol* 50:507–520
5. Garred LJ, Pretlac R (1991) Mathematical modeling of erythropoietin therapy. *ASAIO J* 37:M457–M459

## Chapter 8

# Graphics-based Modeling

*A graph is worth a thousand words (anonymous).*

Graphics greatly enhance the modeling of complex systems. As described in [Sect. 1.2](#), modeling generally starts with a drawing, or a model A, which connects relevant model elements in some logical order. The next step would then be to quantify the interrelationships between system elements, e.g., to assign zero- or first-order reactions, resulting in a dynamic model representation, model B. Graphics-based modeling tools enhance the process of model building through visualization of relationships between model elements. In the following, we will introduce two MATLAB-based modeling tools, SimBiology and Simulink, for the graphical representation of PK-PD and biological (disease) models.

### 8.1 SimBiology

#### 8.1.1 Model Components

SimBiology, a MATLAB toolbox, was designed to model dynamic systems, such as encountered in PK, PD, and systems biology [1]. SimBiology code can be generated via a GUI, named SimBiology Desktop, or using the MATLAB command line. SimBiology also allows the importing of models generated according to the standards defined by the Systems Biology Mark-up Language (SBML) group.

The basic model building elements in SimBiology are species, compartments, reactions, parameters, rules, and events.

*Species*, the state variables of the dynamic system, can be grouped into *compartments* which can contain other compartments. This allows the building of hierarchically structured models.

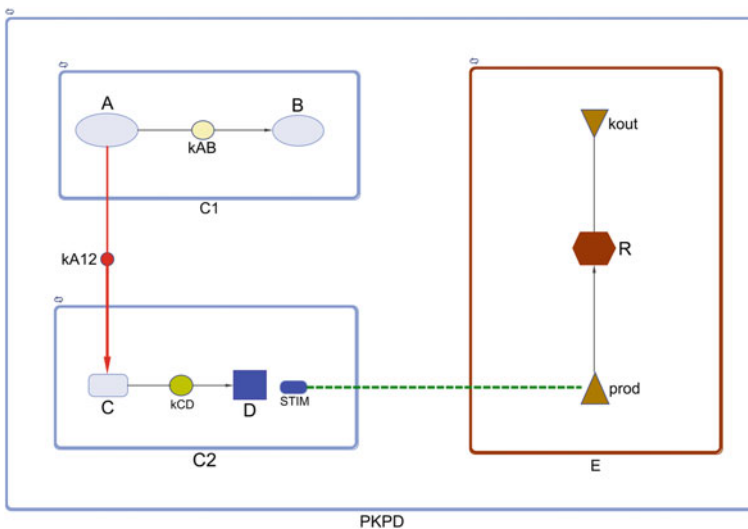
*Reactions* carry information about how species values change over time. If a reaction is directed from species A to species B, A is called reactant and B product.

The simultaneous change of reactant and product is specified in a reaction rate associated with the reaction. Figure 8.1 shows the relationship among species, compartments, and reactions.

*Parameters* are quantities that specify numerical values for rate constants in reactions. SimBiology allows the use of parameters that change during the simulation, as is the case for the model in Fig. 8.1 (STIM acts on prod).

*Rules* are mathematical expressions that algebraically link state variables, e.g., initial values of species, and parameters. An example is given in Fig. 8.1 showing how the state STIM is linked to the state D via a repeated assignment rule. Other examples are the calculation of drug concentrations by dividing the amount of a species by the respective volume. Rules further specify the initial conditions of state variables.

An *event* describes an instantaneous change in a state variable and/or model parameter. Events can be time-driven or state-driven. A time-driven event occurs, e.g., when a drug is administered at a given time which updates the amount at the site of administration. State-driven events depend on conditions of one or more



**Fig. 8.1** Basic model building blocks in SimBiology. In this example there are four compartments: PK-PD, C1, C2, E; five species: A, B, C, D, R, and six reactions labeled:  $k_{A12}$ ,  $k_{AB}$ ,  $k_{CD}$ , prod, kout. Species are the states of the dynamic system defined by the model and can be grouped into compartments. Species are identified by compartment and name. Thus, the same species name can be used in different compartments. Reactions define how species change over time. Here,  $k_{12}$ ,  $k_{AB}$  and  $k_{CD}$  describe the conversion of A into B and of A into D via C. D triggers a stimulatory signal, STIM, which impacts the production of species R. The dashed line indicates that STIM is not changed as a result of acting on the rate labeled *prod*. kout is related to the loss in R. Species can be represented by 15 symbols and user-defined block image files (extension: png, gif, jpg). All elements can be extensively colored, and line width can be varied widely. Overall, this allows a clear representation of a model



states. If the condition is fulfilled a pre-specified action is performed. In SimBiology state events are triggered only on the positive edge, meaning that conditions must change from false to true to elicit an event.

### 8.1.2 The SimBiology Desktop

The SimBiology graphical user interface (GUI) is opened from the MATLAB command line with

```
>> simbiology
```

or using the SimBiology icon under the APPS tab.

Selecting Add Model under the HOME tab and choosing *Create New Blank Model...* together with the Custom VIEW will result in Fig. 8.2.

This Window is used to diagrammatically create a model structure by dragging and dropping species, compartments, and reactions into the open space. Other modeling structures, such as parameters, rules, and events can be inserted via the

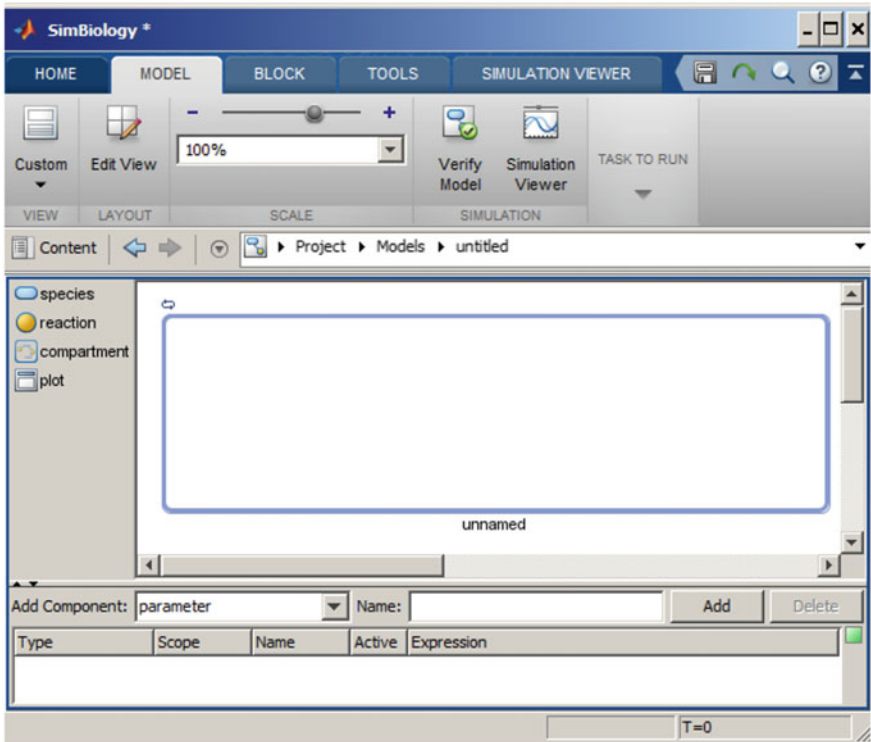
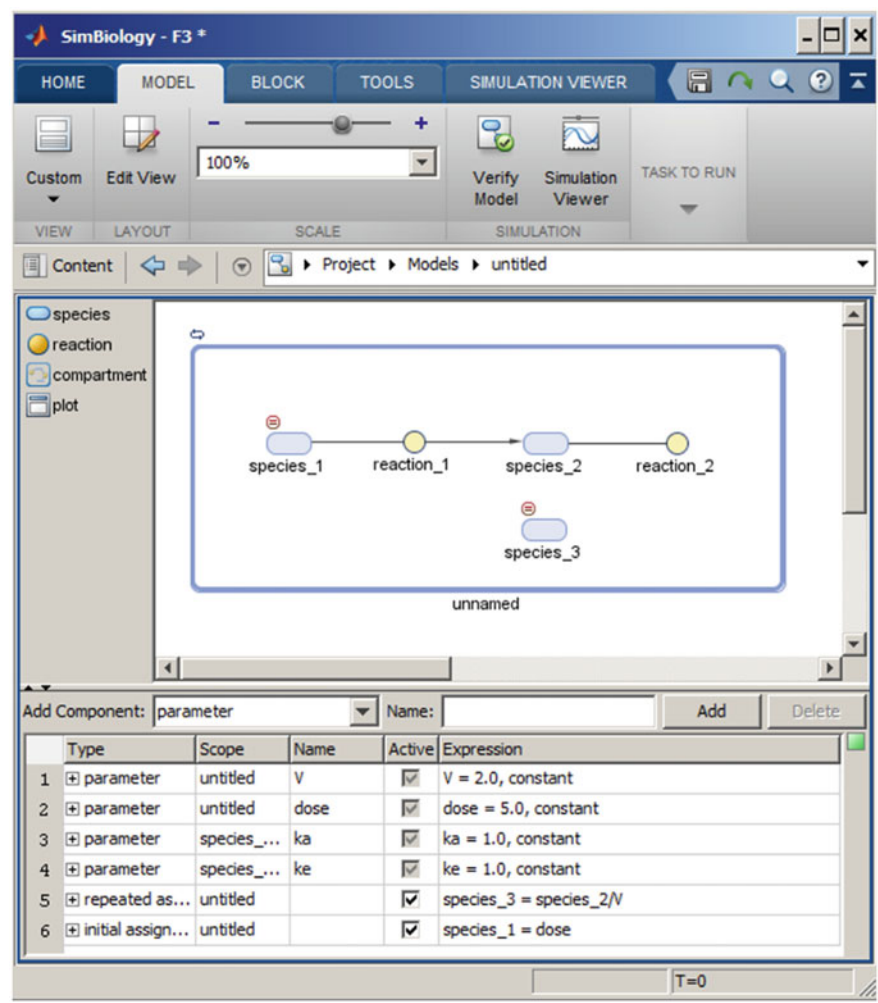


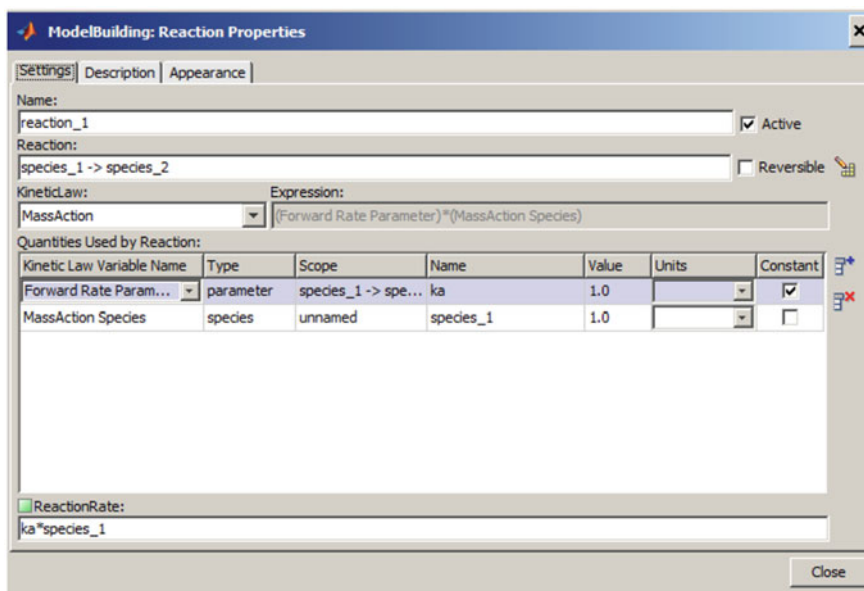
Fig. 8.2 Creating a SimBiology model via the SimBiology desktop

Add Component entry (Fig. 8.3). In addition to the Custom VIEW, there are views that only show the diagram or a table of model structures. The user may switch between these views as needed.

Within the Custom VIEW diagram, reaction rates for reactions can be entered in the reaction properties window which opens after selecting a reaction in the diagram (Fig. 8.4).



**Fig. 8.3** Creating a one-compartment oral PK model in SimBiology via the SimBiology Desktop using Custom VIEW. Initial drug amounts ( $dose = 5$ ) and calculation of drug concentrations from amounts ( $species_2$ ) are determined via rules (indicated by the red circles with the equal sign). The model uses first-order absorption (parameter:  $ka$ ) and first-order elimination (parameter:  $ke$ ); a volume of  $V = 2$  is assigned to  $species_3$

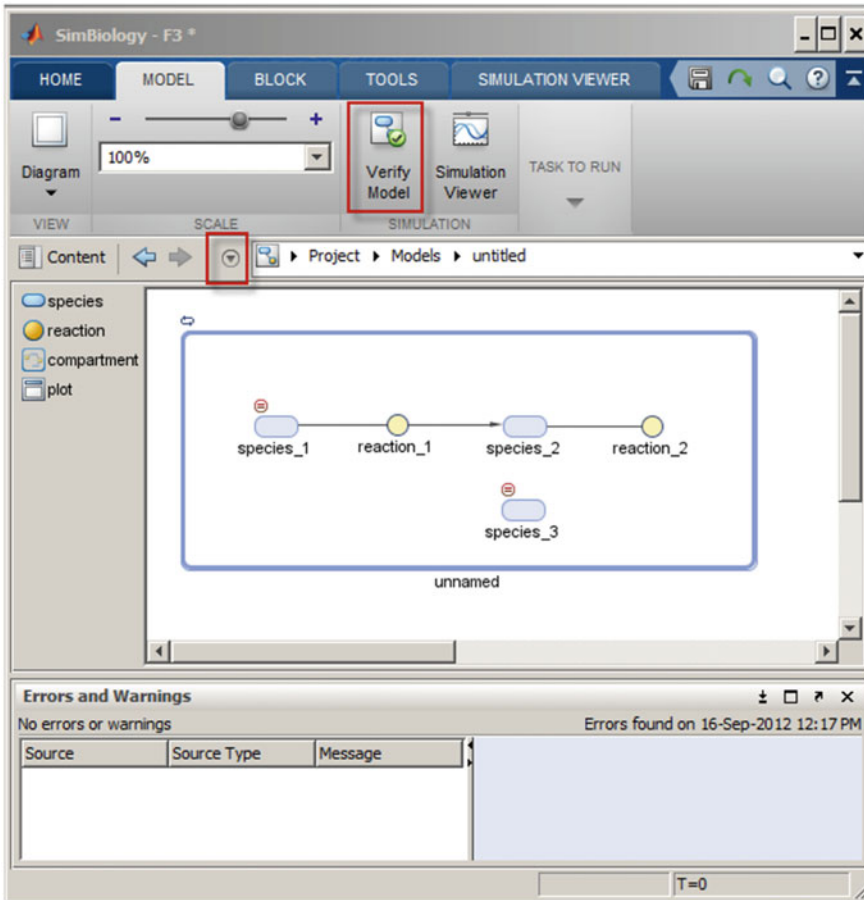


**Fig. 8.4** Definition of reaction properties. After selecting the reaction symbol (yellow circle in Fig. 8.3), reactions can be specified either directly in the *ReactionRate* box or defined by selecting pre-specified *KineticLaws*. This figure shows the respective entries for a *MassAction* kinetic law where the rate of change in a species is proportional to its amount (or concentration). The proportionality factor, in SimBiology *Forward Rate Parameter*, is a model parameter named *ka*

There are two ways to verify a SimBiology Model, both indicated in Fig. 8.5. One is to use the Verify Model button under the Model icon. SimBiology checks the consistency of model entries and reports any problems under Errors and Warnings. The other aims to protect against technically correct but logically faulty models by providing the model equations in standard notation. This is done by selecting the circled triangle button and then *Export Model as Equation....* (Fig. 8.6).

Once the model is verified, it can be explored with the Simulation Viewer. The Simulation Viewer allows the selection of states to be plotted by (right) clicking on the plot area and selecting *Properties...* (Fig. 8.7) or by selecting states directly from the diagram (Fig. 8.8). Generated plots can be overlaid to show potential parameter dependencies. Parameter values can be changed interactively via sliders. Slider properties can be adjusted after clicking on the *Adjust Quantity Values* icon. All these explorations can be done while simultaneously viewing the model diagram.

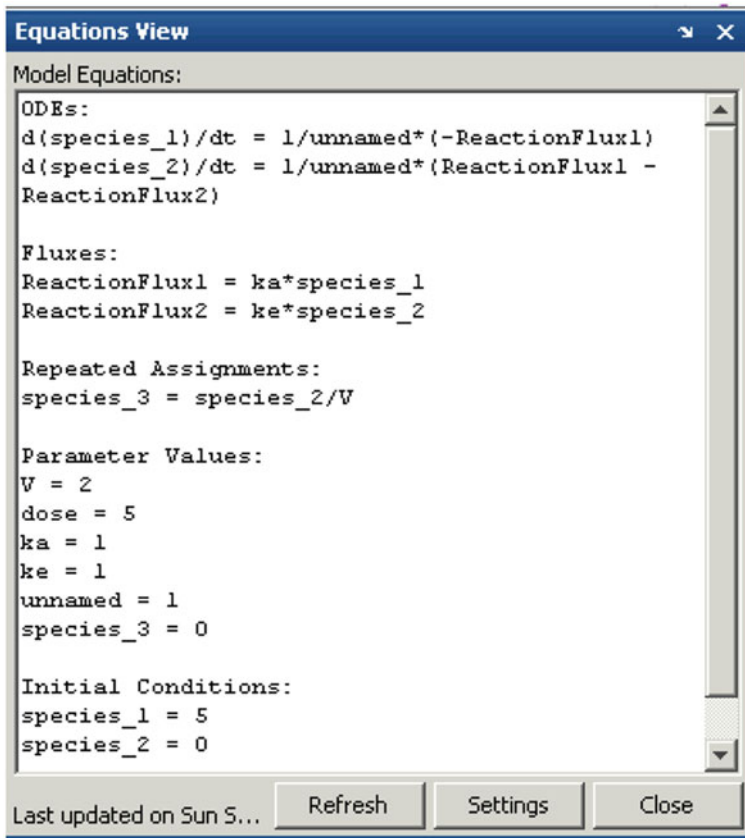
Exploring data with the help of sliders is especially useful when comparing observations with model predictions. Data can be imported into SimBiology from the MATLAB workspace or from files stored outside MATLAB. Each data import is kept separately in SimBiology and multiple imports can be generated (Figs. 8.9, 8.10).



**Fig. 8.5** Verification of a SimBiology model. Verify model lets SimBiology check model consistency. No errors were found for the actual model. Using the circled triangle button allows export of model equations in standard notation

Assume that we wanted to know the area under the curve of *species\_3*, AUC<sub>3</sub>, over the whole integration interval, i.e., from time zero to simulation end time. Calculation of this integral is equivalent to solving the differential equation  $d/dt(\text{AUC}_3) = \text{species}_3$  with  $\text{AUC}_3(0) = 0$  at the end of the simulation interval. A convenient way to express this relationship in SimBiology is to use rate rules (Fig. 8.11).

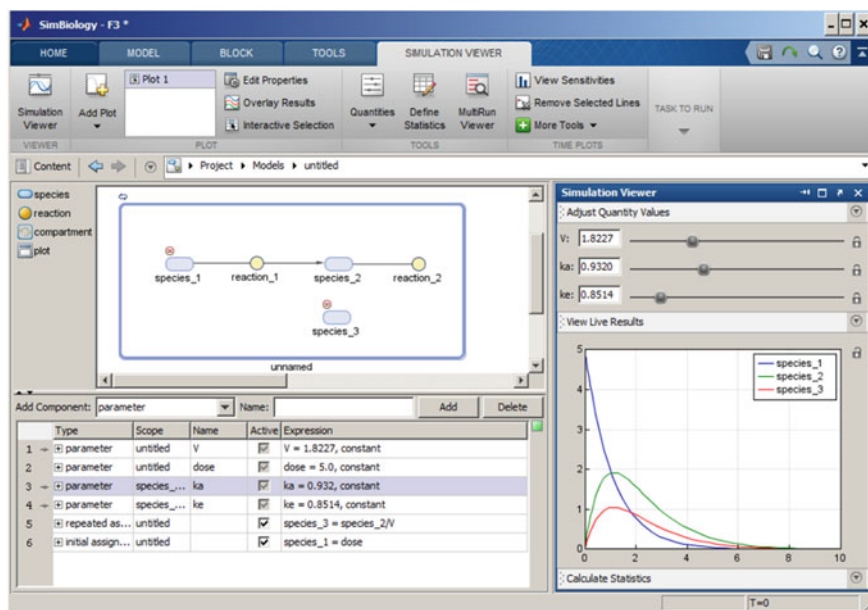
Another important feature of SimBiology is the handling of events. This can be time- or state-dependent. Known time-dependent events are dosing events where doses are applied at sites of administration according to a pre-specified dosage regimen. Such events are implemented in SimBiology via the Doses icon. Dosage regimens can be specified for scheduled and repeated dosing (Fig. 8.12).



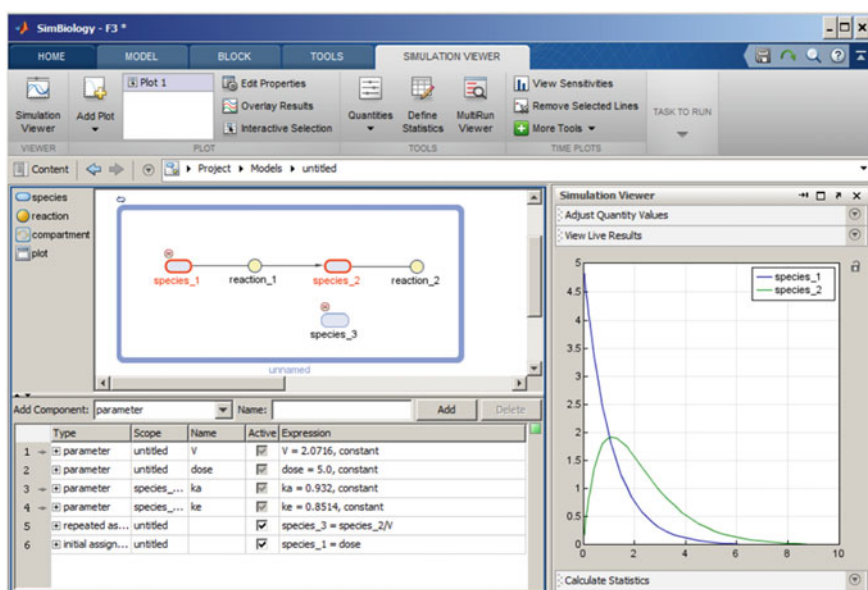
**Fig. 8.6** Verification of a SimBiology model via the model equations. The compartment name *unnamed* is handled as a parameter and can be used to scale amounts in compartments. It is set equal to 1 by default

State-dependent events comprise the definition of an event, i.e., a logical expression of states and parameters, which can be true or false, and the actions caused when the event becomes true. An example is shown in Fig. 8.13 where the dosage regimen (dose and dosing interval) is adapted to threshold values of *species\_3* AUC.

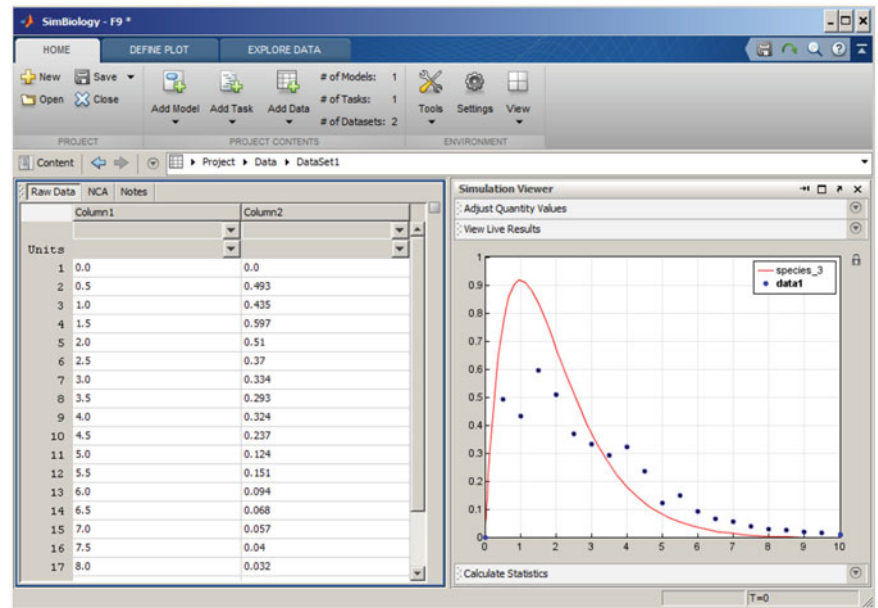
Another useful task provided by SimBiology is sensitivity analysis (Fig. 8.14).



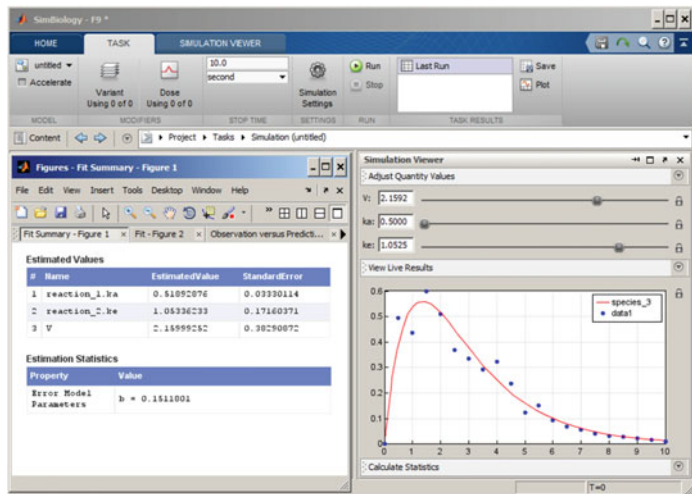
**Fig. 8.7** Exploring a SimBiology model with the SimBiology desktop. Model diagram and non-diagrammatic features (such as parameters, rules) are presented with the computed model outcome. Parameter values can be changed interactively using sliders and the resulting graphs overlaid



**Fig. 8.8** Exploring a SimBiology model with the SimBiology desktop: selection of model states directly from the diagram

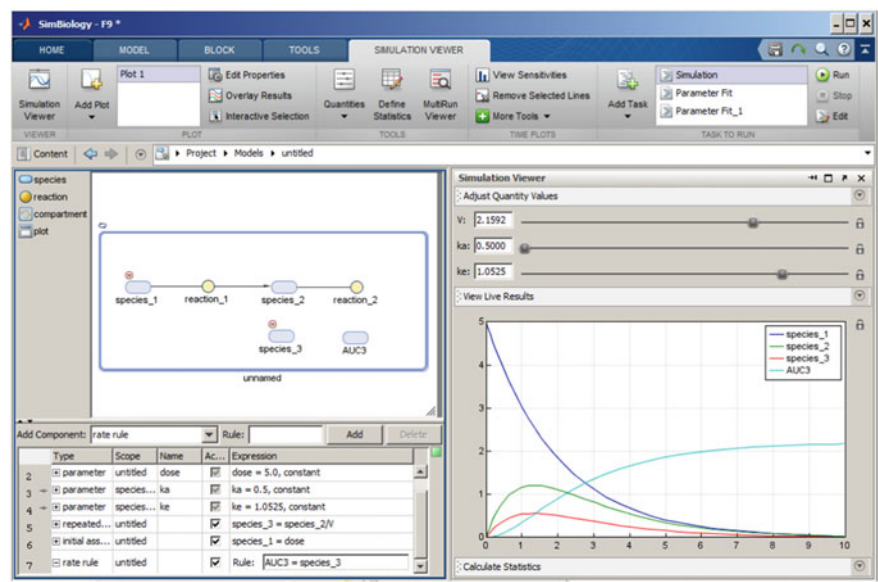


**Fig. 8.9** Data import into SimBiology from file (or MATLAB workspace). Data to be imported are organized such that times and dependent values (observations) appear in columns. More than one dependent value can be imported. Data can be compared with model prediction using the Simulation viewer

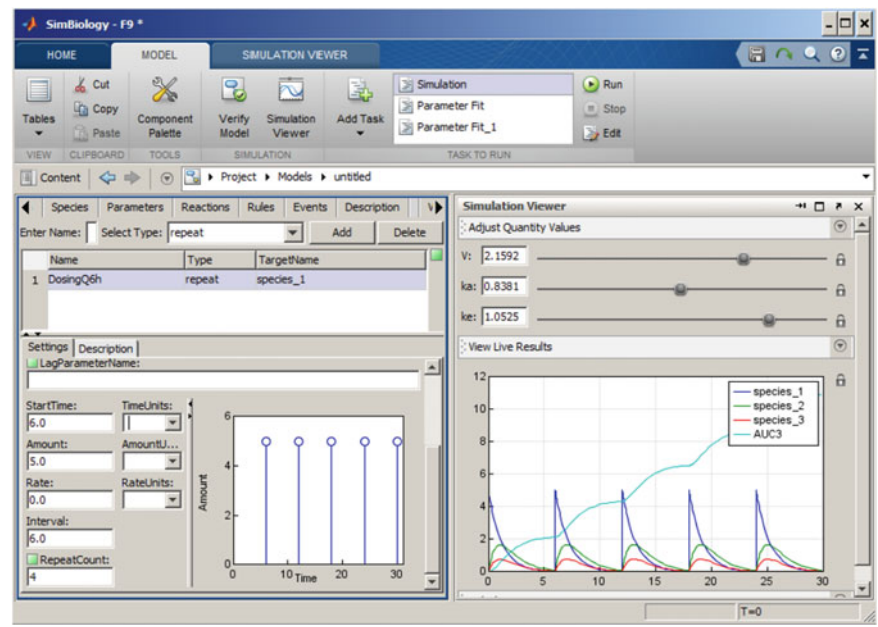


**Fig. 8.10** Imported data can be fitted using the underlying SimBiology model. Detailed model diagnostics are provided as separate graphics. The *Fit Summary* is shown here. For illustration purposes estimated model parameters were entered into the Simulation Viewer to generate the predicted curve



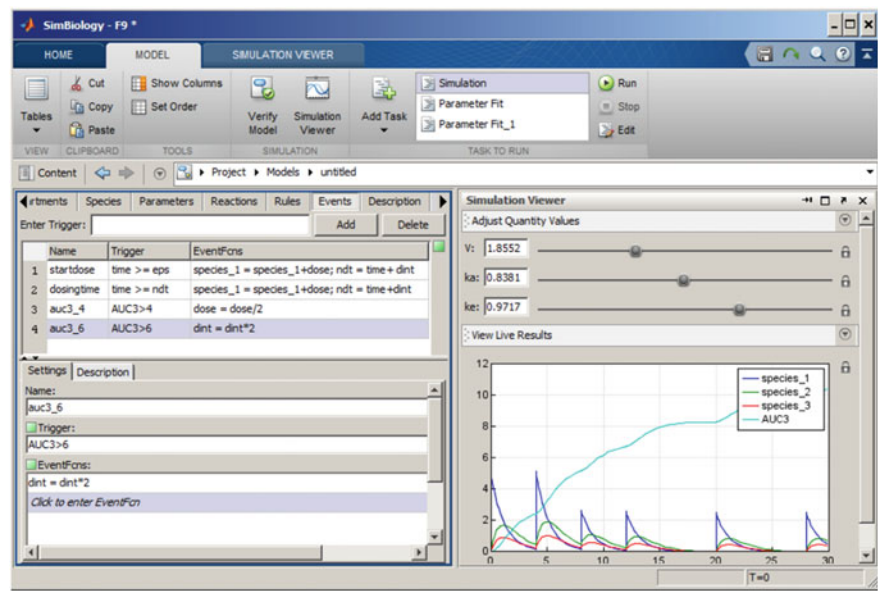


**Fig. 8.11** Application of rate rules to calculate an *AUC*. The expression  $AUC = species\_3$  codes for  $d/dt(AUC3) = species\_3$

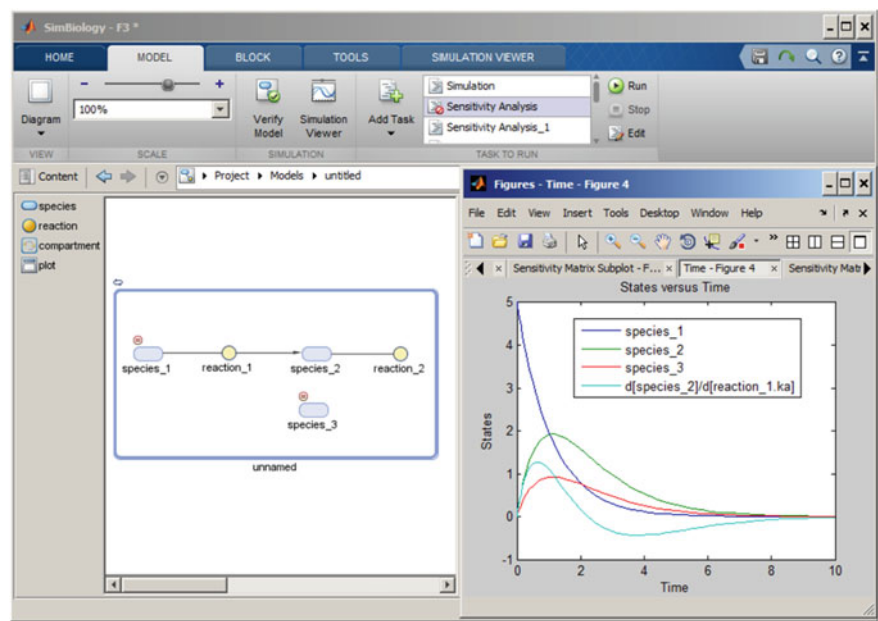


**Fig. 8.12** Scheduling of time events. Implementing q6h dosing. The dosing start time begins at 6 h as the model already implements an initial value for *species\_1*





**Fig. 8.13** Scheduling of time and state events. Triggered by *AUC3*, doses are halved ( $AUC3 > 4$ ) and dosing intervals doubled ( $AUC3 > 6$ )



**Fig. 8.14** Sensitivity analysis of *species\_2* with respect to absorption rate constant *ka*

### 8.1.3 The SimBiology Command Line

While the SimBiology desktop offers a convenient way to create models and run different modeling tasks, SimBiology can also be executed from the command line (Fig. 8.15). A SimBiology scripting language provides all functionality that is available within the SimBiology Desktop. There is also a MATLAB code capture tool that translates SimBiology Desktop activities into SimBiology scripting language commands. This concerns all activities that add information to the model such as defining a state variable, assigning a value to a parameter, etc. The captured commands can be saved as a MATLAB m-file and executed within the MATLAB environment. However, installation of SimBiology toolbox is still needed.

In this chapter we will introduce the SimBiology scripting language and show how some of the tasks described in the previous chapter are coded and executed.

Listing 8.1 shows how compartments, species, reactions, and parameters are defined.

**Listing 8.1** Program **PKmodel1.m**: building a PK model from the SB command line

```
% Create SimBiology Model.
m1 = sbiomodel('PKmodel');

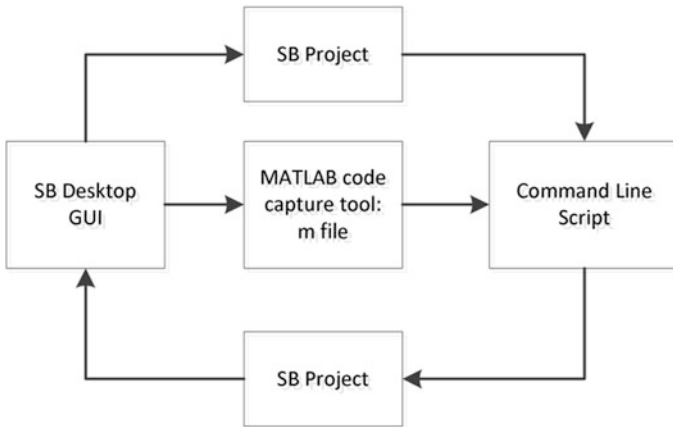
% Add Compartment and Species
c1 = addcompartment(m1, 'unnamed');
s1 = addspecies(c1, 'species_1', 'InitialAmount', 10.0);
s2 = addspecies(c1, 'species_2');
s3 = addspecies(m1, 'species_3');
s4 = addspecies(m1, 'AUC3');

% Add Reactions
r1 = addreaction(m1, 'species_1 -> species_2');
r2 = addreaction(m1, 'species_2 -> null');
set(r1, 'ReactionRate', 'ka*species_1');
set(r2, 'ReactionRate', 'ke*species_2');

% Add Parameters
p1 = addparameter(m1, 'ka', 1.0);
p2 = addparameter(m1, 'ke', 1.0);
p3 = addparameter(m1, 'V', 2.0);

% Add Rules
rule1 = addrule(m1, 'species_3 = species_2/V');
set(rule1, 'RuleType', 'repeatedAssignment');
rule2 = addrule(m1, 'AUC3 =species_3', 'RuleType', 'rate');

% Add Rules
sbiosaveproject('PKmodel1.sbproj', 'm1');
```



**Fig. 8.15** Interplay between SimBiology desktop and SimBiology *Command Line*. Starting from the SimBiology Desktop, work can be stored in a project file or captured in a MATALB m-file. Both entities can be executed and amended from the MATLAB command line which may result in a more advanced script. This script can be converted into a SimBiology project file and loaded into the SimBiology Desktop

The stored SimBiology model **PKmodel1.sbproj1** can then be loaded into SimBiology Desktop (Figs. 8.16, 8.17). As no information about positions of graphical elements on the diagram is provided in the script, SimBiology uses its own algorithm to place the elements on the Desktop.

How to run a simulation from the command line is shown in Listing 8.2, and the simulation result in Fig. 8.18.

**Listing 8.2** Program **PKmodel1c.m**: running a simulation from the command line (continuation of Listing 8.1)

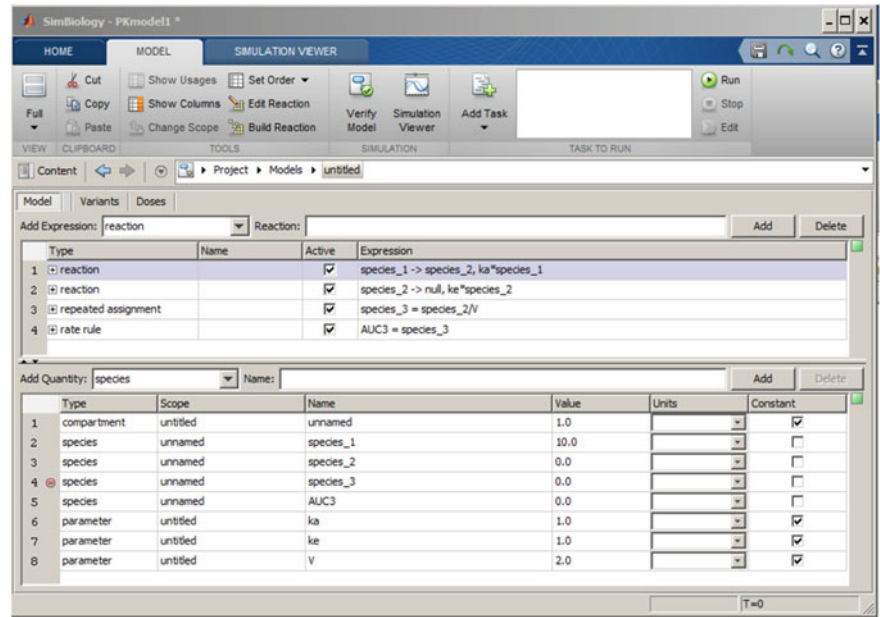
```
% Perform Simulation

% Initialize configset for analysis run.
cs = getconfigset(m1, 'default');

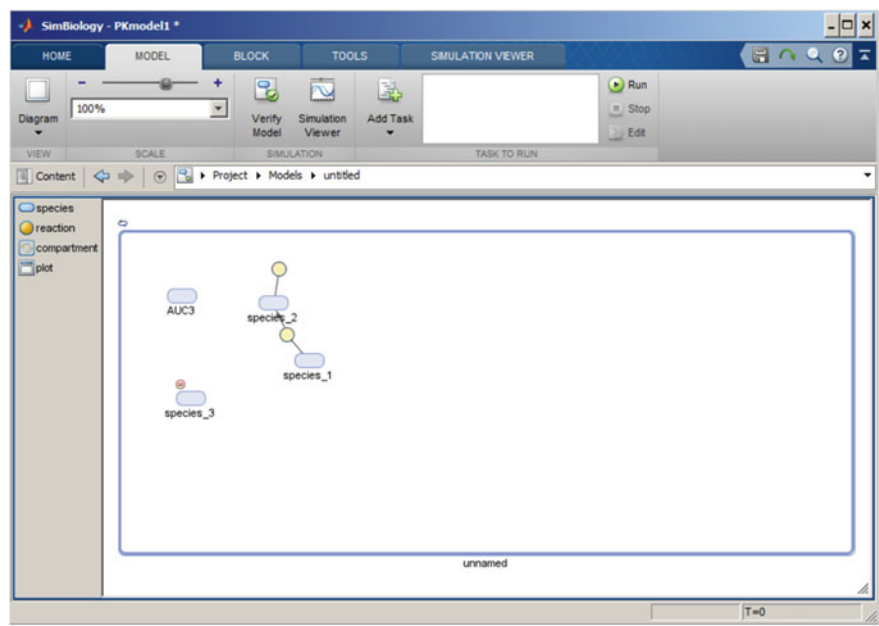
% Run simulation.
data = sbiosimulate(m1, cs, [], []);

% Show graph
set(axes, 'FontSize', 15)
plot(data.time,data.Data, 'LineWidth',2);
xlabel('Time [h]'); ylabel('Value [units]')
legend('species_1','species_2','species_3','AUC3');
print('-r600', '-dtiff', 'PKmodel1')
```

As our concluding example of SimBiology features, we will show how to perform parameter estimation with this tool. Listing 8.3 shows how to generate the data (depicted in Fig. 8.19) which are fitted as in Listing 8.4.

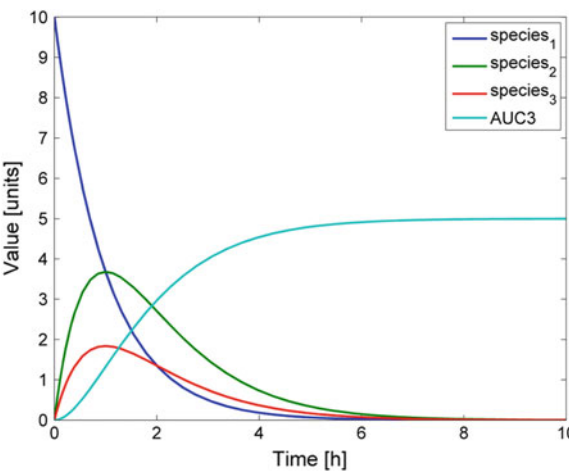


**Fig. 8.16** SimBiology desktop model generated from the *Command Line* language



**Fig. 8.17** SimBiology Desktop model generated from the *Command Line* language. Information about the position of graphical elements is not provided by the *Command Line* language

**Fig. 8.18** Modeling output from code prepared with the *Command Line* language



**Listing 8.3** Program **fitdata.m**: generating data to be fitted with the SimBiology *Command Line* language

```
% Generate Data to Fit
ka = 0.5; ke = 1.2; V = 2;
dose = 10;
times = (0:10)';
conc = dose/V*ka/(ke-ka)*(exp(-ka*times)-exp(-ke*times));
conc_obs = conc .* (1 + 0.01*randn(length(conc),1));
set(axes, 'FontSize', 15);
plot(data.time,data.Data(:,1:3),'LineWidth',2); hold on
plot(times, conc_obs,'or','LineWidth',1.5)
xlabel('Time [h]'); ylabel('Value [units]')
legend('species_1','species_2','species_3','AUC3','Observed');
print('-r800', '-dtiff', 'fitdata.tif')

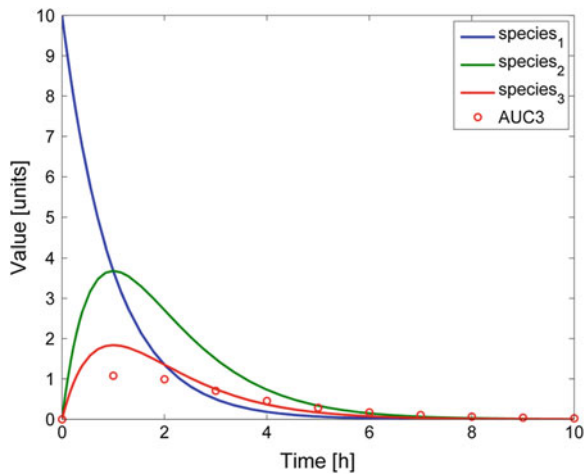
% Fit data
dataToFit = dataset(times,conc_obs) %#ok<NOPTS>
```

**Listing 8.4** Program **estimation.m**: running a parameter estimation from the *Command Line*

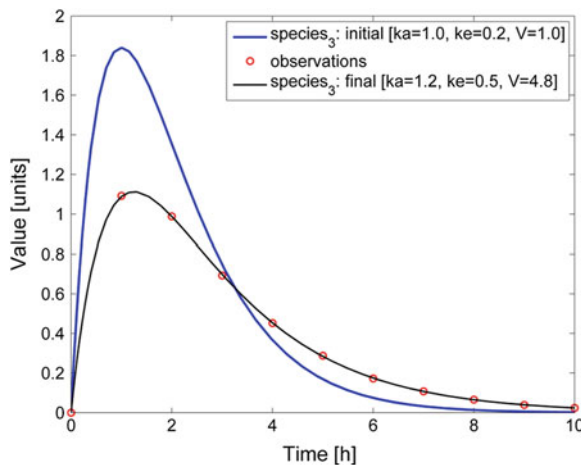
```
% Parameter Estimation
setactiveconfigset(m1, cs);

% PK Data
pkData = PKData(dataToFit);
pkData.IndependentVarLabel = 'times';
pkData.DependentVarLabel = {'conc_obs'};
% Define the parameters being estimated and their initial estimates.
estimatedParameters = {'ka', 'ke', 'V'};
initialEstimate = [1.0, 0.2, 1.0];
% Build a model map.
pkModelMap = PKModelMap;
pkModelMap.Observed = {'unnamed.species_3'};
pkModelMap.Estimated = estimatedParameters;
% Define estimate options.
options.ParamTransform = [0, 0, 0];
options.ErrorModel = 'proportional';
options.Pooled = false;
options.TolX = 1.0E-8;
options.TolFun = 1.0E-8;
options.MaxIter = 100;
% Estimate parameter values.
[results, simdataI] = sbionlinfit(m1, pkModelMap, pkData, ...
    initialEstimate, options);
ka = results.ParameterEstimates.Estimate(1);
ke = results.ParameterEstimates.Estimate(2);
V = results.ParameterEstimates.Estimate(3);
cc = 10*ka/(ke-ka)*(exp(-ka*data.time)-exp(-ke*data.time))/V;
ka1 = num2str(round(10*ka)/10);
ke1 = num2str(round(10*ke)/10);
V1 = num2str(round(10*V)/10);
```

```
% Configure the names of the resulting simulation data.
for i = 1:length(simdataI)
    simdataI(i).Name = 'SimData Individual';
end
% Convert dataToFit to a SimData object for plotting.
dataToFitPlot = SimData(dataToFit, 'times', '');
for i = 1:length(dataToFitPlot)
    dataToFitPlot(i).Name = 'Data To Fit';
end
set(axes, 'FontSize', 15)
plot(data.time,data.Data(:,3),'LineWidth',2)
hold on
plot(dataToFitPlot.time,dataToFitPlot.Data,'or','LineWidth',1.5)
plot(data.time,cc,'k','LineWidth',1.5)
xlabel('Time [h]');
ylabel('Value [units]')
legend('species_3: initial [ka=1.0, ke=0.2, V=1.0]', ...
    'observations', ['species_3: final [ka=',ka1,', ke=',ke1, ...
        ', V=', V1,']'] );
print('-r800','-dtiff','estimation.tif')
```



**Fig. 8.19** Model prediction and data for *species\_3* (to be fitted)



**Fig. 8.20** Result of parameter estimation using the SimBiology *Command Line* language

Figure 8.20 shows the estimation result produced by the **estimation.m** program.

## 8.2 Simulink

Simulink is an extension to MATLAB which uses an icon-driven interface for the construction of block diagrams (models, dynamic systems) [2]. These block diagrams are composed of components, called blocks, which utilize the input–output relationship according to Fig. 8.21.

To build a Simulink model, two strategies can be pursued, the top-down and bottom-up approach as illustrated in Fig. 8.22. Simulink supports these approaches by providing the ability to group blocks into subsystems. This not only provides simplified diagrams, but makes it easier to keep the modeling objectives in mind.

In the previous sections, one-compartment and two-compartment models were analyzed using built-in ODE solvers and the MATLAB programming language. In this section we will show how this can be done but with Simulink.

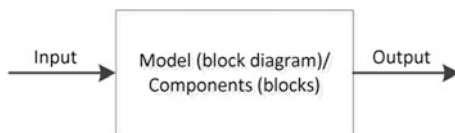
The input–output relationship for the integration of an ODE in Simulink is shown in Fig. 8.23.

In Simulink block diagrams, the integration of the ODE is explicitly indicated according to the following equivalence:

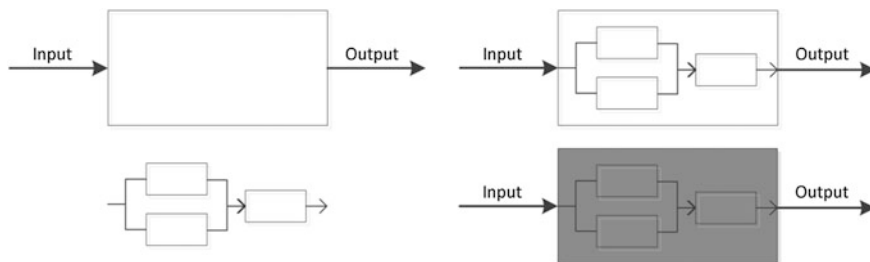
$$\frac{dy}{dt} = \mathbf{f}(\mathbf{y}, \boldsymbol{\beta}, t) \quad \Leftrightarrow \quad \mathbf{y} = \mathbf{y}(t_0) + \int_{t_0}^t \mathbf{f}(\mathbf{y}, \boldsymbol{\beta}, \tau) d\tau \quad (8.1)$$

Please note that quantities on the input and output of an integrator block usually have different dimensions, e.g., amount/time and amount. This is in contrast to



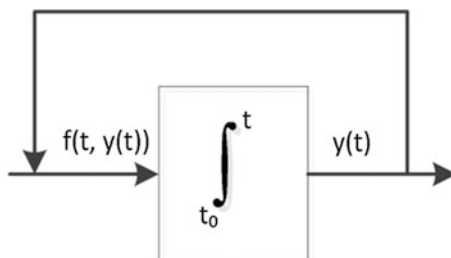


**Fig. 8.21** When building a Simulink model, each component of the model has to be described by a pre-configured input–output block taken from the Simulink Block Library. User-defined blocks can be created



**Fig. 8.22** Top-down (*upper panel*) and bottom-up (*lower panel*) model building strategies. Top-down approaches start from a high level model structure and refine it as required. Bottom-up approaches start from the conceptually lowest level and integrate blocks into subsystems

**Fig. 8.23** Integral block in Simulink to indicate the integration of an ODE. Input is the right-hand side of  $dy/dt = f(t, y(t))$  and output the solution  $y(t)$ .  $y(t)$  is connected back to the input providing an updated argument for the function  $f$



SimBiology where reactions attached to species carry the same dimension. The notion of a compartment is not available in Simulink.

### 8.2.1 One-Compartment Model (Simulink)

A one-compartment PK model with infusion input is given by the following differential equation:

$$\frac{dc}{dt} = \frac{r(t)}{V} - \frac{CL}{V} \cdot c ; \quad t \in [0 ; t_{\max}] \quad (8.2)$$

with a time-dependent infusion rate

$$r(t) = \begin{cases} R ; & t \in [0 ; t_R] \\ 0 ; & t \geq t_R \end{cases} \quad (8.3)$$

and initial value for the concentration

$$c(0) = 0 \quad (8.4)$$

First, we replace the differential equation with an equivalent integral equation

$$\frac{dc}{dt} = \frac{r(t)}{V} - \frac{CL}{V} \cdot c \quad \Leftrightarrow \quad c = c(0) + \int_0^t \left[ \frac{r(\tau)}{V} - \frac{CL}{V} \cdot c(\tau) \right] d\tau \quad (8.5)$$

As shown in Fig. 8.24, the model building can be described in this case in five steps (the number of steps varies with the complexity of the model equations).

- Step 1: At the top, the model is represented with only one *integrator* block, derived from Eq. (8.5). This step, as well as the following, uses the Simulink Block library (drag and drop). The integrator has to be configured, meaning its properties, such as the initial condition,  $c(0)$ , and others, have to be set. This is illustrated in Fig. 8.25 (upper panel).
- Step 2: The expression  $r(t)/V - CL/V \cdot c(t)$  is represented by one *sum* block with two inputs.
- Step 3: The expressions  $r(t)/V$  and  $-CL/V \cdot c(t)$  are represented by two *gain* blocks. The input of the *gain* block labeled with  $-CL/V$  is equal to the output of the integrator,  $c(t)$ . The configuration should include the  $V$  and  $CL$  parameters.
- Step 4: The infusion,  $r(t)$ , is represented by a *Signal Builder*, and the gain block labeled with  $R$ . The signal builder can generate a pulse with amplitude equal to 1, and works as a switcher to start and stop the infusion (*Infusion Switcher*). Besides the configuration of the infusion switcher (Fig. 8.25, lower panel), it should include the  $R$  parameter in the corresponding gain block.
- Step 5: The final model structure is achieved. As final blocks, two *scopes* (*Concentration* and *Infusion*) have been added. They are used to display the simulation results, i.e., the concentration and infusion time courses,  $c(t)$  and  $r(t)$ .

The model we have created will be named **comp1\_inf** and the name is linked to the file name where the model is saved, **comp1\_inf.mdl**. Additionally, we

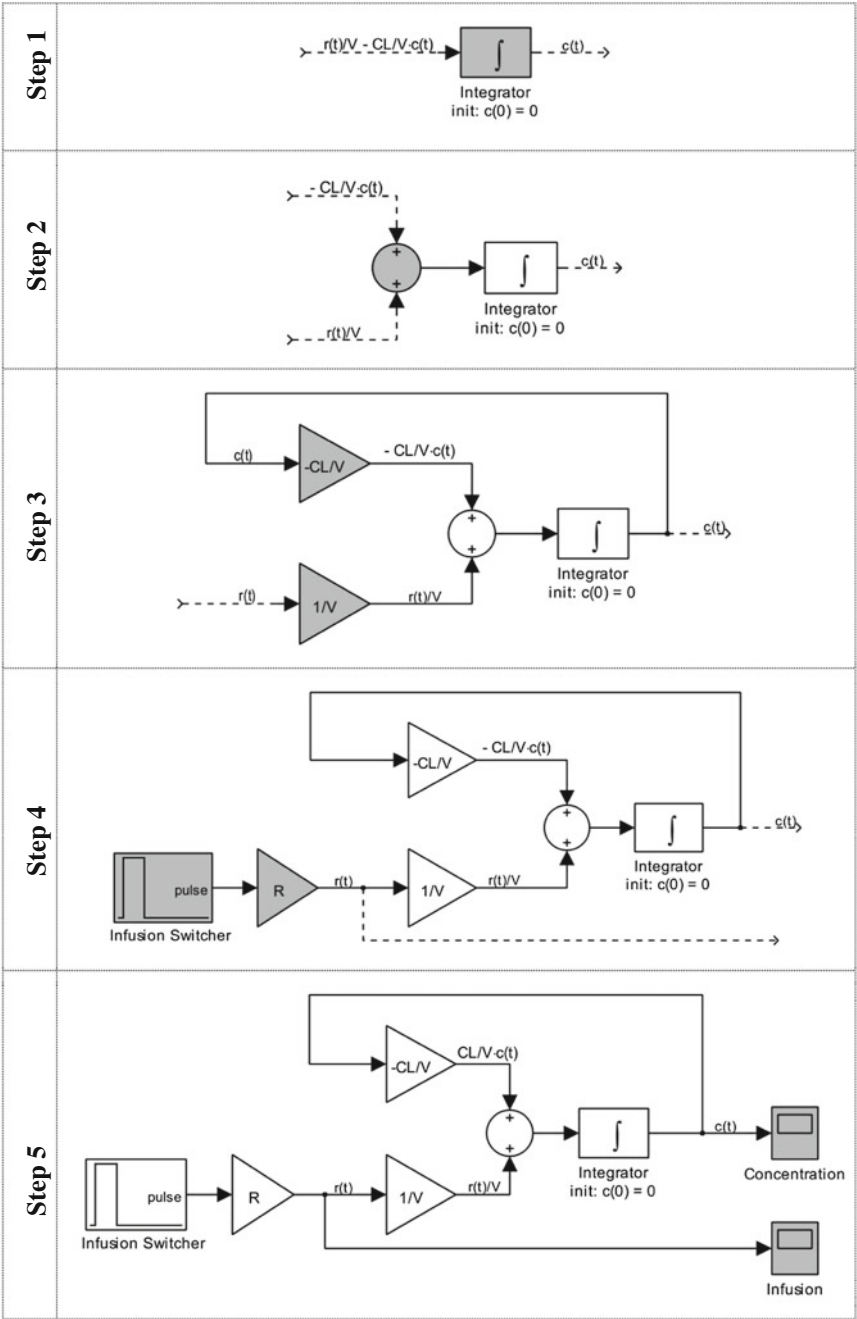
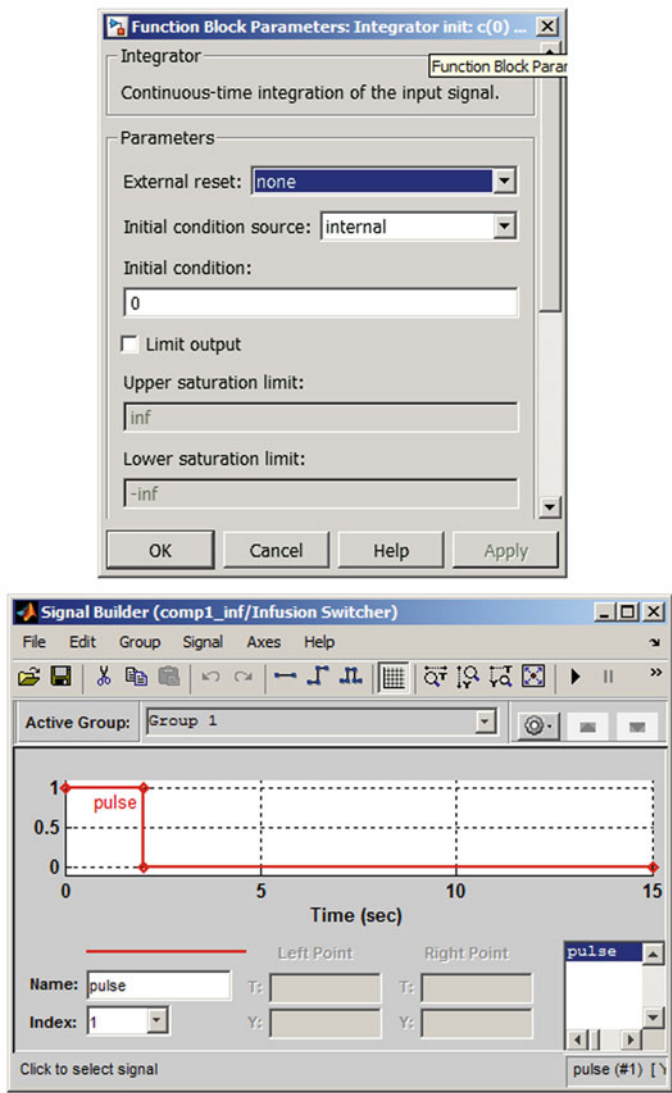


Fig. 8.24 Building a one-compartment PK model with Simulink



**Fig. 8.25** Graphical user interface (GUI) is used to set properties of blocks used in the Simulink model. In the *upper panel*, there is a GUI used to configure the *Integrator*. As configuration parameter, *Initial Condition*: is to be set to 0, the other configuration parameters can be left as shown (default values). In the *lower panel*—GUI for configuration of the *Signal Builder* that generates a square pulse to simulate the infusion. Similar dialog boxes are also used as GUI to set properties of other blocks in the model. More information about possible configuration parameters and their values can be obtained while using *Help* button, available in the dialog box for this block as well as for other blocks used in a Simulink model

will create a data file, **comp1\_inf.mat**, with all values of the model parameters,  $V$ ,  $CL$ ,  $R$ . Before launching the simulation, the data file has to be loaded into the MATLAB Workspace, see Fig. 8.26. There is good reason to keep a Simulink model in **<model-name>.mdl** and the data in **<model-name>.mat**. In the model file we can keep the model structure described with variables and model parameter names. The values for parameters and other data we need for simulations can be kept in the data file. In this way, we have flexibility in launching simulations for various data. The data can be easily changed in Workspace and a new run can be launched without needing to change the configuration parameters in model blocks.

The Simulink Library contains many blocks. Besides the integrator, the *Derivative* block could also be applied to represent a differential equation. However, this is not recommended as the derivative block is sensitive, and its accuracy strongly depends on the size of the time intervals taken into account during the simulation. Particularly, if the input changes rapidly (e.g., infusion start or stop), then the computational accuracy of *Derivative* block might be too low, producing unexpected wrong output.

The simulation results appear in two scope windows shown in Fig. 8.27.

### 8.2.2 Two-Compartment Model (Simulink)

Let us now consider a two-compartment PK model with infusion input, similar to the one in Chap. 3, but with the model equations written in terms of amounts, i.e., instead of

$$\begin{bmatrix} \frac{dc_1}{dt} \\ \frac{dc_2}{dt} \end{bmatrix} = \begin{bmatrix} \frac{r(t)}{V_1} - (k + k_{12}) \cdot c_1 + k_{21} \cdot \frac{V_2}{V_1} \cdot c_2 \\ k_{12} \cdot \frac{V_1}{V_2} \cdot c_1 - k_{21} \cdot c_2 \end{bmatrix}; \quad t \in [0; 15] \quad (8.6)$$

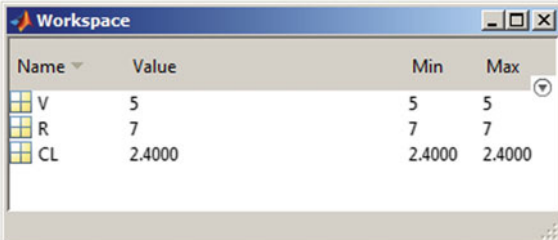
we will use a rearranged ODE system of the following form:

$$\begin{bmatrix} \frac{da_1}{dt} \\ \frac{da_2}{dt} \end{bmatrix} = \begin{bmatrix} r(t) - (k + k_{12}) \cdot a_1 + k_{21} \cdot a_2 \\ k_{12} \cdot a_1 - k_{21} \cdot a_2 \end{bmatrix}; \quad t \in [0; 15] \quad (8.7)$$

where  $a_1 = c_1 \cdot V_1$ ,  $a_2 = c_2 \cdot V_2$ , and the initial values for amounts are

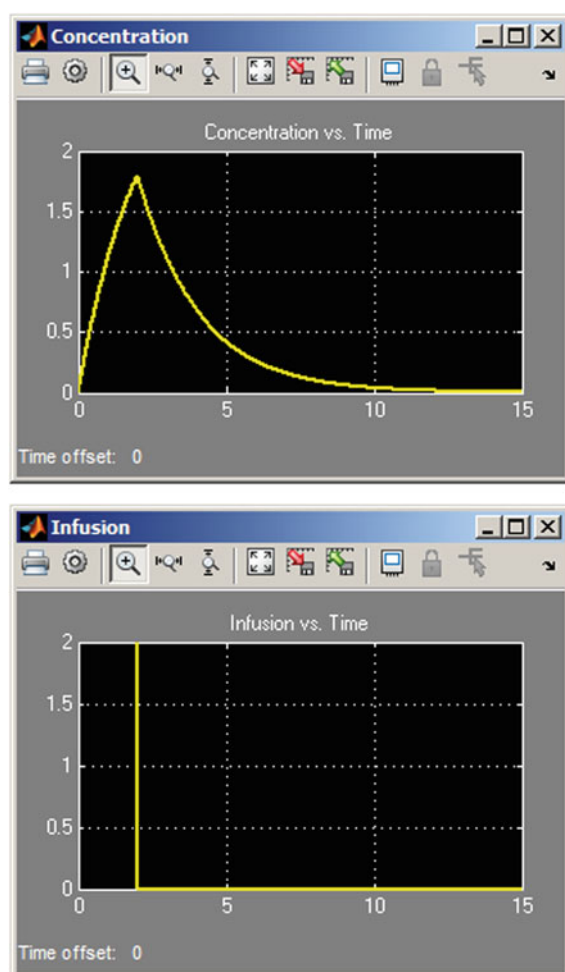
$$a_1(0) = a_2(0) = 0 \quad (8.8)$$

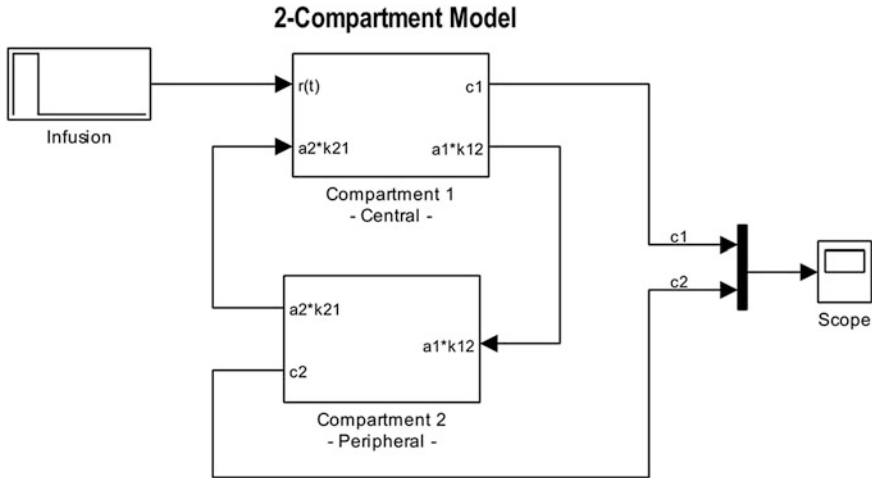
**Fig. 8.26** Workspace with the PK parameters and values assigned to them. The values are available to Simulink as it communicates with Workspace during simulation run time



Name	Value	Min	Max
V	5	5	5
R	7	7	7
CL	2.4000	2.4000	2.4000

**Fig. 8.27** Simulation result with one-compartment PK model. *Upper panel:* time course of concentrations; *lower panel:* infusion rate versus time





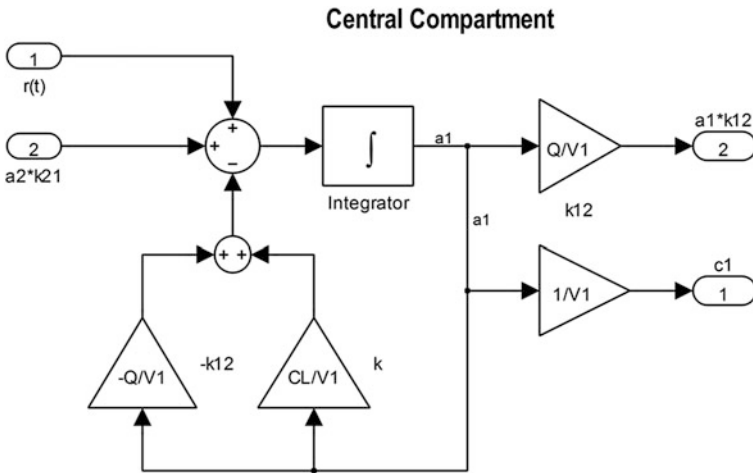
**Fig. 8.28** Schema to the final two-compartment PK model. The output block (*Scope*) shows the concentration time courses in the central and peripheral compartments. Both compartments can be built separately as subsystems, and connected properly at the very end. The infusion, as a control module, can also be considered as additional subsystem

The infusion rate and rate constants are defined as follows:

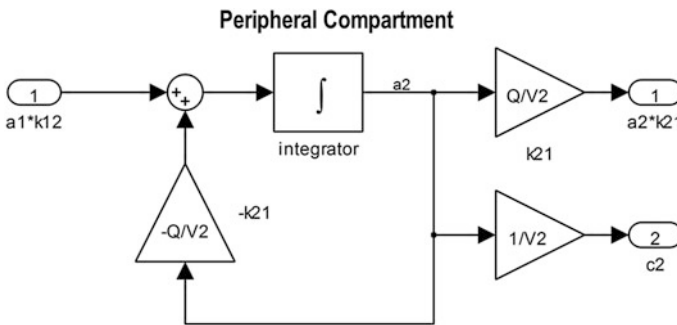
$$\left. \begin{aligned} r(t) &= \begin{cases} R; & t \in [0; t_R) \\ 0; & t \geq t_R \end{cases} \\ k &= CL/V1; \quad k_{12} = Q/V1; \quad k_{21} = Q/V2 \end{aligned} \right\} \quad (8.9)$$

As already shown in the previous sections with regard to the one-compartment PK model, each Simulink model can be understood as a system of several blocks with appropriate functionality. One of the most powerful features of Simulink is modularity, i.e., the ability to split a complex system of several blocks into properly connected subsystems (modules). If possible, we would build at the beginning a number of independent subsystems and establish connections between them at the very end. We will apply this approach to build a two-compartment PK model, described by Eqs. (8.7)–(8.9). Each of the two-compartment can be considered as a subsystem (Fig. 8.28). Based on the mass balance principle it is evident how these two subsystems should be connected. Additionally, the block that simulates infusion is considered as a subsystem.

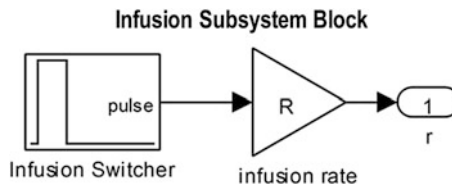
The structures hidden in the subsystem blocks (*Compartment 1—Central*, *Compartment 2—Peripheral* and *Infusion*) can be seen in other Figs. 8.29, 8.30, and 8.31. The created model, **comp2\_inf.mdl**, and corresponding data file (value of  $V1$ ,  $V2$ ,  $R$ ,  $CL$ ,  $Q$ ), **comp2\_inf.mat**, can be now used to simulate the concentration profiles in both compartments. The simulation results are shown in Fig. 8.32.



**Fig. 8.29** The central compartment subsystem of the two-compartment model. Besides the typical functional blocks (integrator, gain, sum), it contains 2 input and 2 output blocks as interfaces to the other subsystems (peripheral and infusion) and to the *scope* blocks



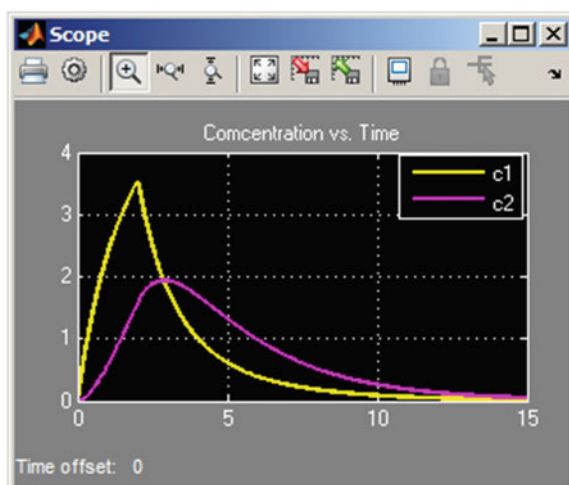
**Fig. 8.30** The peripheral compartment subsystem of the two-compartment PK model. It contains 2 input and 2 output blocks as interfaces to the central subsystems and *scope* blocks



**Fig. 8.31** The infusion subsystem schema to the two-compartment PK model. The infusion,  $r(t)$ , is represented by a signal builder (*Infusion Switcher*), and the gain block labeled with  $R$  (*infusion rate*). The signal builder can generate a pulse with amplitude equal to 1, and works as switcher to start and stop the infusion. The subsystem has one output block, labeled with  $I$ , to interface the subsystem with the central compartment block



**Fig. 8.32** Two-Compartment PK model with constant infusion input and first-order elimination: c1—drug concentrations in central compartment, c2—drug concentrations in peripheral compartment



### 8.2.3 Example: Ibandronate

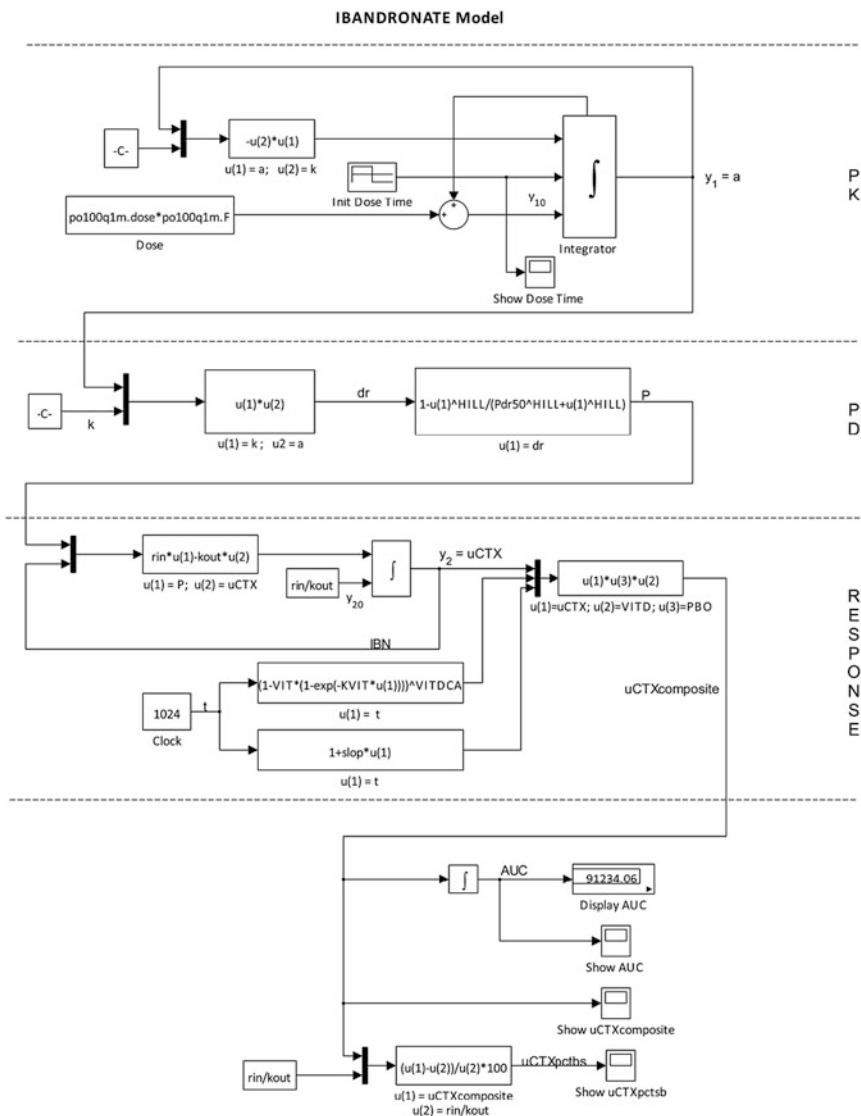
The pharmacologic model for ibandronate was already introduced in Sect. 4.3.4 [3]. A MATLAB ODE implementation of the model was used to investigate the impact of different dosage regimens on the biomarker *uCTX*. Here, we will show how this can be achieved with Simulink. We will start with a Simulink model file, **Iban\_Simul.mdl** (available on MATLAB Central), constructed according to Fig. 8.33. In order to launch a simulation with this model, a data file (**Iban\_Simul.mat**) is needed and must be loaded into the MATLAB *Workspace*. The model implements one dosage regimen (100 mg per *os* monthly) and Figs. 8.34, 8.35 show how this was done.

The **Iban\_Simul.mdl** model generates the results shown in Fig. 8.36.

Figure 8.33 shows the implementation of the ibandronate model for one regimen. In order to simulate the *uCTX* response for other ibandronate regimens, a proper reconfiguration of the blocks responsible for drug administration, *Dose* and *Init Dose Time*, is needed. Simulink simulates all these regimens concurrently. The created model (Fig. 8.33) can be stored in a subsystem block and subsequently used as a template for the multi-process creation of all required regimens. In the next section we will describe how to implement a multi-process of this kind.

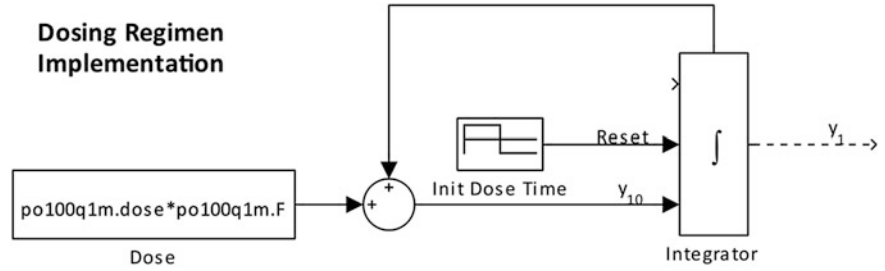
### 8.2.4 Simulink Model as a Multi-process

Let us assume we want to simulate the biomarker response to ibandronate for three different regimens: po25oad, po100q1m, and po150q1m, simultaneously. We can create a subsystem for each regimen specified by the dose and dose time. Each

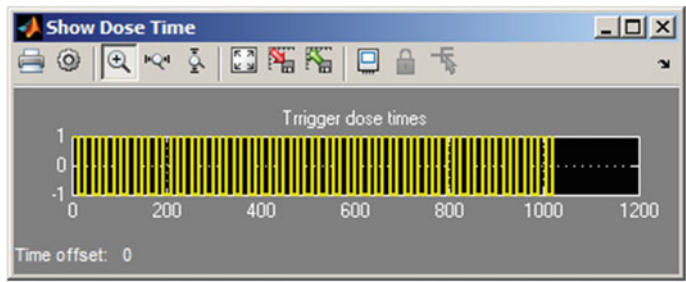


**Fig. 8.33** Simulink model generating the *uCTX* response to ibandronate. The *Dose* and *Init Dose Time* blocks have a special functionality. With this combination of two blocks we can define the required dosage regimen. The initial condition comes from an external source and is equal to the sum of the current value and a new dose amount (constant block, *Dose*). It must be reset every time a new dose is given. For this reason a signal is generated by the signal builder, *Init Dose Time*, and detected by the integrator with output  $y_1$

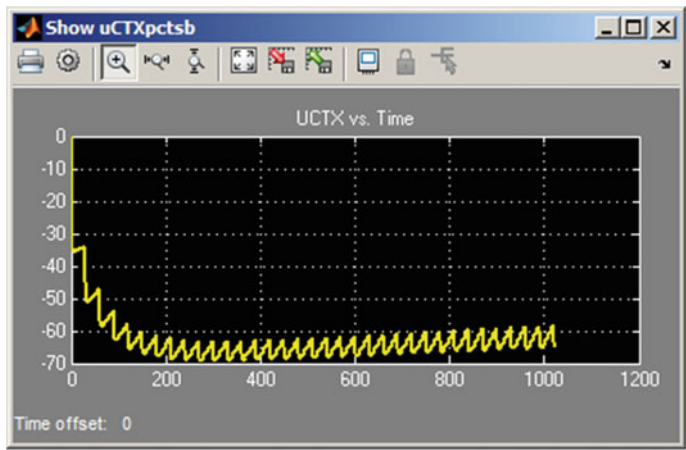
subsystem contains almost the same model structure (see Fig. 8.33), except for configuration of the *Dose* and *Init Dose Time* blocks (Fig. 8.34). Thus, to solve our problem we need only three copies of the created model (Fig. 8.33), and to save



**Fig. 8.34** The dosing regimen implementation that utilizes two blocks, a constant block (*Dose*), and a repeating sequence stair block (*Init Dose Time*)

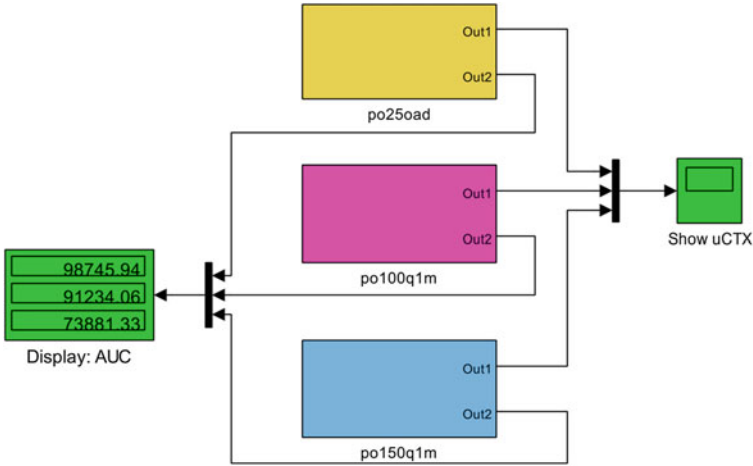


**Fig. 8.35** To trigger the dose time, the repeating sequence stair block can be used

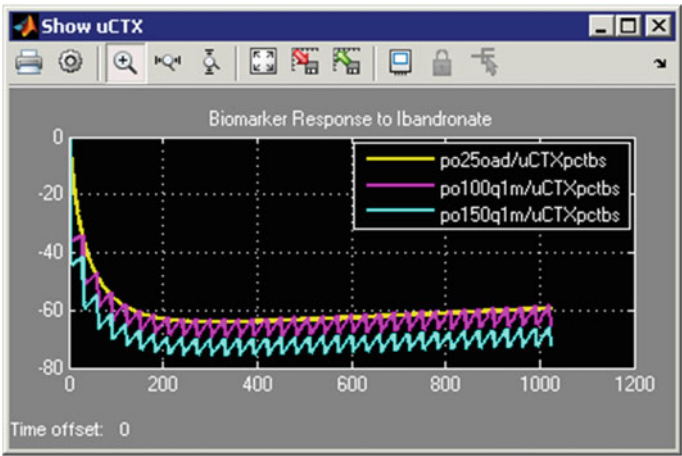


**Fig. 8.36** *uCTX* time course (% change from baseline) following *po100q1m* (time in days)

them as subsystems. The next step is to configure the drug administration mechanism (Fig. 8.34), and lastly create the data set and save it in a data file. The final version of the model, **Iban\_Simul\_Multi.mdl**, is presented in Fig. 8.37.



**Fig. 8.37** Simulink implementation of the ibandronate model as a multi-process. The purpose of this model is a concurrent simulation of *uCTX* responses to three different ibandronate dosage regimens: po25oad, po100q1m, and po150q1m



**Fig. 8.38** The simulation output generated by the **Iban\_Simul\_Multi.mdl** model for three different dosage regimens. The colors on the graph correspond to the colors in the model diagram (Fig. 8.37)

Assuming that the model parameters are saved in **Iban\_Simul\_Multi.mat** we can start the simulation. The simulation output is shown in Fig. 8.38. It corresponds to the results obtained in Sect. 4.3.4 (MATLAB program **ibandronate.m**).

### ***8.2.5 Simulink Models: Summary and Impact***

The Simulink tool provides a user-friendly GUI for building models as block diagrams using drag-and-drop mouse operations. Instead of writing source code, we simply configure the necessary blocks and wire them together to construct the block diagram. As we could see in the examples, the simulations are interactive, i.e., we can change parameters on the fly and immediately see what happens.

Simulink models can be hierarchical which is supported by the subsystem feature. It is also possible to build user-defined subsystems. Finally, we can observe the simulation output while the simulation is running.

## **8.3 Exercises**

### **Exercise 8.1**

Create two versions of a one-compartment PK model with IV bolus input: the first based on the assumption that the whole dose is the initial amount value, and the second considering the dose as the input variable. Show that the model output (concentration–time course) in both cases is the same.

## **References**

1. The MathWorks (2012) SimBiology user's guide (R2012b). The MathWorks, Natick
2. The MathWorks (2012) Simulink user's guide (R2012b). The MathWorks, Natick
3. Pillai G, Gieschke R, Goggin T, Jacqmin P, Schimmer RC, Steimer JL (2004) A semimechanistic and mechanistic population PK-PD model for biomarker response to Ibandronate, a new bisphosphonate for the treatment of osteoporosis. *Brit J Clin Pharmacol* 58:618–631

## Chapter 9

# Outlook

A “virtual man” could be the remedy for the research and development (R&D) problems the pharmaceutical industry is currently facing. For a number of organs (e.g., heart, liver, and brain), there are computational representations that have already proved their worth. The European Community financially supports activities in this area, such as the Drug Disease Model Resource (DDMoRe) as part of the Innovative Medicines Initiative (IMI) and the Human Brain Project.

Both preclinical research and clinical development can benefit from a modeling approach. Preclinical research can expect *in silico* target validation including the mode of action, portfolio assessment and ranking, and inverse drug engineering based on the assessment of multiple target engagements. Clinical development, as broadly advocated, could apply model-based drug development by making use of systems biology (SB) and quantitative systems pharmacology (QSP).

Some hurdles to an effective modeling approach in Pharma have been described: education, computational tools, and mindset. Systems modeling can benefit from graphics tools such as SimBiology and Simulink. Additional pre-specified building blocks for organs and/or physiologic systems, which could be adapted to disease and treatment would be helpful. Regardless of education and tools, mindset determines the final outcome. Development of innovative drugs via modeling remains an empty phrase if not lived at all levels of a pharmaceutical organization.

### 9.1 A Visionary Concept

The pharmaceutical industry remains an important contributor to healthcare. But it is becoming more difficult to create products that sustain the business model. Products should be innovative and effective, and should always outperform the standard of care. Ideas for such products and prototypes are generated mainly by small firms who look for larger partners able to undertake the expensive development phases, including toxicology and clinical development.

Beginning in 2007, PriceWaterhouseCoopers published a series of papers, “Pharma 2020,” in which they reviewed the pharmaceutical business and strategies available to companies [1]. The second paper in this series dealt with R&D.

After noting the decline in R&D productivity, the following factors were considered of vital importance [2]:

- a comprehensive understanding of how the human body works at the molecular level
- a much better grasp of pathophysiology
- greater use of new technologies to “virtualize” the research process and accelerate clinical development; and
- greater collaboration between industry, academia, regulators, governments and healthcare providers.

Thus, the idea is to gain sufficient knowledge about the human body so that a “virtual man” can be created allowing exploration of potential targets, their pharmacologic engagement and their effects on a given disease. Admittedly, this idea looks like science fiction today, but one should keep in mind that only recently a computational model for a whole (bacterial) cell was published [3]. This cell model represents all major cell functions such as metabolism, regeneration, and movement. A decade ago, a computational model of the whole heart was presented [4]. Furthermore, attempts are ongoing by the Virtual Liver Network [5], a major national initiative funded by the German Federal Ministry for Education and Research, to create a virtual liver, i.e., “a dynamic model that represents, rather than fully replicates, human liver physiology morphology and function, integrating quantitative data from all levels of organization”.

Recently, the European Commission [6] granted one billion euro to the Human Brain Project, a brain simulation project intended “to create the world’s largest experimental facility for developing the most detailed model of the brain, for studying how the human brain works and ultimately to develop personalized treatment of neurological and related diseases.

This research lays the scientific and technical foundations for medical progress that has the potential to dramatically improve quality of life for millions of Europeans”.

And if one considers the steady improvements in computational power, such as massive parallelization and cloud computing, the “virtual man” appears less utopic. Although we cannot (yet) rely on a “virtual man,” we can build biological models based on our current insight into cellular functions and cellular networks. And we can always use existing experimental data (our own or published) to build empirical models supporting our development activities. In the following, we would like to present recent suggestions for improving the current R&D process, e.g., in silico target validation, model-based drug development (MBDD), SB, and QSP.

## 9.2 R&D Improvements Based on Mathematical Modeling

Drug development has two defined phases: first, the invention of a molecule that has the potential to treat a disease better than the standard of care, and second, the realization of this potential through preclinical and clinical investigations. As a

matter of fact, the second phase can afford a number of surprises responsible for not every drug development program being a success. Indeed, only 10 % of drugs entering Phase I will be commercially viable. Why so many drugs entering Phase I fail is attributable to the fact that they showed convincing effects on targets that were only loosely validated, or even not validated at all, for the disease for which they were developed [7].

Mathematical modeling can support both development phases. The first phase needs models that describe the pathophysiology of the disease and the molecular action of drugs acting thereupon. These are multiscale biological models that integrate the pharmacology of the compound to be developed and preferably the pharmacology of the standard of care.

The second development phase is characterized by experimental trials in animals and humans delivering a plethora of data. During this phase, often an empirical modeling approach is chosen by creating models from the observed data. There is a good reason for creating empirical models from observed data: Models will fit the data quite well and can be used to extend investigations to the next phase. An empirical model in Phase I is not supposed to predict Phase III results. The disadvantage of the empirical approach is that it normally does not engage the biology of the disease and pharmacology of the compound.

### ***9.2.1 In Silico Target Validation***

Ho suggested biosystems modeling for in silico target validation in 2000 [8]. He introduced the two major approaches to biosystems modeling: the bottom-up and the top-down approaches. Bottom-up approaches usually start at the level of the genes and their interactions and proceed to intracellular networks. An inherent issue with this approach is computability. The top-down approach starts from a set of components that are considered to be necessary to describe the biosystem in question. It then iteratively refines the components until a desired level of granularity is reached. A compromise between these approaches is called “middle-out” [9], which suggests that one should start at a data-rich level and reach up and down to other levels. This approach was selected for modeling the heart [4].

Ho further describes barriers to the adoption of in silico target validation. These are technical (mainly regarding computer power) and mindset-related, both at the scientist (misunderstanding of modeling) and the organization level (modeling seen as a disruptive innovation).

Geerts provided a concrete example of target validation modeling in nervous system disorders [10], describing a computational platform implementing physiologic and pharmacologic processes on biophysically realistic neuronal network structures. Drug-receptor interactions induce receptor activations which are coupled to channel conductivities causing membrane currents and action potentials which are propagated through the network. In this example, the network representing a region of the cortex as readout for working memory is of interest. Not all



parameters of this model are predetermined. Whereas receptor characteristics (distribution, ligand potencies) are taken from animal research, the important receptor-activation membrane-conductivity coupling parameters are obtained from calibrating the model against clinical data. As clinical data comprise dozens of drug-dose combinations with widely varying receptor pharmacology, engaging the platform with these inputs yields a well-calibrated model (as assessed by its correlation coefficient) and enables computational biomarkers for clinical scales to be created.

### ***9.2.2 Model-based Drug Development***

Recognizing pharmaceutical product development problems, the FDA published a report in 2004 entitled “Innovation or Stagnation: Challenge and Opportunity on the Critical Path to New Medical Products” [11]. With reference to Sheiner [12], it claimed that “the concept of model-based drug development, in which pharmacostatistical models of drug efficacy and safety are developed from preclinical and available clinical data, offers an important approach to improving drug development knowledge management and development decision making”. This triggered a series of publications describing the essence of model-based drug development (MBDD) [13–15], its first successes [16], but also the environment in which it can thrive and prosper [17, 18]. Regarding the latter, a radical integration of modeling and simulation into scientific and management processes is advocated. Problems with current empirical R&D are described as being largely caused by the many isolated functional areas. True MBDD could overcome this issue.

Another important initiative in advancing drug development was started by the European Union in 2008, the IMI [19]. One of its projects is Drug Disease Model Resources (DDMoRe) which is divided into different activities. Overall it aims at providing a disease model library coded in a modeling markup language and creating a framework for interoperability of dedicated software products, in addition providing tools for complex problems, enhanced by training programs [20].

### ***9.2.3 Systems Biology***

SB has its roots in general systems theory which was worked out and applied to biology by Ludwig von Bertalanffy (1901–1972) [21]. A system is defined as a complex of interacting parts prototypically described by a set of differential equations. General systems theory is so widely applicable that meanwhile many disciplines use “systems” as an adjunct to their field, such as “systems pharmacology.” From SB new disciplines evolved such as theoretical and mathematical biology, providing the foundation for computational (or “in silico”) biology. SB typically involves iterative cycles where experiments provide data that are

integrated into models which in turn suggest new experiments until the modeling goal is reached [22]. Remarkably, SB approaches have also found application in psychiatry [23], an area at first glance supposed to be refractory to modeling, but at second glance in serious need of it.

### ***9.2.4 Quantitative Systems Pharmacology***

The integration of MBDD (or quantitative pharmacology) with SB seems to be obvious. Indeed, QSP, as documented in an NIH White Paper [24], was suggested as a further attempt to find a way out of the Pharma productivity crisis. The authors recommended research in eight different areas (R1–R8), and those that explicitly involve modeling were:

R1: Quantitative analysis of drug targets, the networks in which they are embedded and the effects of drugs on targets and networks

R2: Investigating the origins of variability in drug response at the single-cell, organ and patient level based on proteomic, genomic, and environmental differences

R7: Developing multiscale computational models of pharmacologic mechanism that span the divide between cell-level biochemical models and organism-level PK/PD models.

Overall, a step toward the “virtual man.”

## **9.3 Prerequisites and Obstacles**

The implementation of mathematical modeling in drug development, as MBDD or QSP, cannot be successful unless some conditions are met. These refer to the education of the people executing the work, to the computational tools accessible to them, and to the mindset of the organization they are working in.

### ***9.3.1 Education***

To advance R&D by mathematical modeling, several capabilities are required such as knowledge in physiology, disease biology, pharmacometrics, mathematics (including numerics and statistics), and applied programming.

Regarding pharmacometrics, a curriculum has been suggested [25] to include not only technical model-building skills, but also the review and presentation of the literature data. In a 2011 publication, the Society for Industrial and Applied Mathematics (SIAM) reviewed the mathematical demands and came to the following assessment: “The mathematical and computational techniques behind these

models include network science, deterministic and stochastic differential equations, Bayesian networks and hidden Markov models, optimization, statistics, control, simulation, and uncertainty quantification” [26].

Last but not least, soft skills are required: modeling is nothing if not a team exercise.

### ***9.3.2 Computational Tools***

The use of mathematical models, their implementation in software, ongoing model evaluation and repeated model revision require careful documentation of all activities. This becomes especially burdensome the more different software is needed to do the job.

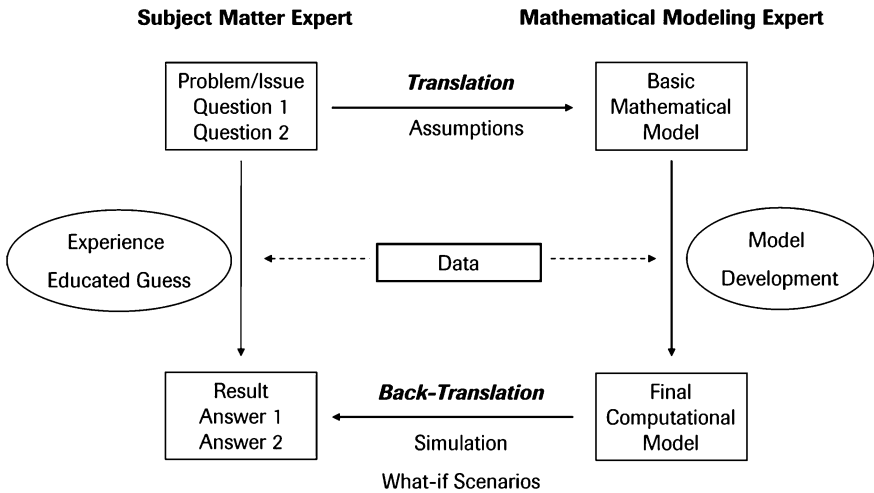
For all our calculations in this book, we have used MATLAB software exclusively. Its versatility leads us to recommend it as the software of choice for pharmaceutical M&S, disease modeling and QSP. Of course, there are areas where MATLAB functionality falls below the “standard of care,” in particular for complex, ODE-based population PK-PD models which are evaluated with the MATLAB `nlmefit` or `nlmefitsa` functions. The ODE and DDE suite, on the other hand, can be considered state-of-the art. Other extremely valuable functionalities are provided with SimBiology and Simulink. These graphics-based modeling tools help to visualize and modularize the modeling approach. These are also much more suited team communication tools than a mathematical description of the model or even software code. Other MATLAB features that deserve highlighting are the graphics, the publishing of user-defined programs, the online help, and the documentation.

### ***9.3.3 Mindset***

Mindset refers to the overarching readiness (or reluctance) on the part of decision-makers regarding how, where and when to use modeling. If the disconnect is too large between the mathematical modeling expert (MME) and subject matter expert (SME), it will be hard to achieve any meaningful result, even after huge investments in time and persuasion. How can this be avoided?

It is a general observation that SMEs with only a superficial mathematical background have barely any chance to understand what is going on within the “mathematical animal” that is presented to them. But do they understand the real animals better? A way out would be to learn the abstract science up to a level where confidence emerges.

If the modeling approach is only marginally understood, naturally doubt and distrust emerge. To overcome this hurdle, collaboration between the SME and MME is mandatory. The SME should check all modeling steps that are related to



**Fig. 9.1** Modeling process from a collaborative viewpoint: translation and back-translation as the key areas of collaboration between subject matter expert and mathematical modeling expert

subject matter items, i.e., assumptions for the conversion of the real problem into a mathematical task, especially the conceptual model, and the back-translation of computational into subject matter results, where simulation of what-if scenarios could clarify understanding (Fig. 9.1). The MME needs to consult the SME to be up-to-date with subject matter developments (publications, etc.).

Another issue regarding mindset is about expectations. Maybe the expectations of the SME in terms of timelines and scope were too high? Or the MME promised too much? Modeling needs time. And future modeling may take even more time. Were the deliverables clearly defined? Often they are provided in many steps.

A feature of current modeling mindset is to sharply question predictions that are counterintuitive but to welcome with open arms those that confirm prior beliefs. This disregards the fact that complex models often yield counterintuitive results [4].

Last, but not least, we would like to mention validation, an ingrained mindset. Mathematical modeling is sometimes confronted with the argument that it is producing results that cannot be validated, i.e., compared to real data outcomes. In pharmacologic modeling, this certainly holds true for all abstract compartment concentrations. However, one may argue that validation is secondary to the modeling process itself, which primarily allows us to make better use of our mental capacities in tackling otherwise insurmountable problems.

*Mathematics is biology's next microscope, only better;  
biology is mathematics' next physics, only better [27].*

## References

1. PricewaterhouseCoopers (2011) Introducing the 2020 pharrma series. <http://www.pwc.com/gx/en/pharma-life-sciences/pdf/introducing-the-pharma-2020-series-july-2011.pdf> (Accessed 15 Mar 2013)
2. PricewaterhouseCoopers (2008) Pharma 2020: Virtual R&D – Which path will you take? [http://www.pwc.ch/user\\_content/editor/files/publ\\_life/pwc\\_pharma2020\\_08\\_e.pdf](http://www.pwc.ch/user_content/editor/files/publ_life/pwc_pharma2020_08_e.pdf) (Accessed 15 Mar 2013)
3. Karr JR, Sanghvi JC, Macklin DN, Gutschow MV, Jacobs JM, Bolival B Jr, Assad-Garcia N, Glass JL, Covert MW (2012) A whole-cell computational model predicts phenotype from genotype. *Cell* 150:389–401
4. Noble D (2002) Modeling the heart - from genes to cells to the whole organ. *Science* 295:1678–1682
5. Virtual Liver Network <http://www.virtual-liver.de> (Accessed 15 Mar 2013)
6. European Comission (2013) [http://europa.eu/rapid/press-release\\_IP-13-54\\_en.htm](http://europa.eu/rapid/press-release_IP-13-54_en.htm)
7. Paul SM, Mytelka DS, Dunwiddie CT, Persinger CC, Munos BH, Lindborg SR, Schacht AL (2010) How to improve R&D productivity: the pharmaceutical industry's grand challenge. *Nat Rev Drug Discov* 9:203–214
8. Ho RL (2000) Biosystems modeling for in silico target validation: challenges to implementation. *Emerg Ther Targets* 4:699–714
9. Noble D (2002) The rise of computational biology. *Nat Rev Mol Cell Biol* 3:460–463
10. Geerts Hugo (2011) Mechanistic disease modeling as a useful tool for improving CNS drug research and development. *Drug Dev Res* 72:66–73
11. US Food and Drug Administration (2004) Innovation or stagnation: Challenge and opportunity on the critical path to new medical products. <http://www.fda.gov/ScienceResearch/SpecialTopics/CriticalPathInitiative/CriticalPathOpportunitiesReports/ucm077262.htm> (Accessed 15 Mar 2013)
12. Sheiner LB (1997) Learning versus confirming in clinical drug development. *Clin Pharmacol Ther* 61:275–291
13. Zhang L, Allerheiligen SR, Lalonde RL, Stanski DR, Pfister M (2010) Fostering culture and optimizing organizational structure for implementing model-based drug development. *J Clin Pharmacol* 50:146S–150S
14. Wetherington JD, Pfister M, Banfield CB, Stone JA, Krishna R, Allerheiligen S, Grasela DM (2010) Model-based drug development: Strengths, weaknesses, opportunities, and threats for broad application of pharmacometrics in drug development. *J Clin Pharmacol* 50:31S–46S
15. Suryawanshi S, Zhang L, Pfister M, Meibohm B (2010) The current role of model-based drug development. *Expert Opin Drug Discov* 5:311–321
16. Stone JA, Banfield C, Pfister M, Tannenbaum S, Allerheiligen S, Wetherington JD (2010) Model-based drug development survey finds pharmacometrics impacting decision making in the pharmaceutical industry. *J Clin Pharmacol* 50:20S–30S
17. Grasela TH, Slusser R (2010) Improving productivity with model-based drug development: an enterprise perspective. *Clin Pharmacol Ther* 88:263–268
18. Grasela TH, Fiedler-Kelly J, Walawander CA, Owen JS, Cirincione BB, Reitz KE, Ludwig EA, Passarell JA, Dement CW (2005) Challenges in the transition to model-based development. *AAPS J* 7:E488–E495
19. Goldman M (2010) The Innovative Medicines Initiative: A European response to the innovation challenge. *Clin Pharmacol Ther* 88:
20. DDMoRe Consortium. <http://www.ddmore.eu/> (Accessed 15 Mar 2013)
21. von Bertalanffy L (1968) General system theory. George Braziller, New York
22. Young DL, Michelson S (2012) Systems biology in drug discovery and development. Wiley, New York
23. Tretter F, Gebicke-Haerter PJ, Mendoza ER, Winterer G (2010) Systems biology in psychiatric research. Wiley, Weinheim

24. Sorger PK (2011) Quantitative and systems pharmacology in the post-genomic era: New approaches to discovering drugs and understanding therapeutic mechanisms. An NIH White Paper by the QSP workshop group. (<http://isp.hms.harvard.edu/wordpress/wp-content/uploads/2011/10/NIH-Systems-Pharma-Whitepaper-Sorger-et-al-2011.pdf>) (Accessed 15 Mar 2013)
25. Holford N, Karlsson MO (2007) Time for quantitative clinical pharmacology: A proposal for a pharmacometrics curriculum. *Clin Pharmacol Ther* 82:103–105
26. Society for Industrial and Applied Mathematics (2012) Mathematics in Industry. <http://www.siam.org/reports/mii/2012/report.pdf> (Accessed 15 Mar 2013)
27. Cohen JE (2004) Mathematics is biology's next microscope, only better; biology is mathematics next physics, only better. *PLoS Biol* 2:e439

# Appendix A: Hints to MATLAB Programs

In this Appendix, we provide additional explanations regarding procedures and programming techniques used in this book. They could also be utilized for pharmacologic modeling and simulation, in general. We do not aim for an *Introduction to MATLAB*, or a *MATLAB Tutorial*. Wide-ranging literature, internet sources, and various MATLAB courses are offered to get started with MATLAB fundamentals as well as to deal with advanced programming techniques. At the end of this Appendix, the reader will find a list of sources that can be very helpful in both learning and improving MATLAB programming and application skills in various areas directly or indirectly associated with pharmacologic modeling [1] (Electronic Edition), [2–13].

We also provide no introduction to MATLAB essentials, given the excellent MATLAB ‘help’ system. It is fully integrated in the whole programming system and can be also used interactively. The user can get help directly while writing the code, and by calling the required help documentation, any MATLAB code contained in the help can be directly evaluated. We will pay more attention to the help system in the next chapters.

Furthermore, MATLAB Central on the internet provides an extensive source of useful programs and tips.

## A.1 Getting Help

There are several ways to access the MATLAB help system. Either directly by typing commands in the Command Window, or indirectly by browsing to MATLAB internet pages maintained by MathWorks.

### Help on the main menu

From the main menu select *Help > Documentation > MATLAB*

### Command help

In the Command Window type `>>help`. This command has numerous options. We will not describe them here, as the full description can be easily obtained when entering `>> helphelp`

**Help on selection**

In the Command Window or Command History Window, mark/highlight the word/term you would like to get help about. Use the right mouse button to get a context menu and then select *Help on Selection* or press [F1].

**Using the browser for functions**

In the Command Window, press button [*f*] which is always available close to the command prompt `>>` or press [Shift] + [F1].

**Command iskeyword**

In the Command Window, type `>> iskeyword`

**Command doc**

In the Command Window, type `>> doc < keyword >`. For a full description enter `>> help doc`.

**Help from MATLAB help Browser**

In the Command Window, type `>> helpwin`. It will generate a list of help topics. For the full description enter `>> help helpwin`.

**Look for the specified topic**

In the Command Window, type `>> lookfor < topic >`. It will look for the specified topic in all m-files in the current directory or on the MATLAB search path. For the full description enter `>> help lookfor`.

**MATLAB homepage**

Browse to the internet page <http://www.mathworks.com/support>.

**MATLAB Central homepage**

Browse to page <http://www.mathworks.ch/matlabcentral>.

## A.2 Creating Self-documenting Online Help

MATLAB has a specific style for the descriptive text called *help section* (or *line HI*) that follows a function header. The help section has the form of a specific constructed comment but with special meaning that enables the user to extend the MATLAB help system. If we take a look inside the code for MATLAB's built-in functions (e.g. typing for example `edit randn`) we will see that they all follow the same general format. There is no obligation to follow this format, but it makes the created scripts and functions more comfortable for the user, as program information is automatically integrated into the MATLAB help system. It also gives some specific advantages, in that it allows users to type e.g. `help derivatives`, and the help text (help section for function `derivatives`) will appear at the Command Window, as it does with built-in functions.

More details are shown in the following Listing A.1.



**Listing A.1** Example of a help section (line H1): It follows the function header and is properly created in the **derivatives.m** m-file

```
function dydt = derivatives(~, y, p)
%DERIVATIVES Compute the right-hand side of the ODE.
%   DYDT = DERIVATIVES(T,Y,P) calculates |DYDT|, the right-hand side
%   of the ODE model, at points defined by the vector of dependent
%   variable |Y|, time |T|, and with parameters |P|.
%
%   See also ode15s
%
% .....
% function statements
% .....
end
```

As we can see, the help section is formatted according to the following general syntax:

```
% <FUNCTION-NAME> <short-description>
%   <FUNCTION-HEADER> <functionality-description>
%   <input-and-output-description>
```

According to MATLAB conventions, the function name and its arguments in the help section should be typed in upper case characters. Say, the **help** command with the argument < function-name > is typed, for example:

```
>> help derivatives
```

This will produce help information, as shown in Listing A.2.

**Listing A.2** Output produced in the Command Window when using the **help** function

```
derivatives Compute the right-hand side of the ODE.
   DYDT = derivatives(T,Y,P) calculates |DYDT|, the right-hand side
   of the ODE model, at points defined by the vector of dependent
   variable |Y|, time |T|, and with parameters |P|.

   See also ode15s
```

The **ode15 s** text appears as a help link (underlined) to this ode solver. It happens because within any help section, **See also** < name > is interpreted as a command to create a corresponding help link to object < name >.

The help functionality works correctly if the corresponding m-file (in this case **derivatives.m**) is on the MATLAB search path or in the current directory, and **derivatives** is the *primary function*, i.e. the first function in the file. But it is also worth using help functionality for the additional functions within an m-file,

so-called *subfunctions*. If **derivatives** were a subfunction, say in **consecutive.m** m-file, we would type

```
>> help consecutive > derivatives
```

and could see the corresponding help text.

### A.3 Dosing Regimen Specification

MATLAB offers various functions that are useful for scheduling a dosing regimen. The following text describes two of them, **reshape** and **repmat**, used in the book for the specification of dosing history.

```
<B> = reshape(<A>, <m>, <n>)
```

This function changes the shape of matrix  $\langle A \rangle$  and stores the reshaped matrix in  $\langle B \rangle$ . Matrix  $\langle B \rangle$  has  $\langle m \rangle$  rows,  $\langle n \rangle$  columns and exactly the same number of elements like  $\langle A \rangle$ . The reshaping can be depicted as in Fig. A.1. The key operation of this procedure is the column-wise handling of matrices involved.

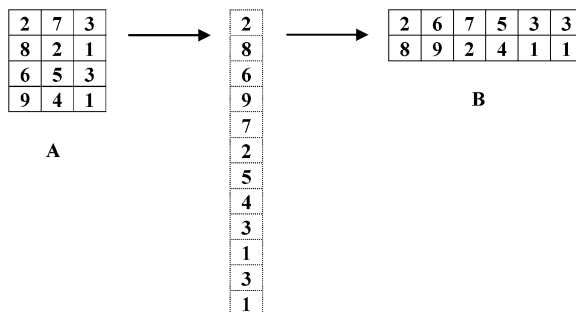
The **repmat** function is used according to the syntax

```
<B> = repmat(<A>, <m>, <n>)
```

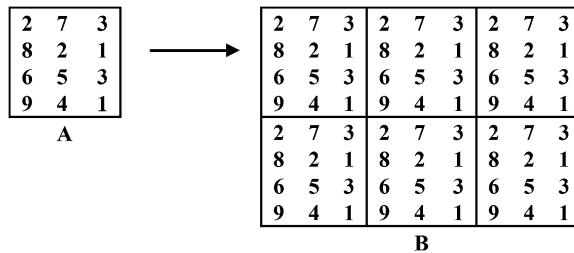
It replicates matrix  $\langle A \rangle$   $\langle m \rangle$  times vertically and  $\langle n \rangle$  times horizontally as shown in Fig. A.2.

The above two functions can be used for a dosing regimen specification. Say, the following dosing regimen is required: Treatment is administered over 252 days, and a dose is given on days: 0, 2,..., 22 within three 90-d periods.

The task is to generate vector **doseTimes** containing all dosing times within the treatment time. In the first step, we create a matrix storing the ‘dosing days’ (rows) within each period (columns).



**Fig. A.1** An example of how the **reshape(A, 2, 6)** function operates. It handles the input matrix **A** (4 rows and 3 columns) in a column-wise manner; we can imagine an intermediate vector (*middle*) created from **A** by taking consecutive elements down each column, and finally putting them into matrix **B** (2 rows and 6 columns), also in a column-wise manner



**Fig. A.2** A simple example of how the `repmat(A,2,3)` function operates. The matrix **A** is replicated  $2 \cdot 3 = 6$  times and configured, 2 times vertically, and 3 times horizontally, into matrix **B**

```
DD = repmat(0:2:22, length(0:90:252),1)
```

It produces a  $12 \times 3$  matrix (12 days within 3 possible periods):

```
DD = [ 0  2  4  6  8 10 12 14 16 18 20 22;  
      0  2  4  6  8 10 12 14 16 18 20 22;  
      0  2  4  6  8 10 12 14 16 18 20 22]
```

The next step generates the reference days (first day within each period) as many times as the number of dosing days within one period.

```
PP = repmat(0:90:252',1,length(0:2:22))
```

It produces a  $12 \times 3$  matrix (12 reference days within 3 possible periods):

```
PP = [ 0  0  0  0  0  0  0  0  0  0  0  0;  
      90 90 90 90 90 90 90 90 90 90 90 90;  
      180 180 180 180 180 180 180 180 180 180 180 180 ]
```

In the last step, the dose days have to be located within each period (**DD + PP**), and finally, the dose days from all periods (**DD + PP**) collected (reshaped) in one row, i.e.

```
doseTimes = ...
```

```
reshape((DD + PP)',1,length(0:2:22)*length(0:90:252))
```

which gives the required dosing times:

```
doseTimes = [ 0 2 4 6 8 10 12 14 16 18 20 22 90 ...  
92 94 96 98 100 102 104 106 108 110 112 180 182 ...  
184 186 188 190 192 194 196 198 200 202 ]
```

This method is also applied in the `concmdd.m` and `ibandronate.m` programs.

## A.4 Multiple Runs: Consecutive Versus Simultaneous

One of the powerful properties of MATLAB is vectorization. Users can conduct several computations for all subjects (patients) simultaneously instead of using a loop to process subjects consecutively.

In the following, we will compare both approaches assuming the model

$$\left. \begin{aligned} \frac{da}{dt} &= -k_a \cdot a \\ \frac{dc}{dt} &= k_a \cdot \frac{F}{V} \cdot a - \frac{CL}{V} \cdot c \end{aligned} \right\}; \quad t \geq 0 \quad (\text{A.1})$$

### Consecutive

Listing A.3 demonstrates consecutive processing. The program uses model (A.1) implemented as the **derivatives** function which specifies two differential equations. The other function, **modelparams**, randomly generates model parameters for simulation.

**Listing A.3** The **consecutive.m** program simulates subjects consecutively

```
function consecutive
%CONSECUTIVE Conduct trials consecutively.

tic
ns = 500;
p = modelparams(ns);
a0 = 10; c0 = 0; y0 = [a0; c0]; tspan = [0 15];
for id = 1:p.numSub
    pind.ka = p.ka(id);
    pind.V = p.V(id);
    pind.F = p.F(id);
    pind.CL = p.CL(id);
    [t,y] = ode15s(@derivatives, tspan, y0, [], pind);
    plot(t,y(:,2),'Color','b'); hold on
end
toc
xlabel('Time'); ylabel('Concentration'); title('Consecutive')
print('-r900', '-dtiff', 'consecutive')

end

function p = modelparams(ns)
%MODELPARAMS Specification of model parameters.
% P = MODELPARAMS(NS) creates a structure |P| storing the model
% parameters for |NS| subjects.
```

```

% Reset the random number generator to the default
rng default
p.numSub = ns;                % number of subjects
% specify model parameters for all subjects
p.ka = 1 + rand(ns,1);        % absorption rates
p.V = 15 + rand(ns,1)*2;      % volume of distribution
p.F = abs(1 - rand(ns,1)*0.5); % bioavailability
p.CL = 5 + rand(ns,1);        % clearance

end

function dydt = derivatives(~, y, p)
%DERIVATIVES Compute the right-hand side of the ODE.
% DYDT ...

% set dependent variables (for one subject)
a = y(1);    % drug amount
c = y(2);    % concentration
% PK Model
dadt = -p.ka*a;
dcdt = (p.ka/p.V/p.F*a - p.CL/p.V*c);
% Derivatives vector of ODE system
dydt = [ dadt
         dcdt ];
end

```

In order to execute the script enter in the Command Window:

»**consecutive;**

MATLAB will then execute a **for** loop and process one subject after the other. The ODE model with two equations will be solved **ns** times. Similarly, a time profile will be generated **ns** times, too.

### Simultaneous

Listing A.4 demonstrates simultaneous processing. This program utilizes also model (A.1). Functionality of **modelparams** is the same but the **derivatives** function is different as it specifies **2\*ns** equations, and is vectorized (element-wise operations **./** and **.\*** are used).

**Listing A.4** The `simultaneous.m` program simulates several subjects simultaneously

```
function simultaneous
%SIMULTANEOUS Conduct trials simultaneously.

tic
ns = 500;
p = modelparams(ns);
ns = p.numSub;
a0 = 10*ones(ns,1); c0 = zeros(ns,1); y0 = [a0; c0]; tspan = [0 15];
[t,y] = ode15s(@derivatives, tspan, y0, [], p);
plot(t,y(:,ns+1 : 2*ns), 'Color', 'r'); hold on
toc
xlabel('Time'); ylabel('Concentration'); title('Simultaneous')
print('-r900', '-dtiff', 'simultaneous')

end

function p = modelparams(ns)
% .....
% .....
% .....

end

function dydt = derivatives(~, y, p)
%DERIVATIVES Compute the right-hand side of the ODE.
%   DYDT ...

ns = p.numSub;           % number of subjects
% set dependent variables simultaneously for all subjects
% (vectorization)
a = y( 1 : ns);          % drug amount
c = y(ns+1 : 2*ns);       % concentration
% PK Model for all subjects
dadt = -p.ka.*a;
dcdt = (p.ka./p.V./p.F.*a - p.CL./p.V.*c);
% Derivatives vector of ODE system
dydt = [ dadt
         dcdt ];

end
```

In order to execute the script enter in the Command Window:

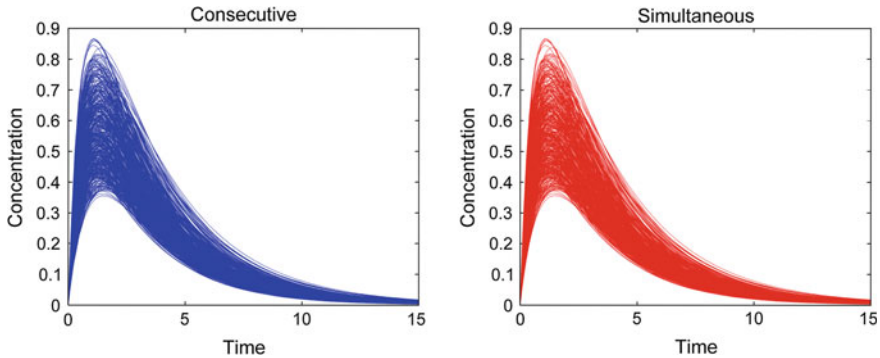
```
» simultaneous;
```

In this case, MATLAB will solve the ODE model only once (no loop is needed), despite the `2*ns` equations contained in the ODE model. The time profile will be generated at once for all `ns` subjects. The results are the same in both cases, as shown in Fig. A.3.

In order to compare the performance we use the commands `tic` and `toc` (see Listing A.3 and Listing A.4) which let us measure the performance times and generate the output:

```
Elapsed time is 7.967654 s.
```

```
Elapsed time is 1.792569 s.
```



**Fig. A.3** *Left panel:* time profiles computed consecutively (elapsed time is 7.967654 s); *right panel:* time profiles computed simultaneously (elapsed time is 1.792569 s). The simulation was carried out with 500 subjects in both cases

for the consecutive and simultaneous methods, respectively. The absolute value of the elapsed time varies on different PCs but the ratio,  $7.967654/1.792569$ , will be approximately the same and reflects the advantage of vectorization.

The vectorization is applied in some applications in this book, for example **esaCTS.m** and **conmodvector.m**.

## A.5 Using ‘for’ Loop Over Any Array

According to **for** loop functionality and description, the iteration index set can be an array of any data type. This property is very powerful because we can assign very complex structures to the loop index and loop statements in each iteration. MATLAB can execute the statements in a loop taking into account each column of the array.

Say we have the following code (based on the **consecutive.m** program in Listing A.3)

```
for id = 1:p.numSub
    pind.ka = p.ka(id);
    pind.V = p.V(id);
    pind.F = p.F(id);
    pind.CL = p.CL(id);
    [t,y] = ode15s(@derivatives, tspan, y0, [], pind);
    plot(t,y(:,2), 'Color', 'b'); hold on
end
```

Instead writing the above loop, we can first specify outside the loop a structure array containing all parameters for all subjects, for example

```

sparams = struct('ka', num2cell(p.ka'), ...
                'V', num2cell(p.V'), ...
                'F', num2cell(p.F'), ...
                'CL', num2cell(p.CL'));

```

and then a loop such as

```

for pind = sparams
    [t,y] = ode15s(@derivatives, tspan, y0, [], pind);
    plot(t,y(:,2), 'Color', 'b'); hold on
end

```

MATLAB loops through each element of the iteration index set, **sparams**, and assign the respective value to the index **pind**. We need neither to specify explicitly the first and the last value for the index, nor know the number of iterations. In this way, we can modify **consecutive.m** program to **consecutiveMod.m**, and compare the results of both. We leave this task to the reader.

This functionality is also demonstrated in the book, for example in the **DDEmar.m**, and **hoghux.m** programs.

## A.6 Calculation of AUC

In [Chap. 4](#), *Pharmacologic Modeling*, the quantity AUC (area under the curve) was introduced. AUC can be numerically calculated if drug concentrations over time are given (in analytical or numerical form). One generally applied calculation method is the trapezoidal rule. However, if the pharmacologic model is ODE-based, it is possible to derive a partial AUC for any drug concentration by just adding one differential equation.

AUC, is defined as

$$AUC \equiv y_{AUC} = \int_0^t c(\tau) d\tau; \quad t \rightarrow \infty \quad (\text{A.2})$$

where  $c$  is the drug concentration and for any time  $t$ , the partial AUC is given by

$$AUC[0, t] \equiv y_{AUC}(t) = \int_0^t c(\tau) d\tau \quad (\text{A.3})$$

The derivative of  $y_{AUC}$  with regard to time,  $t$ , is then equal to

$$\frac{dy_{AUC}}{dt} = c(t) \quad (\text{A.4})$$



The above equation is an ODE, and (A.3) is the solution of this ODE. Equation (A.4) can be now added to the ODE model, as shown for the following ODE system:

$$\left. \begin{array}{l} \text{ODE model:} \quad \frac{dy}{dt} = \mathbf{f}(t, \mathbf{y}) ; \quad \mathbf{y}(0) = \mathbf{y}_0 \\ \text{added equation:} \quad \frac{dy_{AUC}}{dt} = c(t) ; \quad y_{AUC}(0) = 0 \end{array} \right\} \quad (\text{A.5})$$

where  $c(t)$  is one of the components of  $\mathbf{y}$ . Thus, having solved the ODE system (A.5) until time  $t$ , we also have a value for the partial AUC,  $y_{AUC}(t)$ . If the value for the AUC, as defined in (A.2), is finite we can approximately calculate it by integrating over a long enough timespan.

Implementation of this method is shown in the example below, in Listing A.5, and the respective results can be seen in Fig. A.4.

**Listing A.5** Program **AUC.m**: how the AUC value can be computed by adding an ODE

```
function AUC
%AUC Compute area under the curve.

% specify model parameters
p.ka = 1; % absorption rates
p.V = 15; % volume of distribution
p.F = 1; % bioavailability
p.CL = 5; % clearance
% initial values
a0 = 10; c0 = 0; AUC0 = 0; y0 = [a0; c0; AUC0]; tspan = [0 15];
[t,y] = ode15s(@derivatives, tspan, y0, [], p);
set(axes, 'FontSize', 15); plot(t, y(:,2), 'Color', 'k', 'LineWidth', 3);
hold on; grid on; xlabel('Time'); ylabel('Concentration');
area(t, y(:,2), 'FaceColor', [0.9 0.9 0.9])
legend('conc', ['AUC = ', num2str(y(end,3))])
print('-r800', '-dtiff', 'AUC')

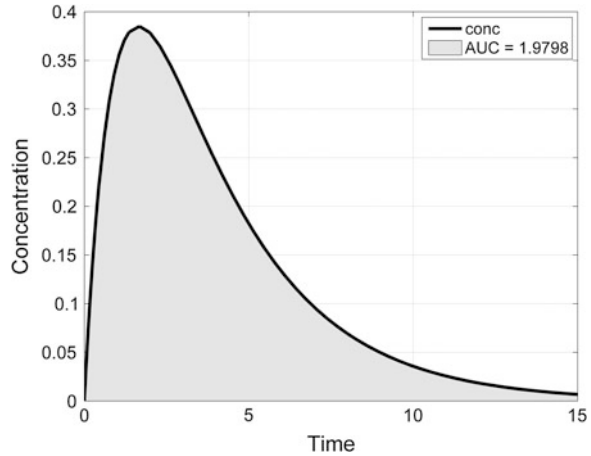
end

function dydt = derivatives(~, y, p)
%DERIVATIVES Compute the right-hand side of the ODE.
% DYDT ...

a = y(1); % drug amount
c = y(2); % concentration
% yAUC = y(3); % variable linked to AUC can be omitted
% PK Model
dadt = -p.ka*a;
dcdt = p.ka/p.V/p.F*a - p.CL/p.V*c;
dyAUCdt = c; % !!! derivative added to compute AUC
% Derivatives vector of ODE system
dydt = [dadt; dcdt; dyAUCdt];

end
```

**Fig. A.4** Area under the curve (concentration time course) derived from one-compartment model equation with first-order absorption and elimination, and computed directly from the ODE solution



Practical examples on how to apply this AUC computation method is demonstrated in [Sect. 8.1](#), *SimBiology*, [Fig. 8.11](#), in the `ibandronate.m` program and Simulink models `Iban_Simul.mdl`, `Iban_Simul_Multi.mdl` (available at MATLAB Central).

## A.7 Passing Additional Arguments in Function Call Statements

MATLAB offers many built-in functions which by default need only the necessary input parameters but optionally they can be called in extended form with any number of additional parameters. For example, the one-compartment model with first-order elimination rate,  $k = CL/V = 2$ ) and a bolus treatment,  $Dose = 50$ , can be solved with the call:

```
ode45(@(t,y) -2*y, [0 2], 50, [])
```

But we can make the call more flexible by adding  $k$  to the argument list

```
k = 2;
ode45(@(t,y,k) -k*y, [0 2], 50, [], k)
```

or even more, by using  $V$  and  $CL$  rather than  $k$

```
V = 2; CL = 4;
ode45(@(t,y,V,CL) -CL/V*y, [0 2], 50, [], V, CL)
```

We could introduce more and more parameters to the right-hand side of the ODE, and the procedure will always work. This flexibility is due to a specific mechanism we would like to discuss in the following.

We have created two simple programs, `locpar.m` in Listing A.6 and `addpars.m` in Listing A.7, containing all key elements needed to explain the issue.

**Listing A.6** Program **locpars.m**: additional parameters are assigned locally

```

function locpars
x = 1:5;
fo(x)
end

function y = fo(x)
eps = rand(1,length(x))*2 - 1;
y = fp(x) + eps;
end

function yhat = fp(x)
a = 1; b = 1;
z(x) = exp(x);
yhat = a*x + b + z(x);
end

```

The **locpars** function contains no possibility of executing the **fo** function with required flexibility regarding model parameters **a**, **b** and function **z(x)**. Each time we have to construct a new form of the **fp(x)** function, as these parameters are assigned locally within this function.

The problem can be avoided by modifying the **fo** function, as shown in Listing A.7. The list of arguments is now extended by the **varargin** parameter. Following a call to **fo**, the number of arguments, **nargin**, is checked, and further execution is properly managed: for more than one argument, the **fp** function is called with **x** and the additional arguments, **a**, **b**, **fun**, are stored properly in cell array **varargin**.

**Listing A.7** Program: **addpars.m**: passing additional parameters as arguments

```

function addpars
x = 1:5; a = 1; b = 1; fun = 'exp(x)'; % additional parameters
fo(x,a,b,fun)
end

function y = fo(x, varargin)
eps = rand(1,length(x))*2 - 1;
if nargin > 1
    y = fp(x,varargin{:}) + eps;
else
    y = fp(x) + eps;
end
end

function yhat = fp(x,a,b,fx)
z = inline(fx);
yhat = a*x + b + z(x);
end

```

The above method, in association with function handles (see the next section, *Handles*), offers very powerful functionality while calling general purpose functions with any number of additional arguments, function names included. It is also used in this book in various chapters, mainly to pass model parameters into functions which compute the right-hand side of ODE, DDE, DAE and PDE systems.

## A.8 Handles (Using @)

Various build-in functions (**ode\***, **pdepe**, **quad**, **fminsearch**, etc.) need a function handle as one of the parameters passed to them. For the analysis of the pharmacological model we mainly use the ODE and PDE solvers, where function handles are essential.

A function handle is a variable that contains full information about the function to which it is linked. In order to create a function handle we use the @ sign followed by a function name, for example **@odefun**. The following example shows how a function handle works. Both commands, **%1** and **%2**, give the same result, **r = 12.1825**.

```
r = exp(2.5);           %1
e = @exp; r = e(2.5);  %2
```

With regard to the final result, we can say that function **exp(x)** and its handle **e** are equivalent; however, the first is a direct function call, and the second a function evaluation by the handle.

Simplifying the problem, and using well-known programming terminology (valid for other than MATLAB programming languages), we can compare a function handle with a ‘powerful pointer’. A pointer is a variable which contains the address (in memory) of other variables or functions, so we can imagine a function handle as a variable that stores the function address, and can evaluate this function.

A simple example in Listing A.8 can help us to explain why a function handle is useful and makes another function powerful. In the example, three calls of the same function, **fo**, are demonstrated, each one with a different handle, **@fp**, **@fpnew** and **@log**.

**Listing A.8** Program `locparsMod.m`: using `@` to create a function handle

```

function locparsMod
x = 1:5;
fo(@fp,x)          % handle @fp as argument
fo(@fpnew,x)       % pass a handle of another function
fo(@log, x)        % a built-in function can also be applied.
end

function y = fo(fany,x)
eps = rand(1,length(x))*2 - 1;
y = fany(x) + eps;
end

function yhat = fp(x)
a = 1; b = 1; z = exp(x);
yhat = a*x + b + z;
end

function yhat = fpnew(x)
a = 1; b = 1; c = 1; z = log(x);
yhat = a*x.^2 + b*x + c + z;
end

```

The `fo` function is properly programmed, and expects a function as the first argument. No further changes are needed. It remains unchanged, and can correctly work with any function passed as argument.

In order to illustrate this functionality, we can follow the information flow as shown in Fig. A.5.

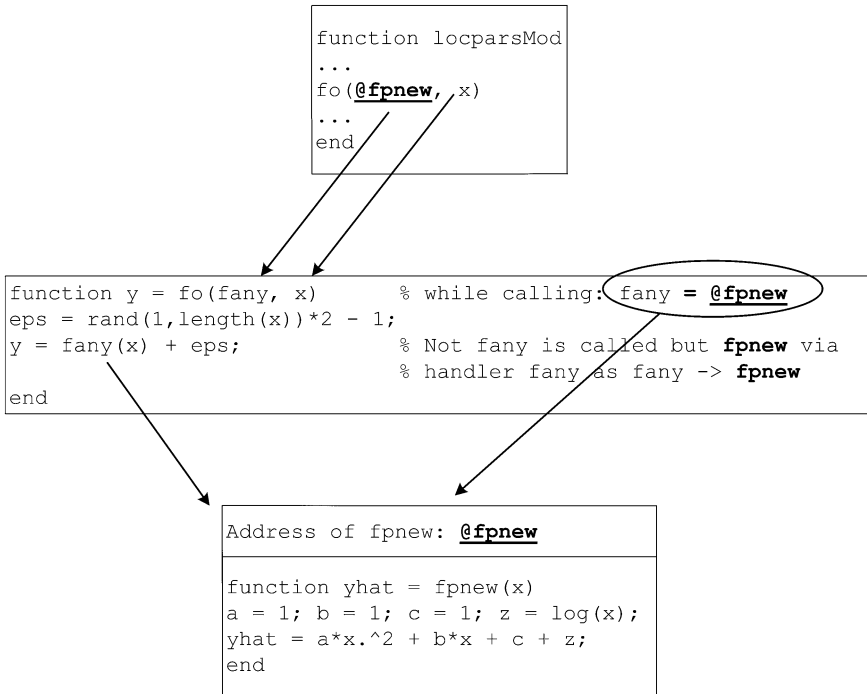
It is also important to emphasize that in above cases the call with a function handle is not only a better option than a call without a handle; it is necessary if we want to keep the key function (here `fo`) unchanged, or if it is a built-in function from MATLAB's library. In our example, the following call will work correctly:

```
fo(@fpnew, x);                % correct
```

but this one:

```
fo(fpnew, x);                % incorrect
```

will not, and will additionally generate an error message. The reason is that instead of calling `fo`, an attempt to call first `fpnew` without any parameters takes place, and this gives rise to the error message: *Not enough input arguments*. The `fo` function will not be entered at all.



**Fig. A.5** Mechanism of a function handle: The **fany** variable is an argument of the **fo** function, and plays a key role while the function is being called. In the above case, the **@fpnew** handle is assigned to **fany**. At this moment **fany** becomes a ‘box’ containing the address of **fpnew** as well as all information needed to evaluate (execute) the function. This box is not only dedicated to function **fpnew** but to any function, for example **fp**, **log**, ..., etc. In this way, function **fo** does not need to be changed anymore, and can process any function

Due to this mechanism, users are free to select any names of functions being passed to **fo** (as the first argument linked to **fany**). This flexibility would not have been possible in the **locpars.m** program (Listing A.6) where the proper function, **fp**, is hard coded within **fo**; in such a case it would have been necessary to write a library of ‘**fo**-like’ functions, each with a different replacement for **fp**. This would have been tedious and is not recommended.

With regards to model implementations in this book, function handles are used in many cases. First of all, any ODE, DDE, DAE and PDE solver is called with at least one function handle as input parameter (argument); these handles represent functions computing the right-hand sides as required for ODE, DDE DAE or PDE; in the DDE solvers they also represent the history functions;

additionally, in the PDE solvers they are linked to the boundary condition. If events have to be considered then a function handle is needed that represents the events function (function describing the events location properties). In the population analysis based on the NLMEM where the **nmlmefit** and **nmlmefitsa** functions are called, function handles represent the model prediction functions, as demonstrated in programs **nlmeBaseAnalysis.m** and **nlmeFinalAnalysis.m**.

In addition, we would like to emphasize that handle usability of is not limited to the cases described above, i.e. function handles. We encourage the reader to investigate and make use of these powerful objects.

## A.9 Handling Events

One of the most important strengths of MATLAB's differential equations solvers is the ability to handle events. In [Chap. 3, \*Differential Equations in MATLAB\*](#), we have already explained how MATLAB can handle events and we demonstrate various examples. We consistently used 2-dimensional arrays for the events descriptions. It means that the variables representing events within the event definition function (usually called here **fevent**) have two indices:

```
isterminal(i,1) = ...
direction(i,1) = ...
value(i,1) = ...
```

The first index is **i = 1:n**; where **n** is the number of events, and the second is always **1**. This means that arrays **isterminal**, **value** und **direction** should be kept as columns and not as rows (the default). Alternatively we could also keep them one-dimensional and then transpose them before they are returned. But is this necessary? Conventional one-dimensional representation usually suffices. A problem can arise when two or more events occur at the same time. An error message is then triggered about inconsistent matrix dimensions being concatenated within standard event-handling MATLAB procedures. Obviously, for column-wise organized vectors **isterminal**, **value** und **direction**, the problem can be managed. Listing A.9 offers an example. Events **2** and **4** happen at the same time and this implementation fails when detecting these events.

**Listing A.9** Program **evMultiProblem.m**: handling events

```

function evMultiProblem
%EVMULTIPROBLEM Event handling in a two-compartment ODE model
%   EVMULTIPROBLEM implements state events in a two-compartment
%   model. Plasma concentration should be kept within a given range
%   by infusion rate: if concentration achieves allowed maximum then
%   decrease infusion; if minimum - infusion has to be increased.

% PK parameters
CL = 1.38E+01; V1 = 1.48E+01; Q = 2.17E+00; V2 = 4.23E+00; R = 50;
% p structure with PK parameters available in several functions
p.R0 = R; p.R = R; p.V1 = V1; p.V2 = V2; p.k = CL/V1;
p.k12 = Q/V1; p.k21 = Q/V2;
options = odeset('Events',@fevents, 'RelTol',1e-9, 'AbsTol', 1e-9);
tmax = 30; tspan = [0 tmax]; t = 0; c0 = [0 0]; % Initial values
while t(end) < tmax
    [t, c, te, ye, ie] = ode45(@derivatives, tspan, c0, options, p);
    plot(t, c, 'LineWidth', 2); hold on
    if isempty(ie)
        break;
    end
    ie = ie(te == te(end)); % eliminate already handled events
    tspan = [te(end); tmax];
    c0 = [ye(end,1) ye(end,2)];
    display(['time: ', num2str(te(end)), ' events: ', ...
            num2str(ie')])
    p = todoevents(ie', p);
end
xlabel('Time [hours]'); ylabel('Concentration [mg/L]');
legend('central compartment','peripheral compartment')
title('Two-Compartment Model / Infusion / Multiple Doses')

end

function dcdt = derivatives(~, c, p)
%DERIVATIVES Compute the right-hand side of the ODE.
%   DCDT = ...

dcdt = [ p.R/p.V1 - (p.k + p.k12)*c(1) + p.k21*p.V2/p.V1*c(2)
        p.k12*p.V1/p.V2*c(1) - p.k21*c(2) ]';

end

function [value,isterminal,direction] = fevents(t,y,~)
%FEVENTS Define events.
%   [VALUE,ISTERMINAL,DIRECTION] = ...

% Event 1
isterminal(1) = 1; % Stop integration
direction(1) = 1; % Positive direction only
value(1) = (t - 3); % after 3 hours
% Event 2: Concentration at lower limit
isterminal(2) = 1; % Stop integration
direction(2) = -1; % Negative direction only

```



```

value(2) = (y(1) - 0.5);
% Event 3: Concentration at upper limit
isterminal(3) = 1;          % Stop integration
direction(3) = 1;          % Positive direction only
value(3) = (y(1) - 2);
% Event 4: test event forced to happen together with event 4
isterminal(4) = 1;          % Stop integration
direction(4) = 1;          % Negative direction only
value(4) = t - 14.459115546074692;

end

function pMod = todoevents(numEvents, p)
%TODOEVENTS ...

pMod = p;
% use 'for' as more than 1 event can happen at the same time
for i = numEvents
    switch i
        case 1
            % stop infusion after 3 hours
            display('event 1: handled')
            pMod.R = 0;
        case 2
            % allow full infusion
            display('event 2: handled')
            pMod.R = p.R0;
        case 3
            % reduce infusion
            display('event 3: handled')
            pMod.R = p.R0/10;
        case 4
            % show a message to event 4
            display('-----')
            display('event 4 - TEST EVENT: handled')
            display('-----')
        otherwise
            messageID = 'Book:infus2comstev:UnknownEvent';
            messageStr = 'Unknown event '%s'.';
            error(messageID, messageStr, num2str(i));
    end
end
end

```

In the `evMultiProblem.m` program, event **2** is ‘artificially provoked’ to happen simultaneously with event **4** which illustrates the problem.

In order to fix the problem, we need only to modify the `fevents` function as shown in Listing A.10, i.e. change the indexing for `isterminal`, `value`, and `direction`.

**Listing A.10** Function **fevents**: handling events

```

function [value,isterminal,direction] = fevents(t,y,-)
%FEVENTS Define events.
%   [VALUE,ISTERMINAL,DIRECTION] = ...

% Event 1
isterminal(1,1) = 1;           % Stop integration
direction(1,1) = 1;           % Positive direction only
value(1,1) = (t - 3);         % after 3 hours
% Event 2: Concentration at lower limit
isterminal(2,1) = 1;           % Stop integration
direction(2,1) = -1;          % Negative direction only
value(2,1) = (y(1) - 0.5);
% Event 3: Concentration at upper limit
isterminal(3,1) = 1;           % Stop integration
direction(3,1) = 1;           % Positive direction only
value(3,1) = (y(1) - 2);
% Event 4: test event forced to happen together with event 4
isterminal(4,1) = 1;           % Stop integration
direction(4,1) = 1;           % Negative direction only
value(4,1) = t - 14.459115546074692;

end

```

In this way the modified program (available at MATLAB Central as **evMultiProblemMod.m**) can handle all the scheduled events.

There are many applications in the book in which events need to be scheduled and handled, for example, the **ibandronate.m**, **esaCTS.m**, **epodosing.m** programs. Furthermore, in [Chap. 3](#), *Differential Equations in MATLAB*, the reader can find numerous examples that use events.

## A.10 Super Label

In [Chap. 6](#), we created graphs with individual time courses for drug concentrations. They depict both individual observations and predictions, and population predictions. The corresponding graphs contain many subplots demonstrating the time profiles of drug concentrations versus time, and they are very tightly stored within the main graph. For that reason, axes descriptions and a title for each subplot are not practical. Instead, we may use only one label for the  $x$  axis, one for the  $y$  axis, and one title for the whole figure. We will call them *super labels*. MATLAB has no single graphical function to place these super labels on a graph. However, on MATLAB Central there is the **suplabel** function and the link to this function is given in our example **indProfilesMOD.m** (Listing A.11).

**Listing A.11** Program **indProfilesMOD.m**: how to implement a super label

```

function indProfilesMod
%INDPROFILESMOD Simulate time profiles and demonstrate 'super labels'
%
%   This function uses suplabel from MATLAB Central:
%   http://www.mathworks.ch/matlabcentral/fileexchange/7772-suplabel
%   Copyright (c) 2004, Ben Barrowes
%   All rights reserved.
%   Code covered by the BSD License: http://www.mathworks.com/
%   matlabcentral/fileexchange/view_license?file_info_id=7772

ns = 20;
p = modelparams(ns);
a0 = 10*ones(1,1); c0 = zeros(1,1); y0 = [a0; c0]; tspan = [0 15];
for id = 1:p.numSub
    pind = struct('ka',p.ka(id), ...
                  'V',p.V(id), ...
                  'F',p.F(id), ...
                  'CL',p.CL(id));
    [t,y] = ode15s(@derivatives, tspan, y0, [], pind);
    subplot(5,4,id)
    plot(t,y(:,2),'Color','k' )
    title(['ID=',num2str(id)]);
end
[~,hx]=suplabel('Time');          set(hx,'FontSize',14)
[~,hy]=suplabel('Drug Concentration','y'); set(hy,'FontSize',14)
[~,ht]=suplabel(['Time Courses', 10], 't'); set(ht,'FontSize',18)
print('-r900', '-dtiff', 'indProfilesMod')

end

function p = modelparams(ns)
% .....
% .....
% .....

end

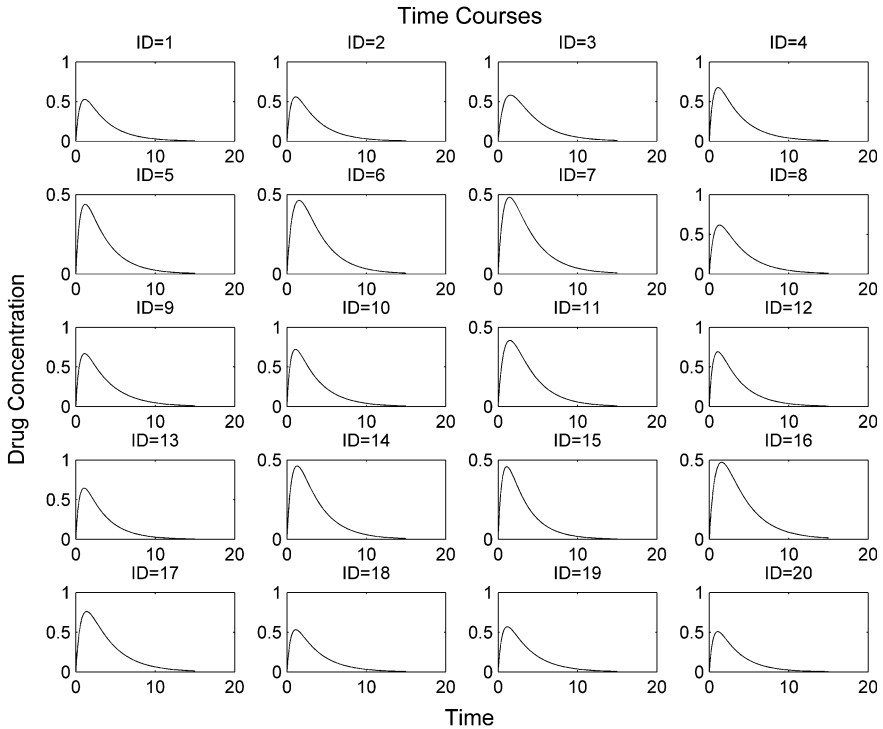
function dydt = derivatives(~, y, p)
% .....
% .....
% .....

end

```

The **indProfilesMOD.m** program produces the graph shown in Fig. A.6.

The **suplabel** function has been applied in this book to demonstrate results of the population analysis in the following functions: **explore** and **predtime**. For more details regarding **suplabel** see the descriptive text that follows the function header, or use the help command: **help suplabel**.



**Fig. A.6** Demonstration of super labels: drug concentration versus time courses were generated for 20 subjects, each in a separate subplot, stored in one figure, and described with common labels, *Time*, *Drug Concentration* and *Time Courses*

## A.11 Handling Legends

The number of lines (and related descriptions) in a legend to a graph is usually equal to the numbers of graphical objects (curves, histogram, scatter plot) generated in the graph. MATLAB stores information for the legend in the same order as that in which the curves are created. For example, **legend('A','B')** will describe two lines in order of appearance.

There are some cases, though, when the legend should describe selected graphical components (e.g. curves) but not all those existing in the figure. For example, a number of curves (say 10) may be drawn for two groups and we would like to label the groups rather than all curves. This scenario is simulated by the **consLegHan.m** program in Listing A.12. We have two cohorts (doses: 100 and 200), and only need to identify each cohort instead of each subject. Knowing that time courses for '**cohort A**' are generated first, followed by '**cohort B**', we only need to select legend information of one representative in each case. This was implemented in the program below with the help of handles.

**Listing A.12** Program **consLegHan.m**

```

function consLegHan
%CONSOLEGHAN Simulation of 2 cohorts and legend demonstration

ns = 10;
p = modelparams(ns);
%cohort A
a0 = 100; c0 = 0; y0 = [a0; c0]; tspan = [0 15];
for id = 1:p.numSub
    pind = struct('ka',p.ka(id), ...
                  'V',p.V(id), ...
                  'F',p.F(id), ...
                  'CL',p.CL(id));
    [t,y] = ode15s(@derivatives, tspan, y0, [], pind);
    hA = plot(t,y(:,2),'Color','b', 'LineWidth',2); hold on
    % AAA
end
%cohort B
a0 = 200; c0 = 0; y0 = [a0; c0]; tspan = [0 15];
for id = 1:p.numSub
    pind = struct('ka',p.ka(id), ...
                  'V',p.V(id), ...
                  'F',p.F(id), ...
                  'CL',p.CL(id));
    [t,y] = ode15s(@derivatives, tspan, y0, [], pind);
    hB = plot(t,y(:,2),'Color','r','LineWidth',2 ); hold on
    % BBB
end
xlabel('Time'); ylabel('Concentration');

legend([hA hB], 'cohort A', 'cohort B')           % COR
title('Legend Specified Correctly')
print('-r900', '-dtiff', 'consecLegHan_COR')

% legend('cohort A','cohort B')                 % INCOR
% title('Legend Specified Incorrectly')
% print('-r900', '-dtiff', 'consecLegHan_INCOR')
end

function p = modelparams(ns)
% .....
% .....
% .....

end

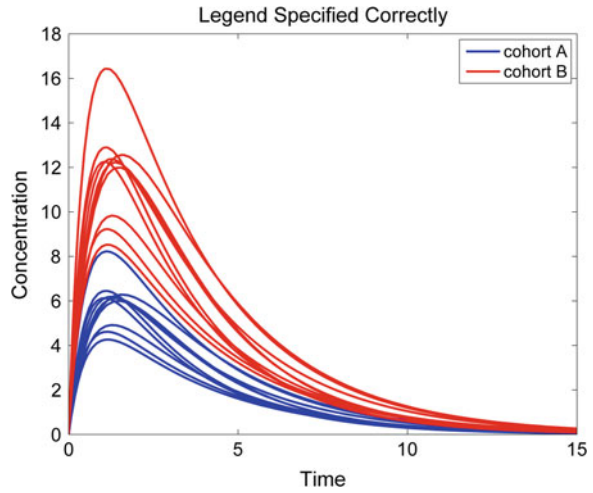
function dydt = derivatives(~, y, p)
% .....
% .....
% .....

end

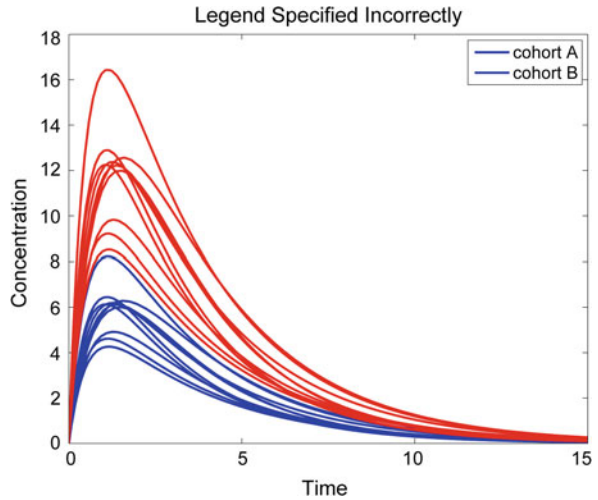
```

The important step is to use handles **hA** and **hB** as input to the **legend** function. Both handles must be created prior to calling **legend** and they represent the groups of time profiles for both cohorts, respectively:

**Fig. A.7** Creating a group legend: the legend describes drug concentrations versus time profiles for two cohorts



**Fig. A.8** Creating a group legend: without using a legend handle the legend is not as expected (same line color for different cohorts)



```
legend([hA hB], 'cohort A', 'cohort B') % COR
```

The correct output is shown in Fig. A.7.

In this scenario, we cannot use the following statement:

```
legend('cohort A', 'cohort B') % INCOR
```

It is not correct and will give rise to the result shown in Fig. A.8. The first and second generated curves both belong to the first cohort, and the above specification will not reflect the correct labeling of both groups.

Another possibility is to eliminate, while executing the loop, the legend information for all curves except one for cohort A and one for cohort B (for example, the first or last ones). One can replace in Listing A.12 the comment lines indicated as % **AAA** and % **BBB** with

```
if id ~= 1
    set(get(get(hA,'Annotation'),'LegendInformation'),...
        'IconDisplayStyle','off');
end
```

and

```
if id ~= 1
    set(get(get(hB,'Annotation'),'LegendInformation'),...
        'IconDisplayStyle','off');
end
```

respectively. In such case, we can use one of both forms:

```
legend([hA hB], 'cohort A', 'cohort B')    % LEG1
legend('cohort A', 'cohort B')             % LEG2
```

Having done the above modifications, the program will produce exactly the same legend to the graph as it was shown in Fig. [A.7](#).

The problem is also not quite trivial if the time courses (curves) are generated simultaneously (Listing A.4). In such cases we make direct use of the first elements of the handle vectors, **hA(1)** and **hB(1)**, returned by plot functions. The **SimuLegHan.m** program shows a possible implementation (Listing A.13), and the output is the same as shown in Fig. [A.7](#).

**Listing A.13** Program **simuLegHan.m**

```

function simuLegHan
%SIMULEGHAN Simulate 2 cohorts and demonstrate creation of legend

ns = 10;
p = modelparams(ns);

%cohort A
a0 = 100*ones(ns,1); c0 = zeros(ns,1); y0 = [a0; c0]; tspan = [0 15];
[t,y] = ode15s(@derivatives, tspan, y0, [], p);
hA = plot(t,y(:,ns+1 : 2*ns), 'Color', 'b', 'LineWidth',2); hold on
% AAA

%cohort B
a0 = 200*ones(ns,1); c0 = zeros(ns,1); y0 = [a0; c0]; tspan = [0 15];
[t,y] = ode15s(@derivatives, tspan, y0, [], p);
hB = plot(t,y(:,ns+1 : 2*ns), 'Color', 'r', 'LineWidth',2); hold on
% BBB

xlabel('Time'); ylabel('Concentration'); title(' Legend Demo')
% Setup a legend
legend([hA(1) hB(1)], 'cohort A', 'cohort B') % LEG 1

print('-r900', '-dtiff', 'simuLegHan')

end

function p = modelparams(ns)
% .....
% .....
% .....

end

function dydt = derivatives(~, y, p)
% .....
% .....
% .....

end

```

Alternatively, we can eliminate from the plotted objects the legend information for all curves excepting any one for cohort A and cohort B, for example the first ones or the last ones. Using Listing A.12, we can replace the comment line indicated as **% AAA** with the following statements

```

h1 = get(hA, 'Annotation');
h2 = get([h1{:}], {'LegendInformation'});
% Exclude lines from legend (except the first)
set([h2{2:ns}], 'IconDisplayStyle', 'off');

```



and % **BBB** with

```
h1 = get(hB,'Annotation');
h2 = get([h1{:}],{'LegendInformation'});
% Exclude lines from legend (except the first)
set([h2{2:ns}], 'IconDisplayStyle','off');
```

and finally use one of the forms:

```
legend([hA(1) hB(1)], 'cohort A', 'cohort B') % LEG 1
legend('cohort A', 'cohort B') % LEG 2
```

The program will produce exactly the same legend to the graph as it was shown in Fig. A.7.

Some applications in the book use the above methods to properly create legends, for example **epodosing.m** or **CTSbasic.m**.

## A.12 Including Axes Handles to Plot

Including the axes handle as an input to **plot** is a procedure that programmatically consists of two steps:

```
% 1 - create axes and the handle to it, e.g. axh
axh = axes;
% 2 - include the handle to plot as an input
plot(axh, t, y)
```

By default the axes handles are not included, nor is this necessary so long as we do not move forth and back from one figure to another. Otherwise we have to tell MATLAB how to distribute the generated graphics object. There are two possibilities: we can call the **figure** statement before the **plot** statement is executed or we can include axes handles as an input to **plot**.

In general, including the axes handles rather than calling the **figure** command prior to the plot avoids the 'flashing' of the Figure Window as each plot gets updated in turn. If the output needs to be observed in the Figure Window during runtime this is the preferred option.

Listing A.14 contains a simple example and shows how the axes handle could be used. The program generates time profiles based on a simulated data set. The time profiles should be split into two groups, the first for subjects having  $V \geq 16$  L, and the second for subjects with  $V < 16$ . Two graphs should be produced, respectively.

**Listing A.14** Program **consAxes.m**: demonstration how to include axes

```

function consAxes
%CONSAxes Include and handle axes.

figure(1)
axesA = axes;
set(axesA,'FontSize',20);
xlabel('Time'); ylabel('Concentration'); title('V => 16 [L]')
ylim([0 1])
hold on

figure(2)
axesB = axes;
set(axesB,'FontSize',20);
xlabel('Time'); ylabel('Concentration'); title('V < 16 [L]')
ylim([0 1])
hold on;

set([figure(1); figure(2)], 'units', 'normalized', ...
    {'Position'}, {[0.1, 0.5, 0.3, 0.4]; [0.6, 0.5, 0.3, 0.4]});

ns = 50;
p = modelparams(ns);
a0 = 10; c0 = 0; y0 = [a0; c0]; tspan = [0 15];
sparams = struct('ka', num2cell(p.ka'), ...
    'V', num2cell(p.V'), ...
    'F', num2cell(p.F'), ...
    'CL', num2cell(p.CL'));
for pind = sparms
    [t,y] = ode15s(@derivatives, tspan, y0, [], pind);
    if pind.V >= 16
        figure(1); plot( t, y(:,2), 'k', 'LineWidth',2); % FIG
        plot(axesA, t, y(:,2), 'k', 'LineWidth',1.5); % AXES
    else
        figure(2); plot( t, y(:,2), 'k', 'LineWidth',2); % FIG
        plot(axesB, t, y(:,2), 'k', 'LineWidth',1.5); % AXES
    end
    pause(1/2)
end
print(figure(1), '-r900', '-dtiff', 'consAxesA')
print(figure(2), '-r900', '-dtiff', 'consAxesB')

end

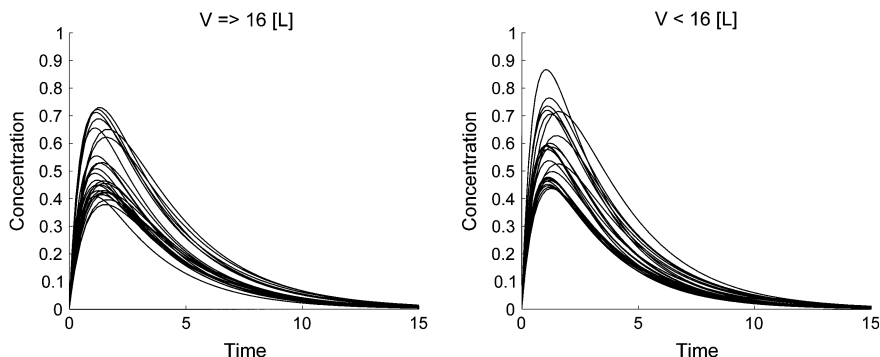
function p = modelparams(ns)
% .....
% .....
% .....

end

function dydt = derivatives(~, y, p)
% .....
% .....
% .....

end

```



**Fig. A.9** Drug concentration—time profiles of 50 subjects generated with the **consAxes.m** program. Curves for subjects with  $V \geq 16$  L (left panel) and  $V < 16$  L (right panel)

In this context (including axes), the deciding role is played by the statements indicated as **% AXES** and **% FIG**, linked to the version including axes in the **plot** function and using the **figure** command, respectively. The reader, via *comment* and *uncomment*, can test both versions, and observe how the figures behave. No ‘flashing’ should be observed when the axes are included.

The time profiles generated by the **consAxes.m** program are shown in Fig. A.9. Of course, the final result is the same, whether the axes are included or not. However, the behavior of both figures during the runtime is different.

The reader can also find some application in the book where axes are included in the **plot** function, for example **concmmod.m**, **esaCTS.m** and **hoghax.m**.

## A.13 Using GUI

In Sect. 5.2.1, *Physiologic Background*, equilibrium and resting potential were mathematically described by two formulas, the Nernst equation and Goldman equation, respectively. Both formulas can be easily implemented as calculator using MATLAB’s GUI builder. The MATLAB help documentation on GUI as well as many other sources describe how a GUI-based application can be built. In this section, the **calpot** calculator will be demonstrated as an example. It is available on MATLAB Central and is used for calculating equilibrium and resting potentials. Unlike a standard MATLAB program, a GUI program generates by default two files which are needed for execution, a **fig**-file (binary file storing the layout and GUI components), and a **m**-file (containing the specifics for the application source code with regards to the initialization procedures and behavior

**Fig. A.10** The GUI-based calculator **calpot** can be used to compute the equilibrium potential,  $E(\text{ion})$ , ( $\text{ion} \equiv \text{K}^+, \text{Na}^+, \text{Cl}^-$ ) and resting potential according to the Nernst and Goldman equations, respectively. The default values, as shown in the figure, are linked to the neuron at resting state and temperature  $T = 20^\circ \text{C}$

The screenshot shows a MATLAB GUI titled 'calpot'. It features a table of ion parameters and a temperature input field. The table has columns for ion type, internal concentration (ion(i)), external concentration (ion(o)), permeability (P), and equilibrium potential (E(ion)). The rows are for K+, Na+, and Cl-. Below the table is a temperature input field set to 20°C. At the bottom, the calculated membrane potential is displayed as -58.1833 mV, with a 'Calculate' button below it.

	ion( i )	ion( o )	P	E(ion)
K+	400	20	1	-75.6731
Na+	50	440	0.05	54.9349
Cl-	52	560	0.45	-60.036

Temp. [°C]: 20

Membrane potential [mV]: V = -58.1833

Calculate

of the GUI). The calculator consists of **calpot.fig** and **calpot.m**. After downloading both files (to the same directory), launch the application by entering

» **calpot**

This will generate the GUI, as shown in Fig. A.10.

Using MATLAB's GUI, the main structure of the source code is automatically created by the programming system. It is for the user (programmer) to write callbacks and the application specific function (usually for numerical calculation).

In the **calpot.m** program, the initialization values (see Listing A.15), and callback to handle "press button Calculate" (Listing A.16), need to be added by the user (see sections **Added (BEGIN)** ... **Added (END)**).

**Listing A.15** Function `calpot_OpeningFcn`: executes just before `calpot` is made visible

```
function calpot_OpeningFcn(hObject, ~, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)
% varargin    command line arguments to calpot (see VARARGIN)

% Choose default command line output for calpot
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes calpot wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% ----- Added (BEGIN) -----
global data
% values at rest
data.KpIn = 400; data.KpOut = 20; data.PK = 1; data.zK = 1;
data.NapIn = 50; data.NapOut = 440; data.PNa = 0.05; data.zNa = 1;
data.ClmIn = 52; data.ClmOut = 560; data.PCl = 0.45; data.zCl = -1;
data.temp = 20;

% mapping initial values to the text boxes in GUI
set(handles.K1, 'String', data.KpIn)
set(handles.K2, 'String', data.KpOut)
set(handles.K3, 'String', data.PK)
set(handles.Na1, 'String', data.NapIn)
set(handles.Na2, 'String', data.NapOut)
set(handles.Na3, 'String', data.PNa)
set(handles.Cl1, 'String', data.ClmIn)
set(handles.Cl2, 'String', data.ClmOut)
set(handles.Cl3, 'String', data.PCl)
set(handles.temp, 'String', data.temp );

% compute according to Nernst and Goldman equations
[rp, epK, epNa, epCl, ~, ~, ~] = resting();
% mapping computed values to the text boxes in GUI
set(handles.Resting, 'String', num2str(rp))
set(handles.EK, 'String', num2str(epK))
set(handles.ENa, 'String', num2str(epNa))
set(handles.ECl, 'String', num2str(epCl))
% ----- Added (END) -----

end
```

**Listing A.16** Function **Calculate\_Callback**: executes on button press in *Calculate*

```
function Calculate_Callback(~, ~, handles) %#ok<DEFNU>
% hObject      handle to Calculate (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% ----- Added (BEGIN) -----
global data
data.KpIn      = str2num(get(handles.K1, 'String')); %#ok<*ST2NM>
data.KpOut     = str2num(get(handles.K2, 'String'));
data.PK        = str2num(get(handles.K3, 'String'));
data.NapIn     = str2num(get(handles.Na1, 'String'));
data.NapOut    = str2num(get(handles.Na2, 'String'));
data.PNa       = str2num(get(handles.Na3, 'String'));
data.ClmIn     = str2num(get(handles.C11, 'String'));
data.ClmOut    = str2num(get(handles.C12, 'String'));
data.PCl       = str2num(get(handles.C13, 'String'));
data.temp      = str2num(get(handles.temp, 'String'));

% compute according to Nernst and Goldman equations
[rp, epK, epNa, epCl, ~, ~, ~] = resting();
% mapping computed values to the text boxes in GUI
set(handles.Resting, 'String', num2str(rp))
set(handles.EK, 'String', num2str(epK))
set(handles.ENa, 'String', num2str(epNa))
set(handles.ECl, 'String', num2str(epCl))
% ----- Added (END) -----

end
```

The additional **resting** function must be specified to calculate the equilibrium and resting potential every time the calculator is launched, i.e. when the *Calculate* button has been pressed (after input data changes). It is shown in Listing A.17.

**Listing A.17** Function **resting**: called by callbacks

```

function [V, EK, ENa, ECl, VK, VNa, VCl] = resting()
%RESTING Compute the equilibrium and resting potential
% [V, EK, ENa, ECl, VK, VNa, VCl] = RESTING() calculates the
% equilibrium and resting potential according to Nernst and
% Goldman equations.
% Input arguments: passed globally
% Output: V - resting potential; EK, ENa, ECl - equilibrium
% potentials for K K Cl channels, respectively. VK, VNa, VCl -
% equilibrium potentials calculated with another fomula.

global data
KpIn = data.KpIn; KpOut = data.KpOut;
PK = data.PK; zK = data.zK;
NapIn = data.NapIn; NapOut = data.NapOut;
PNa = data.PNa; zNa = data.zNa;
ClmIn = data.ClmIn; ClmOut = data.ClmOut;
PCl = data.PCl; zCl = data.zCl;
temp = data.temp;

R = 8.314; % J/(K·mol)
F = 96485.3415; % C/mol or J/(V·mol) 96 485.3415
T = temp + 273.15; % Kelvin = Celsius + 273.15)

% Equilibrium
EK = R*T/F/zK*log(KpOut/KpIn)*1000; % in mV
ENa = R*T/F/zNa*log(NapOut/NapIn)*1000; % in mV
ECl = R*T/F/zCl*log(ClmOut/ClmIn)*1000; % in mV
V = R*T/F*log((NapOut*PNa + KpOut*PK + ClmIn*PCl)/ ...
(NapIn*PNa + KpIn*PK + ClmOut*PCl))*1000; % in mV

% Equilibrium (using an equivalent fomula)
k = 1.38E-23; % Boltzman constant [J/K]
q = 1.602E-19; % magnitude of the electronic charge [C]
VK = k*T/q*log(KpOut/KpIn)*1000; % in mV
VNa = k*T/q*log(NapOut/NapIn)*1000; % in mV
VCl = k*T/q*log(ClmOut/ClmIn)*1000; % in mV

end

```

**References**

1. Moler C (2004) Numerical computing with MATLAB. The MathWorks, Inc. <http://www.mathworks.ch/moler/chapters.html>
2. Hanselman D, Littlefield B (2005) Mastering MATLAB 7. Pearson Prentice Hall, Upper Saddle River
3. MATLAB® Programming Fundamentals, R2012b (2012) The MathWorks, Inc., 3 Apple Hill Drive Natick, 1760–2098. [http://www.mathworks.ch/help/pdf\\_doc/matlab/matlab\\_prog.pdf](http://www.mathworks.ch/help/pdf_doc/matlab/matlab_prog.pdf)

4. MATLAB<sup>®</sup> Mathematics, R2012b (2012) The MathWorks, Inc., 3 Apple Hill Drive Natick, MA 01760–2098; [http://www.mathworks.ch/help/pdf\\_doc/matlab/math.pdf](http://www.mathworks.ch/help/pdf_doc/matlab/math.pdf)
5. [http://www.mathworks.ch/ch/help/pdf\\_doc/allpdf.html](http://www.mathworks.ch/ch/help/pdf_doc/allpdf.html)
6. Xue D, Chen YQ (2009) Solving applied mathematical problems with MATLAB. Chapman & Hall, Boca Raton
7. Ogata K (2008) MATLAB for control engineers. Pearson, Englewood Cliffs
8. Moscinski J, Ogonowski Z (1995) Advanced control with MATLAB & SIMULINK. Ellis Horwood
9. Wallisch P, Lusignan M, Benayoun M, Baker TI, Dickey AS, Hatsopoulos NG (2009) MATLAB for neuroscientists: an introduction to scientific computing in MATLAB. Academic Press, London
10. Gabbiani F, Cox SJ (2010) Mathematics for Neuroscientists. Academic Press
11. Martinez LW, Martinez AR, Solka JL (2011) Exploratory data analysis with MATLAB. CRC Press, A Chapman & Hall Book
12. Demidovich BP, Maron IA (1981) Computational mathematics. Mir Publishers, Moscow
13. Press WH, Teukolsky SA, Vetterling WT, Flannery BP (2007) Numerical recipes: the art of scientific computing. Cambridge University Press, New York



## Appendix B: Solution to Exercises

### Exercise 2.1

Rewrite the `concmold()` program applying vectorization. In this implementation you will have to introduce element-wise vector operators `./`, `.*`, `.^` instead of standard operators `/`, `*`, `^` where necessary, and to remove all loops `for`.

### Solution

It is important to note that thanks to vectorization the proper ODE system has to be solved only once for all subjects simultaneously within the whole treatment time  $T$ , instead of creating a loop for each subject. For example, the `derivatives` function that computes the right-hand sides of the ODE system can be written in the case of vectorization as shown in Listing B.1.

**Listing B.1** Function **derivatives** shows how to specify the ODE right-hand sides concurrently for all subjects (vectorization)

```
function dydt = derivatives( ~, Y, p)
%DERIVATIVES Compute the right-hand side of the ODE.
% DYDT ...

nSub = p.numSubjects;

amount = Y(1+(1-1)*nSub : 1*nSub)'; % drug amount
c1      = Y(1+(2-1)*nSub : 2*nSub)'; % parent drug concentration
c2      = Y(1+(3-1)*nSub : 3*nSub)'; % active metabolite concentration
n       = Y(1+(4-1)*nSub : 4*nSub)'; % tumor growth
pc      = Y(1+(5-1)*nSub : 5*nSub)'; % cells (proliferative compart.)
dc      = Y(1+(6-1)*nSub : 6*nSub)'; % cells (differentiated compar.)
sc      = Y(1+(7-1)*nSub : 7*nSub)'; % cells in stratum corneum

% PK Model
dAmountdt = -p.k01.*amount;

dcdt1 = p.k01./p.V1.*amount - (p.CL12./p.V1 + p.CL10./p.V1).*c1;
dcdt2 = p.CL12./p.V2.*c1 - p.CL20./p.V2.*c2;

% Efficacy Model
kkill = hillEffect(p.tumFactor.*c2, 0, p.kkillMax, p.kCkill50, 1);
dndt  = (p.k0.*log(p.nn00./n) - (p.k + kkill)).*n;

% Toxicity Model
tox = hillEffect(p.toxFactor.*c2, 0, p.toxMax, p.toxC50, 1);
br1 = p.br0.*(1-tox); % birth rate;
dpcdt = br1 - p.kpc.*pc;
ddcdt = p.kpc.*pc - p.kdc.*dc;
dscdt = p.kdc.*dc - p.ksc.*sc;

% Derivatives vector of ODE system
dydt = [ dAmountdt'; % drug amount
         dcdt1'; % parent drug concentration
         dcdt2'; % active metabolite concentration
         dndt'; % tumor growth
         dpcdt'; % changes in proliferative compartment
         ddcdt'; % changes in differentiated compartment
         dscdt' ]; % changes in stratum corneum compartment

end
```

The full solution as the MATLAB **concmodvector.m** program is available at MATLAB Central.

## Exercise 2.2

Visualize the drug concentrations (parent and metabolite) after first intake.

### Solution

We can modify the **concmod.m** program presented in [Chap. 2, First Example of a Computational Model](#). The quantities we would like to visualize, namely the concentrations of parent drug and metabolite, are coded as dependent variable **Y(:,2)** and **Y(:,3)**. These values need to be extracted in the time period between the first and second drug intake. The details are presented in Listing B.2 where the required modifications are commented with **% INCLUDE CONC !**.

**Listing B.2** Program **concmmodconc.m**: the listing contains only the primary function with the necessary changes (statements with **% INCLUDE CONC !**). The remaining segments are the same as in the program **concmmod**

```
function concmodconc(regimenType, numSubjects)
%CONCMODCONC Tumor Growth Model.
% CONCMODCONC(REGIMENTYPE, NUMSUBJECTS) models tumor growth and
% creates graphs of tumor growth and epidermis over time.
% Additionally, concentration time profiles for the parent and
% metabolite compartment are produced.
% |REGIMENTYPE| must be either 'intermittent' or 'continuous'.
% |NUMSUBJECTS| is the number of subjects.

%
% Example:
% Model tumor growth with an intermittent regimen and 10 subjects
% concmodconc('intermittent',10)
% Check input arguments and provide default values if necessary
error(nargchk(0, 2, nargin));      %#ok<*NCHKN>

if nargin < 2
    numSubjects = 10;
end
if nargin < 1
    regimenType = 'intermittent';
end

% Reset the random number generator to the default
rng default

% Calculate dosing times and amount based on regimen
[doseTimes, doseAmount] = doseSchedule(regimenType);

% Set up figures for plotting results
tumorFigure = figure;
tumorAxes = axes; set(tumorAxes,'FontSize',15)
xlabel('Time [h]')
ylabel('Number of tumor cells (relative to baseline)')
title('Tumor Growth')
xlim([0, doseTimes(end)]); hold on; grid on;

epidermisFigure = figure;
epidermisAxes = axes; set(epidermisAxes,'FontSize',15)
xlabel('Time [h]')
ylabel('% Change (relative to baseline)')
title('Epidermis')
xlim([0, doseTimes(end)]); hold on; grid on;

conc1Figure = figure;                                % INCLUDE CONC !
conc1Axes = axes; set(conc1Axes,'FontSize',15)        % INCLUDE CONC !
set(conc1Axes, 'FontSize', 15)                        % INCLUDE CONC !
xlabel('Time [h]')                                    % INCLUDE CONC !
ylabel('Concentration')                               % INCLUDE CONC !
title('Parent Concentration')                         % INCLUDE CONC !
hold on; grid on;                                     % INCLUDE CONC !

conc2Figure = figure;                                % INCLUDE CONC !
conc2Axes = axes; set(conc1Axes,'FontSize',15)        % INCLUDE CONC !
set(conc2Axes, 'FontSize', 15)                        % INCLUDE CONC !
xlabel('Time [h]')                                    % INCLUDE CONC !
ylabel('Concentration')                               % INCLUDE CONC !
```

```

set([tumorFigure; epidermisFigure; conc1Figure; conc2Figure], ...
    'units', 'normalized', ...
    {'Position'}, {[0.1, 0.5, 0.3, 0.4]; [0.6, 0.5, 0.3, 0.4]; ...
    [0.1, 0.05, 0.3, 0.4]; [0.6, 0.05, 0.3, 0.4]}); % INCLUDE CONC !
% Simulate system for each subject
for subjectID = 1:numSubjects
    % Initialize parameters and set initial values
    p = initializeParams;
    y0 = [doseAmount, p.c10, p.c20, p.n0, p.pc0, p.dc0, p.sc0];
    % Allocate variables to store results
    timePoints = [];
    tumorGrowth = [];
    epidermis = [];
    conc1 = []; % INCLUDE CONC !
    conc2 = []; % INCLUDE CONC !
    % Simulate system for each treatment period
    for dose = 1:(length(doseTimes)-1)
        % Set time interval for this treatment period
        tspan = [doseTimes(dose), doseTimes(dose+1)];

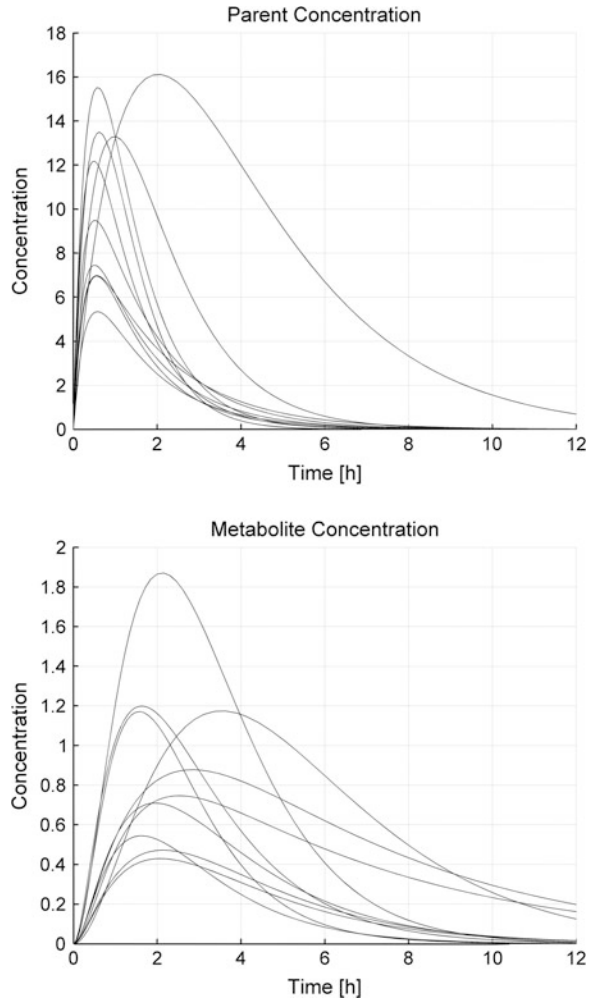
        % Call Runge-Kutta method
        [t,y] = ode45(@derivatives, tspan, y0, [], p);

        % Record values for plotting
        timePoints = [timePoints ; t]; %ok
        tumorGrowth = [tumorGrowth; y(:,4)/p.n0]; %ok
        epidermis = [epidermis ; 100*y(:,7)/p.sc0]; %ok
        if dose == 1
            conc1 = [conc1; y(:,2)]; %ok % INCLUDE CONC !
            conc2 = [conc2; y(:,3)]; %ok % INCLUDE CONC !
            tconc = timePoints;
        end
        % Reset initial values for next treatment period
        % and add next dose
        y0 = y(end,:);
        y0(1) = y0(1) + doseAmount;
    end
    % Plot results for this subject
    plot(tumorAxes, timePoints, tumorGrowth, 'k')
    plot(epidermisAxes, timePoints, epidermis, 'k')
    plot(conc1Axes, tconc, conc1, 'k') % INCLUDE CONC !
    plot(conc2Axes, tconc, conc2, 'k') % INCLUDE CONC !
    drawnow
end
% save graphs as TIFF file
print(tumorFigure, '-r600', '-dtiff', ...
    ['tumor', '_', regimenType])
print(epidermisFigure, '-r600', '-dtiff', ...
    ['epidermis', '_', regimenType])
print(conc1Figure, '-r600', '-dtiff', ...
    ['conc1', '_', regimenType]) % INCLUDE CONC !
print(conc2Figure, '-r600', '-dtiff', ...
    ['conc2', '_', regimenType]) % INCLUDE CONC !
end

```

The program produces output shown in Fig. B.1.

**Fig. B.1** The output produced by the `concmod-conc.m` program, i.e. the concentration–time profiles for the parent and metabolite compartments after the first dose only (in a run with 10 subjects)



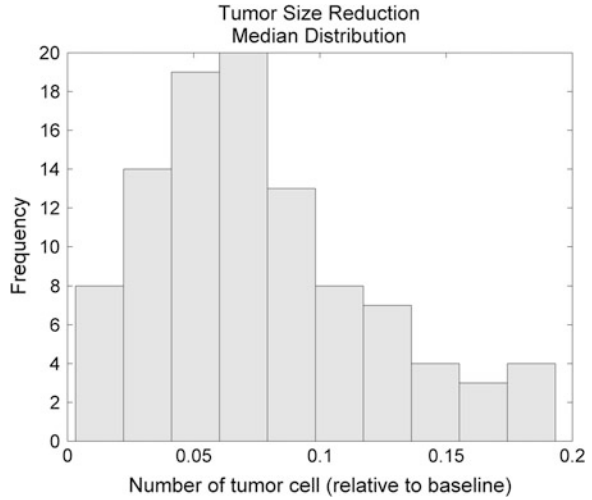
### Exercise 2.3

Determine a distribution of outcomes in tumor size reduction over 100 replicates in 50 patients.

#### Solution

We can utilize the `concmodvector.m` program. It saves `tumorPoints` at `time = 2,016 h` (end of the treatment time) in an external file (`tumsize.STA`) after each replication. The following code shows how to execute 100 replicates in 50 patient (Fig B.2).

**Fig. B.2** The output produced by the code utilizing the `concmovector.m` program for 100 replications in 50 subjects. The histogram shows the distribution of the median for the tumor size reduction, i.e. the rate of number of tumor cells at the end of treatment time related to the baseline value



```
rng default;
for i = 1:100; concmovector('intermittent',50); end
X1 = fscanf(fopen('tumsizes.STA','r'), '%f');
X2 = reshape(X1, 50, 100); Xm = median(X2);
set(axes, 'FontSize', 15)
nx = hist(Xm); hist(Xm); hold on
h = findobj(gca,'Type','patch'); set(h,'FaceColor',[0.9 0.9 0.9])
xlabel('Number of tumor cell (relative to baseline)');
ylabel('Frequency');
title({'Tumor Size Reduction', 'Median Distribution'})
print('-r600', '-dtiff', 'TumorSizeDistr')
```

#### Exercise 2.4

Show that the non-negative function defined by Eq. (2.17) is a density function, i.e.

$$\int_{-\infty}^{\infty} f(x) \, dx = 1 \quad (\text{B.1})$$

Show that this equation is fulfilled, both manually and using MATLAB.

#### Solution

It has to be shown that

$$f(x) = \frac{1}{\sigma \cdot \sqrt{2 \cdot \pi}} \cdot e^{-\frac{(x-\mu)^2}{2 \cdot \sigma^2}} \Rightarrow \frac{1}{\sigma \cdot \sqrt{2 \cdot \pi}} \int_{-\infty}^{\infty} e^{-\frac{(x-\mu)^2}{2 \cdot \sigma^2}} dx = 1 \quad (\text{B.2})$$

We need to calculate

$$P = \frac{1}{\sigma \cdot \sqrt{2} \cdot \pi} \int_{-\infty}^{\infty} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx \quad (\text{B.3})$$

The independent variable of the integrand is  $x$  but we can calculate the same definite integral with regard to variable  $y$ , i.e.:

$$P = \frac{1}{\sigma \cdot \sqrt{2} \cdot \pi} \int_{-\infty}^{\infty} e^{-\frac{(y-\mu)^2}{2\sigma^2}} dy \quad (\text{B.4})$$

Putting together (B.3) and (B.4) we obtain

$$P^2 = \frac{1}{\sigma \cdot \sqrt{2} \cdot \pi} \int_{-\infty}^{\infty} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx \cdot \frac{1}{\sigma \cdot \sqrt{2} \cdot \pi} \int_{-\infty}^{\infty} e^{-\frac{(y-\mu)^2}{2\sigma^2}} dy \quad (\text{B.5})$$

As  $x$  and  $y$  are independent

$$\left. \begin{aligned} P^2 &= \frac{1}{2 \cdot \pi \cdot \sigma^2} \iint_D e^{-\left[\frac{(x-\mu)^2}{2\sigma^2} + \frac{(y-\mu)^2}{2\sigma^2}\right]} dy dx ; \\ D &= \{(x, y) : -\infty < x < \infty ; -\infty < y < \infty\} \end{aligned} \right\} \quad (\text{B.6})$$

Converting the Cartesian coordinate system,  $(x, y)$ , to the polar coordinate system  $(r, \varphi)$ , a new integration region  $\Delta$  will be obtained, where

$$\left. \begin{aligned} x &= \sigma \cdot r \cdot \cos \varphi + \mu \\ y &= \sigma \cdot r \cdot \sin \varphi + \mu \end{aligned} \right\} \Rightarrow \Delta = \{(r, \varphi) : 0 < r < \infty ; 0 < \varphi < 2 \cdot \pi\} \quad (\text{B.7})$$

The Jacobian determinant,  $J$ , (Carl Jacobi, 1804–1851) of the coordinate conversion formula is given by

$$\begin{aligned} J &= \begin{vmatrix} \frac{\partial x}{\partial r} & \frac{\partial x}{\partial \varphi} \\ \frac{\partial y}{\partial r} & \frac{\partial y}{\partial \varphi} \end{vmatrix} = \begin{vmatrix} \sigma \cdot \cos \varphi & -\sigma \cdot r \sin \varphi \\ \sigma \cdot \sin \varphi & \sigma \cdot r \cos \varphi \end{vmatrix} = \sigma^2 \cdot r \cdot (\cos^2 \varphi + \sin^2 \varphi) \\ &= \sigma^2 \cdot r \end{aligned} \quad (\text{B.8})$$

Thus, the integral in polar coordinates is

$$\begin{aligned}
P^2 &= \frac{1}{2 \cdot \pi \cdot \sigma^2} \iint_{\Delta} e^{-(r^2 \cdot \cos^2 \varphi + r^2 \cdot \sin^2 \varphi)} \cdot J \cdot d\varphi \, dr \\
&= \frac{1}{2 \cdot \pi \cdot \sigma^2} \iint_{\Delta} e^{-(r^2 \cdot \cos^2 \varphi + r^2 \cdot \sin^2 \varphi)} \cdot \sigma^2 \cdot r \, d\varphi \, dr \\
&= \frac{1}{2 \cdot \pi \cdot \sigma^2} \int_0^{2\pi} d\varphi \int_0^{\infty} e^{-r^2} \cdot \sigma^2 \cdot r \, dr = \frac{\sigma^2}{2 \cdot \pi \cdot \sigma^2} \cdot \varphi \Big|_0^{2\pi} \cdot \int_0^{\infty} e^{-r^2} \cdot r \, dr \\
&= \frac{\sigma^2}{2 \cdot \pi \cdot \sigma^2} \cdot (2 \cdot \pi - 0) \cdot (-e^{-r^2}) \Big|_0^{\infty} = \frac{\sigma^2}{2 \cdot \pi \cdot \sigma^2} \cdot (2 \cdot \pi - 0) \cdot (0 + 1) = 1 \\
&\Downarrow \\
P &= 1
\end{aligned} \tag{B.9}$$

A similar validation of the density function (B.2) can be carried out with the MATLAB Symbolic Toolbox, as shown in Listing Fig B.3.

### Listing B.3 Program **fdensity.m**

```

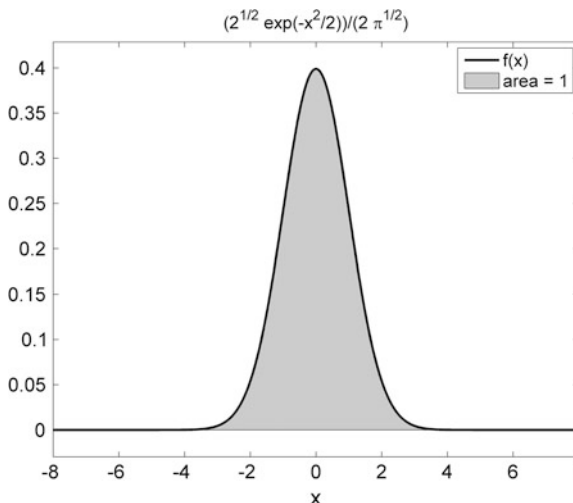
%FDENSITY The density function of the normal distribution.
%  FDENSITY uses the Symbolic Math Toolbox to calculate the area
%  under the density function of the normal distribution and
%  produces a graph based on the analytical form of the function.
%  See also ezplot.

syms f s m x a pis
assume(s>0)
assume(m>0)
m = sym(0);
s = sym(1);
pis = sym(pi);
f = 1/(s*sqrt(2*pis))*exp(-(x-m)^2/s^2/2);
F=int(f,-a,a);
F2 = F*F;
Fa=limit(F,a,inf);
display(['area under the density function = ', char(Fa)])
f = 1/(s*sqrt(2*pis))*exp(-(x-m)^2/s^2/2);
set(axes, 'FontSize', 15)
h = ezplot(char(f), [-8 8]); hold on
xpoints = get(h, 'XData');
ypoints = get(h, 'YData');
set(h, 'Color', 'black', 'LineWidth', 2);
area(xpoints, ypoints, 'face', [0.8 0.8 0.8])
legend('f(x)', ['area = ' char(Fa)])
print('-r600', '-dtiff', ['density_',char(m),'_',char(s)]);

```



**Fig. B.3** The density function  $N(\mu, \sigma^2)$  of a normally distributed variable as produced by the `fdensity.m` program, given  $\mu = 1$  and  $\sigma = 1$ . The calculated value of area under the curve is equal to 1



### Exercise 3.1

Find the analytical solution for the DDE defined by (3.51) with history as in (3.52). Furthermore, write a program to solve this equation numerically by using a MATLAB DDE solver. Finally, incorporate the analytical solution into this program and graphically demonstrate that both solutions are identical.

#### Solution

In order to solve the equation

$$\frac{dx}{dt} = -k \cdot x(t - \tau); \quad t \geq 0 \quad (\text{B.10})$$

with history

$$x(t) = x_0; \quad t \in [-\tau; 0], \quad (\text{B.11})$$

the time domain,  $t \geq 0$ , needs to be divided into intervals of the same length  $\tau$ , i.e. at the beginning a solution has to be found in  $[0; \tau]$ , then  $[\tau; 2 \cdot \tau]$ , ..., etc. Based on (B.10) and (B.11)

$$\left. \begin{array}{l} \frac{dx}{dt} = -k \cdot x(t - \tau); \quad t \in [0; \tau] \\ x(t - \tau) = x_0 \end{array} \right\} \Rightarrow \frac{dx}{dt} = -k \cdot x_0 \quad (\text{B.12})$$

After analytical integration, a general solution of (B.12) is

$$x(t) = -k \cdot x_0 \cdot t + \text{const.} \quad (\text{B.13})$$

This solution must fulfill the boundary condition,  $x(0) = x_0$ , which gives  $\text{const} = x_0$ , and the solution within this interval is equal:

$$x(t) = x_0 \cdot (1 - k \cdot t); \quad t \in [0; \tau] \quad (\text{B.14})$$

Similarly, the solution in the next period,  $[\tau; 2 \cdot \tau]$ , can be found. Based on Eqs. (B.10) and (B.14)

$$\left. \begin{aligned} \frac{dx}{dt} &= -k \cdot x(t - \tau); & t \in [\tau; 2 \cdot \tau] \\ x(t - \tau) &= x_0 \cdot [1 - k \cdot (t - \tau)] \end{aligned} \right\} \Rightarrow \frac{dx}{dt} = k \cdot x_0 \cdot [1 - k \cdot (t - \tau)] \quad (\text{B.15})$$

Equation (B.15) can be integrated analytically, leading to the following solution

$$x(t) = -k \cdot x_0 \cdot t + k^2 \cdot x_0 \cdot \tau \cdot t - \frac{1}{2} \cdot k^2 \cdot x_0 \cdot t^2 + \text{const} \quad (\text{B.16})$$

Again, this solution must fulfill the boundary condition,  $x(\tau) = x_0 \cdot (1 - k \cdot \tau)$ , and after some algebraic simplifications the value *const* can be obtained, i.e.:

$$\text{const} = x_0 \cdot \left(1 - \frac{1}{2} \cdot k^2 \cdot \tau^2\right) \quad (\text{B.17})$$

Substituting *const* in Eq. (B.16) with expression (B.17), the final solution in the current interval can be expressed as follows:

$$x(t) = x_0 \cdot \left[1 - k \cdot t + \frac{k^2}{2} \cdot (t - \tau)^2\right]; \quad t \in [\tau; 2 \cdot \tau] \quad (\text{B.18})$$

The analytic solution is becoming more and more complicated every time while moving to the next period.

To validate the DDE procedure, the obtained solution, Eqs. (B.14, B.15, B.16, B.18), can be now compared with the numerical DDE solver offered by MATLAB. It was done in Listing B.4.

**Listing B.4** Program **DDEconst.m**

```

function DDEconst
%DDECONST Solve a DDE model.
%   DDECONST compares numerical solution vs. analytical solution of
%   a DDE model with a constant history.

tau = 5;
p.x0 = 10;
p.lag = tau;
p.k = 0.1;
tspan = [0 10];
options = ddeset('RelTol',1e-9, 'AbsTol', 1e-9);
sol = dde23(@derivatives, p.lag, @history, tspan, options, p);
% numerical solution
t = sol.x;
x = sol.y;
% ax = axes;
set(axes,'FontSize',15)
plot(t,x,'--b','LineWidth',2); hold on
% numerical solution
x = analyticsol(t,p);
plot(t,x,'+r','LineWidth',2,'MarkerSize',10)
legend('numerical', 'analytical')
title('DDE Solution with Constant History Function');
xlabel('t')
ylabel('x')
print('-r600', '-dtiff', 'ddeconst')
end

function dxdt = derivatives(~, ~, Z, p)
%DERIVATIVES Compute the right-hand side of the DDE.
%   DXDT = ...

% Derivatives vector of the DDE system
xlag = Z(1,1);
dxdt = -p.k*xlag;
end

function xhist = history(~,p)
%HISTORY Provide the history of the solution
%   XHIST = ...

xhist = p.x0;
end

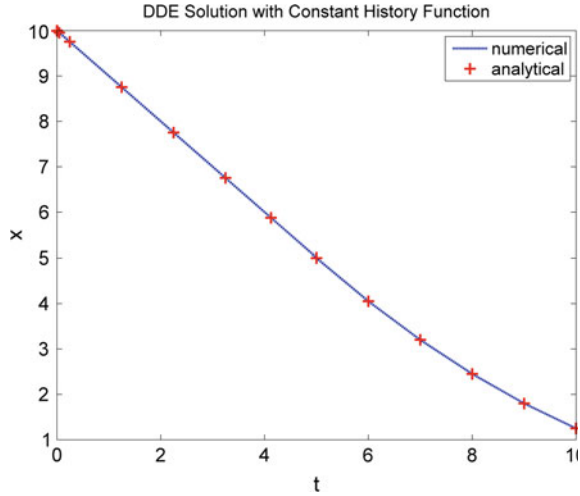
function x = analyticsol(t,p)
%ANALYTICSOL Analytical solution of the DDE
%   X = ANALYTICSOL(T,P) computes the dependent variable |X|, defined
%   at time |T|, and based on the analytical solution with
%   parameters |P|.

x = p.x0*(1-p.k*t) .* (t <= p.lag) + ...
    p.x0*(1 - p.k*t + (p.k^2)/2*(t-p.lag).^2) .* ...
    (p.lag < t & t <= 2*p.lag);
end

```

The output from the program **DDEconst.m** is shown in Fig. B.4.

**Fig. B.4** A Comparison between analytical and numerical solution of Eq. (B.10) with a constant value as the history



### Exercise 3.2

Find the analytical solution for the DDE defined by (3.53) with history as in (3.54). Furthermore, write a program to solve this equation numerically by using a MATLAB DDE solver. Finally, incorporate the analytical solution into this program and graphically demonstrate that both solutions are identical.

#### Solution

In order to solve the equation

$$\frac{dx}{dt} = -k \cdot x(t - \tau); \quad t \geq 0 \quad (\text{B.19})$$

with a history depending linearly on time

$$x(t) = a \cdot t; \quad t \in [-\tau; 0] \quad (\text{B.20})$$

use a similar approach as in Exercise 3.1. The solution for the first two periods is:

$$x(t) = \begin{cases} -k \cdot a \cdot \left( \frac{t^2}{2} - \tau \cdot t \right); & t \in [0; \tau] \\ k^2 \cdot \left[ \frac{(t-\tau)^3}{6} - \frac{(t-\tau)^2}{2} \right] + \frac{k \cdot a}{2}; & t \in [\tau; 2 \cdot \tau] \end{cases} \quad (\text{B.21})$$

In order to validate the DDE procedure, the obtained solution, Eq. (B.21), can be compared with the numerical DDE solver. The comparison is shown in Listing B.5.

Listing B.5 Program **DDElingen.m**

```

function DDElingen
%DDELINGEN Solve a DDE model.
%   DDELINGEN compares numerical solution vs. analytical solution of
%   a DDE model with a linear history function.

tau = 5;
p.a = 1;
p.lag = tau;
p.k = 0.1;
tspan = [0 10];
options = ddeset('RelTol',1e-6, 'AbsTol', 1e-6);
sol = dde23(@derivatives, p.lag, @history, tspan, options, p);
% numerical solution
t = sol.x;
x = sol.y;
% ax = axes;
set(axes, 'FontSize', 15)
plot(t, x, '--b', 'LineWidth', 2); hold on
% numerical solution
x = analytcsol(t, p);
plot(t, x, 'or', 'LineWidth', 2, 'MarkerSize', 10)
legend('numerical', 'analytical')
title('DDE Solution with Linear History Function');
xlabel('t')
ylabel('x')
print('-r600', '-dtiff', 'ddelingen')

end

function dxdt = derivatives(~, ~, Z, p)
%DERIVATIVES Compute the right-hand side of the DDE.
%   DXDT = ...

% Derivatives vector of DDE system

xlag = Z(1,1);
dxdt = -p.k*xlag;

end

function xhist = history(t, p)
%HISTORY Provide the history of the solution
%   XHIST = ...

xhist = p.a*t;

end

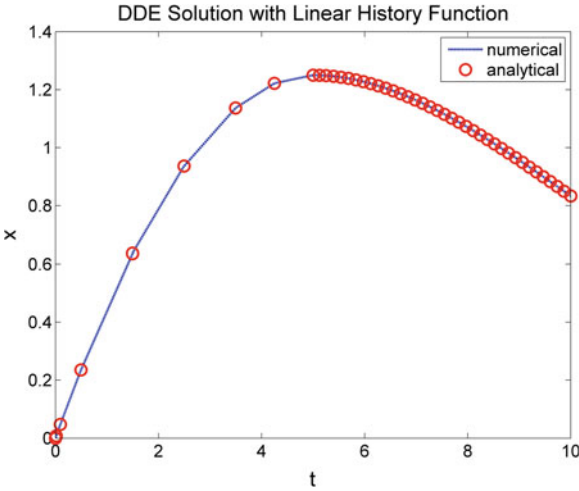
function x = analytcsol(t, p)
%ANALYTICSOL Analytical solution of the DDE
%   X = ...

x = p.k*p.a*(-t.^2/2 + p.lag*t) .* (t < p.lag) + ...
    (p.k^2*p.a*(1/6*(t-p.lag).^3 - 1/2*p.lag*(t-p.lag).^2) ...
    + 1/2*p.k*p.a*p.lag^2) .* (p.lag <= t & t <= 2*p.lag);
end

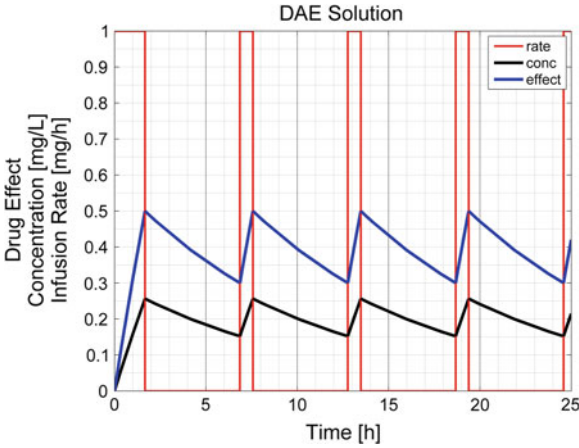
```

Figure B.5 shows the **DDElingen.m** program output.

**Fig. B.5** A comparison between the analytical and numerical solution of (B.19) with a linear function of time as history



**Fig. B.6** The concentration–time course obtained with scheduled infusion rate, according to the rules: if drug effect has increased to 0.5 then stop the infusion; if drug effect  $E$  has decreased to 0.3 then restart the infusion at 1 mg/h. Comparing with Fig. 3.8 in Chap. 3, we can see the identical output but the DAE solver was applied in this case



**Exercise 3.3**

Modify the `emaxevent.m` program considering the same state events:

- State Event: Drug effect  $E$  has increased to 2
- Rule: Stop infusion
- State Event: Drug effect  $E$  has decreased to 1
- Rule: Set infusion to 1 mg/h
- Time Event: Simulation lasts for 25 h
- Rule: Stop simulation.

However, use the DAE solver to find the infusion regimen. Compare the output with Fig. 3.8 that was obtained without a DAE solver.

## Solution

Looking for a solution, we can use the `emaxevent.m` program (already created in Chap. 3) that can be easily modified, as in Listing B.6. The necessary modifications were indicated as **%EXT** (extension in form of new statement) or **%CHA** (changes in existing statement). The output of the newly created program, `emaxeventDAE.m`, is demonstrated in Fig. B.6.

### Listing B.6 Program `emaxeventDAE.m`

```
function emaxeventDAE()
%EMAXEVENTDAE
%   EMAXEVENTDAE() schedules PKPD events, using a DAE solver. The
%   drug effect should be kept within a given range by the infusion
%   rate (control variable) according to the rule: if the drug
%   effect achieves the allowed maximum, then decrease infusion; if
%   the drug effect declines to allowed minimum - infusion has to be
%   increased.

% PKPD parameters
p.R = 1.0;      % infusion rate
p.V = 6;        % volume of distribution
p.k = 0.1;      % elimination rate
p.Emax = 20;     % max effect (in Hill equation)
p.E50 = 10;     % concentration linked to 50% of max effect
% Calling ODE
M = [ 1 0                                %EXT
      0 0 ];                                %EXT
options = odeset('Events', @fevents, 'Mass', M); %EXT
tspan = [0 150];
c0 = 0;        % Initial value for the dependent variable
E0 = hilleffect(c0, 0, p.Emax, p.E50, 1);
ie = 99;
set(axes,'FontSize',14)
while ~isempty(ie)
    [t,c,te,ye,ie] = ode15s(@derivalgeb, tspan, [c0 E0], ... %CHA
                             options,p);
    % Generate graph
    plot(t, p.R*ones(length(t),1),'Color', 'r', 'LineWidth',2)
    hold on;
    plot(t, c(:,1), 'Color', 'k', 'LineWidth', 3);
    hold on;
    plot(t, c(:,2), 'Color', 'b', 'LineWidth', 3) %CHA
    tspan = [te(end) 500];
    c0 = ye(end,1);
    E0 = ye(end,2);
    R1 = p.R; %CHA
    % investigate events
    switch ie(end)
        case 1
            % drug effect E has increased to 2
            % stop infusion
            p.R = 0;
        case 2
            % drug effect E has decreased to 1
            % restart infusion
            p.R = 1; % [mg/h]
        case 3
            % end of observation
            ie = []; % no more events
    end
    R2 = p.R;
    plot([t(end) t(end)], [R1 R2],'Color', 'r', 'LineWidth',2)
end
```

```

grid minor; grid on ;
xlabel('Time [h]');
ylabel({'Drug Effect', 'Concentration [mg/L]', ...
       'Infusion Rate [mg/h]'});
legend('rate','conc','effect')
title('DAE Solution')
% save graphs as TIFF file
print('-r900', '-dtiff', 'emaxeventDAE')

end

function dydt = derivalgeb(~, y, p) %CHA
%DERIVALGEB Compute the right-hand side of the DAE.
% DYDT = ...

dydt = [p.R/p.V - p.k*y(1);
        hilleffect(y(1), 0, p.Emax, p.E50, 1) - y(2)]; %EXT
end

function [value,isterminal,direction] = fevents(t,y,~)
%FEVENTS Define events.
% [VALUE,ISTERMINAL,DIRECTION] = ...

% Event: Drug effect E has increased to the limit (upper) %CHA
value(1,1) = y(2) - 0.5;
isterminal(1,1) = 1; % Stop integration
direction(1,1) = 1; % Positive direction only

% Event: Drug effect E has increased to the limit (lower) %CHA
value(2,1) = y(2) - 0.3;
isterminal(2,1) = 1; % Stop integration
direction(2,1) = -1; % Negative direction only

% Event: End of time integration %CHA
value(3,1) = t - 25 ;
isterminal(3,1) = 1; % Stop integration
direction(3,1) = 1; % Positive direction only

end

function E = hilleffect(c, E0, Emax, EC50, n)
% .....
% .....
% .....

end

```

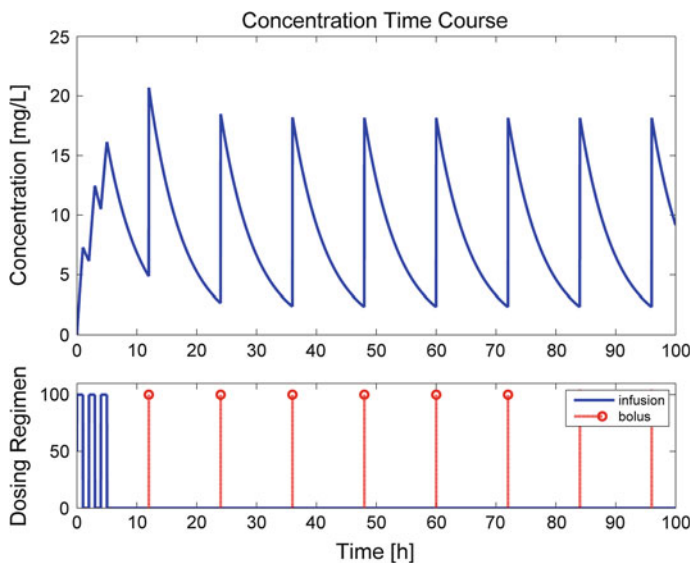
### Exercise 3.4

Create a function that generates the concentration–time profile following three initial infusions and multiple bolus administrations thereafter. Use a one-compartment model with first-order elimination.



### Solution

The solution and result are shown in Listing B.7 and the generated output in Fig. B.7.



**Fig. B.7** Output given by the `laplgraph.m` program: The graph shows the concentration profile response (*upper panel*) to a combined dosing regimen (*lower panel*)

### Listing B.7 Program `laplgraph.m`

```
function laplgraph
%LAPLGRAPH Generate the response to a combined dosing regimen.
% LAPLGRAPH produces the concentration time profile as response
% to a dosing regimen that consist of both infusions and boluses
% in the one-compartment model with first-order elimination

global tbolus
V =12.6;
CL = 2.16;
[y, u] = laplcomodel;
fy = vectorize(inline(char(y)))    %#ok<NOPRT>
fu = vectorize(inline(char(u)))    %#ok<NOPRT>
t = 0:0.01:100;
ffy = fy(CL, V, t);
ffu = fu(t);
subplot(3,1,1:2)
plot(t, ffy, 'LineWidth',2);
title('Concentration Time Course','FontSize',12)
ylabel('Concentration [mg/L]','FontSize',12)
```

```

subplot(3,1,3)
plot(t, ffu, 'LineWidth', 2, 'LineWidth', 2); hold on;
y=(tbolus==tbolus)*100;
stem(tbolus, y, '--r', 'LineWidth', 2);
ylim([0 max(y)*1.1])
xlabel('Time [h]', 'FontSize', 12)
ylabel('Dosing Regimen', 'FontSize', 12)
legend('infusion', 'bolus')
print('-r900', '-dtiff', 'laplgraph');
end

function r = dosing
%DOSING Create a function with the full dosing history.
% R = DOSING specifies a combined administration regimen: amounts.
% and times for infusion or bolus: Dose bolus times, |tbolus|, and
% infusion times, |tinfus|, should be global variables. It returns
% a symbolic function |R| that contains the full dosing history as
% (in analytical form).

syms D R t
global tbolus tinfus

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
D = 200; tbolus = [12 24 36 48 60 72 84 96];
R = 100; tinfus = [0 1; 2 3; 4 5];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
r = 0;
% input of all bolus
for i=1:length(tbolus)
    r = r + D*dirac(t-tbolus(i));
end
% input of all infusions
for i=1:length(tinfus)
    r = r + R*(heaviside(t-tinfus(i,1)) - heaviside(t-tinfus(i,2)));
end
end

function [c,u] = lapl1comodel
%LAPL1COMODEL Solve one-compartment model using Laplace transform.
% [C,U] = LAPL1COMODEL solves one-compartment model equation, using
% the Laplace transform function. Output: |C| - drug concentrations
% as a function of time in analytical form (as response to the
% dosing regimen) and |U| - the dosing as a function of time, in
% analytical form, too.

syms CL V s t c u

Ks = 1/(CL + V*s); % transform function
% specify the input (dosing regimen)
u = dosing;
Us = laplace(u, t, s);
Ys = Ks*Us;
c = ilaplace(Ys, s, t);
display(['c(t) = ' char(c)]);
pretty(c);

end

```

**Exercise 3.5**

Find drug concentrations  $c(t)$  after a bolus dose of 200 mg using the transform function  $K(s)$  of the one-compartment model.

**Solution**

Knowing that the transfer function  $K(s)$  for the one-compartment model exists, i.e.:

$$K(s) = \frac{1}{CL + V \cdot s} \quad (\text{B.22})$$

the input  $u(t)$  is a bolus, meaning

$$u(t) = Dose \cdot \delta(t) \quad (\text{B.23})$$

where  $\delta(\cdot)$  is the delta Dirac function. The Laplace  $U(s)$  transform of the bolus  $u(t)$  is equal:

$$U(s) = \mathcal{L}\{Dose \cdot \delta(t)\} = Dose \cdot \mathcal{L}\{\delta(t)\} = Dose \cdot 1 = Dose \quad (\text{B.24})$$

The concentration function,  $c(t)$ , that we are looking for can be now expressed in Laplace  $C(s)$  form as follows:

$$C(s) = U(s) \cdot K(s) \quad \Leftrightarrow \quad C(s) = Dose \cdot \frac{1}{V} \cdot \frac{1}{s + \frac{CL}{V}}. \quad (\text{B.25})$$

The inverse transform of  $C(s)$  gives the solution  $c(t)$ :

$$c(t) = \mathcal{L}^{-1}\{C(s)\} = \mathcal{L}^{-1}\left\{Dose \cdot \frac{1}{V} \cdot \frac{1}{s + \frac{CL}{V}}\right\} = \frac{Dose}{V} \cdot e^{-\frac{CL}{V}t} \cdot H(t) \quad (\text{B.26})$$

Putting  $Dose = 200$  mg,  $V = 12.6$  L, and  $CL = 2.16$  L/h, the concentration can be computed as follows:

$$c(t) = 15.873 \cdot e^{-0.1714t} ; \quad t \geq 0 \quad (\text{B.27})$$

This is the final solution.

**Exercise 3.6**

Solve the same problem as in Exercise 3.5, but using MATLAB routines for Laplace transformations.

**Solution**

The solution is shown in Listing B.8.

**Listing B.8** Program `lap11combolus.m`

```

function c = lap11combolus()
%LAPL1COMBOLUS One-compartment model with bolus - Laplace solution.
% C = LAPL1COMBOLUS() solves the equation dcdt = r/V - k*c using
% first Laplace transform and then inverse Laplace transform.
% The result is a function |C| = f(t) in analytical form

% declare symbolic variables
syms D CL V s t k dcdt Lc c r Lr
% differential equation in analytic form (symbols)
dcdt = sym('diff(c(t),t)');
c = sym('c(t)');
r = sym('r(t)');
equation = dcdt - r/V + CL/V*c;
transformL = laplace(equation, t, s);
% simplify the Laplace form of the equation
transformL = subs(transformL, {'c(0)', 'laplace(c(t), t, s)', ...
    'laplace(r(t), t, s)'}, {0, Lc, Lr}); %ok<NODEF>
transformL = collect(collect(transformL, V), Lc);
% solution as Laplace transform Lc(s)
Lc = solve(transformL, Lc);
Ks = Lc/Lr; % transform function
% specify the input (dosing)
% r = R*(heaviside(t) - heaviside(t-2)); % valid for infusion
r = D*dirac(t); % valid for bolus
Us = laplace(r, t, s);
Ys = Ks*Us;
c = ilaplace(Ys, s, t);
display(['c(t) = ' char(c)]);
pretty(c);

end

```

The `lap11combolus.m` program gives as output:

$$c = (D \cdot \exp(-(CL \cdot t)/V)) / V$$

that is equivalent to the solution expressed in Eq. (B.26).

**Exercise 3.7**

Show that the equation for the one-compartment model with infusion and first-order elimination, i.e.

$$c(t) = \frac{Rate}{CL} \cdot [(1 - e^{-\frac{CL}{V}t}) \cdot H(t) - (1 - e^{-\frac{CL}{V}(t-t_R)}) \cdot H(t - t_R)] \quad (\text{B.28})$$

approaches the analytical solution for a one-compartment model with bolus input that is given by

$$c(t) = \frac{Dose}{V} \cdot e^{-\frac{CL}{V}t} \cdot H(t) \quad (\text{B.29})$$

if the infusion duration  $t_R$  approaches 0.

**Solution**

According to the formula (B.28) and knowing that  $Dose = Rate \cdot t_R$  we have:

$$c(t) = \frac{Dose}{CL} \cdot \frac{1}{t_R} \cdot [(1 - e^{-\frac{CL}{V}t}) \cdot H(t) - (1 - e^{-\frac{CL}{V}(t-t_R)}) \cdot H(t-t_R)] \quad (B.30)$$

Taking a closer look at this function we can notice that the concentration  $c(t)$  has a form of following type:

$$c(t) = \frac{Dose}{CL} \cdot \frac{f(t) - f(t-t_R)}{t_R} ; \quad f(t) = (1 - e^{-\frac{CL}{V}t}) \cdot H(t) \quad (B.31)$$

The infusion time approaches zero,  $t_R \rightarrow 0$ , thus

$$c(t) = \lim_{t_R \rightarrow 0} \frac{Dose}{CL} \cdot \frac{f(t) - f(t-t_R)}{t_R} = \frac{Dose}{CL} \cdot \lim_{t_R \rightarrow 0} \frac{f(t) - f(t-t_R)}{t_R} \quad (B.32)$$

However, the limiting value (lim) covers the definition of the function derivative,

$$\lim_{t_R \rightarrow 0} \frac{f(t) - f(t-t_R)}{t_R} \stackrel{\text{def}}{=} \frac{df}{dt} ; \quad t > 0 \quad (B.33)$$

On the other hand, the derivative is equal to:

$$\begin{aligned} \frac{df}{dt} &= \frac{dH}{dt} + \frac{CL}{V} \cdot e^{-\frac{CL}{V}t} \cdot H(t) - e^{-\frac{CL}{V}t} \cdot \frac{dH}{dt} \\ &= \delta(t) + \frac{CL}{V} \cdot e^{-\frac{CL}{V}t} \cdot H(t) - e^{-\frac{CL}{V}t} \cdot \delta(t) \end{aligned} \quad (B.34)$$

where  $\delta(t)$  is Dirac's delta function [1]. Hence

$$\frac{df}{dt} = \delta(t) + \frac{CL}{V} \cdot e^{-\frac{CL}{V}t} \cdot H(t) - e^{-\frac{CL}{V}0} \cdot \delta(t) = \frac{CL}{V} \cdot e^{-\frac{CL}{V}t} \cdot H(t) \quad (B.35)$$

and putting it into Eq. (B.32), we obtain

$$\left. \begin{aligned} c(t) &= \frac{Dose}{CL} \cdot \frac{df}{dt} \\ \frac{df}{dt} &= \frac{CL}{V} \cdot e^{-\frac{CL}{V}t} \cdot H(t) \end{aligned} \right\} \Rightarrow c(t) = \frac{Dose}{V} \cdot e^{-\frac{CL}{V}t} \cdot H(t) \quad (B.36)$$

This final form is exactly the one from Eq. (B.29) we wanted to get.

**Exercise 3.8**

Using a one-compartment model with bolus input, show that the solution of the differential equation describing such a model is the same both for dose handled as the initial value or as an input value.

**Solution**

The model equation derived from mass balance is

$$\frac{da}{dt} = -k \cdot a ; \quad a(0) = Dose \quad (B.37)$$

and the Laplace transform of the equation gives

$$A(s) - a(0) = -k \cdot A(s) \quad \Leftrightarrow \quad A(s) - Dose = -k \cdot A(s) \quad (\text{B.38})$$

after a simple rearrangement

$$A(s) - 0 = -k \cdot A(s) + Dose \quad (\text{B.39})$$

Now, we assume that the initial value  $a(0) = 0$ . Then the invert Laplace transform is

$$\frac{da}{dt} = -k \cdot a + Dose \cdot \delta(t); \quad a(0) = 0 \quad (\text{B.40})$$

Of course, the solution  $a(t)$  (or as concentration  $c(t)$ ) of (B.40) is the same as in (B.37), i.e.

$$a(t) = Dose \cdot e^{-k \cdot t} \cdot H(t) \quad \text{or} \quad c(t) = \frac{Dose}{V} \cdot e^{-\frac{CL}{V} \cdot t} \cdot H(t) \quad (\text{B.41})$$

See also Eq. (B.26), where the elimination rate constant  $k = CL/V$ .

#### Exercise 4.1

Show that  $V_{\text{app}}$  is time-independent if ratios of drug concentrations in the body between any two tissues or fluids are constant over time

#### Solution

$V_{\text{app}}$  is defined as the ratio of drug amount in the body to drug concentrations in plasma (blood, serum). Assume that the amount of drug in the body can be expressed as

$$a(t) = \sum_i c_i(t) \cdot V_i \quad (\text{B.42})$$

where summation is taken over all drug-containing compartments. If we divide this equation by  $c_p$ , the drug concentration in plasma, we get

$$V_{\text{app}}(t) = \sum_i \frac{c_i(t)}{c_p(t)} \cdot V_i \quad (\text{B.43})$$

which is independent of time according to our assumption that the ratios  $c_i(t)/c_p(t)$  are time-independent.

#### Exercise 4.2

Show that first-order kinetics implies dose proportionality and that the superposition principle holds.

#### Solution

Dose proportionality is given if drug concentrations at any time  $t$  scale with  $Dose$ . If the kinetics is first order, drug amounts can be expressed as

$$\frac{d\mathbf{m}}{dt} = \mathbf{A} \cdot \mathbf{m}, \quad \mathbf{m}(0) = [Dose \ 0 \ \dots \ 0]^T \quad (\text{B.44})$$

with a matrix  $\mathbf{A}$  containing the rate constants between the compartments. If  $\mathbf{m}_1$  and  $\mathbf{m}_2$  are solutions of (B.44) with initial values  $\mathbf{m}_1(0) = [Dose_1, 0, \dots, 0]^T$  and  $\mathbf{m}_2(0) = [Dose_2, 0, \dots, 0]^T$ , then any linear combination  $\mathbf{m} = a_1 \cdot \mathbf{m}_1 + a_2 \cdot \mathbf{m}_2$  of  $\mathbf{m}_1$  and  $\mathbf{m}_2$  is a solution with initial value  $\mathbf{m}(0) = [a_1 \cdot Dose_1 + a_2 \cdot Dose_2, 0, \dots, 0]^T$ . This is the superposition principle. If we set  $a_2$  to zero,  $a_1 \cdot \mathbf{m}_1$  is a solution with initial condition  $a_1 \cdot Dose_1$ , thus it scales with  $Dose$ .

An immediate consequence of linear kinetics is that the maximum concentration,  $C_{\max}$ , scales with  $Dose$  and that the time of  $C_{\max}$ ,  $t_{\max}$ , is independent of  $Dose$ . Furthermore, also  $AUC$  scales with  $Dose$  as

$$AUC_{Dose_1} = \int c_{Dose_1}(t) dt = \frac{Dose_1}{Dose_2} \int c_{Dose_2}(t) dt = \frac{Dose_1}{Dose_2} \cdot AUC_{Dose_2} \quad (\text{B.45})$$

### Exercise 4.3

Show that dose-proportionality does not imply first-order kinetics.

#### Solution

Consider the following non-linear absorption function, described by [2]

$$a(t) = F \cdot Dose \cdot \lambda_a \cdot s \cdot (t - t_{\text{lag}})^{s-1} \cdot e^{-\lambda_a \cdot (t - t_{\text{lag}})^s} \quad (\text{B.46})$$

If this absorption function is coupled to a one-compartment model with first-order elimination, we achieve dose-proportional concentration profiles but the system is not of first-order.

### Exercise 4.4

Describe the one-compartment model with IV bolus input and zero order elimination.

#### Solution

For a zero-order elimination model, the time profile of the plasma concentration is linear. This means that the same amount of drug is eliminated in every time unit. Mathematically, the mass balance can be described by the form:

$$\frac{da}{dt} = -k \quad \Leftrightarrow \quad \frac{dc}{dt} = -\frac{k}{V} \quad (\text{B.47})$$

with  $c$  as drug concentration,  $a$ —drug amount, and  $a = V \cdot c$ ; where  $V$ —volume of distribution. If we consider a bolus input of amount  $a_0$ , then this amount can be considered at once as the initial condition:

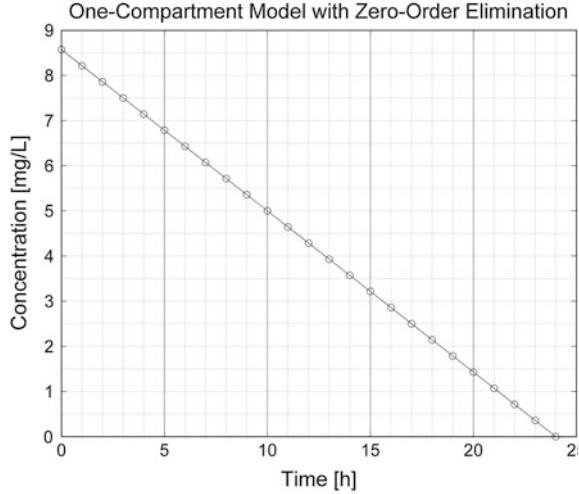
$$a(0) = Dose \quad (\text{B.48})$$

Solving ODE (B.47) with initial condition (B.48) gives:

$$c(t) = -\frac{k}{V} \cdot t + \frac{Dose}{V}; \quad 0 \leq t \leq \frac{Dose}{k} \quad (\text{B.49})$$

Figure B.8 illustrates the change in drug concentration with time for the model with zero-order elimination, assuming that 2.5 mg drug is eliminated every hour, the given bolus is 60 mg, and the volume of distribution is equal to 7 L.

**Fig. B.8** The model with zero-order elimination. The concentration–time after the IV bolus is linear; the graph is produced for  $k = 2.5$  mg/h,  $V = 7$  L,  $a_0 = \text{Dose} = 60$  mg



### Exercise 4.5

Write MATLAB code for multiple oral administrations into a one-compartment model with first-order elimination. Show numerically that the superposition principle holds.

### Solution

We need to apply the model (4.37)

$$c_1(t) = \frac{F \cdot \text{Dose}}{V_1} \cdot \frac{k_{01}}{k_{01} - k_{10}} \cdot (e^{-k_{10} \cdot t} - e^{-k_{01} \cdot t}) \quad (\text{B.50})$$

but for multiple doses,  $\{\text{Dose}_1, \text{Dose}_2 \dots \text{Dose}_n\}$ , given at times  $\{t_1, t_2 \dots t_n\}$ . This gives rise to an extended form of the above equation, i.e.:

$$c_n(t) = \sum_{i=1}^n \frac{F \cdot \text{Dose}_i}{V_1} \cdot \frac{k_{01}}{k_{01} - k_{10}} \cdot [e^{-k_{10} \cdot (t-t_i)} - e^{-k_{01} \cdot (t-t_i)}] \cdot H(t - t_i) \quad (\text{B.51})$$

One can rewrite (B.51) to a recursive form:

$$\left. \begin{aligned} c_1(t) &= \frac{F \cdot \text{Dose}_1}{V_1} \cdot \frac{k_{01}}{k_{01} - k_{10}} \cdot [e^{-k_{10} \cdot (t-t_1)} - e^{-k_{01} \cdot (t-t_1)}] \cdot H(t - t_1) \\ c_n(t) &= c_{n-1}(t) + \frac{F \cdot \text{Dose}_n}{V_1} \cdot \frac{k_{01}}{k_{01} - k_{10}} \cdot [e^{-k_{10} \cdot (t-t_n)} - e^{-k_{01} \cdot (t-t_n)}] \cdot H(t - t_n) \end{aligned} \right\} \quad (\text{B.52})$$

The recursive form of the model can be easily implemented in MATLAB if we use a recursive function specification for this model, as shown in Listing B.9.



**Listing B.9** Program `multiordose.m`

```

function multiordose()
%MULTIORDOSE One-compartment model with multiple oral administration.
% MULTIORDOSE() calculates and produces a graph with concentration-
% time profiles for a one-compartment model with first-order drug
% absorption and first-order elimination, and multiple dose oral
% administration. Additionally, the superposition principle is
% illustrated with the plasma concentration profiles associated
% with each individual dose within the multiple oral
% administration.

p.k01 = 0.5;    % 1/h
p.V = 1.0;     % L
p.CL = 2.0;    % L/h
p.F = 1;       % L/h
tdose = [0 1 2 3 4];
n = length(tdose);
Dose = 200*p.F*ones(1,n); % mg
timeEnd = 5;    % observation time
nd = 5;         % number of doses
% Compute drug concentration directly
ti = [tdose timeEnd]; % add time after the last dose
tPoints = []; cPoints = [];
m0 = 0; c0 = 0;
for i = 1:nd
    m0 = m0 + Dose(i);
    tspan = [ti(i) ti(i+1)];
    [t,y] = ode45(@derivatives, tspan, [m0; c0], [], p);
    tPoints = [tPoints; t]; % #ok<AGROW>
    cPoints = [cPoints; y(:,2)]; % #ok<AGROW>
    m0 = y(end,1); c0 = y(end,2);
end
% Graph preparation
ax = axes;
set(ax,'FontSize',15)
plot(tPoints, cPoints, 'o','LineWidth',1.5, 'Color', 'Black');
hold on; grid on
title({'Superposition Principle', ...
    'One-Compartment Model with Oral Administration'})
ylabel('Drug concentration [mg/L]')
xlabel('Time [h]')
% Compute drug concentration using superposition principle (overall)
t = (0:0.01:timeEnd)';
c1 = ansoln(t, p, nd, Dose, tdose);
plot(t,c1,'LineWidth',3,'Color','red');
% Compute drug concentrations associated with doses
for idose = 1:nd
    ci = ansoln(t, p, 1, Dose(idose), tdose(idose));
    plot(t,ci,'LineWidth', 2);
end
legend('concentration - direct', ...
    'overall concentration - superposition', ...
    'dose associated concentrations', 'Location', 'NorthWest')
print('-dtiff','-r900','multiordose')
end

```

```

function c1 = ansoln(t, p, n, Dose, tdose)
%ANSOLN Apply the the superposition principle.
% C1 = ANSOLN(T, P, N, DOSE, TDOSE) computes the concentration
% values at time points |T|, with model parameters |P|. Dose
% administration is defined by the vectors of dose amounts |DOSE|,
% and dose times |TDOSE|, both of length |N|. The function is
% recursive, i.e., it calls itself as long as |N| > 1.

k01 = p.k01;
k10 = p.CL/p.V;
F = p.F;
dt = t - tdose(n);
D = Dose(n);
if n == 1
    c1 = F*D*k01/(k01-k10)*(exp(-k10*dt)-exp(-k01*dt)).*heaviside(dt);
else
    c1 = ansoln(t, p, n-1, Dose, tdose) + ...
        F*D*k01/(k01-k10)*(exp(-k10*dt)-exp(-k01*dt)).*heaviside(dt);
end
end

function dydt = derivatives(~, y, p)
%DERIVATIVES Compute the right-hand side of the ODE.
% DYDT = ...

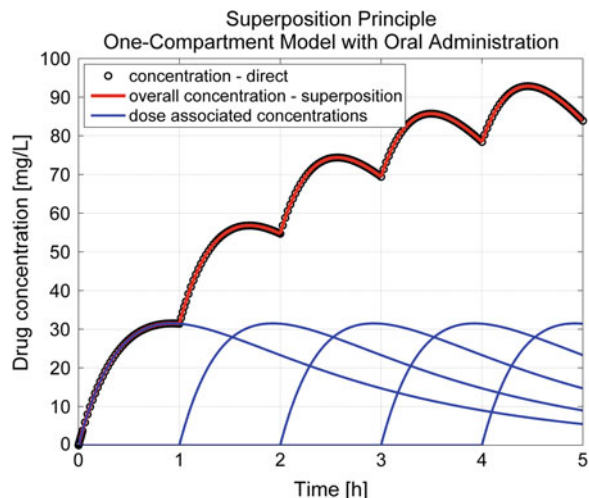
dy1dt = - p.k01*y(1);
dy2dt = p.k01*y(1) - p.CL/p.V*y(2);
dydt = [dy1dt; dy2dt];

end

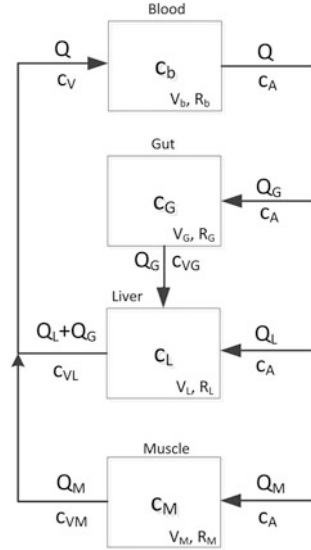
```

Figure B.9 shows the output generated by **multiordose.m**. It illustrates the time course of the plasma concentration that, according to the superposition principle, is simple a sum of all (in this case 5) plasma concentration values linked to the individual doses.

**Fig. B.9** The output of the **multiordose.m** program: within the treatment time of 5 h, 5 doses were given, 200 mg every hour. The overall drug concentration is the sum of concentrations profiles associated with the doses according to the superposition principle. The graph also shows the drug concentration computed directly with an ODE solver; it is identical to the overall concentration, thus confirming the correctness of the superposition method used



**Fig. B.10** Basic PBPK model including drug absorption:  $c_A$  ( $c_{VG}$ ,  $c_{VL}$ ,  $c_{VM}$ ,  $c_V$ )—drug concentration in the arterial (venous) system;  $Q$ ,  $Q_G$ ,  $Q_L$ ,  $Q_M$ —blood flow through the organs with volume  $V_B$ ,  $V_G$ ,  $V_L$ ,  $V_M$  and partition coefficient  $R_b$ ,  $R_G$ ,  $R_L$ ,  $R_M$ , respectively;  $c_b$ ,  $c_G$ ,  $c_L$ ,  $c_M$ —drug concentrations in blood and organs (B - blood, G - gut, L - liver, M - muscle)



#### Exercise 4.6

Construct a PBPK model comprising blood, muscle, liver, and gut. Develop equations and MATLAB code.

#### Solution

According to Fig. B.10, the following equations describe this model:

$$\left. \begin{aligned}
 V_b \cdot \frac{dc_b}{dt} &= Q \cdot (c_V - c_A); & c_b(0) &= 0 \\
 V_G \cdot \frac{dc_G}{dt} &= Q_G \cdot (c_A - c_{VG}); & c_G(0) &= 10 \\
 V_L \cdot \frac{dc_L}{dt} &= Q_L \cdot c_A + Q_G \cdot c_{VG} - (Q_L + Q_G) \cdot c_{VL}; & c_L(0) &= 0 \\
 V_M \cdot \frac{dc_M}{dt} &= Q_M \cdot c_A - Q_M \cdot c_{VM}; & c_L(0) &= 0 \\
 c_A &= c_b, \quad c_{VG} = \frac{c_G}{R_G}, \quad c_{VL} = \frac{c_L}{R_L}, \quad c_{VM} = \frac{c_M}{R_M} \\
 c_V &= \frac{(Q_L + Q_G) \cdot c_{VL} + Q_M \cdot c_{VM}}{Q}; \quad Q = Q_L + Q_G + Q_M
 \end{aligned} \right\} \quad (B.53)$$

As we are going to use a DAE solver (ODE with option **Mass**), the mass matrix has to be derived from (B.53). It is specified (array **M**) in the MATLAB implementation **pbpk4DAE.m**, as shown in Listing B.10. Output is demonstrated in Fig. B.11. This graph contains only the concentration–time course in plasma and included organs. However, thanks to the applied DDE solver, the solution vector **y(:,5:9)** also contains the time profile of  $c_A$  (and  $c_{VG}$ ,  $c_{VL}$ ,  $c_{VM}$ ,  $c_V$ ) for drug concentration in the arterial (and venous) system. Putting **y** rather than **y(:,5:1)** in the **plot** function also reveals the other curves.

**Listing B.10** Program **pbpk4DAE.m**

```

function pbpk4DAE
%PBPK4DAE Solve a 3-organ PBPK model.
% PBPK4DAE calculates the time courses of the drug concentration
% in blood and in a PBPK model representing blood and 3 organs
% (gut, liver, muscle), and produces the respective graphs.
% The DAE solver is applied.

% flows (gut, liver, muscle)
    p.QG = 2;    p.QL = 1;    p.QM = 1;
% volumes (blood, gut, liver, muscle)
p.Vb = 5;  p.VG = 4;    p.VL = 4;    p.VM = 4;
% partition coefficients (gut, liver, muscle)
    p.RG = 0.5;  p.RL = 2;    p.RM = 0.5;
% initial values for organs (blood, gut, liver, muscle)
cbi = 0;  cGi = 10;  cLi = 0;  cMi = 0;
% drug concentration for arterials (A) and venous (V)
cAi = cbi;  cVGi = cGi/p.RG;  cVLi = cLi/p.RL;  cVMi = cMi/p.RM;
cVi = ((p.QL + p.QG)*cVLi + p.QM*cVMi)/(p.QL + p.QG + p.QM);
% vector of initial values
y0 = [cbi; cGi; cLi; cMi; cAi; cVGi; cVLi; cVMi; cVi];
% specify Mass matrix
M = [ p.Vb 0 0 0 0 0 0 0 0
      0 p.VG 0 0 0 0 0 0 0
      0 0 p.VL 0 0 0 0 0 0
      0 0 0 p.VM 0 0 0 0 0
      0 0 0 0 0 0 0 0 0
      0 0 0 0 0 0 0 0 0
      0 0 0 0 0 0 0 0 0
      0 0 0 0 0 0 0 0 0 ];
options = odeset('Mass', M, 'RelTol', 1e-9, 'AbsTol', 1e-9);
timeEnd = 24; tspan = [0 timeEnd];
% p.CL = 1;
[t, y] = ode15s(@derivalgeb, tspan, y0, options, p);
set(axes, 'FontSize', 15); plot(t, y(:, 1:4), 'LineWidth', 2);
hold on
xlabel('Time [h]'); ylabel('Drug Concentration [mg/L]');
legend('Blood', 'Gut', 'Liver', 'Muscle'); title('PKPB Model')
print('-dtiff', '-r900', 'pbpk4DAE.tif')

end

function dydt = derivalgeb(~, y, p)
%DERIVALGEB Compute the right-hand side of the DAE.
% DYDT = ...

% drug concentrations
cb = y(1); % blood
cG = y(2); % gut
cL = y(3); % liver
cM = y(4); % muscle
% drug concentration in the arterial (A) and venous (V) systems.
cA = y(5); %
cVG = y(6); % gut

```

```

cVL = y(7);          % liver
cVM = y(8);          % muscle
cV = y(9);           % muscle
% Mass balance
Q = p.QL + p.QG + p.QM;
% rout = p.CL*c2;
% differential equations
dcbdt = Q*(cV - cA);
dcGdt = p.QG*(cA - cVG);
dcLdt = p.QL*cA + p.QG*cVG - (p.QL + p.QG)*cVL;
dcMdt = p.QM*cA - p.QM*cVM;
% algebraic equations
dcAdt = cA - cb;
dcVGdt = cVG - cG/p.RG;
dcVLdt = cVL - cL/p.RL;
dcVMdt = cVM - cM/p.RM;
dcVdt = cV - ((p.QL + p.QG)*cVL + p.QM*cVM)/Q;
% vector DAE = DE and AE
dydt = [dcbdt; dcGdt; dcLdt; dcMdt; ...
        dcAdt; dcVGdt; dcVLdt; dcVMdt; dcVdt];
end

```

### Exercise 4.7

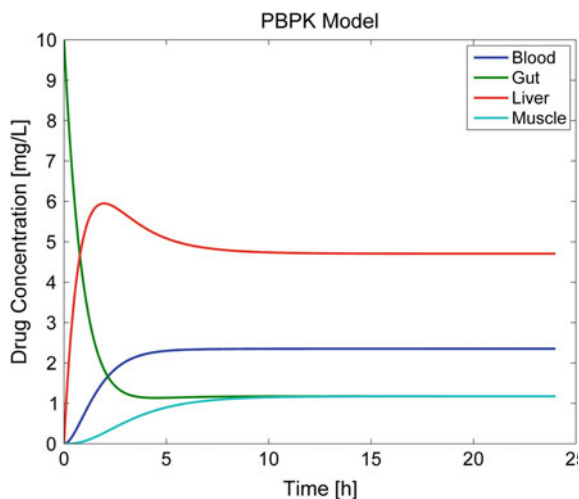
Derive the equation for equilibrium concentrations for the reaction  $D + R \leftrightarrow DR$  under the assumption that total drug and total receptor are conserved.

### Solution

The reaction  $D + R \leftrightarrow DR$  is described by the following equation (see (4.46))

$$\frac{d}{dt}[DR] = k_{\text{on}} \cdot [D] \cdot [R] - k_{\text{off}} \cdot [DR] \quad (\text{B.54})$$

**Fig. B.11** PKPB model including blood, gut, liver and muscle: the output demonstrates time course of drug in plasma and included organs



If both  $D$  and  $R$  are conserved, we have

$$[D] = [D0] - [DR] \quad \text{and} \quad [R] = [R0] - [DR] \quad (\text{B.55})$$

where  $[D0]$  and  $[R0]$  are the total drug and receptor concentrations, respectively. Substituting (B.55) into (B.54), we get

$$\frac{d}{dt}[DR] = k_{\text{on}} \cdot ([D0] - [DR]) \cdot ([R0] - [DR]) - k_{\text{off}} \cdot [DR] \quad (\text{B.56})$$

At equilibrium  $\frac{d}{dt}[DR]_{\text{ss}} = 0$  and (B.54) can be solved for  $[DR]_{\text{ss}}$ :

$$[DR]_{\text{ss}} = 0.5 \cdot ([R0] + [D0] + K_D - \sqrt{([R0] + [D0] + K_D)^2 - 4 \cdot [R0] \cdot [D0]}) \quad (\text{B.57})$$

where  $K_D = K_{\text{off}}/K_{\text{on}}$ . The equilibrium concentrations for  $R$  and  $D$  are given by

$$[D]_{\text{ss}} = [D0] - [DR]_{\text{ss}} \quad \text{and} \quad [R]_{\text{ss}} = [R0] - [DR]_{\text{ss}} \quad (\text{B.58})$$

### Exercise 4.8

Calculate the time-dependent solution for  $[DR]$  in Exercise 4.7.

#### Solution

Equation (B.56) can be simplified by using the transformation  $\tau = k_{\text{on}} \cdot t$ . Setting  $y = [DR]$  we arrive at

$$\frac{dy}{d\tau} = y^2 - 2 \cdot a \cdot y + b^2 \quad (\text{B.59})$$

which is a Riccati-type ODE (Jacopo Riccati, 1676–1754) with constant coefficients and initial condition  $y(0) = 0$ . To solve this equation, a particular solution needs to be identified first. Obviously, solutions of the quadratic term,  $y_1$  and  $y_2$ , are particular (time independent) solutions of Eq. (B.59).

$$y_1 = a + \sqrt{a^2 - b^2}, \quad y_2 = a - \sqrt{a^2 - b^2} \quad (\text{B.60})$$

Any solution  $y$  of Eq. (B.59) can then be expressed as

$$y = y_1 + 1/z \quad (\text{or} \quad y = y_2 + 1/z) \quad (\text{B.61})$$

where  $z$  satisfies the following linear differential equation:

$$\frac{dz}{d\tau} = -2 \cdot (y_1 - a) \cdot z - 1, \quad z(0) = \frac{1}{y(0) - y_1} \quad (\text{B.62})$$

This can easily be verified by calculating the derivative of

$$z = \frac{1}{y - y_1} \quad (\text{B.63})$$

$$\begin{aligned}
\frac{dz}{d\tau} &= -\frac{y' - y'_1}{(y - y_1)^2} = -\frac{y^2 - y_1^2 - 2 \cdot a \cdot (y - y_1)}{(y - y_1)^2} \\
&= -\frac{y - y_1 + 2 \cdot y_1 - 2 \cdot a}{y - y_1} = -2 \cdot (y_1 - a) \cdot z - 1
\end{aligned} \tag{B.64}$$

Equation (B.62) can be readily solved. A linear differential equation

$$\frac{dz}{d\tau} = -A \cdot y + B \quad z(0) = z_0 \tag{B.65}$$

has the solution

$$z = \frac{B}{A} + \left(z_0 - \frac{B}{A}\right) \cdot e^{-A \cdot \tau} \tag{B.66}$$

Thus,

$$\begin{aligned}
z &= \frac{-1}{2 \cdot (y_1 - a)} + \left[ \frac{-1}{y_1} + \frac{1}{2 \cdot (y_1 - a)} \right] \cdot e^{-2 \cdot (y_1 - a) \cdot \tau} \\
&= \frac{-1}{y_1 - y_2} + \left[ \frac{-1}{y_1} + \frac{1}{y_1 - y_2} \right] \cdot e^{-(y_1 - y_2) \cdot \tau} \\
&= \frac{-y_1 + y_2 \cdot e^{-(y_1 - y_2) \cdot \tau}}{y_1 \cdot (y_1 - y_2)}
\end{aligned} \tag{B.67}$$

The solution  $y$  is calculated as follows

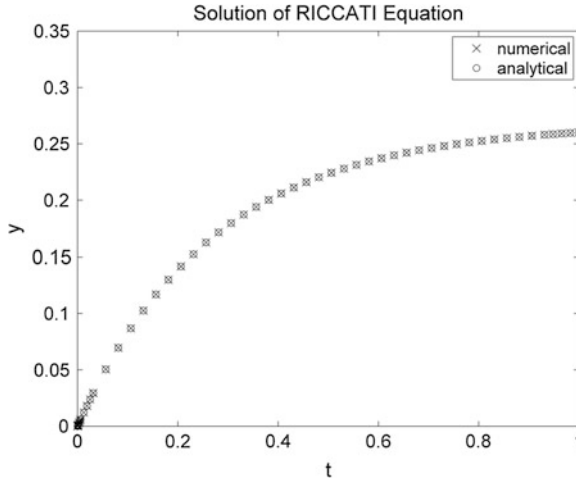
$$y = y_1 + \frac{1}{z} = y_1 + \frac{1}{\frac{-y_1 + y_2 \cdot e^{-(y_1 - y_2) \cdot \tau}}{y_1 \cdot (y_1 - y_2)}} = \frac{y_1 \cdot y_2 \cdot (-1 + e^{-(y_1 - y_2) \cdot \tau})}{-y_1 + y_2 \cdot e^{-(y_1 - y_2) \cdot \tau}} \tag{B.68}$$

Applying the following identities in the denominator

$$\begin{aligned}
y_1 &= \frac{y_1 + y_2}{2} + \frac{y_1 - y_2}{2} \\
y_2 &= \frac{y_1 + y_2}{2} - \frac{y_1 - y_2}{2}
\end{aligned} \tag{B.69}$$

we finally get

**Fig. B.12** The output of `bindingriccati.m` program (Listing B.11) as a graphical comparison of two solutions of the Riccati equation: one obtained with the formula in Eq. (B.71), the other the numerical solution obtained with an ODE solver. The presented solution is applied for building mathematical models of immunoradiometric assays [3]



$$\begin{aligned}
 y &= \frac{y_1 \cdot y_2 \cdot (-1 + e^{-(y_1 - y_2) \cdot \tau})}{\frac{y_1 + y_2}{2} \cdot (-1 + e^{-(y_1 - y_2) \cdot \tau}) - \frac{y_1 - y_2}{2} \cdot (1 + e^{-(y_1 - y_2) \cdot \tau})} \\
 &= \frac{y_1 \cdot y_2}{\frac{y_1 + y_2}{2} + \frac{y_1 - y_2}{2} \cdot \frac{1 + e^{-(y_1 - y_2) \cdot \tau}}{1 - e^{-(y_1 - y_2) \cdot \tau}}} \\
 &= \frac{y_1 \cdot y_2}{\frac{y_1 + y_2}{2} + \frac{y_1 - y_2}{2} \cdot \coth[(y_1 - y_2) \cdot \tau]} \\
 &= \frac{b^2}{a + \sqrt{a^2 - b^2} \cdot \coth(\sqrt{a^2 - b^2} \cdot \tau)}
 \end{aligned} \tag{B.70}$$

With the original transformation,  $\tau = k_{\text{on}} \cdot t$ , we get

$$y = \frac{b^2}{a + \sqrt{a^2 - b^2} \cdot \coth(k_{\text{on}} \cdot \sqrt{a^2 - b^2} \cdot t)} \tag{B.71}$$

This is the solution to the exercise. Now, we can compare analytical solution (B.71) to a numerical solution using the ODE solver in MATLAB. The implementation, Listing B.11, gives the output shown in Fig. B.12.



**Listing B.11** Program **bindingriccati.m**

```

function bindingriccati()
%BINDINGRICCATI validate the analytical solution of Riccati equation
% BINDINGRICCATI() compares two solutions of Riccati equation,
% the analytical form to the numerical solution with ODE solver,
% and creates as output a graph with both solution. The presented
% type of equation is applied for building a mathematical models of
% immunoradiometric assays.

p.a = 2;
p.b = 1;
tspan = [0 1];
y0 = 0 ;
[t,y] = ode45(@derivatives, tspan, y0, [], p);
set(axes,'FontSize',15)
plot(t, y, 'xk','Markersize', 10 ); hold on
y = fan(t,p);
plot(t, y, 'ok')
legend('numerical', 'analytical')
xlabel('t')
ylabel('y')
title('Solution of RICCATI Equation')
print('-r600', '-dtiff', 'riccatti')

end

function dydt = derivatives(~,y, p)
%DERIVATIVES Compute the right-hand side of the ODE.
% DYDT = ...

dydt =y(1)^2-2*p.a*y(1)+p.b^2;

end

function y = fan(t, p)
%FAN compute the analytical solution of Riccati equation
% FAN(T,P) calculates values of Riccati solution at points |T|,
% and given parameters in structure |P|

s=-sqrt(p.a^2-p.b^2);
y = p.b^2./(p.a + s*coth(s*t));

end

```

**Exercise 4.9**

Derive the Michaelis–Menten rate law from enzyme kinetics considerations.

**Solution**

An enzyme  $E$  turns a substrate  $S$  into a product  $P$  without being changed by this reaction:



Let  $k_1$  and  $k_{-1}$  be the reaction rates for the first reversible reaction, and  $k_2$  the reaction rate for the formation of the product. The change in the enzyme-substrate complex,  $ES$ , can then be written as

$$\frac{d}{dt}ES = k_1 \cdot E \cdot S - k_{-1} \cdot ES - k_2 \cdot ES \quad (\text{B.73})$$

Under equilibrium assumption,  $ES$  is constant, thus  $\frac{d}{dt}ES = 0$ . Hence

$$\frac{E \cdot S}{ES} = \frac{k_{-1} + k_2}{k_1} = K_M \quad (\text{B.74})$$

$K_M$  is called the Michaelis–Menten constant. The total enzyme concentration,  $ET$ , is the sum of  $E$  and  $ES$

$$ET = E + ES \quad (\text{B.75})$$

Thus,

$$ES = \frac{ET \cdot S}{S + K_M} \quad (\text{B.76})$$

Equation (B.72) yields  $\frac{dP}{dt} = k_{23} \cdot ES$  and therefore the rate of formation of the product  $P$ ,  $v$ , is given by

$$v = \frac{k_2 \cdot ET \cdot S}{S + K_M} = \frac{v_{\max} \cdot S}{S + K_M} \quad (\text{B.77})$$

Equation (B.77) is called the Michaelis–Menten equation and  $v_{\max}$  the maximum velocity of product formation.

#### Exercise 4.10

Show that multiple nested  $E_{\max}$  models yield a further  $E_{\max}$  model.

##### Solution

Suppose  $E$  and  $F$  are  $E_{\max}$  models with parameters  $E_{\max}$ ,  $EC_{50}$  and  $F_{\max}$ ,  $FC_{50}$ , respectively.  $F$  is nested in  $E$ , if the output of  $E$  is used as input in  $F$ . According to the definition of an  $E_{\max}$  model

$$F = \frac{F_{\max} \cdot c}{c + FC_{50}}, \quad E = \frac{E_{\max} \cdot c}{c + EC_{50}} \quad (\text{B.78})$$

If we substitute  $c$  in  $F$  with  $E$ , we get another  $E_{\max}$  model:

$$F(E) =: G = \frac{G_{\max} \cdot c}{c + GC_{50}} \quad (\text{B.79})$$

with

$$G_{\max} = \frac{F_{\max} \cdot E_{\max}}{F_{\max} + FC_{50}} \quad (\text{B.80})$$

and

$$GC_{50} = \frac{FC_{50} \cdot EC_{50}}{F_{\max} + EC_{50}} \quad (\text{B.81})$$

### Exercise 4.11

Describe empirical criteria that allow differentiation between the direct link, the effect compartment and indirect response PK-PD models.

#### Solution

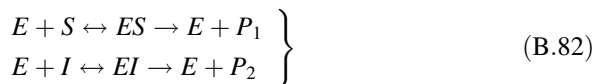
Only the direct link model shows no hysteresis when plotting concentrations against response. For the effect-compartment model, effect compartment concentrations are calculated from plasma concentrations via a linear ODE. Therefore, the maximum drug effect increases with dose and is time-independent. For the indirect-response model, the time of maximum drug effect changes with dose as parameters of the differential equation describing the response are affected in a nonlinear way.

### Exercise 4.12

Develop the equations for competitive inhibition of an enzyme.

#### Solution

Suppose  $S$  and  $I$  are competing for the same enzyme  $E$



with rate constants  $k_{1s}$ ,  $k_{-1s}$ ,  $k_{2s}$  and  $k_{1i}$ ,  $k_{-1i}$ ,  $k_{2i}$  for the substrate and inhibitor, respectively. According to Exercise 4.9, the Michaelis-Menten equations for  $S$  and  $I$  are

$$\left. \begin{array}{l} \frac{E \cdot S}{ES} = K_{Ms} \\ \frac{E \cdot I}{EI} = K_{Mi} \end{array} \right\} \quad (\text{B.83})$$

As the enzyme is conserved

$$ET = E + ES + EI \quad (\text{B.84})$$

Equations (B.83) and (B.84) can be solved for  $ES$ , yielding

$$ES = \frac{ET \cdot S}{S + K_{Ms} \cdot \left(1 + \frac{I}{K_{Mi}}\right)} \quad (\text{B.85})$$

or

$$v = \frac{v_{\max} \cdot S}{S + K_{Ms} \cdot \left(1 + \frac{I}{K_{Mi}}\right)} \quad (\text{B.86})$$

This shows that the maximum velocity is not decreased by a competitive inhibitor, but the substrate concentration for half-maximum velocity is increased.

### Exercise 5.1

Show how to obtain the transformation (5.14)  $\rightarrow$  (5.15), and then check units for correctness, assuming that model parameters are given as in Table 5.6, and voltage  $V$  in mV.

### Solution

The equation

$$\frac{1}{R} \cdot \frac{\partial^2 V}{\partial x^2} = C \cdot \frac{\partial V}{\partial t} + G_K \cdot (V - E_K) + G_{Na} \cdot (V - E_{Na}) + G_L \cdot (V - E_L) \quad (\text{B.87})$$

describes the membrane current per unit of length as the parameters  $R$ ,  $C$ ,  $G$  are measured per length. On the other hand, the corresponding axon parameters,  $R_{\text{spec}}$ ,  $C_{\text{spec}}$ ,  $G_{\text{spec}}$ , are expressed per area, and  $R_{\text{spec}}$  in resistance unit times length unit. Dependency is given in the following form:

$$C = C_{\text{spec}} \cdot 2 \cdot \pi \cdot a, \quad G = G_{\text{spec}} \cdot 2 \cdot \pi \cdot a, \quad R = \frac{R_{\text{spec}}}{\pi \cdot a^2} \quad (\text{B.88})$$

In fact, capacitance  $C$  and conductivity  $G$  are proportional to the circumference of the axon ( $2 \cdot \pi \cdot a$ ), and resistance  $R$  depends inversely on the cross-sectional area ( $\pi \cdot a^2$ ). Physiologically, the above formula is reasonable, as the bigger the circumference the larger the axon area ( $2 \cdot \pi \cdot a \cdot L$ ;  $L$  - length), and in this manner the greater the ability to store electrical charge (capacity  $C$ ), and conduct ion currents (conductivity  $G$ ). Regarding the longitudinal resistance  $R$  of the cytosol, we have a similar statement: the larger the cross-sectional area, the weaker the resistance  $R$ .

Regarding Eqs. (B.87) in (B.88), we obtain after simplification:

$$\frac{a}{2 \cdot R_{\text{spec}}} \cdot \frac{\partial^2 V}{\partial x^2} = C_{\text{spec}} \cdot \frac{\partial V}{\partial t} + \sum_{i \equiv K, Na, L} G_{\text{spec}(i)} \cdot (V - E_i) \quad (\text{B.89})$$

For simplicity, we can now ignore the index  $(\cdot)_{\text{spec}}$  and handle all axon parameters as specific with the proper units. Furthermore, we need to regard the gating variables  $n$ ,  $m$ ,  $h$  in a manner similar to that in model (5.4), i.e.

$$G_K \equiv g_K \cdot n^4, \quad G_{Na} \equiv g_{Na} \cdot h \cdot m^3, \quad G_L \equiv g_L \quad (\text{B.90})$$

Thus, we have

$$\begin{aligned} \frac{a}{2 \cdot R} \cdot \frac{\partial^2 V}{\partial x^2} = & C \cdot \frac{\partial V}{\partial t} + g_K \cdot n^4 \cdot (V - E_K) + g_{Na} \cdot h \cdot m^3 \cdot (V - E_{Na}) \\ & + g_L \cdot (V - E_L) \end{aligned} \quad (\text{B.91})$$

and this the final form we wanted to obtain.

To ensure the units are correct we have to derive the output unit of both the left and right sides of Eq. (B.91) based on the units in Table B.1.

**Table B.1** Parameter and units

Parameter	Units
$E_K$	mV
$E_{Na}$	mV
$E_L$	mV
$V$	mV
$t$	ms
$g_K$	mS/cm <sup>2</sup>
$g_{Na}$	mS/cm <sup>2</sup>
$g_L$	mS/cm <sup>2</sup>
$C$	uF/cm <sup>2</sup>
$R$	$\Omega \cdot \text{cm}$
$a$	cm

The unit of the left side,  $S_L$ :

$$[S_L] = \frac{\text{cm}}{\Omega \cdot \text{cm}} \cdot \frac{\text{mV}}{\text{cm} \cdot \text{cm}} = \frac{\text{mV}}{\Omega} \cdot \frac{1}{\text{cm} \cdot \text{cm}} = \frac{\text{mA}}{\text{cm}^2} \quad (\text{B.92})$$

The unit of the right side,  $S_R$ :

$$\begin{aligned}
[S_R] &= \frac{\mu\text{F}}{\text{cm} \cdot \text{cm}} \cdot \frac{\text{mV}}{\text{ms}} + \frac{\text{mS} \cdot \text{mV}}{\text{cm} \cdot \text{cm}} + \frac{\text{mS} \cdot \text{mV}}{\text{cm} \cdot \text{cm}} + \frac{\text{mS} \cdot \text{mV}}{\text{cm} \cdot \text{cm}} \\
&= \frac{10^{-6} \cdot \text{F}}{\text{cm} \cdot \text{cm}} \cdot \frac{\text{mV}}{\text{ms}} + \frac{10^{-3} \text{mV}}{\text{cm} \cdot \text{cm} \cdot \Omega} + \frac{10^{-3} \text{mV}}{\text{cm} \cdot \text{cm} \cdot \Omega} + \frac{10^{-3} \text{mV}}{\text{cm} \cdot \text{cm} \cdot \Omega} \\
&= \frac{10^{-6} \cdot \text{A} \cdot \text{s}}{\text{cm} \cdot \text{cm} \cdot \text{V}} \cdot \frac{\text{mV}}{10^{-3} \cdot \text{s}} + \frac{10^{-3} \cdot \text{mA}}{\text{cm}^2} + \frac{10^{-3} \cdot \text{mA}}{\text{cm}^2} + \frac{10^{-3} \cdot \text{mA}}{\text{cm}^2} \\
&= \frac{10^{-3} \cdot \text{mA} \cdot \text{s}}{\text{cm} \cdot \text{cm} \cdot 10^3 \cdot \text{mV}} \cdot \frac{\text{mV}}{10^{-3} \cdot \text{s}} + \frac{10^{-3} \cdot \text{mA}}{\text{cm}^2} + \frac{10^{-3} \cdot \text{mA}}{\text{cm}^2} + \frac{10^{-3} \cdot \text{mA}}{\text{cm}^2} \\
&= \frac{10^{-3} \cdot \text{mA}}{\text{cm}^2} + \frac{10^{-3} \cdot \text{mA}}{\text{cm}^2} + \frac{10^{-3} \cdot \text{mA}}{\text{cm}^2} + \frac{10^{-3} \cdot \text{mA}}{\text{cm}^2}
\end{aligned} \quad (\text{B.93})$$

But

$$\left[ \frac{10^{-3} \cdot \text{mA}}{\text{cm}^2} + \frac{10^{-3} \cdot \text{mA}}{\text{cm}^2} + \frac{10^{-3} \cdot \text{mA}}{\text{cm}^2} + \frac{10^{-3} \cdot \text{mA}}{\text{cm}^2} \right] = 10^{-3} \cdot \frac{\text{mA}}{\text{cm}^2} \quad (\text{B.94})$$

In Eqs. (B.92, B.93, and B.94), note that the operation  $[x]$  means *unit of  $x$* . Finally, we can see that

$$[S_L] = 10^3 \cdot [S_R] \quad (\text{B.95})$$

Conclusion: Using the standard values of parameters (Table B.1) in the HH model (B.91), we need the multiplier  $10^3$  on the right side.

The above checking is done manually. Using the Symbolic Math Toolbox the same unit evaluation task can be conducted in MATLAB, as shown in Listing B.12.

**Listing B.12** Program `units.m`

```

function units
%UNITS Evaluate units in the HH model.
%   UNITS checks the general correctness of units in the HH model,
%   given the model parameters in 'standard units'. The procedure
%   compares the output unit on the left side with components of the
%   right one.

syms F S cm mA mV ohm s uF uA
% basic units
OMEGA = mV/mA;           % ohm
S = 1/ohm;                % siemens
mS = S/1000;              % milisiemens
F = mA/mV*s;              % farad
uF = F/1000000;           % microfarad
ms = s/1000;              % milisecond
% Put standard units and compare both sides of the HH model equation
SL = cm/(OMEGA*cm)*mV/cm^2; % Left side of HH
SRC = uF/cm^2*mV/ms;       % right side of HH: ref. to capacitor
SRK = mS/cm^2*mV;          % right side of HH: ref. to K channel
SRNa = mS/cm^2*mV;         % right side of HH: ref. to Na channel
SRL = mS/cm^2*mV;          % right side of HH: ref. to L channel
display(['SL = ', char(simplify(SL))])
display(['SRC = ', char(simplify(SRC))])
display(['SRK = ', char(simplify(SRK))])
display(['SRNa = ', char(simplify(SRNa))])
display(['SRL = ', char(simplify(SRL))])

end

```

The above function produces:

```

SL = mA/cm^2
SRC = mA/(1000*cm^2)
SRK = mV/(1000*cm^2*ohm)
SRNa = mV/(1000*cm^2*ohm)
SRL = mV/(1000*cm^2*ohm)

```

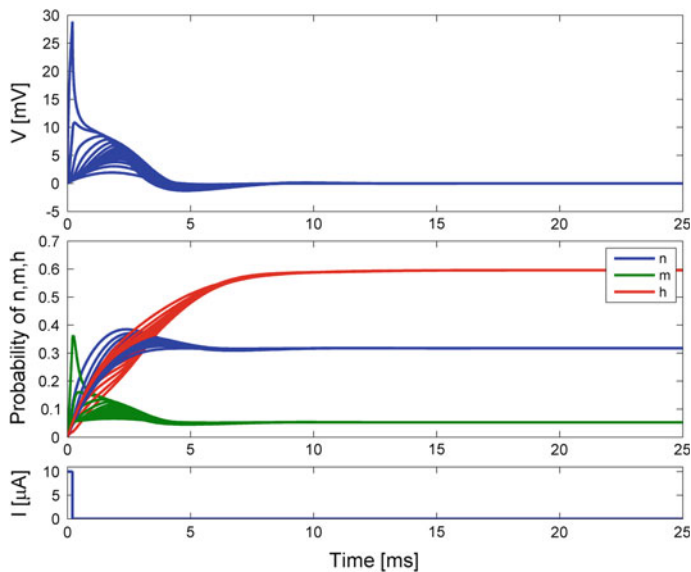
This result gives rise to the same conclusion as manual evaluation.

**Exercise 5.2**

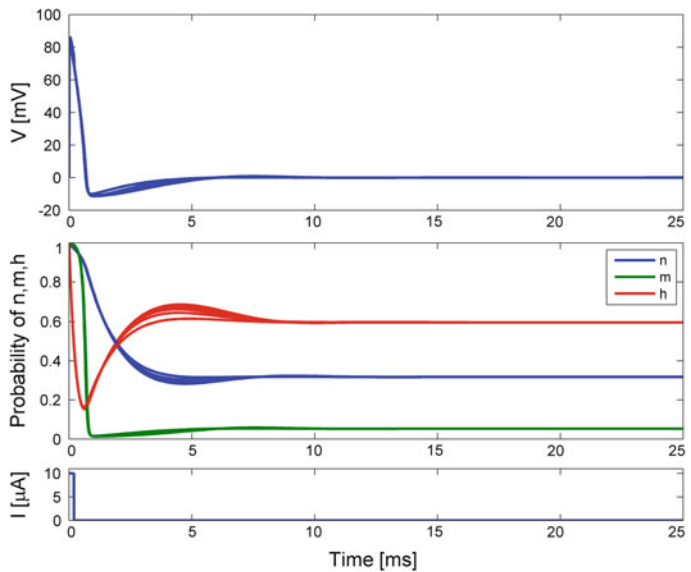
Assuming that the initial values of gating variables  $n$ ,  $m$ , and  $h$  in Eq. (5.15) are unknown, how we can find these values with help of the `hodhux.m` program?

**Solution**

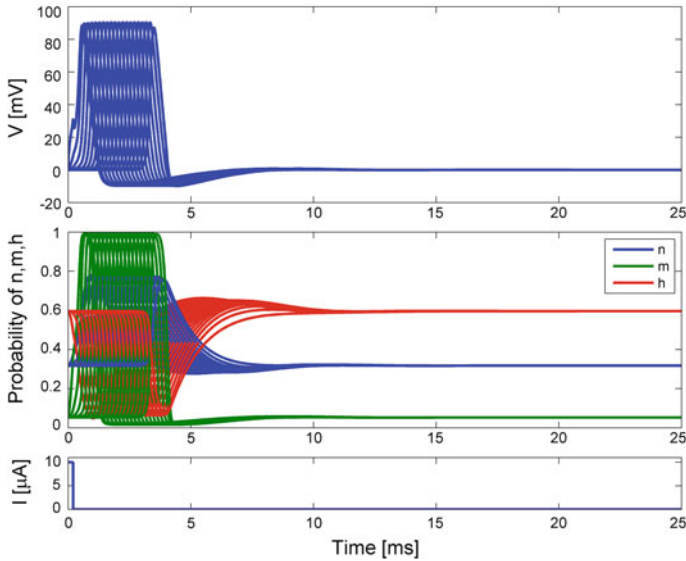
As mentioned in [Chap. 5](#), the initial values of gating variables  $n$ ,  $m$ , and  $h$  are probabilities, and are equal to the values at steady state  $n_\infty$ ,  $m_\infty$  and  $h_\infty$ . In fact, if we run the HH model (5.15) using any initial values from the interval  $[0; 1]$  (as a possible interval for probability), then the system will always come back to the same values, namely to  $n_\infty$ ,  $m_\infty$ , and  $h_\infty$ . This property can be used to solve this exercise. Selecting any initial values from  $[0; 1]$  (variables coded as **n0**, **m0**, **h0**)



**Fig. B.13** Steady-state values (0.3177, 0.0529, 0.5961) of gating variables ( $n$ ,  $m$ ,  $h$ ) obtained when  $n(0) = m(0) = h(0) = 0$



**Fig. B.14** Steady-state values (0.3177, 0.0529, 0.5961) of gating variables  $n, m, h$  obtained when  $n(0) = m(0) = h(0) = 1$



**Fig. B.15** Having launched the model with initial values  $n(0) = 0.3177$ ,  $m(0) = 0.0529$ ,  $h(0) = 0.5961$ , and assuming a sufficient long observation time, we can see that steady-state gating variables always come back to the same values, i.e.;  $n_{\infty} = 0.3177$ ,  $m_{\infty} = 0.0529$ ,  $h_{\infty} = 0.5961$ ; this happens for any location ( $x$ ) within the axon

and a sufficiently long observation time (coded as **p.tmax**), we can launch the **hodhux.m** program and register the steady-state values. Even if we assume such extreme value as (0, 0, 0) or (1, 1, 1), as shown in Figs. B.13 and B.14, steady-state values  $n_{\infty}$ ,  $m_{\infty}$ , and  $h_{\infty}$  are always equal to 0.3177, 0.0529, and 0.5961, respectively. Finally, if we use these steady-state values as initial values, we get the output shown in Fig. B.15, which is the standard case when using the HH model.

### Exercise 6.1

Write a program for the NLMEM analysis of a two-compartment model with infusion as described in Chap. 3, in (3.17, 3.18, and 3.19). Include the covariates weight and sex. The simulated dataset for the analysis, **drugCDData.mat**, is available at MATLAB Central. Compare the model with covariates to the model without covariates.

### Solution

The given model is

$$\begin{bmatrix} \frac{dc_1}{dt} \\ \frac{dc_2}{dt} \end{bmatrix} = \begin{bmatrix} \frac{r(t)}{V_1} - (k + k_{12}) \cdot c_1 + k_{21} \cdot \frac{V_2}{V_1} \cdot c_2 \\ k_{12} \cdot \frac{V_1}{V_2} \cdot c_1 - k_{21} \cdot c_2 \end{bmatrix}; \quad t \in [0; 15] \quad (\text{B.96})$$



**Table B.2** Dataset structure for the PK-PD analysis (shows only 2 subjects from 60)

ID	TIME	AMT	RATE	DV	PKPD	EVID	WGT	SEX
1	0	10	5	0.0000	1	1	66	1
1	0.5	0	0	0.1276	1	0	66	1
1	0.5	0	0	0.0245	2	0	66	1
1	1	0	0	0.2592	1	0	66	1
1	1	0	0	0.0743	2	0	66	1
1	2	0	0	0.2597	1	0	66	1
1	2	0	0	0.1932	2	0	66	1
1	3	0	0	0.0829	1	0	66	1
1	3	0	0	0.2785	2	0	66	1
1	4	0	0	0.0314	1	0	66	1
1	4	0	0	0.2847	2	0	66	1
1	6	0	0	0.0078	1	0	66	1
1	6	0	0	0.2338	2	0	66	1
1	8	0	0	0.0022	1	0	66	1
1	8	0	0	0.2210	2	0	66	1
1	12	0	0	0.0005	1	0	66	1
1	12	0	0	0.1455	2	0	66	1
2	0	10	5	0.0000	1	1	86	0
2	0.5	0	0	0.1096	1	0	86	0
2	0.5	0	0	0.0170	2	0	86	0
...	...	...	...	...	...	...	...	...
2	12	0	0	0.0012	1	0	86	0
2	12	0	0	0.1822	2	0	86	0
...	...	...	...	...	...	...	...	...

where the initial values for concentrations are

$$c_1(0) = c_2(0) = 0 \quad (\text{B.97})$$

The infusion rate and rate constants are the following

$$\left. \begin{aligned} r(t) &= \begin{cases} 50; & t \in [0; 2) \\ 0; & t \geq 2 \end{cases} \\ k &= \frac{CL}{V_1}; \quad k_{12} = \frac{Q}{V_1}; \quad k_{21} = \frac{Q}{V_2} \end{aligned} \right\} \quad (\text{B.98})$$

For visualization purposes, the structure of the simulated dataset, **drugCData.mat**, is shown in Table B.2 in reduced form (only 2 subjects are demonstrated).

The full MATLAB implementation, **xnlmedrugccov.m**, is available at MATLAB Central. Below, in Listing B.13, as a clue to the solution, we demonstrate how to implement the **model** function. It is passed to the **nlmefitsa** (or **nlmefit**) function as a parameter in form of a handle, **@model**, and specified according to Eqs. (B.96, B.97, and B.98).

**Listing B.13 Function model**

```

function [varargout] = model(phi,t,covar)
%MODEL compute model output values
% C = MODEL(PHI,T,COVAR) calculates plasma concentration values
% |C| at observation time points |T|, given model parameters |PHI|
% and covariates |COVAR|.
%
% [TMOD,C] = MODEL(PHI,T,COVAR) calculates plasma concentration
% values |C| at observation interval of length |T|, given
% model parameters |PHI| and covariates |COVAR|. The function
% output consists of concentration values |C| at time points
% |TMOD|.
%
% [TMOD,C,P] = MODEL(PHI,T,COVAR) calculates plasma concentration
% values |C| at observation interval of length |T|, given
% model parameters |PHI| and covariates |COVAR|. The function
% output consists of concentration values |C| at time points
% |TMOD|, structure |P| with model parameters

p.tR = 2;           % infusion stop
p.R = covar(1);     % rate
p.WGT = covar(2);   % weight as covariate
p.SEX = covar(3);   % weight as covariate

p.CL = phi(1);      % central clearance
p.V1 = phi(2);      % volume of distribution in central compartment
p.Q = phi(3);        % inter-compartmental clearance
p.V2 = phi(4);      % volume of distribution in peripheral compartment
p.k = p.CL/p.V1;    % rate constant of elimination
p.k12 = p.Q/p.V1;   % rate constant from central to peripheral
p.k21 = p.Q/p.V2;   % rate constant from peripheral to central

% Initial values
options = odeset('RelTol',1e-3, 'AbsTol',[1e-4 1e-4]);
tObs = [0; t];      % 0 + observation times
c0 = [0 0];         % Initial value for the dependent variable
if nargin == 1
    [~,y] = ode45(@derivatives, tObs, c0, options, p);
    conc = y(2:end,1);
    if max(isnan(conc)+ isinf(conc)) == 1
        % use penalty value as conc = NaN
        conc = 1e6;
        display('penalty factor')
    end
    varargout(1) = {conc};
else
    % solve during infusion - (interval 1)
    tspan = [0 p.tR];
    c0 = [0 0];      % Initial value for the dependent variable
    [t0R, y0R] = ode45(@derivatives, tspan, c0, options, p);
    % solve during infusion - (interval 2)
    tspan = [t0R(end) t];
    c0 = y0R(end,:); % Initial value for the dependent variable
    [tRT, yRT] = ode45(@derivatives, tspan, c0, options, p);
    % (interval 2) + (interval 1)

```

```

tmod = [t0R; tRT];
conc = [y0R; yRT];
if max(isnan(conc)+ isinf(conc)) == 1
    % use penalty value as conc = NaN
    conc = 1e6;
    display('penalty factor')
end
varargout(1) = {tmod};
varargout(2) = {conc};
varargout(3) = {p};
end
end

```

### Exercise 7.1

Modify the **esaCTS.m** program by adding one event that will enable you to interrupt the CTS process after 168 days and then decide, by entering (**Y** or **N**), if the procedure has to be continued or stopped.

### Solution

In order to solve the exercise, two functions have to be extended (modified), **fevents**, and **todoevents**. The changes linked to the extension are commented on in Listing B.14 and Listing B.15 as **% EXT**. This should illustrate what to do if some new events have to be added to trial simulator **esaCTS.m**.

#### Listing B.14 Function **fevents**

```

function [value,isterminal,direction] = fevents(t, ~ , ~, p)
%FEVENTS Specify events for the DDE model.
%   [VALUE,ISTERMINAL,DIRECTION] = ...

ns = p.numSubjects;

% Event: end of live-span (LS)
direction1(1:ns,1) = 1;
isterminal1(1:ns,1) = 1;
value1(1:ns,1) = t - p.LS;

% Event: Dosing
direction2(1,1) = 1;
isterminal2(1,1) = 1;
value2(1,1) = t - p.doseTimes(p.doseNext);

% Event: Interrupt the process after 180 days           %EXT
direction3(1,1) = 1;                                   %EXT
isterminal3(1,1) = 1;                                   %EXT
value3(1,1) = t - 168;                                   %EXT

% Event 1 + 2
direction = [direction1; direction2; direction3];       %EXT
isterminal = [isterminal1; isterminal2; isterminal3];   %EXT
value = [value1; value2; value3];                       %EXT

end

```

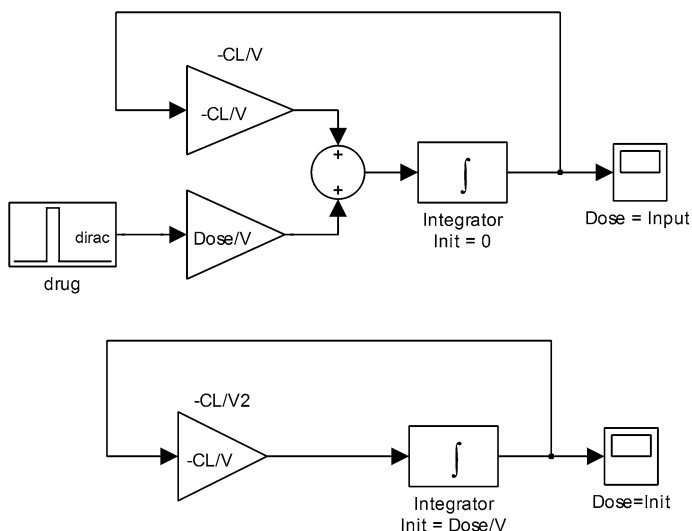
**Listing B.15** Function `todoevents`

```

function pMod = todoevents(y, numEvents, p)
%TODOEVENTS Handle detected events.
%   PMOD = ...

pMod = p;
ns = pMod.numSubjects;
% use 'for' as more than 1 event can happen at the same time
for i = numEvents
    % identify subject number
    switch i
        case num2cell(1:ns)
            % end of live-span (LS)
            subject = p.numRun*~(p.numRun==1) + i*(p.numRun==1);
            display(['time:', num2str(p.te), ...
                ', event:end of live-span, ', 'subject:', ...
                num2str(subject)])
            pMod.switcher = 0;
            pMod.inity(1:ns) = y(end, 1:ns);
        case num2cell(ns+1)
            % dosing time
            display(['time:', num2str(p.te), ', event:dosing time'])
            [pMod.doseTrue pMod.lastAdapt] = doseAdaptation(p, y);
            pMod.doseAmount = pMod.doseAmount.*pMod.doseTrue;
            pMod.doseAmount(pMod.doseAmount == 0 & ...
                pMod.doseTrue > 3/4) = 1;
            pMod.inity(1:ns) = y(end, 1:ns) + pMod.doseAmount;
            pMod.doseNext = pMod.doseNext + 1;
        case num2cell(ns+2) %EXT
            % dosing time %EXT
            display(['time:', num2str(p.te), ... %EXT
                ', event:INTERRUPTION !!! (%EXT)']) %EXT
            if input('Stop the process ? (y/n)', 's') == 'y' %EXT
                pMod.tmax = 168; %EXT
            end %EXT
        otherwise
            messageID = 'Book:CTS:UnknownEvent';
            messageStr = 'Unknown event '%s'.';
            error(messageID, messageStr, num2str(i));
    end
end
end

```



**Fig. B.16** Comparison of two models with different approaches to dose. *Upper panel* Model considering the dose as the input variable. *Lower panel* Model structure based on assumption that the whole dose is the initial amount value

### Exercise 8.1

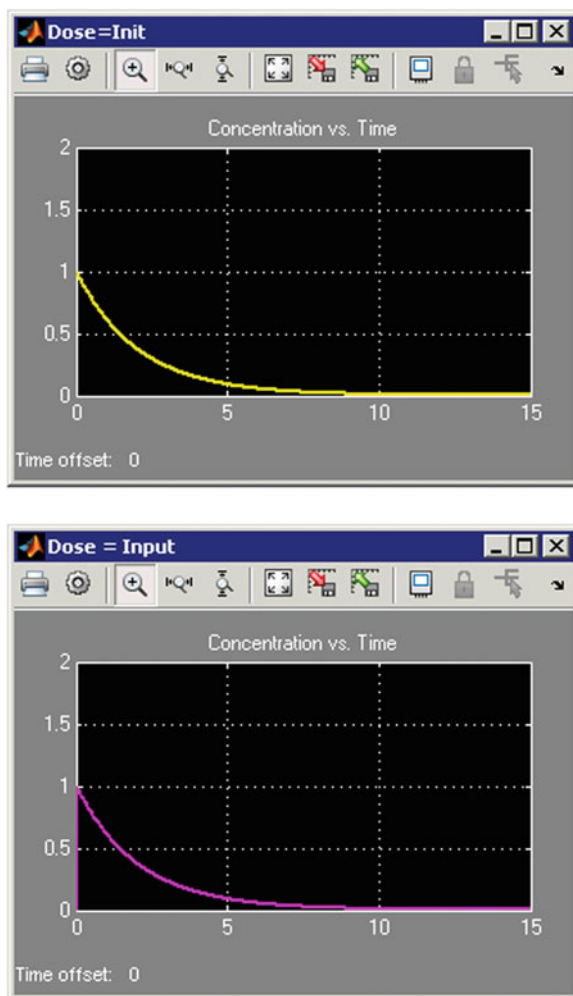
Create two versions of a one-compartment PK model with IV bolus input: the first based on the assumption that the whole dose is the initial amount value, and the second considering the dose as the input variable. Show that the model output (concentration–time course) in both cases is the same.

### Solution

Figure B.16 shows how to create a one-compartment model with IV bolus input in two different ways based on the different assumptions described above.

The results generated by these models are shown in Fig. B.17. Both Simulink models are stored in the `comp1_bolus_init.mdl` model file and the data (model parameters) in `comp1_bolus_init.mat`, and are available at MATLAB Central.

**Fig. B.17** Results of comparison of two models with different approaches to dose, as initial value (*upper panel*) and as input variable (*lower panel*). The time courses are the same in both cases and confirm the equivalence of both models



## References

1. [http://web.mst.edu/~hale/courses/411/411\\_notes/Chapter1.Appendix.Dirac.Delta.pdf](http://web.mst.edu/~hale/courses/411/411_notes/Chapter1.Appendix.Dirac.Delta.pdf)
2. Piotrovskii VK (1987) The use of Weibull distribution to describe the in vivo absorption kinetics. J Pharmacokin Biopharm 15:681–686
3. Rodbard D, Weiss GH (1973) Mathematical theory of immunoradiometric (Labeled Antibody) assays. Anal Biochem 52:10–44

# Index

## A

- Absorption, [88](#)
  - oral, [105](#)
- Action potential (AP), [148](#), [152](#)
  - absolute refractory period, [173](#)
  - conduction velocity, [172](#)
  - relative refractory period, [173](#)
  - threshold, [153](#)
  - time course, [153](#), [172](#)
- Akaike, H., [121](#)
- Alzheimer, A., [201](#)
  - Alzheimer's disease, [201](#)
  - disease progression model, [202](#)
- Anemia, [180](#)
  - renal anemia, [180](#)
- Area under the curve (AUC), [91](#), [324](#)
- Axes handle, [341](#)
- Axon, [148](#)

## B

- Beal, S., [209](#)
- Bernard, C., [143](#)
  - milieu intérieur, [143](#)
- Bioavailability, [90](#)
- Biomarker, [8](#)
- Biophase, [123](#)
- Black, J.
  - drug discovery by design, [5](#)
- Blinding, [240](#)
  - double-blind, [240](#)
  - single-blind, [240](#)
- Bohr, N.
  - Bohr model, [10](#)
- Bolus, [90](#), [367](#)
- Bottom-up approach, [290](#)
- Box, G.E.P., [11](#)

## C

- Cannon, W., [143](#)
- Catenary model, [92](#)
- Chain, E., [5](#)
- Chronic kidney disease (CKD), [183](#)
- Clearance, [91](#)
- Clinical trial simulation (CTS), [17](#), [239](#), [242](#)
  - dose adaptation table, [253](#)
- Compartment, [89](#)
- Computational biomarker, [308](#)
- Crick, F., [10](#)

## D

- De Laplace, P.-S.
  - Laplace transform, [72](#)
- Delay differential equation (DDE), [47](#)
  - DDE solvers, [50](#)
- Depolarization, [153](#)
- Diabetes, [191](#)
- Differential algebraic equation (DAE), [52](#)
  - mass option, [52](#)
- Differential equation, [39](#)
  - autonomous, [40](#)
- Dirac, P., [27](#)
  - Dirac delta function, [90](#)
- Disease model, [145](#)
- Disease progression model, [145](#), [299](#)
- Disposition, [89](#)
- Distribution, [88](#)
- DNA double helix model, [10](#)
- Dosage regimen, [207](#)
- Dose proportionally, [93](#)
- Dosing history, [74](#)
- Drug Disease Model Resource (DDMoRe), [305](#)
- Drug receptor, [114](#)

Drug-disease model, 145  
 Dynamic system, 39

## E

Eccles, J., 146  
 Effect compartment, 124  
 Ehrlich, P., 2  
 Elimination, 88  
 $E_{\max}$  model, 119  
   nestd, 140  
 Empirical PK modeling, 91  
 Enterohepatic cycling, 105  
 Enzyme inhibition, 140  
 Error tolerance, 46  
 Erythropoiesis, 180  
 Erythropoietin, 180  
 Erythropoietin stimulating agent (ESA), 183  
   dose adaptation, 190  
   SC versus IV route, 189  
 ESA replacement, 250  
   computational model, 253  
   conceptual model, 251  
   trial simulations, 264  
 Euler, L.  
   Euler method, 41  
 Event  
   handling, 331  
   state event, 61  
   time event, 55  
 Exploratory analysis, 215  
 Exponential matrix, 70  
 Extraction ratio, 107

## F

First-order kinetics, 92, 139  
 First-order reaction law, 92  
 First-pass effect, 113  
 Fleming, A., 3  
 Florey, H., 5  
 Food and Drug Administration (FDA), 9  
 Forcing function, 43  
 Function handle, 328

## G

Gauss, C. F., 13  
   normal (Gaussian) density function, 13, 357  
 Glucose effectiveness, 192  
 Glucose minimal model, 191

Glucose regulation, 191  
   integrated model, 195  
 Goldman, D.  
   Goldman equation, 149  
 Gompertz, B.  
   growth model, 24

## H

Harvey, W., 143  
 Heaviside, O.  
   Heaviside function, 73  
 Hemoglobin (Hb), 180  
   pathophysiologic model, 183  
   physiologic model, 181  
 Hill, A., 23  
   Hill coefficient, 119  
   Hill equation, 23  
 Hippocrates, 2  
 Hodgkin, A., 10  
 Hodgkin-Huxley (HH) model, 154, 388  
   basic model, 155  
   cable equation, 157, 159  
   computational implementation, 161, 167  
   electrical circuit representation, 155  
   gating functions, 156, 174, 388  
   threshold, 175  
   traveling wave, 160  
 Homeostasis, 143  
 Human Brain Project, 306  
 Human physiology, 144, 145  
 Huxley, A., 10  
 Hyperpolarization, 152

## I

Ibandronate, 130  
   computational model, 133  
   K-PD model, 132  
   PK model, 130  
   PK-PD model, 131  
   simulink model, 299  
 Imbalance, 208  
 Infusion, 89  
   rate, 43, 89  
 Initial value problem (IVP), 40  
 In-silico target validation, 307  
 Insulin sensitivity, 192  
 Intravenous glucose tolerance test (IVGTT), 193  
 Ion channel  
   conductance, 153, 179  
   voltage-gated, 153



**J**

- Jacobi, C., 355
- Jacobian determinant, 355

**K**

- Kinetics of PD response (K-PD), 128
- Kirchhoff, G., 156
  - junction rule, 156
  - loop rule, 157
- Kutta, M., 41

**L**

- Legends, 336
  - plot handles, 336
- Likelihood function, 212
- Lotka, A., 39

**M**

- Mammillary model, 92
- Markov, A.
  - Markov chain, 213, 237
- Mathematical model
  - dynamic, 12
  - model types, 12
- Mathematical modeling, 307
- Medical concepts
  - cellular pathology, 2
  - drug receptors, 2
- Membrane potential, 149
  - equilibrium, 150
  - resting, 149, 151
- Memory trace
  - computational model, 179
- Menten, M., 93
- Michaelis, L., 93
- Michaelis–Menten
  - equation, 382
  - kinetics, 93
- Mindset, 310
- Model
  - applications, 16
  - computational, 16
  - conceptual, 15
  - mathematical, 15
  - of systems, 14
  - parameter, 12
  - pharmacostatistical, 15
  - validation, 18, 311
- Model-based drug development (MBDD), 310
- Modeling and understanding, 11
- Modeling process, 14, 311

**Model selection criteria**

- Akaike information criterion (AIC), 121
- Schwarz criterion (SC), 121

**N**

- Nernst, W.
  - Nernst equation, 150
- Neuraminidase inhibitors, 198
- Neurocircuitry, 178
- Neuron, 147, 148
  - postsynaptic, 178
  - presynaptic, 178
- Neurotransmitter, 178
- Newton, I., 39

**O**

- Objective function, 79, 80
- One-compartment PK model
  - IV bolus input, 93
  - first-order input, 103
- Ordinary differential equation (ODE), 40
  - ODE solvers, 42

**P**

- Pandemic, 198
- Paracelsus, 1
- Parameter estimation, 78
- Partial differential equation (PDE), 40, 68
  - PDE solver, 68, 167
- Passive response, 152
- Pharmaceutical industry, 3, 305
  - clinical research, 7
  - drug development, 6, 9, 306
  - drug discovery, 4
  - mergers and acquisitions, 4
  - preclinical research, 6
- Pharmacodynamics (PD), 114
  - pharmacodynamic models, 118
  - model differentiation, 383
- Pharmacokinetics (PK), 87
  - compartmental representation, 90
  - conceptual model, 88
  - linear, 93
  - organ model, 107
  - physiologically-based (PB), 106
- Pharmacologic response, 114
  - response integration, 116
- Pharmacometrics curriculum, 309
- Pharmacon, 1, 87
- Pharmacostatistical model, 209, 212
  - covariate model, 210

Pharmacostatistical model (*cont.*)

- covariates, 210
- structural model, 210

## PK-PD relationship

- direct-link, 123
- indirect-link, 124
- indirect-response (IDR), 125

Poincare, H., 10

## Population analysis, 210

- covariate inclusion, 224
- individual prediction, 224, 229, 230
- model diagnostics, 230
- naïve pooled data approach, 208
- nonlinear mixed-effects modeling (NLMEM), 209
- one-stage approach, 209
- population prediction, 224, 229, 230
- two-stage approach, 209

## Population PK-PD, 208

## Physiologically-based PK (PBPK) model, 105

- drug absorption, 375

PriceWaterhouseCoopers, 305

**Q**

## Quantitative systems pharmacology (QSP), 309

**R**

## Rate-limitation

- metabolite release, 113

## Receptor

- binding, 116
- binding, equilibrium, 377
- binding, time course, 378
- occupancy, 117
- synaptic, 178

## Red blood cell (RBC), 180

- regulation, 181

## Regression procedure, 79

## Renal anemia

- drug-disease model, 184

## Riccati, J.

- Riccati equation, 378

## Runge, C.

- Runge–Kutta ODE solvers, 41

**S**

Saquinavir, 120

Schwarz, G., 121

## Screening

- phenotype-based, 5
- target-based, 5

## Sequential model-building, 113

Sertuerner, F., 5

Sheiner, L., 209

## SimBiology, 273

- command line, 284
- model exploration, 280
- model fitting, 281
- model verification, 277, 278, 279
- one-compartment PK model, 276
- parameter estimation, 290
- sensitivity analysis, 283
- state event scheduling, 283
- time event scheduling, 282

## Simulation protocol, 242

## Simulink, 290

- ODE representation, 290

- one-compartment PK model, 291

- signal builder, 294

- subsystems, 297

- two-compartment PK model, 295

## Sparseness, 208

Stephenson, R., 119

## Stimulus, 119

## Stochastic approximation expectation-maximization (SAEM), 232

## Structural identifiability, 103

## Study period, 241

## Superposition principle, 93, 370, 374

## Symptomatic treatment, 3

## Systems biology, 308

**T**

Tamiflu, 112

## Target validation, 5

Thalidomide, 9

## Time course of drug response, 123

## Top-down approach, 290, 307

## Transfer function, 73

## Treatment effects

- disease modifying, 200
- symptomatic, 199

## Treatment sequence, 240, 241

## Trial protocol, 240

Two-compartment PK model  
IV bolus input, 99**U**

## U-shape dose-response, 116

**V**Vectorization, [253](#), [320](#)Viral infection, [195](#)Viral kinetics (VK), [196](#), [199](#)Virchow, R., [2](#)Virtual man, [306](#)Visual predictive check (VPC), [236](#)Volterra, V., [39](#)Volume of distribution, [89](#)Von Bertalanffy, L., [308](#)**W**Watson, J., [10](#)World Health Organization (WHO), [3](#)**Z**Zero-order reaction law, [92](#)