Nonlinear mixed-effects modelling with nlmixr2

Justin J. Wilkins, Occams Matthew Fidler, Novartis

2026-02-11

Table of contents

Welcome

This is the website for the work-in-progress **“Nonlinear mixed-effects modelling with nlmixr2”**.

Our book will describe the installation of `nlmixr2`, and walk the reader through its use in pharmacological modelling and simulation using pharmacokinetic and pharmacodynamic data. We intend to start with simple examples and work towards more complex and useful applications. As well as the use of `nlmixr2` as a routine tool in academic and industrial drug research and development, the book will elaborate on the structure of the tool so that readers interested in contributing to the project or developing extensions have a good starting point for doing so.

Readers will need some familiarity with pharmacology and biostatistics to get the most out of the book.

Part I

Introduction

1 Introduction

1.1 Pharmacokinetics and Pharmacodynamics

In the vast landscape of medical science and pharmaceutical development, understanding the intricate relationship between drugs, the body, and their interactions is paramount. Before we start delving into the meat and potatos of this book, two fundamental concepts in pharmacology need to be touched on: pharmacokinetics and pharmacodynamics.

Pharmacokinetics (PK) and pharmacodynamics (PD) form the dual pillars of pharmacology, elucidating the journey of a drug from its initial administration to its final effect on the body. These disciplines are tightly intertwined, and indispensable for developing effective, safe, and personalized treatments.

1.1.1 Pharmacokinetics (PK)

Pharmacokinetics is the study of how drugs are absorbed, distributed, metabolized, and eliminated from the body. It examines the kinetics — rates and patterns — of the passage of drugs through the body and their interactions with biological systems. It is, colloquially put, the study of what the body does to the drug.

The **absorption** phase involves the uptake of drugs into the bloodstream from various routes of administration (oral, intravenous, subcutaenous, *et cetera*). Factors like solubility, pH, and the integrity of the gastrointestinal tract influence this process.

Distribution takes place once the drug enters the bloodstream, and describes the processes by which it is distributed to different tissues and organs. The *volume of distribution (Vd)* is a convenient measure of how extensively a drug is distributed in the body, reflecting the drug's affinity for different tissues.

Metabolism encompasses the processes whereby drugs are transformed from one form to another by enzymes in the liver and other organs. Hepatic and renal function, genetic polymorphisms, and the presence and effects of other drugs in the system can affect metabolic rates.

Finally, **elimination** refers to the processes that remove the drug from the body, including excretion in urine and feces, and *clearance* rates. Factors such as body size and health status can influence elimination pathways.

1.1.2 Pharmacodynamics (PD)

Pharmacodynamics, by contrast, focuses on what the drug does to the body. It investigates how drugs interact with biological systems to produce therapeutic or adverse effects. Understanding how drugs bind to receptors on cell surfaces is fundamental to PD; the nature of the binding, whether it's competitive, non-competitive, or allosteric, dictates how the drug functions.

Once bound, drugs can alter cellular processes such as gene expression, enzyme activity, and ion channel function, leading to physiological changes. PD also considers the impact of drugs on organ function, including metabolic processes, immune responses, and neurotransmitter systems.

Finally, PD provides a toolkit for translating these biological interactions into clinical outcomes, including therapeutic benefits, side effects, and drug interactions, amongst others.

1.1.3 Interplay between PK and PD

The efficacy and safety of a drug are fundamentally determined by the interplay between PK and PD. A drug's pharmacokinetic properties dictate its availability and duration of effect, while its pharmacodynamic profile determines the therapeutic response and potential adverse effects. A drug with rapid absorption and rapid metabolism, for instance, might require frequent dosing to maintain therapeutic levels. Similarly, a drug with a narrow therapeutic index (a small difference between therapeutic and toxic doses) would need to be carefully dosed to avoid toxicity. A good idea of the PK and PD of both of these drugs would be essential in order to be able to use them effectively, and being able to predict how they behave in people of differing physical characteristics and backgrounds would be even better.

Enter pharmacometrics.

1.2 Pharmacometrics

Wikipedia defines pharmacometrics (PMx) as a field of study of the methodology and application of mathematical models for disease and pharmacological measurement. It applies mathematical models of biology, pharmacology, disease, and physiology to describe and quantify interactions between xenobiotics (drugs) and patients (human and non-human), including both beneficial and adverse effects. It is normally applied to combine data from drugs, diseases and clinical trials to aid efficient drug development, regulatory decisions and rational drug treatment in patients.

Pharmacometrics rolls up modeling and simulation for pharmacokinetics, pharmacodynamics, and disease progression, with a focus on populations and variability. A major focus is to understand variability in drug response, which can be predictable (e.g. due to differences in body weight or kidney function) or unpredictable (differences between subjects seem random, but likely reflect a lack of knowledge or data).

Quantitative systems pharmacology (QSP) is also considered to be a part of the PMx ecosystem, but applies a more theoretical and less data-driven approach to building models. QSP models are often much more complex than PK/PD models, with less of a populations focus.

What this boils down to is using mathematical/statistical models to help explain and predict PK and PD - these are often combined to produce PKPD or exposure-response (ER) models. We build these using data collected from clinical trials (e.g. blood samples, clinical observations, scores, X-rays and suchlike - multiple samples, over time, from many subjects), which we use to build compartmental models which approximate what is happening over time using ordinary differential equations (ODEs).

This sounds complicated - and it can be - but it's based on the well-stirred compartmental model for PK, a well-established set of principles for how systems like these can be approximated.

Alcohol, interestingly, is a pretty good example. PK describes what happens to the alcohol (ethanol) you consume between the glass and the bathroom, and PD describes what it does while it's circulating in your blood (quite a few things, including making you tipsy). It is eye-wateringly complex. The "DrinkMe" simulation on Nick Holford's website is a fun interactive example of how it fits together! You can find it at http://holford.fmhs.auckland.ac.nz/research/ethanol.

So pharmacometrics can help us understand how drugs behave in different people. The "DrinkMe" model includes body weight - the bigger you are, the bigger your organs are (usually) and the more machinery you have for metabolizing substances like ethanol, so the slower you get drunk, and if you've eaten something, the alcohol will take longer to get into your system (although these are just two aspects of a very complex system).

These principles apply to every drug we take, from aspirin to metformin (which is commonly used for treating diabetes). We use these models to figure out what an appropriate dose is, and what might affect it.

We can use pharmacometric models like these to simulate clinical trials, dose regimens and so on, *in silico*, so that we can predict what will happen when we actually give a drug to a human, and whether the design we have proposed for our clinical trial will actually work when we run it.

Later on in drug development, as we get close to registration, we can use these models to identify covariates whcih might inform differences in exposure and effect between patients (like age, weight, and sex), and to quantify the relationships between dose, exposure, and response for efficacy (e.g. how well the drug does at reducing or eliminating a tumour) and safety (e.g. how many unwanted side effects the drug generates at a useful dose).

It's not just about the drugs themselves. Drug-disease and disease progression models are also an area in which pharmacometrics continues to have an impact - FDA maintains a list the ones they've developed internally (https://www.fda.gov/about-fda/center-drug-evaluation-and-research-cder/division-pharmacometrics), including examples for Alzheimer's disease and diabetes, although there are many, many more.

So far we've mostly talked about empirical, data-driven models, but pharmacometrics goes further, especially now that the computers are getting so fast (models take time to fit to data, and the more complex they are, and the more patients you have, the longer they take).

Physiologically-based PK (PBPK) models, for example, find the middle ground between PK and QSP, having a more mechanistic bent by taking into account anatomical, physiological, physical, and chemical descriptions of the phenomena involved in complex absorption, distribution, metabolic and elimination (ADME) processes, while remaining fundamentally driven by observed data.

1.3 `nlmixr2`

`nlmixr2` is a set of packages - let's call it the "mixrverse" - for R that provides an open source alternative for nonlinear mixed-effects (NLME) model development, which are the core of most pharmacometrics workflows (amongst others).

Modeling tools in our area are largely closed-source and massively expensive, and are a gigantic entry barrier for new people, especially in low and middle-income countries (and borderline unaffordable even for CROs like mine). `nlmixr2` is intended to be a solution to this problem.

1.4 What you will learn

This book is intended to be a guide to using `nlmixr2` and its constellation of supporting and allied packages in R to develop and use nonlinear mixed-effect pharmacometric models. It is not going to teach you pharmacology, or the core tenets of pharmacometrics. You can learn about those elsewhere.

You will, however, learn to construct datasets for analysis, to write models in `rxode2` and `nlmixr2`, to fit them using `nlmixr2`, to use `shinyMixR` for tracking model development steps, to use `xpose.nlmixr2` for model evaluation, to use `babelmixr2` to cross-convert models from different tools, and to use `PKNCA` for figuring out initial estimates. You'll also learn how the "mixrverse" ecosystem has been constructed and how to work with it efficiently.

Throughout the book, we'll point you to resources where you can learn more.

1.5 How this book is organised

We start off with a summary of `nlmixr2` and all its dependencies, and how they're built and work together. This is essential for understanding why things have been set up in the way they have, and how to drill down into the source code to figure out what is actually happening under the hood. It is not, however, essential if you want to dive straight into modeling.

We then get into datasets - how they should be structured, how events like doses are handled, visualization, and what variables should be.

Next up, we look at a simple PK model, to illustrate how models can be written - both with closed-form solutions and ODEs - as well as how `nlmixr2` objects are constructed, and how to extract information from them. We'll use this example to explore the various minimization algorithms that are available and how to tune them.

We'll then move on to a more complex PK example, to illustrate some of `nlmixr2`'s niftier features, like transit absorption models, and how to use the various diagnostics that are available, as well as `shinyMixR`. We'll then segue into simulation (using `rxode2`) to see how pharmacometric models can be used to predict clinical trial outcomes, for example.

PK/PD models will be demonstrated using a version of the legendary haematological toxicity ("hemtox") model, along with a practical demonstration of how it can be used to predict neutropenia rates.

Finally, we'll wrap up with a demonstration of using `babelmixr` to import models from NONMEM, and `PKNCA` for providing credible initial estimates, and some guidelines on how you can contribute to the project if you so wish. `nlmixr2` is, after all, an open-source project and relies entirely on volunteers for its development and maintenance.

Within each chapter, we try and adhere to a similar pattern: start with some motivating examples so you can see the bigger picture, and then dive into the details. Each section of the book is paired with exercises to help you practice what you've learned.

Although it can be tempting to skip the exercises, there's no better way to learn than practicing on real problems.

1.6 What you won't learn

There are some topics that this book doesn't cover, simply because there isn't space.

1.6.1 Pharmacology

This is quite a big one. You can't be an effective pharmacometrician unless you're up to speed with basic pharmacology, which you can't pick up in an afternoon. We'll be touching on pharmacology concepts throughout, but we're assuming you already know the theory. There are quite a few good books that can serve as an introduction to the topic - we particularly like Rowland & Tozer (1).

1.6.2 Pharmacometrics

Even bigger. Although you'll be able to infer a lot of things as we go, it would help if you already know what compartmental nonlinear mixed-effects models are and how they can be used to model the behaviour of drugs. Mould & Upton published a nice overview of the field a decade or so ago (2–4), and there are good textbooks as well (5,6).

1.6.3 Big data

This book assumes you're working with relatively small in-memory datasets. The kinds of models we talk about here don't work well with bigger ones.

1.6.4 Data science

We are dealing with specifically pharmacometric data analysis round these parts. If it's pure data science you're interested in, we heartily recommend R for Data Science, which provides a comprehensive grounding.

1.6.5 Python/Julia/Matlab/SAS/Ruby on Rails/etc

In this book, you won't learn anything about Python, Julia, JavaScript or any other language outside of R. This is because `nlmixr2` is written in R.

R is an environment designed from the ground up to support quantitative science, being not just a programming language, but also an interactive environment. It is - in our opinion - a much more flexible language than many of its peers.

1.7 Prerequisites

There's some things we assume you know to get the most out of this book. We expect you to know your way around numbers and math, and to have at least basic experience with programming in R. If you're new to R programming, Hands on Programming with R is a highly-recommended place to start.

You need a computer running a recent version of Windows, macOS or Linux with a decent amount of RAM, and some software.

1.7.1 R

R is free and open source, and can be freely downloaded from CRAN, the **c**omprehensive **R** **a**rchive **n**etwork. CRAN is composed of a vast collection of mirrored servers located around the world and is used to distribute R and R packages. Rather than trying to pick the nearest server, use the cloud mirror, https://cloud.r-project.org, which automatically does the heavy lifting for you. New major releases come once a year, interspersed with 2-3 minor releases. It's a good idea to keep current, but we know that people in the pharma industry aren't necessarily able to do this. That being said, you need version 4.2.2 or better for this book.

1.7.2 RStudio

RStudio is an integrated development environment, or IDE, for R and Python. You can get it from https://posit.co/download/rstudio-desktop/. You'll need at least version 2022.07.2+576.

1.7.3 `nlmixr2` and friends

It goes without saying that you'll need to install some additional R packages. An R package is, essentially, a bundle of functions, data, and documentation that can be added to base R to extend its capabilities. As of today, there are tens of thousands of them.

Install `nlmixr2` and its many dependencies from CRAN by entering the following code into R (or RStudio):

```
install.packages("nlmixr2","sessioninfo","pmxTools","PKNCA","babelmixr2","xpose.nlmixr2")
```

Once installed, it can be loaded as follows. Note that you can't use it until it's been loaded.

```r
library(nlmixr2)
```

```
Loading required package: nlmixr2data
```

1.8 Acknowledgements

We have a lot of people to thank. `nlmixr2` is the product of countless hours of hard work by many, many contributors.

First and foremost, Wenping Wang is in many ways the father of `nlmixr`. It was his work that provided the foundation for this tool, and although he has since moved on from the core development team, everything we've built started with him.

Teun Post was there at the very beginning, and was responsible for a lot of the initial documentation for `nlmixr`. Although he too has moved on, his contribution was large and fundamental.

The `nlmixr` project has relied heavily on support from our day jobs. Matt, Mirjam, Yuan, Huijuan and Wenping were given the time they needed to work on this project by Novartis, which remains an enthusiastic core sponsor of the team and the project. Mick Looby, Lisa Hendricks and Etienne Pigeolet are worthy of special mention here. Justin and Rik have had their time sponsored by Occams, Richard and Teun were and are supported by LAP&P, Johnson & Johnson, Seattle Genetics, Avrobio, Human Predictions and Certara have all donated their associates' time to help us build this tool. We are grateful to all of them.

Without `RxODE`, there would be no `nlmixr`. We would be remiss in not mentioning the early contributions of Melissa Hallow, David James and Wenping in the development of this, the engine that drives `nlmixr` and `nlmixr2`.

We'd all like to thank our families and colleagues for putting up with the odd hours and late nights and copious amounts of swearing emanating from our offices, both at work, and more often, recently, at home.

Those we're most grateful to, though, are our users: the early adopters, the curious, and most importantly, the bug reporters who have put `nlmixr` through its paces over the years and helped it become what it is today. Thank you, and we hope you'll stay with us as we grow.

1.9 Colophon

This book is powered by Quarto which makes it easy to write books that combine text and executable code.

This book was built with:

```r
sessioninfo::session_info(c("nlmixr2"))
```

```
- Session info ---------------------------------------------------------------
 setting  value
 version  R version 4.4.2 (2024-10-31 ucrt)
 os       Windows 11 x64 (build 22631)
 system   x86_64, mingw32
 ui       RTerm
 language (EN)
 collate  English_United Kingdom.utf8
 ctype    English_United Kingdom.utf8
 tz       Europe/Berlin
 date     2024-11-27
 pandoc   3.2 @ C:/Program Files/RStudio/resources/app/bin/quarto/bin/tools/ (via rmarkdown)

- Packages -------------------------------------------------------------------
 ! package        * version    date (UTC) lib source
   askpass          1.2.1      2024-10-04 [1] RSPM (R 4.4.0)
   assertthat       0.2.1      2019-03-21 [1] RSPM
 P backports        1.5.0      2024-05-23 [?] RSPM
   base64enc        0.1-3      2015-07-28 [1] RSPM
   BH               1.84.0-0   2024-01-10 [1] RSPM
   binom            1.1-1.1    2022-05-02 [1] RSPM
   bit              4.5.0      2024-09-20 [1] RSPM (R 4.4.0)
   bit64            4.5.2      2024-09-22 [1] RSPM (R 4.4.0)
   bitops           1.0-9      2024-10-03 [1] RSPM (R 4.4.0)
   boot             1.3-31     2024-08-28 [2] CRAN (R 4.4.2)
   bslib            0.8.0      2024-07-29 [1] RSPM
 P cachem           1.1.0      2024-05-16 [?] RSPM
   cellranger       1.1.0      2016-07-27 [1] RSPM
 P checkmate        2.3.2      2024-07-29 [?] RSPM
   class            7.3-22     2023-05-03 [2] CRAN (R 4.4.2)
   classInt         0.4-10     2023-09-05 [1] RSPM
 P cli              3.6.3      2024-06-21 [?] RSPM
   clipr            0.8.0      2022-02-22 [1] RSPM
   cluster          2.1.6      2023-12-01 [2] CRAN (R 4.4.2)
 P colorspace       2.1-1      2024-07-26 [?] RSPM
   commonmark       1.9.2      2024-10-04 [1] RSPM (R 4.4.0)
   cpp11            0.5.0      2024-08-27 [1] RSPM (R 4.4.0)
   cpp11armadillo   0.3.3      2024-09-02 [1] RSPM
   crayon           1.5.3      2024-06-20 [1] RSPM
   curl             6.0.1      2024-11-14 [1] RSPM (R 4.4.0)
   data.table      1.16.2    2024-10-10 [1] RSPM (R 4.4.0)
   Deriv           4.1.6     2024-09-13 [1] RSPM (R 4.4.0)
   DescTools       0.99.58   2024-11-08 [1] RSPM (R 4.4.0)
 P digest          0.6.37    2024-08-19 [?] RSPM
   dparser         1.3.1-13  2024-10-22 [1] RSPM (R 4.4.0)
 P dplyr           1.1.4     2023-11-17 [?] RSPM
   e1071           1.7-16    2024-09-16 [1] RSPM (R 4.4.0)
   evaluate        1.0.1     2024-10-10 [1] RSPM (R 4.4.0)
   Exact           3.3       2024-07-21 [1] RSPM
   expm            1.0-0     2024-08-19 [1] RSPM
 P fansi           1.0.6     2023-12-08 [?] RSPM
   farver          2.1.2     2024-05-13 [1] RSPM
 P fastmap         1.2.0     2024-05-15 [?] RSPM
   fontawesome     0.5.3     2024-11-16 [1] RSPM (R 4.4.0)
   forcats         1.0.0     2023-01-29 [1] RSPM
   foreign         0.8-87    2024-06-26 [2] CRAN (R 4.4.2)
   Formula         1.2-5     2023-02-24 [1] RSPM
   fs              1.6.5     2024-10-30 [1] RSPM (R 4.4.0)
 P generics        0.1.3     2022-07-05 [?] RSPM
 P ggplot2         3.5.1     2024-04-23 [?] RSPM
   ggtext          0.1.2     2022-09-16 [1] RSPM
   gld             2.6.6     2022-10-23 [1] RSPM
   glue            1.8.0     2024-09-30 [1] RSPM (R 4.4.0)
   gridExtra       2.3       2017-09-09 [1] RSPM
   gridtext        0.1.5     2022-09-16 [1] RSPM
   gtable          0.3.6     2024-10-25 [1] RSPM (R 4.4.0)
   haven           2.5.4     2023-11-30 [1] RSPM
   highr           0.11      2024-05-26 [1] RSPM
   Hmisc           5.2-0     2024-10-28 [1] RSPM (R 4.4.0)
   hms             1.1.3     2023-03-21 [1] RSPM
   htmlTable       2.4.3     2024-07-21 [1] RSPM
 P htmltools       0.5.8.1   2024-04-04 [?] RSPM
   htmlwidgets     1.6.4     2023-12-06 [1] RSPM
   httr            1.4.7     2023-08-15 [1] RSPM
   inline          0.3.20    2024-11-10 [1] RSPM (R 4.4.0)
   isoband         0.2.7     2022-12-20 [1] RSPM
   jpeg            0.1-10    2022-11-29 [1] RSPM
   jquerylib       0.1.4     2021-04-26 [1] RSPM
   jsonlite        1.8.9     2024-09-20 [1] RSPM (R 4.4.0)
   KernSmooth      2.23-24   2024-05-17 [2] CRAN (R 4.4.2)
   knitr           1.49      2024-11-08 [1] RSPM (R 4.4.0)
   labeling        0.4.3     2023-08-29 [1] RSPM
 P lattice         0.22-6    2024-03-20 [?] CRAN (R 4.4.2)
```

```
   lazyeval         0.2.2      2019-03-15 [1] RSPM
   lbfgsb3c         2024-3.5   2024-09-17 [1] RSPM (R 4.4.0)
P  lifecycle        1.0.4      2023-11-07 [?] RSPM
   lmom             3.2        2024-09-30 [1] RSPM (R 4.4.0)
P  lotri            1.0.0      2024-09-18 [?] RSPM
P  magrittr         2.0.3      2022-03-30 [?] RSPM
   markdown         1.13       2024-06-04 [1] RSPM
   MASS             7.3-61     2024-06-13 [2] CRAN (R 4.4.2)
   Matrix           1.7-1      2024-10-18 [2] CRAN (R 4.4.2)
P  memoise          2.0.1      2021-11-26 [?] RSPM
   mgcv             1.9-1      2023-12-21 [2] CRAN (R 4.4.2)
   mime             0.12       2021-09-28 [1] RSPM
   minpack.lm       1.2-4      2023-09-11 [1] RSPM
   minqa            1.2.8      2024-08-17 [1] RSPM
P  munsell          0.5.1      2024-04-01 [?] RSPM
   mvtnorm          1.3-2      2024-11-04 [1] RSPM (R 4.4.0)
   n1qn1            6.0.1-12   2024-09-17 [1] RSPM (R 4.4.0)
P  nlme             3.1-166    2024-08-14 [?] CRAN (R 4.4.2)
   nlmixr2        * 3.0.1      2024-10-28 [1] RSPM (R 4.4.0)
P  nlmixr2data    * 2.0.9      2024-01-31 [?] RSPM
   nlmixr2est       3.0.2      2024-11-23 [1] RSPM (R 4.4.0)
   nlmixr2extra     3.0.1      2024-10-29 [1] RSPM
   nlmixr2plot      3.0.0      2024-09-18 [1] RSPM (R 4.4.0)
   nnet             7.3-19     2023-05-03 [2] CRAN (R 4.4.2)
   numDeriv         2016.8-1.1 2019-06-06 [1] RSPM
   openssl          2.2.2      2024-09-20 [1] RSPM (R 4.4.0)
   pander           0.6.5      2022-03-18 [1] RSPM
P  pillar           1.9.0      2023-03-22 [?] RSPM
P  pkgconfig        2.0.3      2019-09-22 [?] RSPM
   png              0.1-8      2022-11-29 [1] RSPM
P  PreciseSums      0.7        2024-09-17 [?] RSPM
   prettyunits      1.2.0      2023-09-24 [1] RSPM
   progress         1.2.3      2023-12-06 [1] RSPM
   proxy            0.4-27     2022-06-09 [1] RSPM
   purrr            1.0.2      2023-08-10 [1] RSPM
   qs               0.27.2     2024-10-01 [1] RSPM (R 4.4.0)
P  R6               2.5.1      2021-08-19 [?] RSPM
   RApiSerialize    0.1.4      2024-09-28 [1] RSPM (R 4.4.0)
   rappdirs         0.3.3      2021-01-31 [1] RSPM
   RColorBrewer     1.1-3      2022-04-03 [1] RSPM
   Rcpp             1.0.13-1   2024-11-02 [1] RSPM (R 4.4.0)
   RcppArmadillo    14.2.0-1   2024-11-18 [1] RSPM (R 4.4.0)
   RcppEigen        0.3.4.0.2  2024-08-24 [1] RSPM
```

```
PD RcppParallel     5.1.9       2024-08-19 [?] RSPM
   RCurl            1.98-1.16   2024-07-11 [1] RSPM
   readr            2.1.5       2024-01-10 [1] RSPM
   readxl           1.4.3       2023-07-06 [1] RSPM
   rematch          2.0.0       2023-08-30 [1] RSPM
   rex              1.2.1       2021-11-26 [1] RSPM
P  rlang            1.1.4       2024-06-04 [?] RSPM
   rmarkdown        2.29        2024-11-04 [1] RSPM (R 4.4.0)
   rootSolve        1.8.2.4     2023-09-21 [1] RSPM
   rpart            4.1.23      2023-12-05 [2] CRAN (R 4.4.2)
   rstudioapi       0.17.1      2024-10-22 [1] RSPM (R 4.4.0)
   rxode2           3.0.2       2024-10-30 [1] RSPM (R 4.4.0)
   rxode2ll         2.0.12      2024-11-21 [1] RSPM (R 4.4.0)
   sass             0.4.9       2024-03-15 [1] RSPM
P  scales           1.3.0       2023-11-28 [?] RSPM
   sitmo            2.0.2       2021-10-13 [1] RSPM
   StanHeaders      2.32.10     2024-07-15 [1] RSPM
P  stringfish       0.16.0      2023-11-28 [?] RSPM
   stringi          1.8.4       2024-05-06 [1] RSPM
   stringr          1.5.1       2023-11-14 [1] RSPM
   survival         3.7-0       2024-06-05 [2] CRAN (R 4.4.2)
P  symengine        0.2.6       2024-02-25 [?] RSPM
   sys              3.4.3       2024-10-04 [1] RSPM (R 4.4.0)
P  tibble           3.2.1       2023-03-20 [?] RSPM
   tidyr            1.3.1       2024-01-24 [1] RSPM
P  tidyselect       1.2.1       2024-03-11 [?] RSPM
   tinytex          0.54        2024-11-01 [1] RSPM (R 4.4.0)
   tzdb             0.4.0       2023-05-12 [1] RSPM
P  utf8             1.2.4       2023-10-22 [?] RSPM
P  vctrs            0.6.5       2023-12-01 [?] RSPM
   viridis          0.6.5       2024-01-29 [1] RSPM
   viridisLite      0.4.2       2023-05-02 [1] RSPM
   vpc              1.2.2       2021-01-11 [1] RSPM
   vroom            1.6.5       2023-12-05 [1] RSPM
   withr            3.0.2       2024-10-28 [1] RSPM (R 4.4.0)
   xfun             0.49        2024-10-31 [1] RSPM (R 4.4.0)
   xgxr             1.1.2       2023-03-22 [1] RSPM (R 4.4.0)
   xml2             1.3.6       2023-12-04 [1] RSPM
P  yaml             2.3.10      2024-07-26 [?] RSPM

[1] C:/Users/justin/Documents/GitHub/nlmixr2_book/renv/library/windows/R-4.4/x86_64-w64-ming
[2] C:/Users/justin/AppData/Local/R/cache/R/renv/sandbox/windows/R-4.4/x86_64-w64-mingw32/e
```

```
 P -- Loaded and on-disk path mismatch.
 D -- DLL MD5 mismatch, broken installation.

-------------------------------------------------------------------------------
```

2 Prerequisites

2.1 Hardware

It probably goes without saying that you're going to need a computer of some kind.

At a minimum, you're going to need a reasonably recent workstation or laptop (PC or Mac) running Windows, macOS or Linux, and equipped with a 64-bit processor with x86-compatible architecture (such as AMD64, Intel 64, x86-64, IA-32e, EM64T, or x64 chips), with as many cores as you can get away with. You'll need sufficient disk space, but in this day and age you'll probably be fine with what you have as long as it's more than 5Gb or so. You can theoretically run R with 2 Gb of RAM, but in practice we think 16 Gb is an absolute drop dead minimum. R is a memory hog, so the more the better.

2.2 Core software for a local installation

2.2.1 R

R is the living, breathing heart of `nlmixr2`, and with Python and Julia, is at the centre of modern data science. R was started by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand, as a teaching tool. In the beginning it was heavily based on the proprietary S language, but has since surpassed it in almost every conceivable way. It was first made public in mid-1993, and version 1.0 was released as free and open-source under the GNU General Public License in February 2000. Its key strength lies in the tens of thousands of add-on packages that are available for it, filling every imaginable need. The Comprehensive R Archive Network (CRAN) was founded in 1997 to be a repository for R itself (both source code and platform-specific binaries), documentation, and third-party packages. Per Wikipedia, as of December 2022, it has 103 mirrors and almost 20,000 contributed packages. Today R is maintained by the R Core Team, an association of 20-odd statisticians and computer scientists, and supported by the R Foundation, a non-profit organization founded in 2003.

For Windows and Mac users, R is best downloaded from r-project.org, the central hub of CRAN, where the most recent stable and development versions may be found. Following the `CRAN` link in the navigation bar on the left of the landing page will allow to select a convenient mirror, from which you can download the latest version for your system (4.2.2 at the time of writing - the latest version is almost always recommended, since older versions may not necessarily work). Although ready-to-use versions are provided for Debian, Fedora/Redhat and Ubuntu-based distributions, Linux users may find it more convenient to use their own package managers to obtain it, although the versions available through these channels are often older. In most cases, you can add R-specific repositories that will get you the latest and greatest R release, but describing how to do this is beyond the scope of this book. Google is your friend. The very bravest among you might choose to compile R from its source code, but this too is beyond our scope.

2.2.2 Build tools

`nlmixr2` and some of its dependencies are partially written in C, and thus require compilation in order to work. The tools and toolchains needed for this are, frustratingly, not built into most operating systems by default, so we will need to install them ourselves under most circumstances. Making things even more complicated, you need the right versions, or things will break. Helpfully, CRAN has made these tools available from their website for Windows and Mac users.

2.2.2.1 Windows

For Windows, you will need RTools, which can be found on CRAN at cran.r-project.org/bin/windows/Rtools. Just to make things exciting, the version of RTools you need depends on the version of R you have, so for example, if you have R 4.2.x, you'll need RTools 4.2. Full details are provided on the website - download and install.

2.2.2.2 macOS

For macOS, you'll need Xcode - download it free from the App Store - and a suitable Fortran compiler. CRAN provides a compatible version of gfortran at mac.r-project.org/tools. Add the gfortran directory to your path using `export PATH=$PATH:/usr/local/gfortran/bin` (thus allowing R to find it when it needs it).

2.2.2.3 Linux

For Linux, there are many ways to get the tools you need installed and working. For Ubuntu, for example, you can install gcc and other essential build tools using `sudo apt-get install r-base-dev`; for (recent versions of) Fedora, the command is `sudo dnf install R-core-devel`.

2.2.3 Installing packages and their dependencies

In principle, you should be able to install everything using one command in R: `install.packages("nlmixr2", dependencies = TRUE)`. Assuming your environment is properly set up, everything will download and install itself. Get some coffee (or tea, or whatever beverage you like) because this step will take a bit of time. There are a lot of packages.

2.2.3.1 `rxode2`

`rxode2` is a set of R packages for solving and simulating from models based on ordinary differential equations (ODEs), and is the computational core that powers `nlmixr2`. Models are expressed in `rxode2`'s coding shorthand and subsequently compiled via C into dynamic link libraries optimized for speed. `rxode2` has several key components, split across several R packages (why this was done is complex, but boils down to CRAN objections to the time it took a single unified package to compile):

- `rxode2`: The core package
- `rxode2et`: Event table functions
- `rxode2parse`: The rxode2 parser
- `rxode2ll`: Log-likelihood functions for a wide range of statistical distributions
- `rxode2random`: Random-number generators for a wide range of statistical distributions

2.2.3.2 `nlmixr2`

`nlmixr2` is a set of R packages that performs parameter estimation based on an `rxode2`-format model and an input dataset. Like `rxode2`, `nlmixr2` is split into several smaller packages to accommodate CRAN concerns about the size of a single monolithic package. The core packages are:

- `nlmixr2`: The core package
- `nlmixr2data`: Datasets
- `nlmixr2est`: Estimation routines
- `nlmixr2plot`: Plotting functions
- `nlmixr2extra`: Supporting functions and miscellaneous add-ons

If you run into any unexpected issues, you can have a look at www.nlmixr2.org to see whether there's anything specific you need to do for your platform or version of R. If you don't have any luck, submit an issue on the `nlmixr2` GitHub site (github.com/nlmixr2/nlmixr2) and one of the developers will be in touch to help.

2.2.4 Supplemental packages

Although these packages are not absolutely required for `nlmixr2` to work, they add additional functionality, and you'll need them to get the most out of this book, so we strongly recommend that you install them as well.

- `ggPMX`: a toolkit providing a set of standardized diagnostic plots, designed from the ground up to play well with `nlmixr2`. Install using `install.packages("ggPMX", dependencies = TRUE)`.
- `shinyMixR`: a `shiny`-based graphical user interface for `nlmixr2`. The package provides a dashboard-like interface and helps in managing, running, editing and analysing `nlmixr2` models. `install.packages("shinyMixR", dependencies = TRUE)`.
- `xgxr`: a toolkit for exploring PKPD data. `install.packages("xgxr", dependencies = TRUE)`
- `xpose.nlmixr2`: an interface to the absurdly useful pharmacometric model goodness-of-fit toolkit `xpose`, it provides an array of pretty diagnostic plots. `install.packages("xpose.nlmixr2", dependencies = TRUE)`
- `nlmixr2lib`: a library full of sample models for `nlmixr2`, and the framework for creating new ones. `install.packages("nlmixr2lib", dependencies = TRUE)`
- `nlmixr2rpt`: a package for automating the reporting of `nlmixr2` analyses. This is accomplished by creating a YAML file that contains reporting options, figure and table generation code, and general content for Word and PowerPoint files. `install.packages("nlmixr2rpt", dependencies = TRUE)`

2.2.5 RStudio

RStudio is an integrated development environment (IDE) for R (and Python), and includes a console, a syntax-highlighting editor that supports direct code execution, and a plethora of tools for plotting, history, debugging, and workspace management. It comes in open-source and commercial packages, but the free "Desktop" version is perfectly sufficient for our needs and can be obtained from Posit at posit.co/download/rstudio-desktop. It is available for Windows, macOS and Linux and is pretty great.

2.2.6 Emacs

GNU Emacs is an extensible, customizable, free/libre text editor available for any platform you can think of most likely a few that you can't. It is both massively powerful and massively challenging to get to grips with, but `nlmixr2` was largely developed with it, and many of the development team swear by it (the rest of us swear at it). Still, if console-based editors are your thing, have at it. There is no better tool of its type. You can download it for your platform from www.gnu.org/software/emacs. Don't say we didn't warn you.

To get the most out of Emacs, you'll need the Emacs Speaks Statistics (ESS) add-on package. Visit ess.r-project.org to get started. Windows users, for example, would be best served downloading an all-in-one Emacs distribution that includes ESS baked in. Linux users can use their package managers to get up and running.

2.2.7 Other editors

There are several other IDEs that we're told work well, including Visual Studio Code (code.visualstudio.com) and its fully open-source variant VSCodium (vscodium.com). At the end of the day you can use anything you feel comfortable with, including the built-in R IDE. You might also have a look at Positron (positron.posit.co), Posit's new R and Python environment based on VS Code - it's still quite early in development, though, and we haven't tested it at all.

2.3 Docker

If all this local installation stuff looks daunting, you could always use our Docker image, which comes with everything pre-installed.

Docker (docker.com) describes itself as an open platform for developing, shipping, and running applications. "Dockerizing" a software package effectively separates the tool from the underlying operating system, by placing it in a loosely isolated environment called a container. Containers are lightweight and contain everything needed to run the application, so you can completely forget about dependencies and conflicts and just get on with the job.

To use the Dockerized version of `nlmixr2`, which comes with the latest versions of R, nlmixr2 and all its dependencies and relevant add-ons, as well as RStudio Server (a browser-based version of RStudio Desktop, also free and just as powerful), you need to install Docker for your platform, and then execute `docker run -v /myfiles/:/home/rstudio/myfiles -d -p 8787:8787 -e PASSWORD=nlmixr nlmixr/nlmixr2prod:V0.2`. This will download all the necessary components of the system and run the container. Note that RStudio Server cannot see the host operating system - it's running inside a virtual machine - so you need to "mount" a folder on your local system, in this case assumed to be `/myfiles`, to a folder in side the VM (`/home/rstudio/myfiles` in this example) if you intend to use your own files and transfer results back to the host. You'll need to change this suit your local system. This example also sets a password (`nlmixr`).

The first time will take a few minutes, but after that it will start up much more quickly. You can reach the system by navigating to https://127.0.0.1:8787. Full details and help with any troubleshooting that might be necessary can be found at github.com/RichardHooijmaijers/nlmixr.docker.

3 History

This book has been a long time in coming.

3.1 In the beginning, there was `RxODE`

Our story begins with `RxODE`. `RxODE` was developed by Melissa Hallow and Wenping Wang as an R package for facilitating quick and efficient simulations of ODE models (7), and when it was presented by Melissa Hallow at the PAGE meeting in Crete in 2015 (8), the idea was floated to use its machinery for parameter estimation using `nlme`, the R implementation of nonlinear mixed-effects models by Pinheiro and Bates (9). As it turned out, work on developing this concept was already pretty advanced by that time, and parameter estimation with both `nlme` and stochastic annealing expectation maximization (SAEM) (10) was implemented by Wenping by the end of that year.

3.2 Stan

The next milestone came at ACoP6 the same year, when Yuan Xiong first presented `PMXstan` (11). Applying fully Bayesian approaches to pharmacometric modeling has always been a challenging task, and `PMXstan` was proposed as a way to bridge the gap. Stan (12) implements gradient-based Markov chain Monte Carlo (MCMC) algorithms for Bayesian inference, stochastic, gradient-based variational Bayesian methods for approximate Bayesian inference, and gradient-based optimization for penalized maximum likelihood estimation, and can easily be run from R. However, before `PMXstan`, pharmacometricians had to write their own Stan code to describe PKPD models, and preparing data files was arduous and counter-intuitive for those used to event-based data files like those used in NONMEM. Also, there were no efficient ODE solvers that could handle stiff systems that would work with its No-U-Turn Sampler (NUTS) (13). `PMXstan` solved this by providing wrappers for the more unfriendly parts of the process, closed-form solutions for common PK systems written in Stan code, and a NUTS-compatible template LSODA solver to deal with stiff ODE systems. Significantly, these were components that would become quite important for a more general nonlinear mixed-effects (NLME) model fitting tool. (Since then, our colleagues at Metrum Research Group have taken things in all kinds of new and interesting directions with `Torsten`, a toolkit providing explicit pharmacometric functionality to Stan. But that, dear reader, is another story.)

3.3 GitHub

The first `nlmixr` commit to GitHub was on 19 October 2016, and by then a small team had sprung up around the project, with Wenping Wang and Yuan Xiong at its core within Novartis, and a small group of interested parties including Teun Post and Richard Hooijmaaijers at LAP&P and Rik Schoemaker and Justin Wilkins at Occams.

In December 2016, `nlmixr` was presented to the modeling group at Uppsala University, where the implementation of the first-order conditional estimation method with interaction (FOCEI) by Almquist and colleagues (14) was first discussed.

3.4 CRAN

Matt Fidler joined the team at Novartis in January 2017, and implemented the FOCEI method, bringing the number of available algorithms to three. June 2017 saw the introduction of a unified user interface across all three algorithms, a major milestone, and our first CRAN release was `nlmixr` 0.9.0-1 on 9 November 2017. An official 1.0 would follow in August 2018. By now the team had widened to include Mirjam Trame, who, together with Wenping, was using `nlmixr` as the core of a series of pharmacometric training courses in Cuba and elsewhere in Central and South America.

3.5 First peer-reviewed publications

Although `nlmixr` had been a regular fixture at PAGE and ACoP in the intervening years, our first major publication would arrive in 2019, in the form of a tutorial introducing `nlmixr` to the wider pharmacometric world (15), and two months later, a comparison of algorithms between `nlmixr` and its gold standard commercial alternatives (FOCEI in NONMEM and SAEM in Monolix) followed (16).

3.6 Streamlining and modularization

Installing `nlmixr` was, at this time, a complicated and daunting undertaking, and although many in the pharmacometrics community had taken to `nlmixr` with enthusiasm, this was a large disadvantage that, to be frank, was turning people off. It had long been necessary to use Python for handling some aspects of FOCEI fitting, and getting it to work properly together with R was *hard*. This was further complicated by CRAN's effective but very rigid package review and approval system, which was leading to endless problems with keeping the various dependencies `nlmixr` had in sync with one another. In April 2021, `nlmixr` 2.0 was unleashed upon the world, and Python was left behind forever. To say this was a relief to the development team was to understate the emotional catharsis that took place.

Although this solved one problem, another had been brewing. `nlmixr` had become a large package by R standards, and compile times at CRAN had begun to irk its administrators, leading to significant delays in approval. This eventually led to the decision to reimplement `nlmixr` as a series of closely linked, modular packages as opposed to a single monolithic unit. Rather than reverse-engineer the original `nlmixr`, the decision was taken to fork the project, and `nlmixr2` was born in February 2022. `nlmixr` would remain on GitHub, but would no longer be developed actively, while new features and ongoing improvements would be applied to `nlmixr2`. The first CRAN release of `nlmixr2` took place in June 2022.

Up to 26 March 2022, the date on which the last commit was made to the original version of `nlmixr`, there were 2,403 commits to the nlmixr repository and 17 more CRAN releases. `RxODE` had 4,860 commits and 33 CRAN releases (some before `nlmixr`'s time, but we're just going to go ahead and count them anyway).

3.7 Community enthusiasm

Over the years, we've hosted numerous tutorials at the major pharmacometrics meetings (PAGE, ACoP, PAGANZ and WCoP), and used `nlmixr` as the centrepoint for a series of well-received pharmacometrics courses in Cuba and elsewhere. We've also managed to publish a bit (17), as have others (we'll get into this later on).

Our tutorial in *Clinical Pharmacology & Therapeutics: Pharmacometrics & Systems Pharmacology* (15) was one of that journal's top ten most-read articles in 2021, with over 4,000 downloads. Our article had been one of the top 10% most-downloaded papers in 2018-2019, and having such interest for the second time in a row is tremendously encouraging for all of us! We hope it's a reflection of the enthusiasm the community is building for our tool, and hope that it will continue.

3.8 But why?

It's not about the money. Well, it is, but not in the way you think. It's safe to say that commercial gain is not a motivation for this project.

The money argument is twofold. Pharmacometrics is a small market, so developing commercial software in this space is very risky, and even if you succeed, any tool that is successfully developed will need to be very expensive to recoup one's investment, and there is always an indefinite commitment to support, which is a deceptively massive overhead. So it was clear right from the beginning that we were not going to go that way. On top of this, software licenses are a non-trivial operating cost, especially for smaller players like some of us, so there is a large incentive to find cheaper ways to do our work. But it wasn't completely this either, to be honest.

Starting out in pharmacometrics is challenging. It's a tough field, since it requires one to have a wide range of highly technical skills: pharmacology, math, statistics, computers, and so on. If you're sitting in a stuffy office in an underfunded university in a low-or-middle-income country 10,000 kilometres (literally) from the nearest pharmacometrics center of excellence, you have additional challenges. One: who is going to help you learn this stuff? Two: how are you going to afford the software tools you need? (Even at academic rates, NONMEM, Monolix and company are expensive, and hard to justify in a resource-poor environment when only a few students and staff are going to use it. Especially when hardware and power costs are also taken into account.) There was a crying need for an accessible, low-to-no cost tool to reduce this not-insignificant barrier to entry into our field.

So: Cost! Free software makes pharmacometric modelling (more) accessible in low to middle income countries. Not needing to buy a license makes preparing and giving courses much easier (many pharmacometrics courses are already using `nlmixr2`). Academic licenses might appear to solve this issue to an extent, but these usually cannot be used for commercial work making actual drug development in a low income environment difficult.

Curiosity! Can we make NLME parameter estimation work in R, where others have tried and failed? So far the answer seems to be yes.

Convenience! Having inputs, analyses and outputs in a single environment (R) is super attractive and makes workflows very efficient.

Creativity and collaboration! Whereas open science - of which we are massive fans, it should go without saying - allows one to 'stand on the shoulders of giants', adding open source to the mix allows everyone to take the software and run with it, developing applications we would never have thought of.

Finally: Concern! What will happen to NONMEM when its sole lead developer retires? Having a backup solution is very reassuring.

This book is the next step in our journey; we hope you'll take it with us.

Part II

Population Pharmacokinetics and Pharmacodynamics

4 A brief introduction to pharmacokinetics

It is not possible to give a comprehensive overview of pharmacokinetics (PK) and pharmacodynamics (PD) in a single chapter. Entire textbooks have been written on the subject, and it is a field that is constantly evolving. However, we present here a brief overview of the key concepts and how they relate to one another, with a view to using them as a springboard for `rxode2` and `nlmixr2`. For the interested reader, a much deeper exploration of these topics may be found in seminal works such as Rowland and Tozer (1) and Gabrielsson and Weiner (6), upon which this short summary is heavily based.

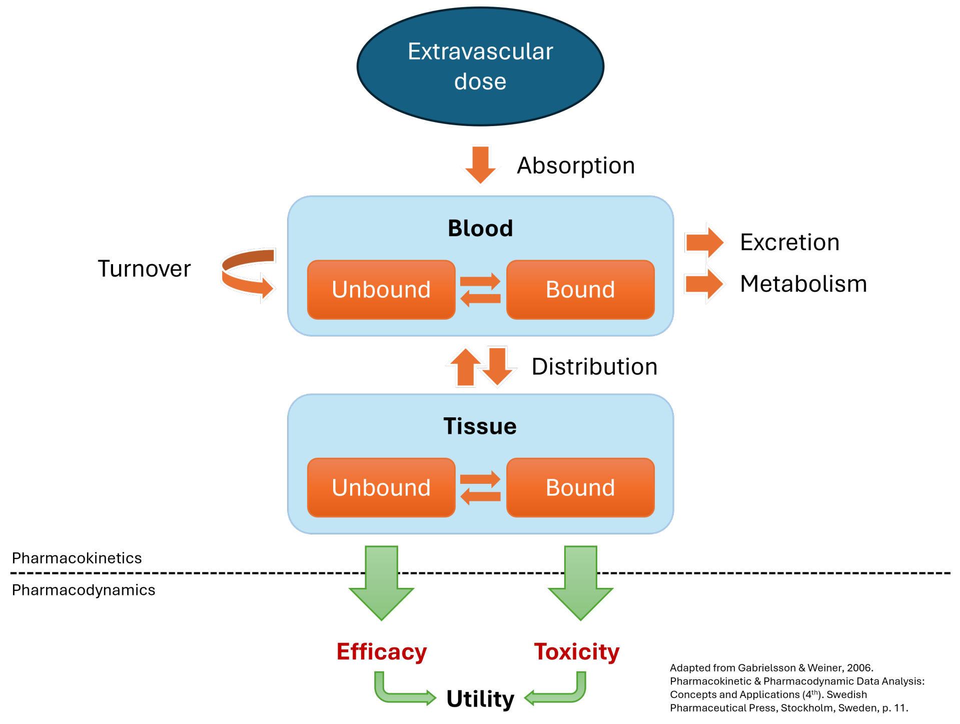
When the rest of the book is done we can add links to the various sections throughout this chapter.

4.1 Pharmacokinetics

Pharmacokinetics, or PK as it is usally referrred to, is the study of the time course of drug concentrations in different body spaces, such as plasma, blood, urine, cerebrospinal fluid, and tissues, and how the effect the time course of drug action, such as the onset, intensity and duration of action (6). Figure 4.1 provides an overview of the key processes involved in PK: absorption, distribution, metabolism and excretion. It's very important to understand these concepts clearly before we move into trying to model them. Figure 4.2 illustrates the path a drug might follow from administration to elimination.

4.1.1 Site of administration

Sites of drug administration are typically classified as intravascular (in which the drug is administered directly into the circulation, either intravenously or intra-arterially), or extravascular (in which the drug is administered outside the circulation, including oral, subcutaneous, sublingual, buccal, intramuscular, dermal, pulmonary, and rectal routes). Drugs administered extravascularly must be absorbed into the circulation before they can be distributed to their site of action.



Adapted from Gabrielsson & Weiner, 2006. Pharmacokinetic & Pharmacodynamic Data Analysis: Concepts and Applications (4th). Swedish Pharmaceutical Press, Stockholm, Sweden, p. 11.

Figure 4.1: PK in a nutshell: absorption, distribution, metabolism and excretion.

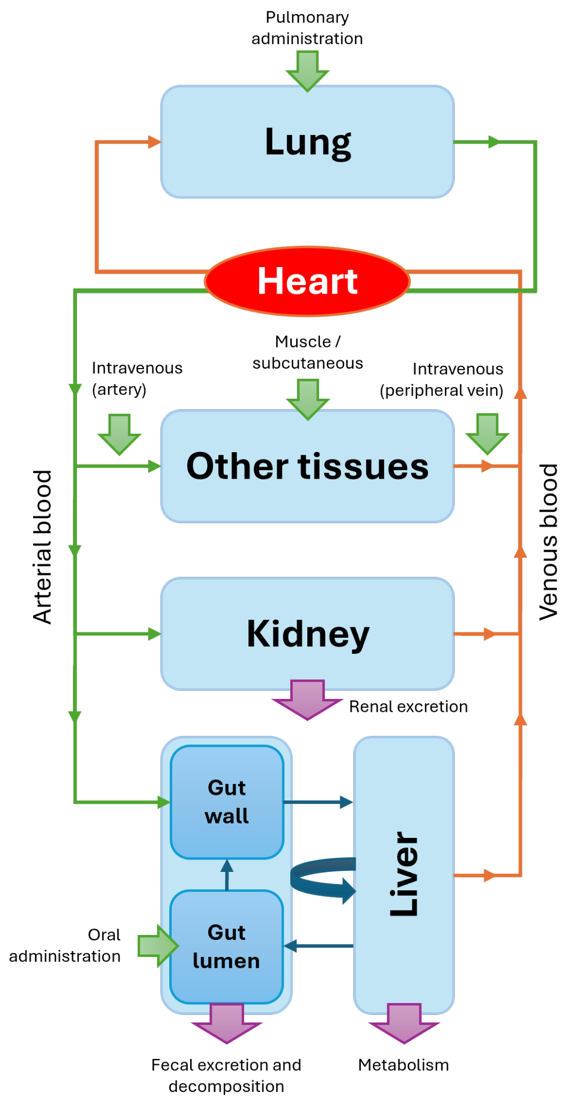


Figure 4.2: Schematic view of the body, illustrating the pathway taken by a drug from administration to elimination.

4.1.2 Absorption

Rowland and Tozer define *absorption* as the process by which unchanged drug moves from the site of administration to the site of measurement in the body (1). Absorption can be a surprisingly complex process that depends on the physicochemical properties of the drug, the route of administration, and the physiology of the recipient. The rate and extent of absorption can be influenced by a range of factors, including the solubility of the drug, the pH of the environment, the presence of food, the presence of other drugs, and the integrity of the gastrointestinal tract. In the case of oral administration, for example, drug travels from the gastrointestinal lumen through the intestinal wall and into the portal vein, where it is transported to the liver before entering the circulation, where it can be measured. In some circumstances, a fraction of the drug is lost as it is metabolized in the liver before it reaches the systemic circulation, a phenomenon known as the *first-pass effect*.

4.1.3 Disposition

After absorption, the drug is distributed throughout the body, delivered simultaneously to all tissues by the arterial blood. *Disposition* refers to both distribution and elimination, and is a blanket term covering all processes occurring subsequent to absorption.

Distribution is defined as the reversible transfer of drug to and from the site of measurement, which is usually blood or plasma. A simple example is the movement of drug between blood and muscle. A less simple example is enterohepatic cycling, in which drug is excreted into the bile by the liver, stored in the gall bladder, released with the bile into the small intestine and reabsorbed into the circulation. Distribution depends heavily on a range of factors, including organ blood perfusion, organ size, permeability of tissue membranes, blood and tissue binding, and the physicochemical properties of the drug.

Distribution primarily determines the early rapid decline in plasma concentration subsequent to dose administration. Eventually, however, equilibrium is reached between the concentration of drug in the plasma and the concentration in the tissues, and the body can be thought of as a single container, or *compartment.* Since decline in observed concentrations is due only to elimination once this state is reached, this phase is often termed the *elimination phase.*

Elimination is defined as the irreversible loss of drug from the site of measurement, and occurs by two processes: excretion and metabolism. Elimination takes place mostly via the kidneys and the liver, with smaller contributions from the lungs, skin, and other routes. Metabolism takes place primarily in the liver, while excretion of unmetabolized drug occurs primarily via the kidneys.

4.1.4 Routes of administration, and some important nomenclature

4.1.4.1 Intravascular dosing

Figure 4.3 provides an illustration of concentration over time for an intravenously-administered drug with first-order elimination. The concentration of drug in the plasma decreases exponentially over time, with the rate of decrease proportional to the concentration of drug in the plasma. This is known as *first-order elimination*, and is the most common type of elimination process observed in PK.

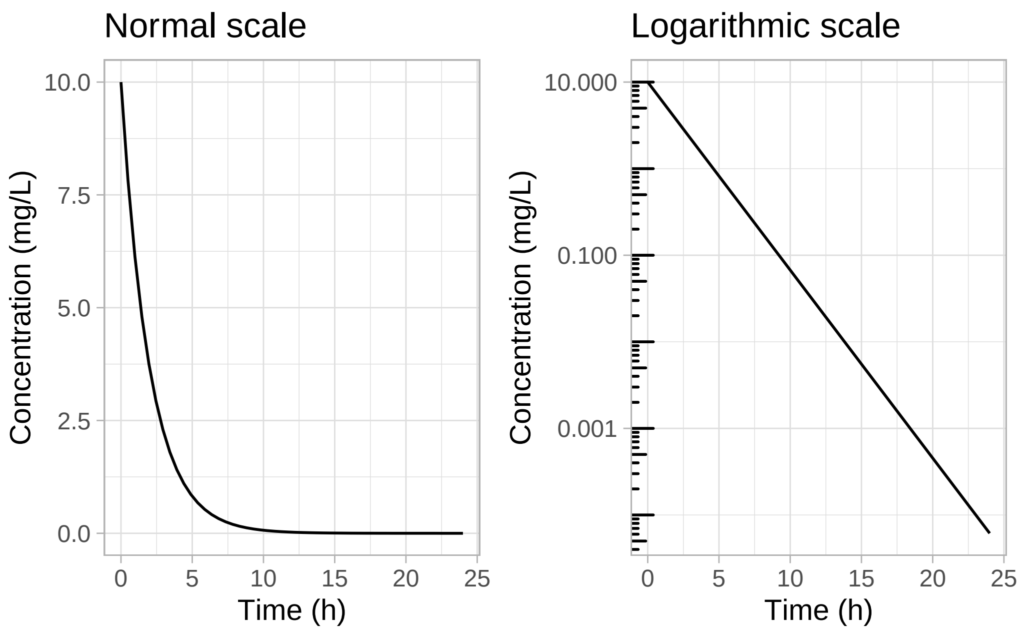


Figure 4.3: Concentration over time after an intravascular dose.

The elimination phase is described by two mathematical parameters: the *elimination half-life* ($t_{1/2}$) and the *apparent volume of distribution* (Vd).

- **Elimination half-life** ($t_{1/2}$): the time required for the plasma concentration of drug (and the amount of drug in the body) to decrease by half, and is a useful measure of the rate of elimination. The elimination half-life is related to the *elimination rate constant*, k, by the equation $t_{1/2} = \frac{ln(2)}{k}$ (assuming a one-compartment system with first-order elimination - we will get to this later).
- **Volume of distribution** (Vd): the hypothetical volume of fluid into which a drug would have to be dissolved to produce the observed concentration. It is a function of the dose and the extent of distribution of a drug into tissues. Calculation of Vd requires that concentration be at equilibrium between the plasma and the tissues.

- **Clearance** (CL): the volume of plasma from which the drug is completely removed per unit time. It is a measure of the efficiency of elimination of a drug from the body, and is calculated as the product of the elimination rate constant k and the apparent volume of distribution Vd: $CL = k \cdot Vd$.
- **Area under the concentration-time curve** (AUC): the total amount of drug that has entered the systemic circulation over time over a given period, a measure of the extent of exposure to the drug. It is calculated by integrating the concentration-time curve over time. It may also be calculated as

  $$AUC = \frac{Dose}{CL}$$

  , where $Dose$ is the amount of drug administered.

4.1.4.2 Extravascular dosing

The key difference between intravascular and extravascular dosing in PK is the absorption process that must take place before the drug reaches the systemic circulation. Figure 4.4 provides an illustration of concentration over time for an orally-administered drug with first-order absorption and elimination.

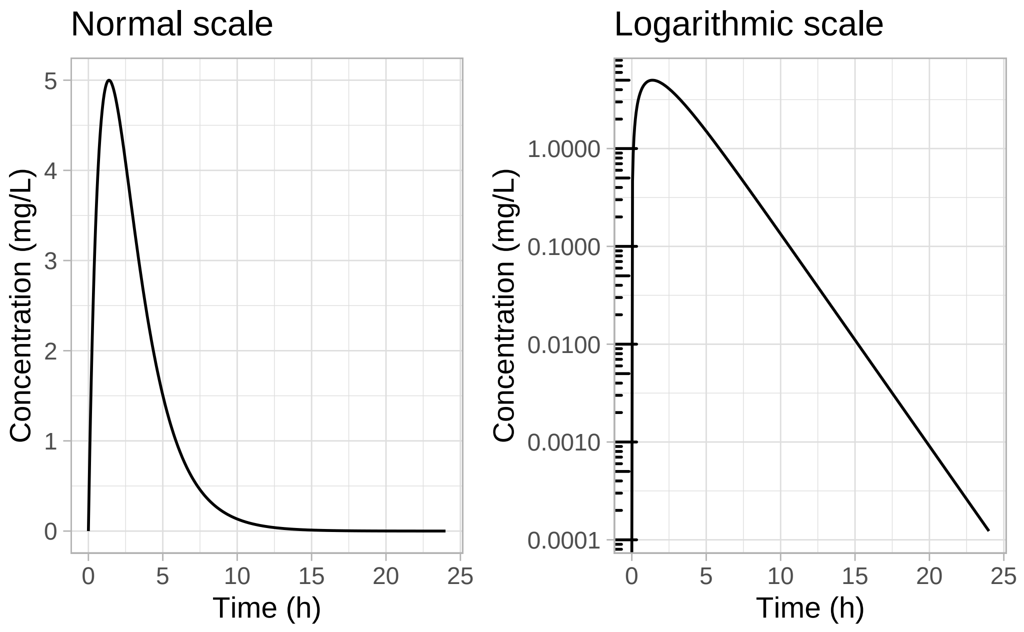


Figure 4.4: Concentration over time after an extravascular dose.

Although often very complex, absorption processes can commonly be approximated as a first-order process.

Key concepts here include -

- The **absorption rate constant** (ka), which governs the absorption process. In some circumstances, zero-order absorption processes are observed, in which the rate of absorption is constant over time.
- **Bioavailability**, which is a measure of the fraction of the administered dose that reaches the systemic circulation, the remainder being lost to elimination processes taking place during absorption. Bioavailability is calculated as the ratio of the AUC after extravascular dosing to the AUC after intravascular dosing:

$$F = \frac{AUC_{extravascular}}{AUC_{intravascular}}$$

4.1.5 PK metrics

In addition to AUC, $t_{1/2}$, Vd, and CL, a number of other metrics are often used to describe the PK of a drug, including:

- **Peak concentration** (C_{max})
- **Time to peak concentration** (T_{max})
- **Minimum concentration** (C_{min} or C_{trough}, typically evaluated immediately before the next dose)
- **Average concentration** (C_{avg}, calculated as the AUC divided by the dosing interval)

4.1.6 Noncompartmental analysis

Noncompartmental analysis (NCA) is not a topic we shall be spending much time on, but since we'll be illustrating how we can use NCA for automatically generating initial estimates for population PK modeling later on, let's take a quick look.

NCA is applied to estimate PK metrics and parameters directly from observed concentration-time measurements. The NCA approach is relatively versatile in that no specific model is assumed, and it is usually sufficiently accurate to for many purposes. Total drug exposure can be estimated by through AUC, using the trapezoidal rule (numerical integration). The number of time points available in order to perform a successful NCA analysis should be enough to cover the absorption, distribution and elimination phase to accurately characterize the drug, and the timing of samples must be adequate to ensure reasonably accurate estimation of AUC. Besides AUC, NCA can be applied to provide estimates of a range of PK metrics and parameters, including C_{max}, T_{max}, C_{min}, CL and Vd.

4.1.7 Compartmental analysis

Compartmental analysis is a more sophisticated approach to PK modeling, in which the body is conceptualized as a series of interconnected compartments, each representing a different tissue or organ, or set of tissues and organs. The simplest compartmental model is the one-compartment model, in which the body is treated as a single compartment in which drug is distributed and eliminated. The one-compartment model is the most commonly used model in PK, and is often sufficient to describe the PK of many drugs. More complex models, such as the two-compartment model, the three-compartment model, and the physiologically-based pharmacokinetic (PBPK) model are used when the one-compartment model is insufficient to describe the PK of a drug.

Ordinary differential equations (ODEs) are used to define compartmental models.

4.1.7.1 One-compartment model

The one-compartment model is the simplest compartmental model, and is often sufficient to describe the PK of many drugs. The one-compartment model is illustrated in Figure 4.5.

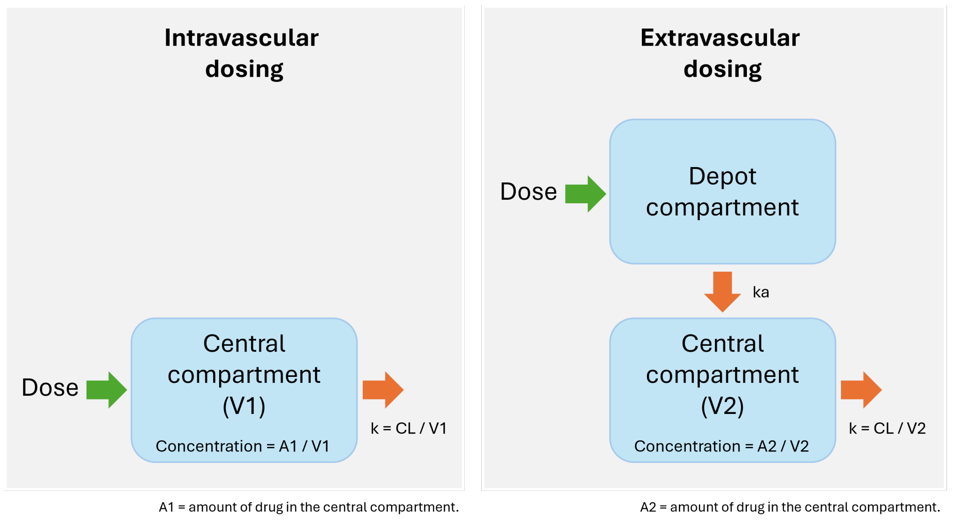


Figure 4.5: The one-compartment model.

Here, the body is considered to be a single large bucket in which drug is dissolved. Drug enters as a single bolus or infusion (in the intravascular case) or from the depot compartment (in the extravascular case, via absorption rate constant ka) and is thereafter eliminated according to elimination rate constant k. The movement of drug is governed by ODEs.

For the intravascular case,

$$\frac{dA_{Central}}{dt} = -k \cdot A_{Central}$$

where A_{Depot} is the amount of drug in the central compartment.

For the extravascular case,

$$\frac{dA_{Depot}}{dt} = -ka \cdot A_{Depot}$$

$$\frac{dA_{Central}}{dt} = ka \cdot A_{Depot} - k \cdot A_{Central}$$

where A_{Depot} is the amount of drug in the depot compartment, and $A_{Central}$ is the amount of drug in the central compartment.

Plotting concentration against time on the semilogarithmic scale will yield a straight line, with the slope of the line equal to k (Figure 4.6).

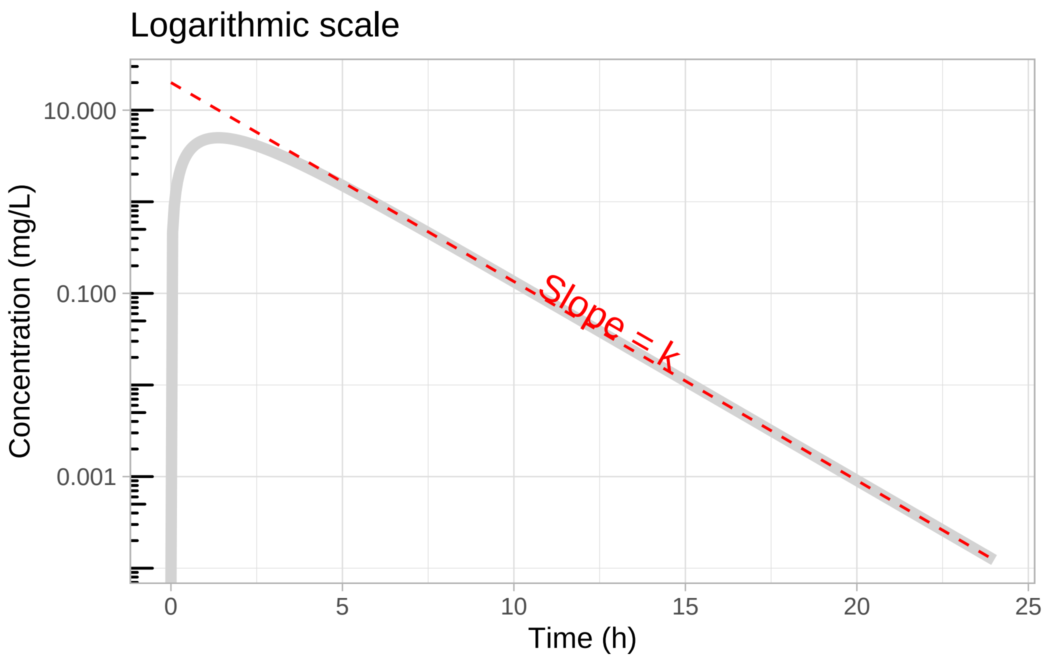


Figure 4.6: Concentration plotted against time for the one-compartment model.

4.1.7.2 Two-compartment model

The two-compartment model is a more complex model that is used when the one-compartment model is insufficient to describe the PK of a drug. The two-compartment model is illustrated in Figure 4.7.

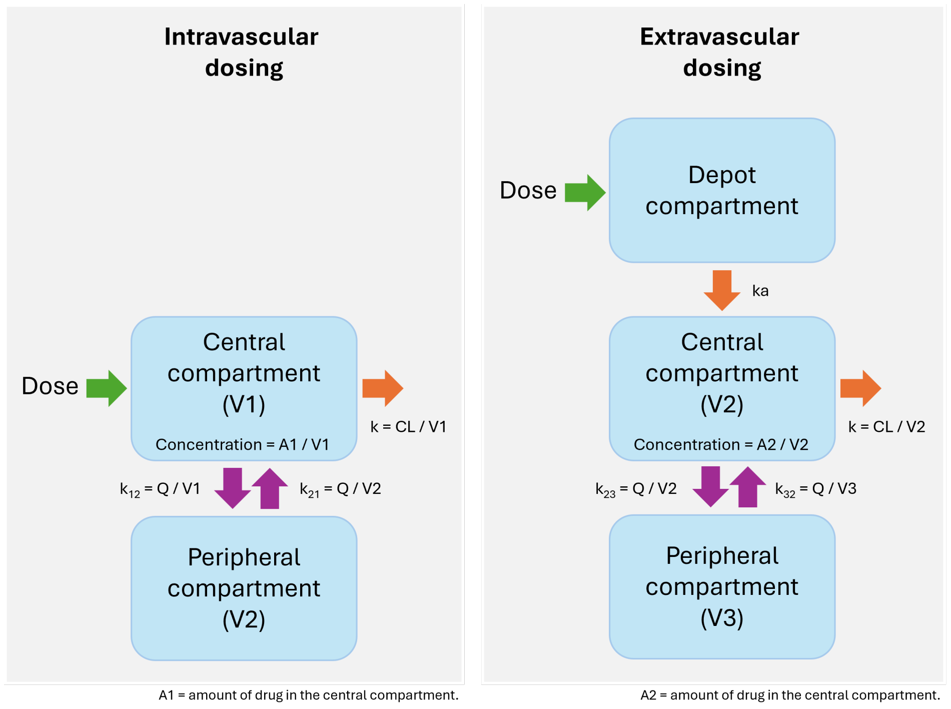


Figure 4.7: The two-compartment model.

Here, the body is considered to be divided into two compartments: the central compartment, which represents the plasma and has volume of distribution V1, and the peripheral compartment, which represents the tissues and has volume of distribution V2. Drug enters the central compartment as a single bolus or infusion (in the intravascular case) or from the depot compartment (in the extravascular case, via absorption rate constant ka) and is thereafter distributed to the peripheral compartment. Drug is eliminated from the central compartment according to elimination rate constant k, and is transported between the peripheral compartment and the central compartment according to rate constants k_{12} and k_{21}, which may be calculated from the *intercompartmental clearance* (Q) as $Q/V1$ and $Q/V2$, respectively. It is worth noting that for the extravascular case, V1 and V2 may sometimes be called V2 and V3, respectively (since the depot compartment is considered to be the first compartment).

The two-compartment model for an orally-dosed drug is defined by the following ODEs:

$$\frac{dA_{Depot}}{dt} = -ka \cdot A_{Depot}$$

$$\frac{dA_{Central}}{dt} = ka \cdot A_{Depot} - k \cdot A_{Central} - k_{12} \cdot A_{Central} + k_{21} \cdot A_{Peripheral}$$

$$\frac{dA_{Peripheral}}{dt} = k_{12} \cdot A_{Central} - k_{21} \cdot A_{Peripheral}$$

where A_{Depot} is the amount of drug in the depot compartment, $A_{Central}$ is the amount of drug in the central compartment, $A_{Peripheral1}$ is the amount of drug in the first peripheral compartment, k_{12} is the rate constant governing the transfer of drug from the central to the first peripheral compartment, k_{21} is the rate constant governing the transfer of drug from the first peripheral compartment to the central compartment, $A_{Peripheral2}$ is the amount of drug in the second peripheral compartment, k_{13} is the rate constant governing the transfer of drug from the central to the second peripheral compartment, and k_{31} is the rate constant governing the transfer of drug from the second peripheral compartment to the central compartment.

Concentration versus time is plotted for an oral two-compartment model in Figure 4.8. Notice that concentration on the semilogarithmic plot no longer declines at a constant rate, as it did in the one-compartment model. Plotting the observed data thus often provides a straightforward way of choosing an initial number of compartments to explore.

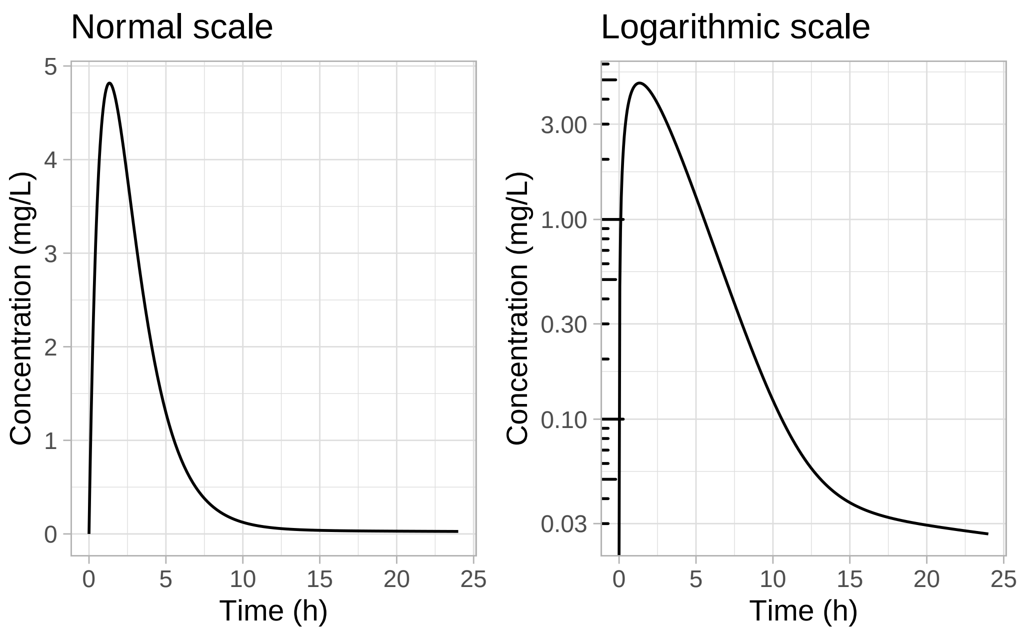


Figure 4.8: Concentration over time in a two-compartment model.

4.1.7.3 Three-compartment model

The three-compartment model (Figure 4.9) is less commonly applied than the one- and two-compartment models, but can be used when less complex models are inadequate.

It is very seldom that more complex structural models are necessary to describe the PK of a drug, unless PBPK modeling is undertaken. The differences in concentration-time curves generated by two- and three-compartment models can often be subtle, and differentiating them from one another requires rich data.

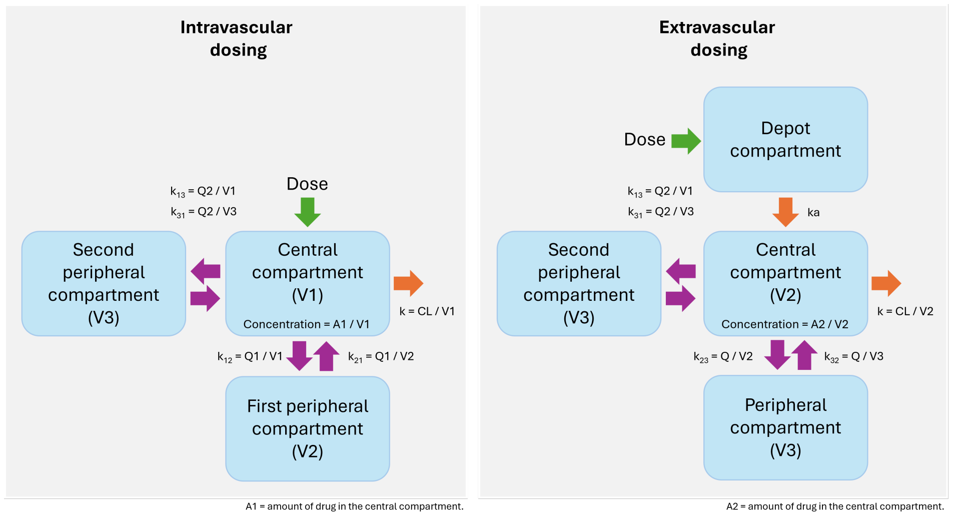


Figure 4.9: The three-compartment model.

The three-compartment model adds an additional volume of distribution for the second peripheral compartment (V3), and an additional intercompartmental clearance parameter governing the transfer of drug to and from the central compartment (Q2). For an orally-dosed drug, the system is defined by the following ODEs:

$$\frac{dA_{Depot}}{dt} = -ka \cdot A_{Depot}$$

$$\frac{dA_{Central}}{dt} = ka \cdot A_{Depot} - k \cdot A_{Central} - k_{12} \cdot A_{Central} + k_{21} \cdot A_{Peripheral1} - k_{13} \cdot A_{Central} + k_{31} \cdot A_{Peripheral2}$$

$$\frac{dA_{Peripheral1}}{dt} = k_{12} \cdot A_{Central} - k_{21} \cdot A_{Peripheral1}$$

$$\frac{dA_{Peripheral2}}{dt} = k_{13} \cdot A_{Central} - k_{31} \cdot A_{Peripheral2}$$

where A_{Depot} is the amount of drug in the depot compartment, $A_{Central}$ is the amount of drug in the central compartment, $A_{Peripheral1}$ is the amount of drug in the first peripheral compartment, k_{12} is the rate constant governing the transfer of drug from the central to the first peripheral compartment (calculated as $Q1/V1$), k_{21} is the rate constant governing the transfer of drug from the first peripheral compartment to the central compartment (calculated as $Q1/V2$), $A_{Peripheral2}$ is the amount of drug in the second peripheral compartment, k_{13} is the rate constant governing the transfer of drug from the central to the second peripheral compartment (calculated as $Q2/V1$), and k_{31} is the rate constant governing the transfer of drug from the second peripheral compartment to the central compartment (calculated as $Q2/V3$).

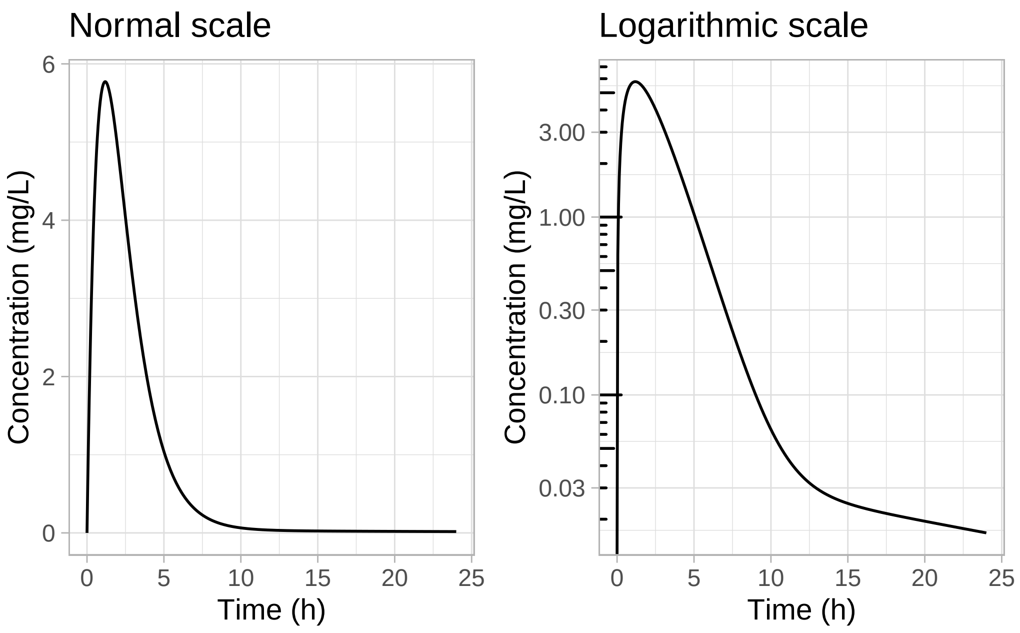


Figure 4.10: Concentration over time in a three-compartment model.

4.1.7.4 Physiologically-based pharmacokinetic (PBPK) modeling

PBPK models, by contrast to simpler empirical compartmental models, are informed by the anatomical, physiological and biochemical characteristics of the body. They are composed of compartments corresponding to organs or tissues, with connections corresponding to blood or lymph flows, or less commonly, diffusion processes. ODEs and their parameters represent blood flows, pulmonary ventilation rate, and organ volumes, using a priori information extracted from the scientific literature. PBPK models, although much more complex than other types of PK model, are particularly useful for describing and predicting inter-species and paediatric scaling, and extrapolation from one mode of administration to another. Figure 4.11 is a schematic of a simple perfusion-limited PBPK model (permeability-limited drug uptake into the tissues is also possible, but is slightly more complex).

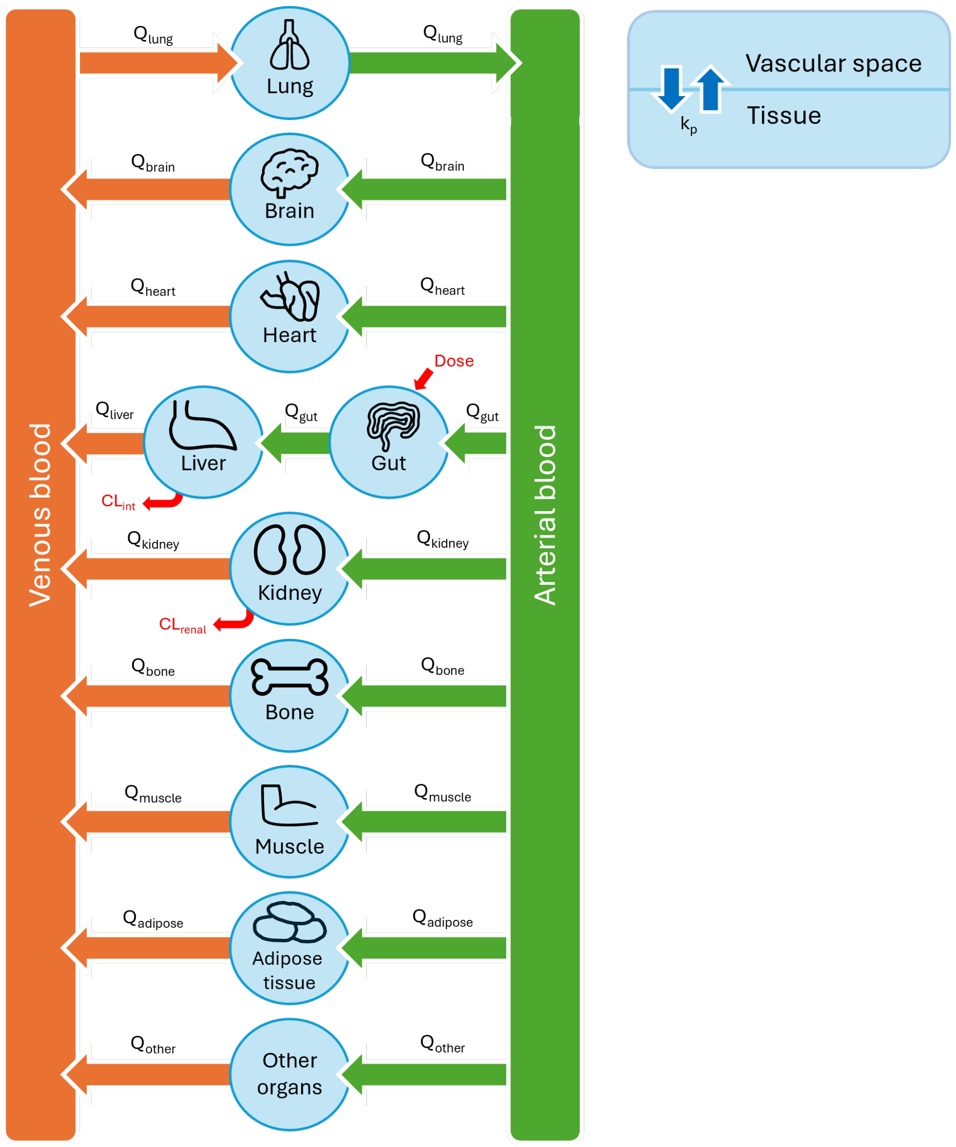


Figure 4.11: Example of a physiologically-based PK model.

The dynamic change in drug concentration in each tissue compartment is thus defined by:

$$\frac{dC_T}{dt} = \frac{Q_T}{V_T} \cdot \left(C_{ven/art} - \frac{C_T}{k_{b,T}} \right)$$

where C_T is the concentration in the tissue, Q_T is the blood flow to the tissue, V_T is the volume of the tissue, $C_{ven/art}$ is the concentration in the venous (for the lung) or arterial blood (for all other tissues), and $k_{b,T}$ is the tissue-blood partition coefficient.

Concentration in the arterial blood is defined by:

$$\frac{dC_{art}}{dt} = \frac{Q_{lung}}{V_{art}} \cdot \left(\frac{C_{lung}}{k_{B,lung}} - C_{art} \right)$$

where Q_{lung} is the blood flow to the lung, V_{art} is the volume distribution of the arterial blood, C_{lung} is the concentration in the lung, and $k_{B,lung}$ is the lung-blood partition coefficient. Concentration in the venous blood is defined by:

$$\frac{dC_{ven}}{dt} = \left(\sum \frac{Q_T}{V_{ven}} \cdot \frac{C_T}{k_{B,T}} \right) - \frac{Q_{lung}}{V_{ven}} \cdot C_{ven}$$

where V_{ven} is the volume of distribution of the venous blood. In this example, $\sum \frac{Q_T}{V_{ven}} \cdot \frac{C_T}{k_{B,T}}$ includes all tissues except the gut and the liver.

4.2 Population PK

Population PK modeling is an approach used to describe the PK of a drug in a population of individuals, rather than in a single individual. Population PK models are used to describe the variability in PK parameters between individuals, and to identify factors that influence the PK of a drug. Population PK models are often used to optimize dosing regimens, to predict the PK of a drug in a new population, and to identify factors that influence the PK of a drug. Models for populations will be the focus of this book.

4.3 Pharmacodynamics

Pharmacodynamics, or PD as it is usually referred to, is the study of the time course of the biological effects of a drug, how these effects are related to the exposure of the drug, and associated mechanisms of drug action (6). Responses can be graded (such as blood pressure or heart rate) or quantal (such as the occurrence of an adverse event).

Some terms should be defined before we delve any deeper: we have based our summary on Gabrielsson and Weiner (6).

- **Receptors** are macromolecules that interact with drugs to produce a biological response through signaling between and within cells.

- **Agonists** are ligands that bind to receptors and activate them, producing a biological response. Conventional agonists increase the proportion of activated receptors, thus increasing the biological response, whereas inverse agonists reduce it.
- **Antagonists** are ligands that act by blocking the effects of other compounds.
- **Potency** describes the activity of a compound, usually in terms of the concentration or amount of a drug necessary to produce a given effect.
- $\mathbf{EC_{50}}$ is the concentration of a drug that produces 50% of the maximum effect, and is a measure of potency.
- $\mathbf{IC_{50}}$ is the concentration of a drug that reduces the effect of interest to 50% of its former value. It, too, is a measure of potency.
- **Efficacy** is the degree to which different agonists produce varying responses, even when the same proportion of receptors is occupied.

4.3.1 Law of Mass Action

The idea of receptor occupancy - the proportion of receptors occupied by a drug - is based on the Law of Mass Action, which states that the rate of a chemical reaction is proportional to the product of the concentrations of the reactants. The Law of Mass Action can be applied to describe the relationship between drug concentration and the biological response.

$$[Drug] + [Receptor] \rightleftharpoons [Drug - Receptor] \rightarrow E$$

where E is a biological effect. We can reformulate this as a differential equation:

$$\frac{d[RC]}{dt} = k_{on} \cdot [R] \cdot [C] - k_{off} \cdot [RC]$$

where $[RC]$ is the concentration of the drug-receptor complex in molar units, $[R]$ is the concentration of the receptor in molar units, $[C]$ is the concentration of the drug in molar units, k_{on} is the association rate constant, and k_{off} is the dissociation rate constant. At equlibrium, the rate of change of [RC] is zero, so we can rearrange as follows:

$$\frac{[C] \cdot [R]}{[RC]} = \frac{k_{off}}{k_{on}} = K_d$$

$\mathrm{K_d}$ is the equilibrium dissociation constant, and is a measure of the drug-receptor *affinity*. The lower the $\mathrm{K_d}$, the higher the affinity.

If we assume that the total number of receptors in the system is $[R_{tot}]$, and is equal to $[RC]$ + $[R]$, we can rearrange to get:

$$\frac{[C] \cdot ([R_{tot}] - [RC])}{[RC]} = \frac{k_{off}}{k_{on}} = K_d$$

Rearranging a bit further, we get:

$$\frac{[RC]}{[R_{tot}]} = \frac{[C]}{[C] + K_d}$$

Response can be thought of as being directly proportional to the fractional occupancy of receptors by drug, or put another way, to the concentration of the drug-receptor complex (18). This gives us the *intrinsic efficacy* ϵ, described by the following equation:

$$\epsilon = \frac{[RC]}{E}$$

The maximum possible response attributable to drug, E_{max}, occurs when all the receptors are occupied:

$$E_{max} = \alpha \cdot [R_{tot}]$$

where α is efficacy.

Thus:

$$\frac{E}{E_{max}} = \frac{[C]}{[C] + K_d}$$

so

$$E = \frac{E_{max} \cdot [C]}{[C] + K_d}$$

This equation ought to be relatively familiar if you've ever encountered the *Hill equation*, which is a special case of this expression.

We could go into further detail here, but we'll leave it at that for now. The interested reader is encouraged to consult Gabrielsson and Weiner (6) for a more detailed exploration of the topic.

4.3.2 Pharmacodynamic models

In vitro binding models are all very well, but what we are really interested in - for the purposes of this book, anyway - is models for *in vivo* responses. For most purposes, we can assume we are at equilibrium or steady state.

4.3.2.1 Linear effect-concentration models

These are pretty simple - in effect, straight lines. All we need to describe the relationship between drug concentration C and effect E is a linear equation:

$$E = S \cdot C$$

where S is the slope of the line. This assumes an intercept of zero, which is almost never the case, so we would normally add an intercept:

$$E = E_0 + S \cdot C$$

where E_0 is the baseline value of the effect we are describing.

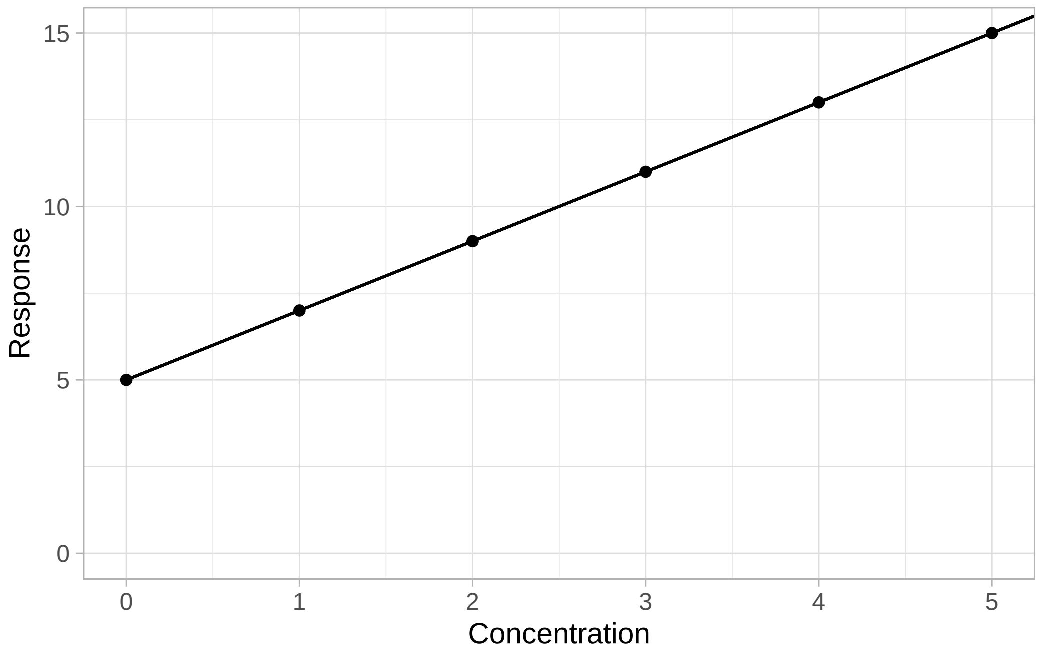


Figure 4.12: Linear concentration-effect model with a baseline effect.

Biology, of course, almost never follows straight lines.

4.3.2.2 Log-linear effect-concentration models

Sometimes, the relationship between concentration and effect may not be linear. Natural log-transformation of the concentration can sometimes help:

$$E = E_0 + S \cdot \ln(C)$$

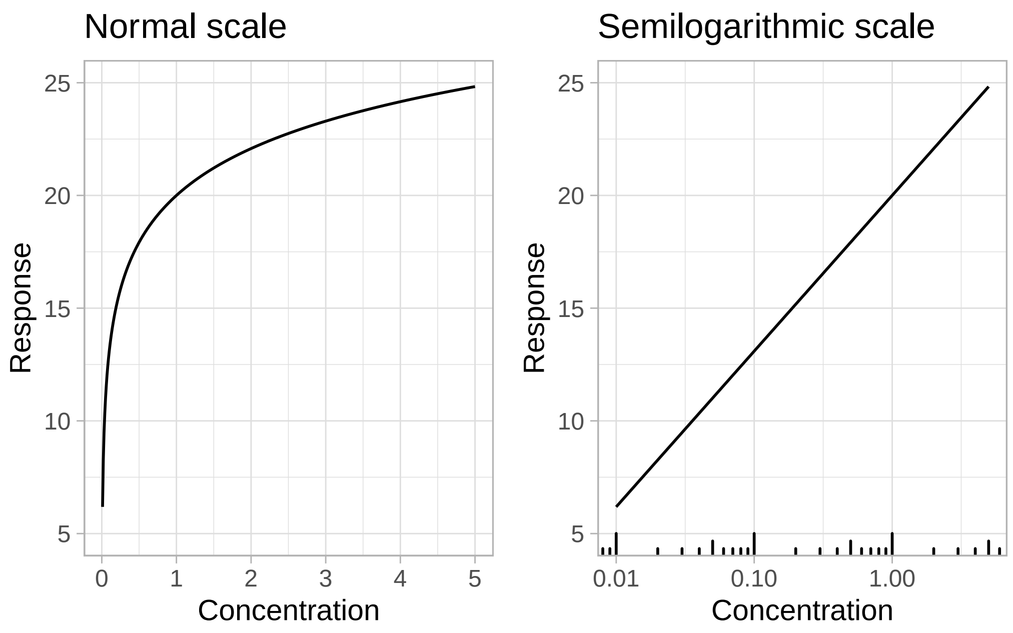


Figure 4.13: Linear concentration-effect model with a baseline effect.

This model, like its linear counterpart, assumes that the effect is proportional to the concentration of the drug. Its simplicity is attractive, but it has no biological basis, and in the specific case of the log-linear model, C can never be zero.

4.3.2.3 Emax model

The $\mathrm{E}_{\mathrm{max}}$ model is the workhorse of pharmacodynamic models, being relatively simple but based on biological theory.

K_d is often referred to as EC_{50} in situations in which we need to correlate the concentration of a drug with a biological effect, and in this context is the concentration corresponding to the half-maximal difference between the baseline E_0 and maximum effect E_{max}.

So, then, $\mathrm{E}_{\mathrm{max}}$ can be expressed as -

$$E = E_0 + \frac{E_{max} \cdot C}{EC_{50} + C}$$

This reflects an agonistic effect, but we can also model antagonistic effects. In this case, we can use the following equation:

$$E = E_0 - \frac{I_{max} \cdot C}{IC_{50} + C}$$

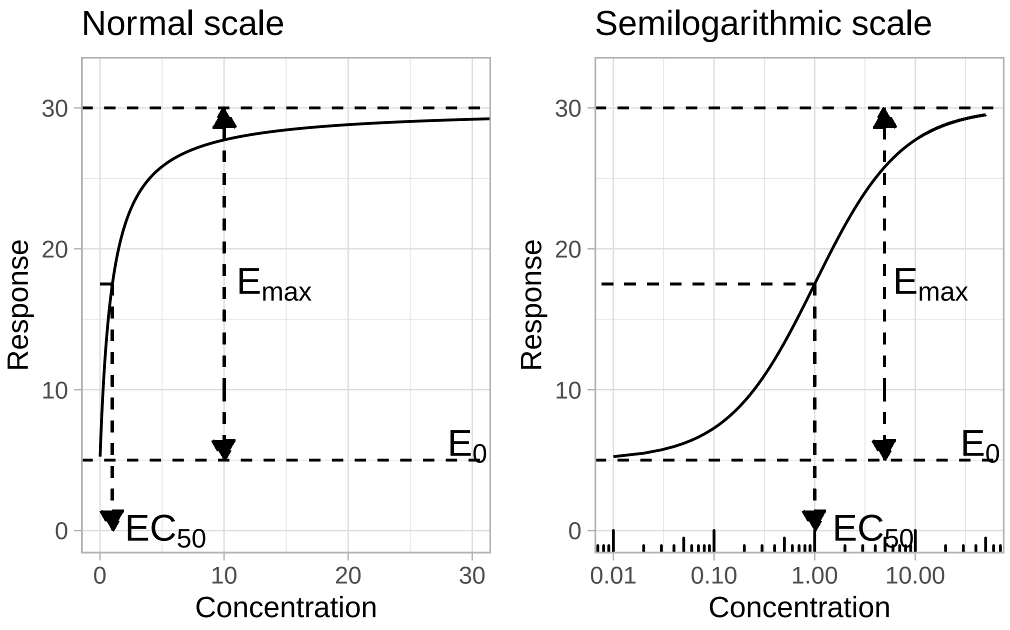


Figure 4.14: E_{max} model with a baseline effect.

where I_{max} is the maximum inhibitory effect, and IC_{50} is the concentration of the drug that produces 50% of I_{max}.

One could also parameterize the E_{max} model such that E_{max} is the fractional increase in effect over baseline, rather than the absolute increase:

$$E = E_0 \cdot \left(1 + \frac{E_{max} \cdot C}{EC_{50} + C}\right)$$

(and similarly for the antagonistic model).

4.3.2.4 Sigmoid Emax model

The E_{max} model is a special case of the sigmoid E_{max} model, which is a more general model that can describe a wider range of concentration-effect relationships. The sigmoid E_{max} model (or *Hill equation*, as it is often known) is defined as:

$$E = \frac{E_{max} \cdot C^n}{EC_{50}^n + C^n}$$

where n is the Hill coefficient, which describes the steepness of the concentration-effect curve. A Hill coefficient of 1 indicates a linear relationship between concentration and effect (the basic E_{max} model, in other words), while a Hill coefficient greater than 1 indicates a sigmoidal relationship. The Hill coefficient can also be less than 1, in which case the curve is sigmoidal but concave.

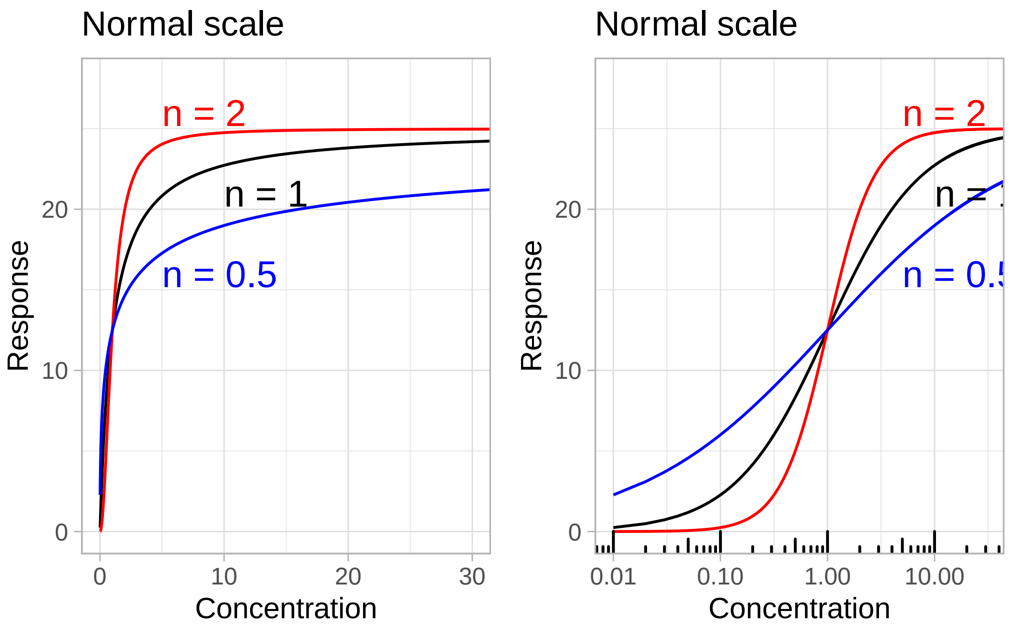


Figure 4.15: Sigmoid E_{max} model.

When two drugs exert effects on a given system, or when a single drug exerts an effect on a single receptor, things become more complicated. Composite E_{max} models, in which two E_{max} models with different E_{max}/I_{max} and EC_{50}/IC_{50} parameters can be applied in situations in which biphasic response is observed. An example would be *u-shaped* or *bell-shaped* exposure-response curves. More detail can be found in Gabrielsson and Weiner (6).

4.3.3 Kinetics of response

Pharmacodynamic models are all very nice, but it is is crucial to understand the concentration changes over time, and thus, so does response. Pharmackinetic-pharmacodynamic (PK-PD) models provide a useful way to tackle this.

4.3.3.1 Direct effect models

With direct effect models, concentration and response are assumed to be in instant equilibrium, and changes in concentration instantaneously affect response. Figure 4.16 provides an illustration.

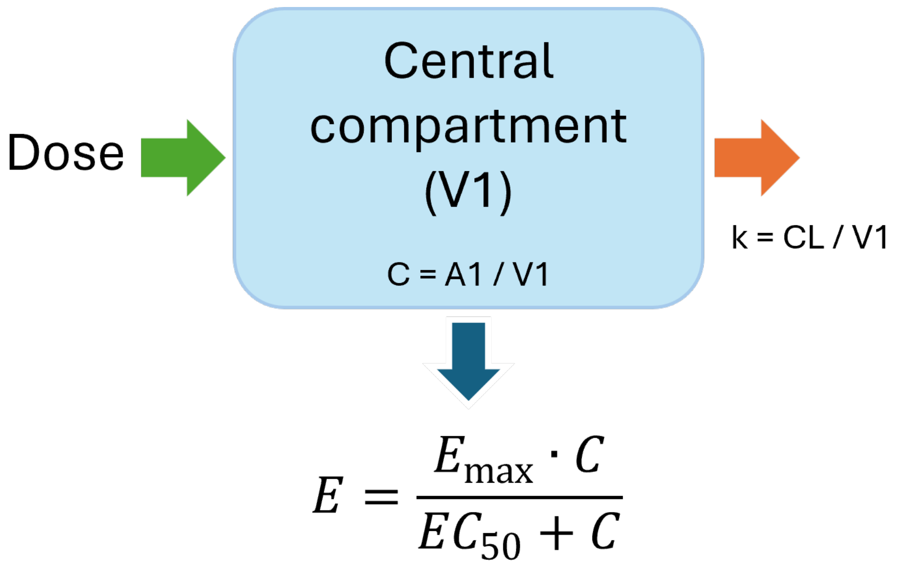


Figure 4.16: The direct effect model.

4.3.3.2 Indirect effect models

Sometimes, response does not occur instantaneously, but instead takes time to develop, without an obvious link to the observed plasma concentration, a state of affairs called *hysteresis* Figure 4.17. There are a number of ways to handle this time delay.

Indirect effect models, also called turnover models or effect compartment models, describe the "turnover" of observed responses in response to stimuli, in this case drug concentration. They tend to be (more) physiologically based and their parameters can often take a mechanistic meaning. Per the seminal paper by Dayneka et al (19), there are four main types of indirect effect models Figure 4.18.

Figure 4.19 shows examples of our four models. The choice of which one to use should be informed by the analyst's knowledge of the underlying biology more than anything else - what do we know about what the drug is doing at the receptor level? These models reflect reversible binding; irreversible binding is also possible, but is beyond the scope of this discussion. The article by Dayneka, Garg and Jusko (19) provides an excellent overview of the topic.

4.3.3.3 Effect compartment (link) models

Effect compartment models are a special case of indirect effect models, in which the drug effect is delayed due to the time it takes for the drug to reach the site of action (the biophase). The effect compartment is a hypothetical compartment that represents the biophase - drug effect is assumed to be proportional to the concentration of the drug in this compartment. After the

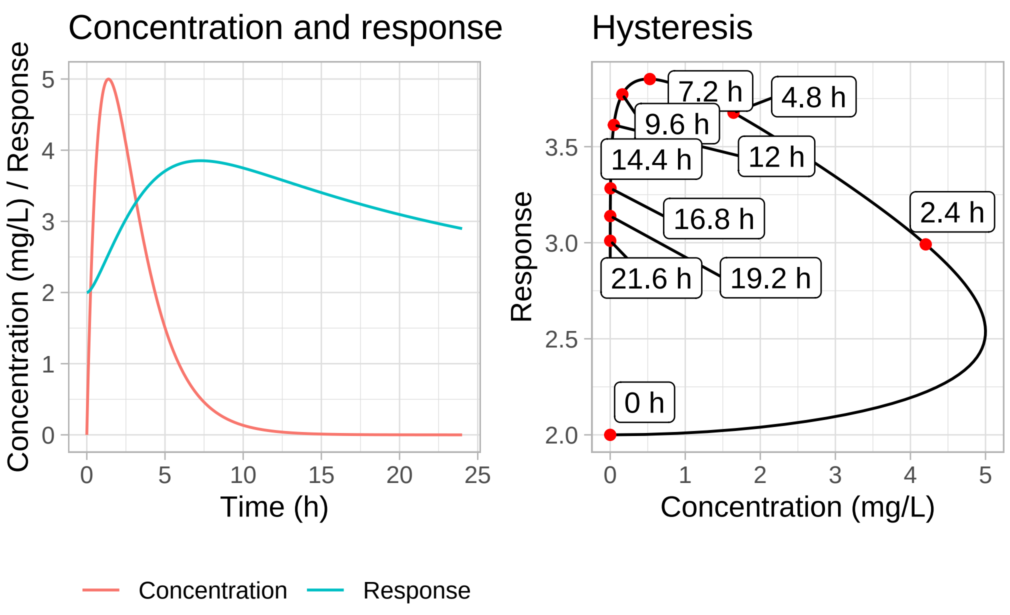
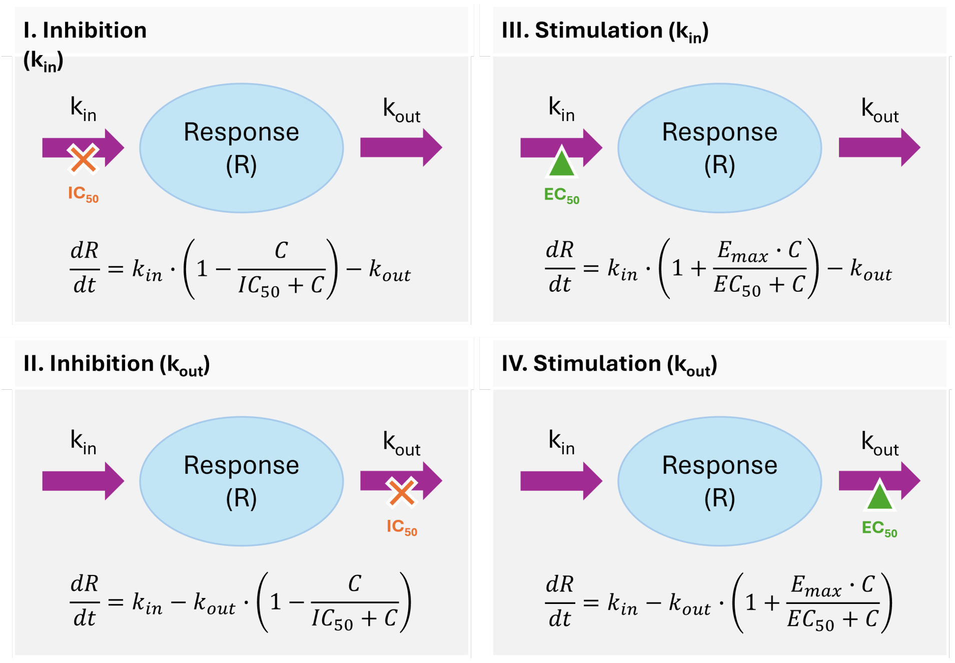


Figure 4.17: Sigmoid E_{max} model.



I. Inhibition (k_{in})
III. Stimulation (k_{in})
II. Inhibition (k_{out})
IV. Stimulation (k_{out})
k_{in}
k_{out}
Response (R)
IC_{50}
EC_{50}

$$\frac{dR}{dt} = k_{in} \cdot \left(1 - \frac{C}{IC_{50} + C}\right) - k_{out}$$

$$\frac{dR}{dt} = k_{in} \cdot \left(1 + \frac{E_{max} \cdot C}{EC_{50} + C}\right) - k_{out}$$

$$\frac{dR}{dt} = k_{in} - k_{out} \cdot \left(1 - \frac{C}{IC_{50} + C}\right)$$

$$\frac{dR}{dt} = k_{in} - k_{out} \cdot \left(1 + \frac{E_{max} \cdot C}{EC_{50} + C}\right)$$

Figure 4.18: Four basic models of indirect pharmacodynamic responses.

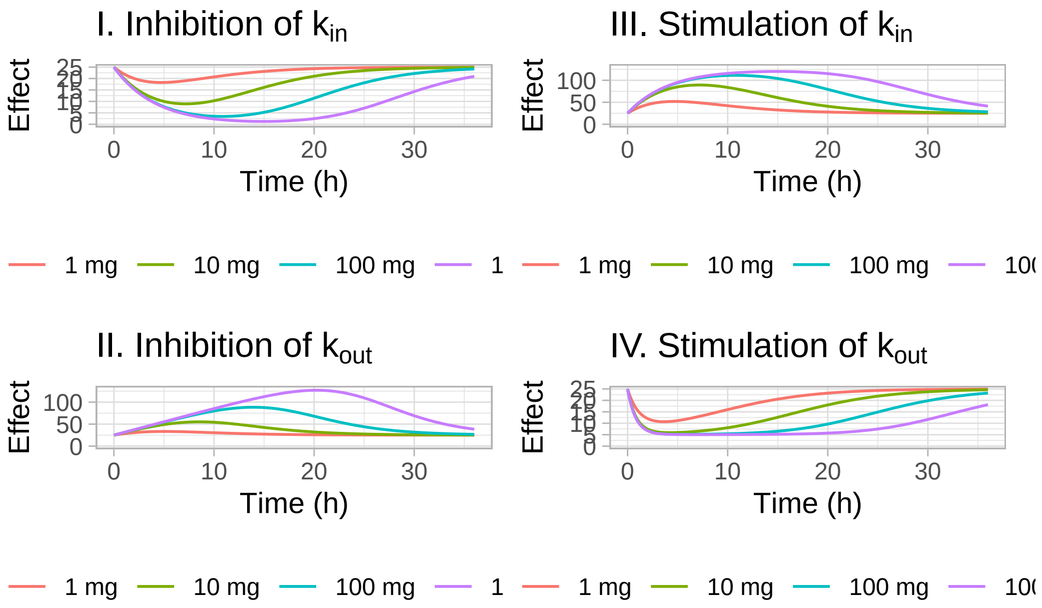


Figure 4.19: Four basic models of indirect pharmacodynamic responses, plotted.

drug binds the receptor, a series of events takes place that eventually results in the observed effect. The effect compartment model is a useful way to describe the time course of drug effect.

4.3.3.4 Tolerance and rebound models

Tolerance is the phenomenon in which the response to a drug *decreases* over time, while rebound is the phenomenon in which the response to a drug *increases* over time. Tolerance and rebound can be captured using modeling - there are a large variety of methods by which this can be done, including systems approaches, tolerance compartments, and turnover of receptor regulation. Gabrielsson and Weiner provide a comprehensive overview in their book.

4.3.3.5 Baseline models

Up until now, we have only considered situations in which effects are exerted on response variables that do not vary independently of drug concentration over time. Most physiological variables, however, show chronobiologic rhythm, such as blood pressure, heart rate, and body temperature. Baseline models are used to describe the time course of these variables, so that drug effects on systems like these can be accurately estimated.

5 A brief introduction to nonlinear mixed-effects modeling

This introductory chapter serves as a brief overview of mixed-effects models as they're used in the pharmaceutical industry. It is by no means complete or exhaustive. If you'd like to learn more, we direct you to the seminal work on this topic, Pinheiro and Bates (9), as well as the comprehensive summary of the topic by Ezzet and Pinheiro (20). Mould and Upton provide a digestible summary (2,3).

For illustrative purposes, we'll focus on population PK for the moment. Sheiner et al (21) first introduced the concept of population PK modeling in 1972, and since then, the field has grown rapidly. Population PK models are typically developed using nonlinear mixed-effects modeling, which is a type of mixed-effects modeling that allows for the estimation of both fixed and random effects. In population PK modeling, the PK of a drug is described by a structural model, which is a mathematical model that describes the relationship between drug concentration and time, and a statistical model, which describes the variability in PK parameters between individuals. The structural model is typically a compartmental model, and the statistical model is typically a mixed-effects model.

5.1 Mixed effects models

Longitudinal data such as plasma concentrations or effect measurements are often collected as repeated measures - more than one measurement is collected from a single subject over time. Measurements collected from the same subject are usually correlated to some degree, which means that the usual ways of applying statistical models to data are not really appropriate, since they assume that every single sample is independent of every other sample, which is of course not the case in this situation. Mixed-effects models handle this by allowing the correlation between samples taken from a discrete individual to be taken into account (or to put it another way, they allow for between-subject variability). They do this through what are called random effects.

5.1.1 Linear mixed effect (LME) models

Linear mixed-effects (LME) models usually deal with situations in which one's response variable can be described as a linear function of both the fixed effects (things which do not vary, such as predictors, or typical values of model parameters) and the random effects. LMEs extend simple linear models to allow both fixed and random effects, and are most often used when observations are not independent of one another, as might arise from a hierarchical structure. For example, patients could be sampled from within treatment groups or study sites.

When there are multiple levels to be considered, such as patients receiving the same dose of a drug but at different sites, the variability in the response can be thought of as being either "within group" or "between group". Patient-level observations are not independent, since (following our example) they are expected to be more similar within a study site. Units sampled at the highest level, however (in our example, a treatment arm) are independent.

To analyse these data, one would need to consider outcome in terms of both study site and dose.

Consider what we might assume is the true regression slope in the population, β, and that we're able to estimate it as $\hat{\beta}$. This is a fixed effect. In contrast, random effects are parameters that can be thought of as random variables. For example, we could say that β is a normally-distributed random variable with mean μ and standard deviation σ. So, formally:

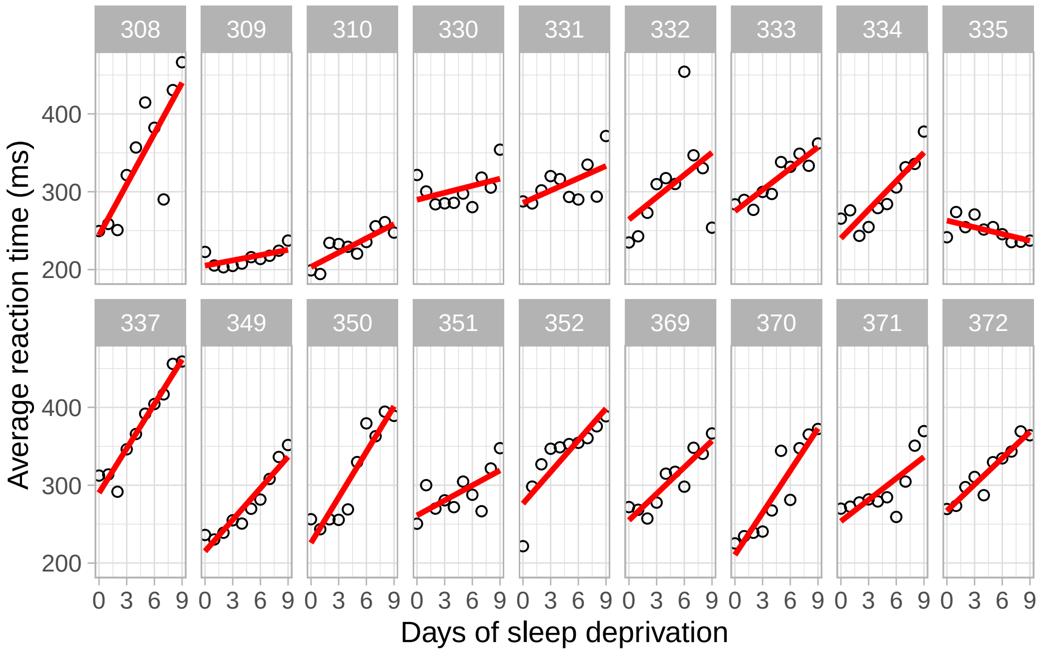
$$\beta \sim N(\mu, \sigma)$$

A LME model for outcome y might formally be defined as follows:

$$y = X\beta + Zu + \epsilon$$

Here, y is the outcome variable we're interested in. We can think of it as a matrix of N x 1, where N is the number of patients. X is an N x p matrix of p predictor variables (such as age, weight, and sex, variables that take unique values for each of the N patients). β is a p x 1 vector of the regression coefficients for each of the p predictors. Z is the N x qJ design matrix for the q random effects and the J groups of interest; u is a qJ x 1 vector of q random effects for J groups. Finally, ϵ is an N x 1 vector describing the residuals (the part of y not described by the rest of the model, $X\beta + Zu$).

This sounds very complex, especially if you haven't got a PhD in statistics. It really isn't. Let's illustrate it with a simple example from the `lmer` package developed by Bates and colleagues based on a sleep deprivation study by Belenky *et al* (22,23). In this study, 18 subjects had their normal amount of sleep on Day 0, but starting that night they were restricted to 3 hours of sleep per night. Average reaction time in milliseconds (`Reaction`) was the outcome, recorded through a series of tests given each `Day` to each `Subject`.



As we can see here, reaction time for most subjects increases with duration of sleep deprivation, but the slope is different. Reaction time observations within each subject are not independent of one another, so we can't just pool everything. This is where mixed effects come in.

This is a fairly basic example, since we only have three bits of data we can use (Subject, Day and Reaction). So:

$$y_{i,j} = Day_{i,j} \cdot Slope_i + \epsilon$$

where

$$Slope_i = \theta_{Slope} + \eta_{Slope,i}$$

Here, $y_{i,j}$ is reaction time in individual i on day j, $Slope_i$ is the regression slope relating reaction time and day, θ_{Slope} is the typical slope in the population, η_{Slope} is between-subject variability in slope, defined as being normally distributed with mean 0 and variance ω^2, and ϵ, as before, is residual variability, defined as being normally distributed with mean 0 and variance σ^2. The `nlmer` function in `lme4` can handle this pretty easily:

```r
lmer(Reaction ~ Days + (Days | Subject), sleepstudy)
```

```
Linear mixed model fit by REML ['lmerMod']
Formula: Reaction ~ Days + (Days | Subject)
```

```
   Data: sleepstudy
REML criterion at convergence: 1743.628
Random effects:
 Groups   Name        Std.Dev. Corr
 Subject  (Intercept) 24.741
          Days         5.922   0.07
 Residual             25.592
Number of obs: 180, groups:  Subject, 18
Fixed Effects:
(Intercept)         Days
     251.41        10.47
```

This output indicates that our fixed effects, β, are 251.4 ms and 10.47 ms/day, corresponding to the typical intercept and slope in the population. In this model, we've also been able to estimate random effects on both of these parameters: their standard deviations are estimated to be 24.74 ms and 5.92 ms/day, respectively. This tells us that each subject can have a different intercept and a different slope, but on average, reaction time increases by 10.47 ms per day, and reaction time with no sleep deprivation is 251.4 ms in this small population.

So far, so good. But pharmacology is a bit more complicated than this, and we're going to need more firepower. Enter nonlinear mixed-effects models.

5.1.2 Nonlinear mixed-effect (NLME) models

Nonlinear mixed-effects (NLME) modeling has emerged as a powerful statistical framework that addresses the inherent variability observed in real-world data. Rooted in the field of mixed-effects modeling, which accounts for both fixed effects (population-level parameters, things which are common across populations) and random effects (individual-specific deviations from what is typically seen in a population), as we've briefly looked at above, the nonlinear variant takes on the challenge of describing complex and nonlinear relationships between variables. This approach found its origins in the realms of pharmacology and biology, where it was first employed to analyze drug concentrations over time and has been extended subsequently to such diverse domains such as ecology, engineering, and the social sciences.

At its core, NLME modeling embraces the idea that underlying dynamic processes - such as changes in drug concentration in an individual, or a population of individuals, over time - can be captured accurately by sets of mathematical equations. Unlike linear models, which assume constant relationships, nonlinear models account for intricate nonlinear dynamics that better reflect the complexities of biological, physical, and social systems. The incorporation of random effects acknowledges that while general trends are present across individuals or subjects, deviations from these trends can arise due to inherent biological variability, measurement errors, or other unobserved factors. Consider: if persons A and B are given the same dose of a specific drug at the same time, plasma drug concentration at some later time T in person

A is not going to be the same as plasma drug concentration at the same time T in person B, although it might be similar. There are all kinds of reasons why this might be the case, but we can never measure all of them, and that's where random effects come in. They help us understand just how similar - or different - those concentrations are, and whereabouts in our modelled system these differences might be coming from.

Pharmacokinetics (PK) and pharmacodynamics (PD) provide one of the most prominent and impactful applications of nonlinear mixed-effects modeling. These models facilitate the estimation of critical parameters such as clearance (CL), volume of distribution (V), and absorption rate (ka) at the population level, which are crucial for optimizing drug dosing regimens.

In drug development, NLME models enable researchers to better understand the variability in drug response across different individuals, allowing the optimization of drug therapy for a given population, and the application of personalized dosing strategies. This has profound implications for enhancing treatment efficacy while minimizing adverse effects. Additionally, these models facilitate the exploration of drug-drug interactions and the impact of patient characteristics (such as age, weight, genetics) on drug response.

Beyond pharmacology, nonlinear mixed-effects models find applications in diverse fields. In ecology, they capture population dynamics influenced by both intrinsic growth and extrinsic environmental factors. In social sciences, they analyze longitudinal data with individual-specific trajectories influenced by time-varying covariates. By providing a nuanced understanding of complex systems, nonlinear mixed-effects modeling empowers researchers to make informed decisions and predictions in the face of intricate variability and nonlinear dynamics.

But it's pharmacology we're here for, isn't it! Let's have a look at a single-dose oral one-compartment model, one of the simplest cases we're likely to encounter.

$$C(t) = \frac{F \cdot Dose \cdot k_a}{V(k_a - k)}[e^{-kt} - e^{-k_a t}]$$

Here, $C(t)$ is drug concentration in the central compartment at time t, which is a function of bioavailability F, drug dose $Dose$, absorption rate k_a, central volume of distribution V and elimination rate k. In this system, we could imagine having two major sources of error to consider: random unexplained error, which could derive from errors in assays, measurement times, and so on, and between-subject error, which relate to differences in model parameters (k_a, V, k, F) between subjects (which might derive from differences in body size, body composition, enzyme expression and any number of other factors).

We can express $C(t)$ formally as follows:

$$C_{ij}(t) = f(\theta_i, Dose_j, t)[1 + \epsilon_{ij}(t)]$$

where $C_{ij}(t)$ is the measured concentration of drug in individual i at dose j at time t, $f(\theta_i, Dose_j, t)$ is its corresponding prediction, θ_i is the vector of model parameters for individual i (i.e. $k_{a,i}$, V_i, k_i and F_i), $Dose_j$ is dose j, and $\epsilon_{ij}(t)$ is residual variability, defined as being normally distributed with mean 0 and variance σ^2. In this example, it is proportional to concentration. The model parameters in this example are considered to be log-normally distributed with mean 0 and variance ω^2, for example:

$$V_i = V \cdot \exp(\eta_{V,i})\eta_{V,i} \sim N(0, \omega_V)$$

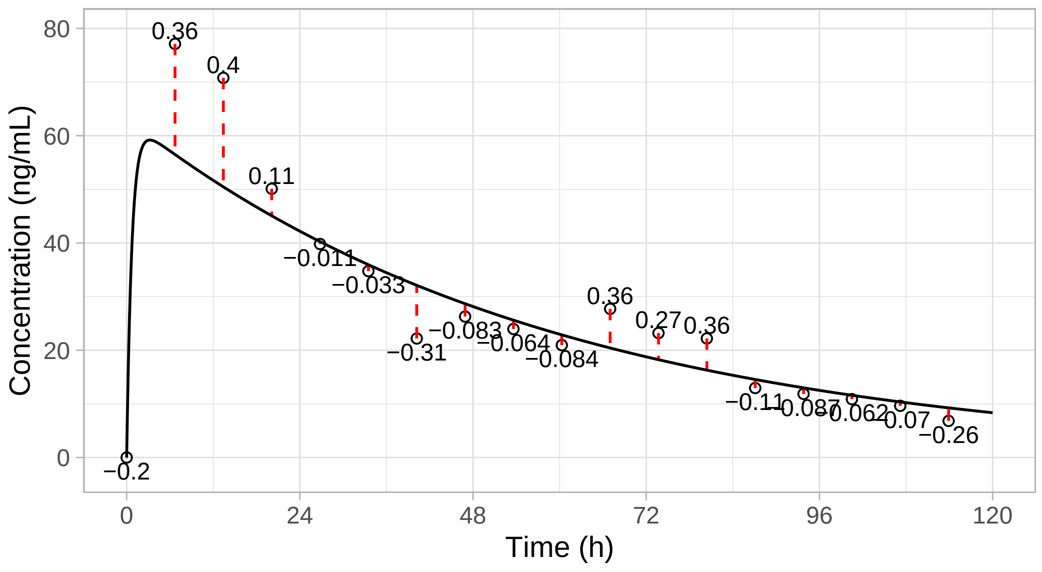
The functional form of f can of course vary depending on the model being considered (in this case, still the oral one-compartmental PK model).

Less commonly used, and the subject of heated debate in some quarters, is between-occasion variability - the variability observed within patients but between sampling occasions (24). At the time of writing, between-occasion variability is not supported by `nlmixr2`, but we include it here for completeness. Continuing with our example, it can be expressed as follows:

$$V_i = V \cdot \exp(\eta_{V,i} + \kappa_{V,ij})\eta_{V,i} \sim N(0, \omega_V)\kappa_{V,ij} \sim N(0, \pi_V)$$

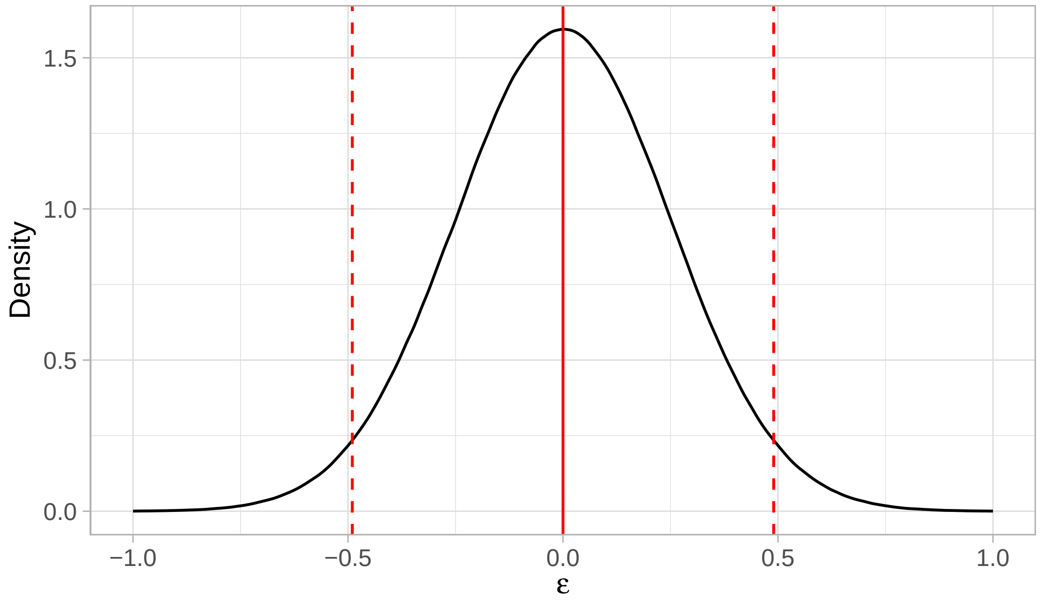
Here we have added the independently-distributed random effect $\kappa_{V,ij}$, representing between-occasion variability in individual i at measurement occasion j, assumed normally distributed with mean 0 and variance π^2.

So what does this all mean in practice? Let's look at our simple 1-compartment model. First, consider the residual error, which describes the difference between a prediction and an observation.



partment oral, proportional residual error. Dose = 500 mg, ka = 1.39 /h, CL = 0.350 L/h, V = 8.00 L
nodel−predicted concentration, points are observations, red dashed lines are residual magnitudes,
annotations are values of epsilon (proportions of predicted values).

Assuming a normal distribution of ϵ, and a σ^2 value of 0.0625 (corresponding to a standard deviation of 0.25) our residuals are distributed like this:



).0625. Solid red vertical line is the median (0). Dashed vertical red lines correspond to 95% range.

Every observation has an associated model prediction, and the difference between them is the residual, which in our model is proportional to the size of the observation. That proportion can lie anywhere within this distribution.

Residual variability can be defined in a number of ways. The first, in which an additive relationship is assumed, is defined formally as:

$$DV_{obs,i,j} = DV_{pred,i,j} + \sigma_{i,j}\sigma \sim N(0, \epsilon)$$

where $DV_{obs,i,j}$ is the osberved value of the dependent variable, $DV_{pred,i,j}$ is the predicted value of the dependent variable, and $\sigma_{add,i,j}$ is additive residual error, defined as being normally distributed with mean 0 and variance ϵ^2_{add}.

Residual error can also be modelled to be proportional:

$$DV_{obs,i,j} = DV_{pred,i,j} \cdot (1 + \sigma_{i,j})\sigma \sim N(0, \epsilon)$$

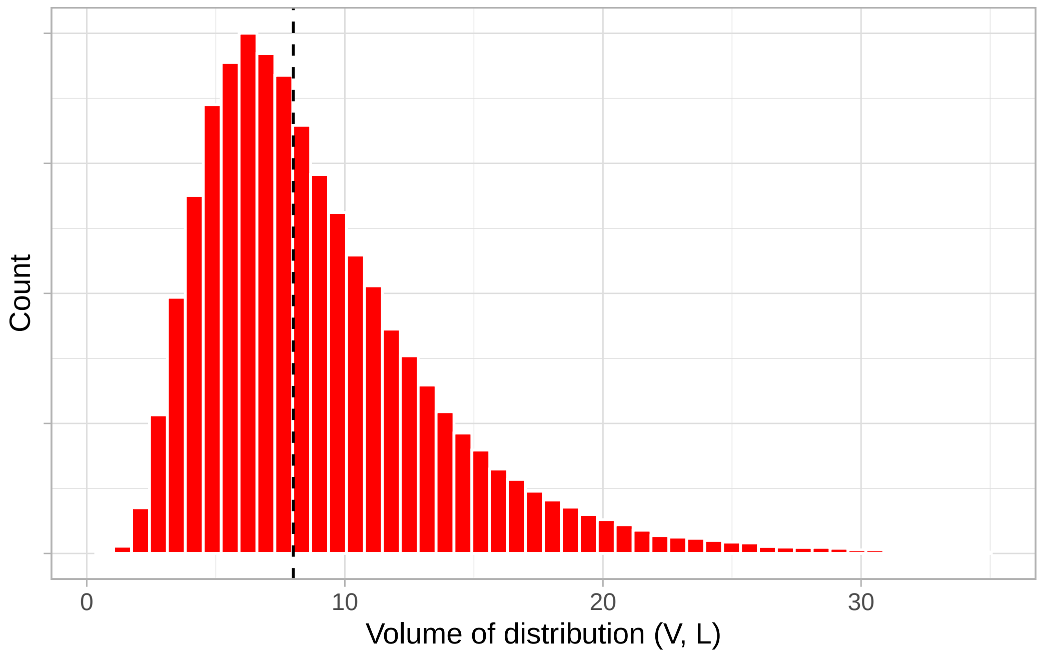
Or both:

$$DV_{obs,i,j} = DV_{pred,i,j} \cdot (1 + \sigma_{prop,i,j}) + \sigma_{add,i,j}\sigma_{prop} \sim N(0, \epsilon_{prop})\sigma_{add} \sim N(0, \epsilon_{add})$$

The model parameters work much the same way. Let's look at volume of distribution (V) one more time. Here we have the distribution of $\eta_{V,i}$, assuming that ω_V^2 is 0.25 (this is a variance, corresponding to a standard deviation of 0.5). Recall

$$V_i = V \cdot \exp(\eta_{V,i})$$

so if the typical value of V is 8 L, we would expect V to be log-normally distributed, like so.



This has the advantage of never being less than 0 (negative values of physiological parameters like CL and V are impossible) and looking a lot like what we actually see in biology. So we use this formulation a lot. Some model parameters may actually be normally, rather than log-normally, distributed. For those we might use an additive model:

$$Baseline_i = Baseline + \eta_{Baseline,i}$$

which would give us a normal distribution. In some circumstances it might still be important to ensure it never dips below 0, though - here we're considering a baseline value in a pharmacodynamic model.

So this is a very, very high-level outline of how one could think about NLME models of the kind used in pharmacometrics. Let's see how we might use them in practice.

Part III

Simulations with `rxode2`

6 Specifying models in `rxode2` and `nlmixr2`

6.1 Model syntax

Models can take four forms in `rxode2`.

Consider a very simple three-compartment system. Let's assume we know the values of CL, V2, Q and KA.

First, and most simply, `rxode2({})` block statements can be used.

```
library(rxode2)

mod <- rxode2({
  C2 <- central/V2

  # time-derivative
  d/dt(central) <- F*KA*depot - CL*C2 - Q*C2 + Q*C3;
})
```

One could also put the model inside an `rxode2("")` string statement:

```
mod <- rxode2("
  C2 <- central/V2

  # time-derivative assignment
  d/dt(central) <- F*KA*depot - CL*C2 - Q*C2 + Q*C3;
")
```

We could load the model code from an external file:

```
writeLines("
  C2 <- central/V2

  # time-derivative assignment
  d/dt(central) <- F*KA*depot - CL*C2 - Q*C2 + Q*C3;
",
  "modelFile.rxode2")

mod <- rxode2(filename='modelFile.rxode2')
```

Or we could specifiy the model as part of a model function:

```
mod <- function() {
  model({
    C2 <- central/V2

    # time-derivative assignment
    d/dt(central) <- F*KA*depot - CL*C2 - Q*C2 + Q*C3;
  })
}

mod <- rxode2(mod) # or simply mod() if the model is at the end of the function
```

This last formulation is the most convenient for our purposes, since it is easy to read and write, can be modified by using piping, and is the format used by `nlmixr2`. Model functions often have residual components and initial conditions attached as well - the classical theophylline model, for example, can be defined as:

```
one.compartment <- function() {
  ini({
    tka <- 0.45   # log KA
    tcl <- 1      # log CL
    tv  <- 3.45   # log V

    eta.ka ~ 0.6
    eta.cl ~ 0.3
    eta.v  ~ 0.1

    add.sd <- 0.7
  })
  model({
    ka <- exp(tka + eta.ka)
    cl <- exp(tcl + eta.cl)
    v  <- exp(tv + eta.v)

    d/dt(depot) = -ka * depot
    d/dt(center) = ka * depot - cl / v * center

    cp = center / v

    cp ~ add(add.sd)
  })
}

# after parsing, compile
mod <- one.compartment()
```

`rxode2` translates the ODE system into C, compiles it, and loads it into the R session where it can be used. The call to `rxode2` produces an object of class `rxode2` which consists of a list-like structure (environment) with various member functions.

6.1.1 Statements

A basic model specification consists of one or more statements, optionally terminated by semi-colons (`;`) and optional comments (comments are delimited by `#` and an end-of-line).

A *block* of statements is a set of statements delimited by curly braces (`{...}`).

Statements can be either assignments, conditionals (`if/else`), `while` loops, special statements, or printing statements (which can be used for debugging and testing).

Assignment statements can be:

- *Simple assignments*, in which the left side is an identifier (a variable) and the right is the value it is assigned.
- *Time-derivative assignments*, in which the left side specifies the change of the amount in the corresponding state variable (compartment) with respect to time (e.g. `d/dt(depot)`), and the right is the corresponding expression.
- *Initial-condition assignments*, in which the left side specifies the compartment of the for which an initial condition is being defined, and the right is the initial value (e.g. `depot(0) = 0`).
- *Model characteristic assignments*, in which a characteristic of the model can be set, including:
  - bioavailability (e.g. `f(depot) = 1`),
  - lag time (e.g. `alag(depot) = 0`),
  - modeled rate (e.g. `rate(depot) = 2`) and
  - modeled duration (e.g. `dur(depot) = 2`).
- *Change point* assignments, in which time-based changes to the system can be modeled (e.g. `mtime(var) = time`).

- *Jacobian-derivative assignments*, in which the left side specifies the change in the compartment ODE with respect to a variable, and the right is the corresponding expression. For example, if `d/dt(y) = dy`, a Jacobian for this compartment can be specified as `df(y)/dy(dy) = 1`. This is sometimes useful for very stiff ODE systems, but speed is adversely affected.
- *String value declarations*, which specify the values a given string variable will take within an `rxode2` solving structure. These values direct corresponding factors to be created for this variable on solving the `rxode2` model (e.g. `labels(a) <- c("a1", "a2")`).
- *Special assignments*, including:

  - *Compartment declaration* statements, which can be used to change the default dosing compartment and the assumed compartment number(s) as well as add extra compartment names at the conclusion of the solving or fitting process (useful for multiple-endpoint `nlmixr2` models). These are specified by `cmt(compartment name)`.
  - *Parameter declaration* statements, which can be used to specify that input parameters are encoded in an explicit order instead of in the order in which they are parsed. This is useful for keeping the parameter order the same when using different ODE models. These are specified by `param(par1, par2,...)`.
  - *Variable interpolation* statements, which specify the interpolation method for specific covariates. These include `locf(cov1, cov2, ...)` for last observation carried forward, `nocb(cov1, cov2, ...)` for next observation carried backward, `linear(cov1, cov2, ...)` for linear interpolation and `midpoint(cov1, cov2, ...)` for midpoint interpolation.

Expressions in assignment and conditional statements can be numeric or logical (`TRUE` or `FALSE`).

Numeric expressions can include common numeric operators (`+`, `-`, `*`, `/`, `^`) and mathematical functions defined in the C or the R mathematics libraries (including but not limited to `fabs`, `exp`, `log`, `sin`, `abs`). R's internal functions may also be used, such as `lgammafn` for the log gamma function. The modulo operator (`%%`) is currently unsupported.

`rxode2` syntax is case-sensitive, i.e., ABC is different than abc, Abc, ABc, etc.

6.1.2 Identifiers

As in R, identifiers (variable names) may consist of one or more alphanumeric, underscore (`_`) or period (`.`) characters, although the first character cannot be a digit or an underscore.

Identifiers in a model specification can refer to:

- *State variables* in the dynamic system (such as compartments in a pharmacokinetic or pharmacodynamic model).

- *Implied input variables*, including `t` (time), `tlast` (last time point), and `podo` (oral dose, in the case of absorption transit models).
- *Special constants* such as `pi` or predefined constants inherited from R.
- *Model parameters* such as `ka` (rate of absorption) and `CL` (clearance).
- *Other identifiers*, as created by assignments as part of the model specification; these are referred as LHS (left-hand side) variables.

Currently, the `rxode2` modeling language only recognizes system state variables and parameters, so any values that must be passed from R to the model (covariates such as age, for example) should be either passed in the `params` argument of the integrator function or be present in the event dataset (or the `nlmixr2` dataset).

6.1.3 Reserved terms

`rxode2` models use a number of reserved terms (internal variables, or used to define event records) which cannot be used as variable names and cannot be assigned, or used as a state, but can be accessed in the `rxode2` code. These include `addl`, `amt`, `cmt`, `dur`, `dvid`, `id`, `print`, `printf`, `rate`, `Rprintf` and `ss`. The variables `evid` and `ii` are also reserved and may never be used anywhere in model code. Details of what these are may be found in Chapter 7.

In addition to these event-related terms, the following are also reserved (although most can be accessed).

Reserved Name	Meaning	Alias
`time`	Solver time	`t`
`podo`	Time of last dose	
`tlast`	Last dose amount (transit models)	
`pi`	`pi`	
`NA`	`NA`	
`NaN`	Not a number	
`Inf`	Infinite	
`newind`	1: First record of individual; 2: Subsequent record of individual	`NEWIND`

Reserved Name	Meaning	Alias
`rxFlag`	Flag for the part of the model being run. 1: ddt; 2: jac; 3: ini; 4: F; 5: lag; 6: rate; 7: dur; 8: mtime; 9: matrix exponential; 10: inductive linearization; 11: lhs	
`M_E`	`exp(1)`	
`M_LOG2E`	`log2(e)`	
`M_LOG10E`	`log10(e)`	
`M_LN2`	`log(2)`	
`M_LN10`	`log(10)`	
`M_PI`	`pi`	
`M_PI_2`	`pi/2`	
`M_PI_4`	`pi/4`	
`M_1_PI`	`1/pi`	
`M_2_PI`	`2/pi`	
`M_2_SQRTPI`	`2/sqrt(pi)`	
`M_SQRT2`	`sqrt(2)`	
`M_SQRT1_2`	`1/sqrt(2)`	
`M_SQRT_3`	`sqrt(3)`	
`M_SQRT_32`	`sqrt(32)`	
`M_LOG10_2`	`log10(2)`	
`M_2PI`	`2*pi`	
`M_SQRT_PI`	`sqrt(pi)`	
`M_1_SQRT_2PI`	`1/(sqrt(2*pi))`	
`M_LN_SQRT_PI`	`log(sqrt(pi))`	
`M_LN_SQRT_2PI`	`log(sqrt(2*pi))`	
`M_LN_SQRT_PId2`	`log(sqrt(pi/2))`	

`rxode2` and `nlmixr2` generate variables that are used internally as part of the solving or estimation process, typically with the prefixes `rx` and `nlmixr`. To avoid any unexpected issues, it is (strongly) suggested that variables starting with either `rx` or `nlmixr` not be used.

6.1.4 Logical operators

The standard R logical operators `==`, `!=`, `>=`, `<=`, `>`, and `<` are supported. Like R, these can be used in `if()`, `while()` and `ifelse()` expressions. They can also be used in standard assignments. For instance, the following is valid:

```
cov1 = covf*(sexf == "female") + covm*(sexf != "female")
```

Character expressions can be used in comparisons. This convenience comes at a cost, however, since character comparisons are slower than numeric expressions.

6.1.5 Supported functions

All the functions supported in `rxode2` can be returned using `rxSupportedFuns()`. Since there are more than 200 of them, we won't discuss them all, but suffice it to say that whatever you might need is available in some form. The assortment includes trigonometry, virtually all statistical distributions of any note, and many, many log-likelihoods.

6.1.6 Residual error

6.1.6.1 Simple form

The tilde (`~`) is used to specify a residual output or endpoint. For variable `var`, the variable that represents the individual central tendency of the model's dependent variable as well as the compartment specification in the dataset, a residual error statement can be written like this:

```
var ~ add(add.sd)
```

which corresponds to

$$y = f + a \cdot \epsilon$$

where `y` is the value of the dependent variable in the current individual, `f` is the central tendency of the observation in the current individual, `a` is the standard deviation of the residual error, and ϵ is normally-distributed with mean 0 and standard deviation 1, or

```
var ~ prop(prop.sd)
```

$$y = f + b \cdot f^c \cdot \epsilon$$

where `b` is the coefficient describing the the extent to which ϵ is proportional to `f`, and `c` is a power coefficient (usually fixed to 1).

We can use additive (`add()`) or proportional (`prop()`) residual error models, or both; here, they are expressed as standard deviations.

Sometimes we'd like to change the compartment the residual error is applied to. We can do this as follows:

```
var ~ add(add.sd) | cmt
```

Here, `cmt` represents the compartment we'd like the residual error to apply to. This is particularly useful for models with multiple endpoints.

6.1.6.2 Combined additive and proportional residual error

Combined additive and proportional models can be expressed in two different ways, which we will define as `combined1` and `combined2` (to avoid between-tool confusion, we have adopted the naming convention used by Monolix).

The first variant, `combined1`, assumes that the additive and proportional components are on the standard deviation scale, or:

$$y = f + (a + b \cdot f^c) \cdot err$$

This represents a linear combination of a constant error term (`a`) and a term proportional to the structural model `f` (`b`). The `c` parameter in this expression is usually fixed to 1.

The second variant, `combined2`, assumes that additive and proportional components of the residual error are combined on the variance scale:

$$y = f + \sqrt{(a^2 + b^2 \cdot f^{2c})} \cdot \epsilon$$

Here we have a combination of constant term `a` and a term proportional to the structural model `f` (`b`). Again, `c` is typically fixed to 1. If not otherwise specified, `combined2` is the default, and is considered to have "better" statistical properties. Combined models can be specified like this:

```
var ~ add(add.sd) + prop(prop.sd) + combined1()
```

or like this for `combined2`:

```
var ~ add(add.sd) + prop(prop.sd) # combined2 is the default
```

6.1.6.3 Transformations

Sometimes, if our residual error distribution is not normal, using the default normal distribution for the residual error model is just not going to be adequate. Fortunately, `rxode2` and `nlmixr2` support several transformations that might be of use. Be warned, though: here there be tygers.

Transformation	Usage
Box-Cox (25)	`+boxCox(lambda)`
Yeo-Johnson (26)	`+yeoJohnson(lambda)`
logit-normal (27)	`+logitNorm(logit.sd, low, hi)`
probit-normal (28)	`+probitNorm(probit.sd, low, hi)`
log-normal (29)	`+lnorm(lnorm.sd)`
T-distribution	`+dt(df)`
Cauchy	`+dcauchy()`

By default, the likelihood for all of these transformations is calculated on the untransformed scale.

These functions require additional estimated parameters `lambda` (Box-Cox, Yeo-Johnson), a standard deviation (logit-normal, probit-normal and log-normal) and, optionally, lower and upper bounds (logit-normal and probit-normal). For bounded transformations like the logit-normal or the probit-normal the low (`lo`) and high (`hi`) values default to 0 and 1 respectively if missing.

`lnorm()`, `probitNorm()` and `logitNorm()` can be combined with the `yeoJohnson()` transformation if necessary (it usually isn't). dt() and dcauchy() can be combined with the other transformations if needed. An example of a proportional error model with Box-Cox and Cauchy in a PK model could be coded like this:

```
cp ~ prop(add.err) + boxCox(lambda) + dcauchy()
```

where `cp` is modeled concentration, and `lambda` is a parameter in the model.

6.1.7 Likelihood distributions

6.1.7.1 Generalized likelihoods

The likelihood for a compartment, if not the default, can be specified as:

```
ll(cmt) ~ llik specification
```

6.1.7.2 Ordinal likelihoods

Let's consider a situation in which we have an ordered categorical dependent variable, such as a score, with 4 discrete values, 0-3. Ordinal likelihoods and simulations can be specified in 2 ways. The first is:

```
cmt ~ c(p0, p1, p2)
```

Here `cmt` represents the compartment, and `p0`, `p1` and `p2` represent the probabilities of being in specific category 0, 1, 2, or 3:

Category	Probability
0	`p0`
1	`p1`
2	`p2`
3	`1 - p0 - p1 - p2`

`p0` + `p1` + `p2` must add to less than 1 for this model to work. Using this formulation, scores are often not integer values, which doesn't really make sense. We can fix this as follows:

```
cmt ~ c(p0=0, p1=1, p2=2, 3)
```

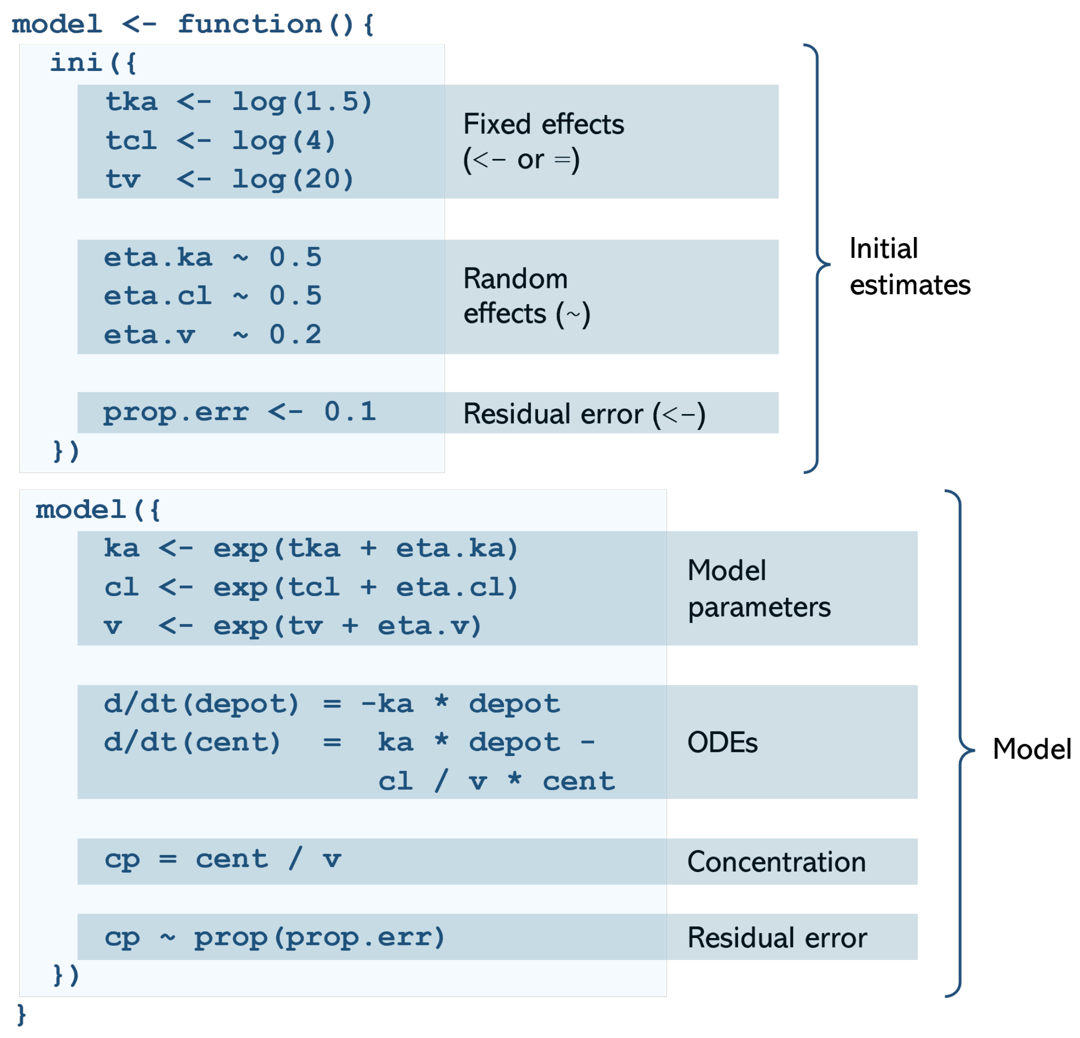
Here the numeric categories are specified explicitly, while the probabilities remain the same.

6.1.8 Working with strings

Strings are converted to double values inside of `rxode2`, and can therefore be referred to as integers corresponding to the string value, or the string values themselves. For covariates, these are calculated on the fly based on your data, and it is not necessary (or advisable) to do anything additional. For strings defined in the model, however, either can be used. That said, it is never anything other than confusing, so we do not recommend it.

6.2 Model blocks

`rxode2` and `nlmixr2` models are composed of two parts, the `ini` block, which contains initial estimates of the fixed and random effect parameters, and the `model` block, which contains the model definition. Figure 6.1 provides a very simple description of how models are constructed.



```
model <- function(){
  ini({
     tka <- log(1.5)
     tcl <- log(4)
     tv  <- log(20)

     eta.ka ~ 0.5
     eta.cl ~ 0.5
     eta.v  ~ 0.2

     prop.err <- 0.1
  })
  model({
     ka <- exp(tka + eta.ka)
     cl <- exp(tcl + eta.cl)
     v  <- exp(tv + eta.v)

     d/dt(depot) = -ka * depot
     d/dt(cent)  =  ka * depot -
                    cl / v * cent

     cp = cent / v

     cp ~ prop(prop.err)
  })
}
```

Figure 6.1: Anatomy of a model.

6.2.1 The ini block

The `ini` block specifies initial conditions, including initial estimates. When estimation is required (for `nlmixr2`), boundaries may be specified for those algorithms that support them. Model parameters are typically expressed as follows:

```
parameter = initial estimate
parameter = c(lower boundary, initial estimate)
parameter = c(lower boundary, initial estimate, upper boundary)
```

`nlmixr2` supports both the standard R assignment operator (`<-`) and `=` for assignments. Simple expressions that evaluate to a number can be used for defining both initial conditions and boundaries, and `nlmixr2` allows specification of infinite parameters, as in:

```
parameter = c(-Inf, log(70), 7)
```

Importantly, initial conditions cannot be assigned directly using local or global variable values in R, because the function is parsed rather than evaluated directly. Furthermore, other parameter values cannot be used as boundaries in the `ini` block. You can see how `nlmixr2` parses the model by “evaluating” it, as follows:

```r
library(nlmixr2)
```

```
Loading required package: nlmixr2data
```

```r
one.cmt <- function() {
  ini({
    ## You may label each parameter with a comment
    tka <- 0.45                # Log Ka
    tcl <- log(c(0, 2.7, 100)) # Log Cl
    ## This works with interactive models
    ## You may also label the preceding line with label("label text")
    tv <- 3.45; label("log V")
    ## the label("Label name") works with all models
    eta.ka ~ 0.6
    eta.cl ~ 0.3
    eta.v  ~ 0.1
    add.sd <- 0.7
  })
  model({
    ka <- exp(tka + eta.ka)
    cl <- exp(tcl + eta.cl)
    v <- exp(tv + eta.v)
    linCmt() ~ add(add.sd)
  })
}

nlmixr(one.cmt)
```

```
i parameter labels from comments are typically ignored in non-interactive mode

i Need to run with the source intact to parse comments

 -- rxode2-based solved PK 1-compartment model with first-order absorption ------
 -- Initalization: --
Fixed Effects ($theta):
      tka       tcl        tv    add.sd
0.4500000 0.9932518 3.4500000 0.7000000

Omega ($omega):
       eta.ka eta.cl eta.v
eta.ka    0.6    0.0   0.0
eta.cl    0.0    0.3   0.0
eta.v     0.0    0.0   0.1
 -- mu-referencing ($muRefTable): --
  theta    eta level
1   tka eta.ka    id
2   tcl eta.cl    id
3    tv  eta.v    id

 -- Model (Normalized Syntax): --
function() {
    ini({
        tka <- 0.45
        tcl <- c(-Inf, 0.993251773010283, 4.60517018598809)
        tv <- 3.45
        label("log V")
        add.sd <- c(0, 0.7)
        eta.ka ~ 0.6
        eta.cl ~ 0.3
        eta.v ~ 0.1
    })
    model({
        ka <- exp(tka + eta.ka)
        cl <- exp(tcl + eta.cl)
        v <- exp(tv + eta.v)
        linCmt() ~ add(add.sd)
    })
}
```

We'll see this one-compartment model again later on, but for the moment, let's concentrate on the syntax.

Model parameter names can be specified using almost any name that R will accept. Variable names are case-sensitive, and a number specific of model parameter names (like `CL` and `V` for an IV bolus one-compartment model) can be used to define solved systems (we'll discuss this in detail in a bit).

Comments (using `#`, or more explicitly defined using the `label` function) on these population and residual parameters are captured as parameter labels. These are used for populating parameter table output by `nlmixr2` and by the graphical user interface package `shinyMixR`.

Multivariate normal individual deviations from typical population values of parameters (analogous to "ETA" parameters in NONMEM) are estimated, along with the variance-covariance matrix of these deviations (the "OMEGA" matrix). These inter-individual random-effects parameters take initial estimates and in `nlmixr2` are specified using the tilde (`~`). In R statistical parlance, the tilde is used to represent "modeled by" and was chosen to distinguish these estimates from the population and residual error parameters.

An example specifying the initial estimate of variability in absorption rate constant (`ka`) as having a variance of 0.6 is:

```
eta.ka ~ 0.6
```

The addition operator `+` may be used to specify the variance/covariance matrix of jointly-distributed random effects, with the right-hand side of the expression specifying the initial estimates as a lower triangular matrix. An example for `CL` and `V` might look like:

```
eta.cl + eta.v ~ c(0.1,
                   0.005, 0.1)
```

This sets initial estimates for the variability of `CL` and `V` of 0.1, with a covariance of 0.005. Note that only the parameters that are assumed to be correlated need to have covariances specified in this manner.

6.2.2 The `model` block

The model block specifies the model itself. Models are defined in terms of the parameters defined in the `ini` block and covariates of interest from the data. Individual model parameters are defined before the model itself is specified. For example, to define individual clearance in terms of the typical value in the population and an individual random-effect estimate, you could use:

```
cl = exp(tcl + eta.cl)
```

This assumes that the initial estimate of CL (`tcl`) has been specified on the logarithmic scale, and that the parameter has a log-normal distribution in the population.

As in the case of the `ini` block, `=` and `<-` operators are equivalent.

For optimal compatibility between estimation methods, it is recommended to express parameters on the logarithmic scale (`exp(LOGTVPAR + IIV + LOGCOV*PAREFF)`, where `LOGTVPAR` is the log-transformed typical population value of the parameter, `IIV` represents the interindividual variability, `LOGCOV` represents a log-transformed covariate variable, and `PAREFF` represents the estimated covariate effect).

The order of these parameters does not matter and is analogous to "mu-referencing" in NONMEM. (Note that because interoccasional variability is not yet supported, mu-referencing applies on an individual level.) For the SAEM algorithm, the traditional parameterization of the form `PAR = TVPAR * exp(IIV)` is not allowed, and `PAR = exp(LOGTVPAR + IIV)` should be used instead. In general, numerical stability is better with mu-referencing for all fitting algorithms, and we recommend it be used wherever possible.

After defining the individual model parameters, the model can be defined directly using equations or by using `rxode2` code. The `rxode2` method of specifying the equations is based on the Leibnitz form of writing differential equations, using "d/dt" notation. Defining a set of ODEs for a "depot" compartment and its interaction with the "central" compartment in which observations are made, a concentration (`cp`) in terms of these state values could be calculated using:

```
d/dt(depot)   = -ka*depot
d/dt(central) =  ka*depot - cl/v*central

cp = central/v
```

Initial conditions for states (i.e. compartments) can be defined by defining `state(0)`:

```
depot(0) = 0
```

The order of appearance in the model code defines the compartment number to be used in `nlmixr2`'s `evid` variable (although the name of the compartment may also be used). In this case, the `depot` compartment would be 1 and the `central` compartment would be 2.

The tilde (`~`) is used to define residual error terms in the `model` block. Residual error models may be defined using either `add(parameter)` for additive error, `prop(parameter)` for proportional error, or `add(parameter1) + prop(parameter2)` for combined additive and proportional errors. The unexplained variability parameters are estimated as a standard deviation (SD) for additive errors and as a fractional coefficient of variation for proportional errors.

In the above example, we specify additive error on concentration as:

```
cp ~ add(add.err)
```

6.3 Other model types

`rxode2` supports a range of model types. So far, we've just looked at ODE systems.

6.3.1 "Prediction-only" models

"Prediction-only" models are analogous to $PRED models in NONMEM, which some readers may be familiar with. Here's a very simple example - a one-compartment model with bolus dosing.

```r
mod <- rxode2({
    ipre <- 10 * exp(-ke * t);
})
```

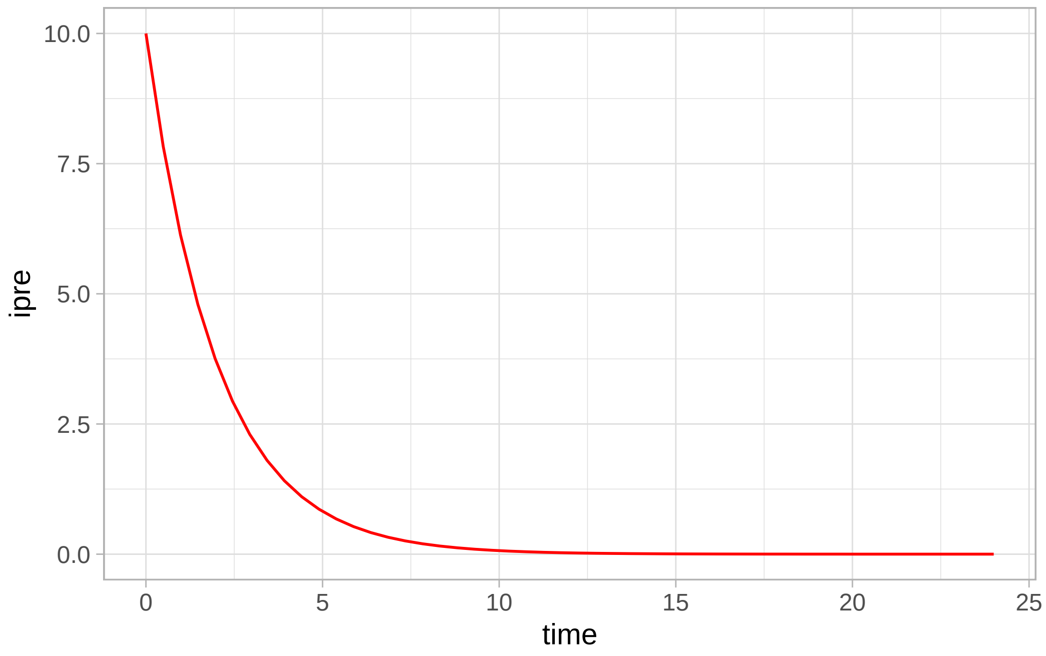
Solving prediction-only models is done just the same way as for ODE systems, but is faster.

```r
et   <- et(seq(0, 24, length.out=50))
cmt1 <- rxSolve(mod, et, params = c(ke=0.5))
cmt1
```

```
-- Solved rxode2 object --
-- Parameters (x$params): --
 ke
0.5
-- Initial Conditions (x$inits): --
named numeric(0)
```

```
-- First part of data (object): --
# A tibble: 50 x 2
   time  ipre
  <dbl> <dbl>
1 0     10
2 0.490  7.83
3 0.980  6.13
4 1.47   4.80
5 1.96   3.75
6 2.45   2.94
# i 44 more rows
```

```r
library(ggplot2)
library(patchwork)
ggplot(cmt1, aes(time, ipre)) +
  geom_line(col="red") +
  theme_light()
```



6.3.2 Solved systems

As well as using ODEs to define models, shortcuts for solved systems in the cases of one-compartment, two-compartment, and three-compartment linear PK model variants with bolus or infusion administration, possibly combined with a preceding absorption compartment, are available. Estimation is typically faster when solved systems are used. When a solved system is specified, `nlmixr2` deduces the type of compartmental model based on the parameter names defined in the `ini` block. To collapse an entire system of ODEs describing a one-compartment model and an additive residual error model to a single, simple statement, one could write the following:

```
linCmt() ~ add(add.err)
```

In this example, concentration is calculated automatically as the amount of drug in the central compartment divided by the central volume.

`rxode2` has a library of models which have been pre-solved, and thus do not need to be defined in terms of ODEs. These can be used by including `linCmt()` function in model code, along with a set of model parameters that fits the model you wish to use. For example:

Model	*Parameters*	*Microconstants*	*Hybrid constants*
One-compartment, bolus or IV dose	`cl`, `v`	`v`, `ke`	`v`, `alpha`
One-compartment, oral dose	`cl`, `v`, `ka`	`v`, `ke`, `ka`	`v`, `alpha`, `ka`
Two-compartment, bolus or IV dose	`cl`, `v1`, `v2`, `q`	`v1`, `k12`, `k21`, `ke`	`v`, `alpha`, `beta`, `aob`
Two-compartment, oral dose	`cl`, `v1`, `v2`, `q`, `ka`	`v1`, `k12`, `k21`, `ke`, `ka`	`v`, `alpha`, `beta`, `aob`, `ka`
Three-compartment, bolus or IV dose	`cl`, `vc`, `vp`, `vp2`, `q1`, `q2`	`v1`, `k12`, `k21`, `k13`, `k31`,`ke`	`a`, `alpha`, `b`, `beta`, `c`, `gamma`
Three-compartment, oral dose	`cl`, `vc`, `vp`, `vp2`, `q1`, `q2`, `ka`	`a`, `alpha`, `b`, `beta`, `c`, `gamma`, `ka`	

Most parameters can be specified in a number of ways. For example, central volume in a one-compartment model can be called `v`, `v1` or `vc`, and will be understood as the same parameter by `linCmt()`. Here's a very simple example…

```
mod_solved <- rxode2({
    ke <- 0.5
    V <- 1
    ipre <- linCmt();
})

mod_solved
```

```
rxode2 NA model named rx_cb43d14feea14a71013438c9d9d84019_x6 model (ready).
x$stateExtra: central
x$params: ke, V
x$lhs: ipre
```

We can treat this the same way as an ODE model:

```
et  <- et(amt=10, time=0, cmt=depot) %>%
    et(seq(0, 24, length.out=50))
cmt_solved <- rxSolve(mod_solved, et, params=c(ke=0.5))
cmt_solved
```

```
-- Solved rxode2 object --
-- Parameters (x$params): --
 ke   V
0.5 1.0
-- Initial Conditions (x$inits): --
named numeric(0)
-- First part of data (object): --
# A tibble: 50 x 2
   time  ipre
  <dbl> <dbl>
1 0     10
2 0.490  7.83
3 0.980  6.13
4 1.47   4.80
5 1.96   3.75
6 2.45   2.94
# i 44 more rows
```

6.3.3 Combining ODEs and solved systems

Solved systems and ODEs can be combined. Let's look at our two-compartment indirect-effect model again.

```
## Set up parameters and initial conditions

theta <- c(KA   = 0.294,
           CL   = 18.6,
           V2   = 40.2,
           Q    = 10.5,
```

```r
           V3   = 297,
           Kin  = 1,
           Kout = 1,
           EC50 = 200)

inits <- c(eff = 1);

## Set up dosing event information

ev <- eventTable(amount.units='mg', time.units='hours') %>%
    add.dosing(dose=10000, nbr.doses=10, dosing.interval=12, dosing.to=1) %>%
    add.dosing(dose=20000, nbr.doses=5, start.time=120,dosing.interval=24, dosing.to=1) %>%
    add.sampling(0:240);

## Set up a mixed solved/ODE system

mod2 <- rxode2({
  ## the order of variables do not matter
  ## the type of compartmental model is determined by the parameters specified

    C2         = linCmt(KA, CL, V2, Q, V3);  # you don't need to provide the parameters, but 
    eff(0)     = 1   ## The initial amount in the effect compartment is 1
    d/dt(eff) =  Kin - Kout*(1 - C2/(EC50 + C2))*eff;
})
```

```r
mod2
```

```
rxode2 NA model named rx_893a1b9c542a31101b3500a8291bf850_x6 model (ready).
x$state: eff
x$stateExtra: depot, central
x$params: CL, V2, Q, V3, KA, Kin, Kout, EC50
x$lhs: C2
```

Concentration output from the 2-compartment model is assigned to the `C2` variable and is subsequently used in the indirect response model.

Ntoe that when mixing solved systems and ODEs, the solved system's "compartment" is always the last one. This is because the solved system technically isn't a compartment as such. Adding the dosing compartment to the end will not interfere with the actual ODE to be solved.

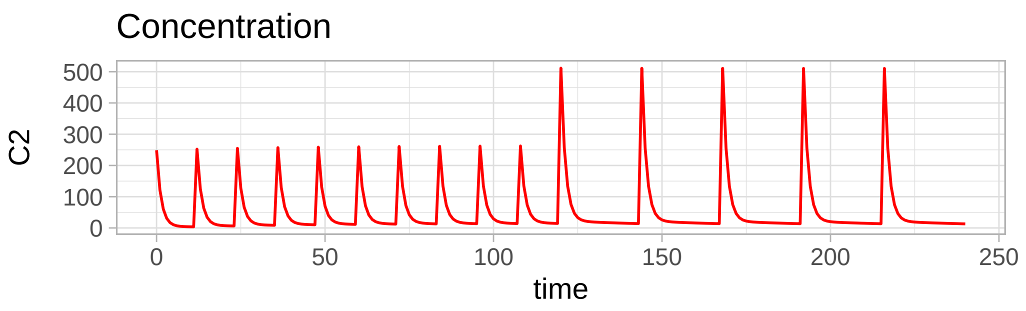
In this example, therefore, the effect compartment is compartment #1 while the PK dosing compartment for the depot is compartment #2.

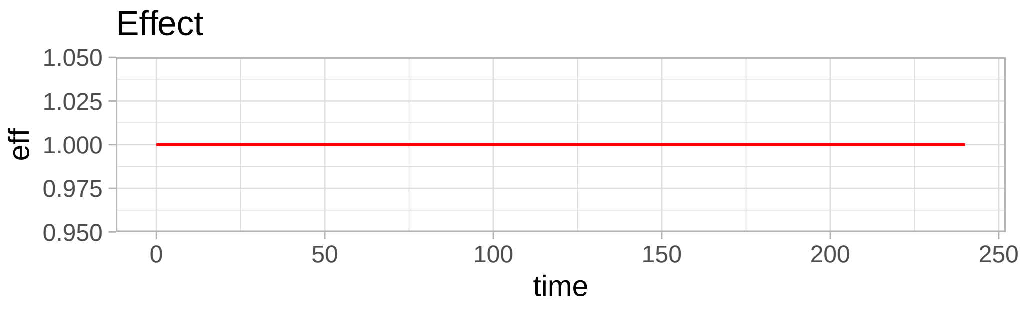
Let's solve the system and see what it looks like.

```r
x <- mod2 %>%  solve(theta, ev)
print(x)
```

```
-- Solved rxode2 object --
-- Parameters ($params): --
     CL      V2       Q      V3      KA     Kin    Kout    EC50
 18.600  40.200  10.500 297.000   0.294   1.000   1.000 200.000
-- Initial Conditions ($inits): --
eff
  1
-- First part of data (object): --
# A tibble: 241 x 3
   time    C2   eff
  <dbl> <dbl> <dbl>
1     0 249.      1
2     1 121.      1
3     2  60.3     1
4     3  31.0     1
5     4  17.0     1
6     5  10.2     1
# i 235 more rows
```

```r
p1 <- ggplot(x, aes(time, C2)) +
  geom_line(col="red") +
  theme_light() +
  labs(title="Concentration")

p2 <- ggplot(x, aes(time, eff)) +
  geom_line(col="red") +
  theme_light()+
  labs(title="Effect")

p1 + p2 + plot_layout(nrow=2)
```





6.4 Preparing a model for use

If not already defined in the `ini` block, parameters can be specified via the `params` argument in the `solve()` method to differentiate them from dynamic state variables.

```
m1 <- rxode2(model = ode, modName = "m1")

# model parameters -- a named vector is required
theta <- c(KA   = 0.29,
           CL   = 18.6,
           V2   = 40.2,
           Q    = 10.5,
           V3   = 297,
           Kin  = 1,
           Kout = 1,
           EC50 = 200)

# state variables and their amounts at time 0 (the use of names is
# encouraged, but not required)
inits <- c(depot=0, centr=0, peri=0, eff=1)

# qd1 is an eventTable specification with a set of dosing and sampling
# records (code not shown here)
```

```
rxSolve(m1, params = theta, events = qd1, inits = inits)
```

The values of these variables at pre-specified time points are saved during model fitting/integration and returned as part of the fitted values (see the function `et()`, to define a set of time points when to capture the values of these variables) and returned as part of the modeling output.

7 Events in `rxode2` and `nlmixr2`

7.1 Understanding events

Before we wade into writing models for `rxode2`, we should first understand events, which describe what happens and when, and provides the model with inputs it can work with. Events in `rxode2` are defined in event tables.

7.1.1 Event table structure

`rxode2` event tables contain a set of data items which allow fine-grained control of doses and observations over time. Here is a summary.

Data Item	Meaning	Notes
`id`	Individual identifier	Can be a integer, factor, character, or numeric.
`time`	Individual time	Numeric for each time.
`amt`	Dose amount	Positive for doses, zero or `NA` for observations.
`rate`	Infusion rate	When specified, the infusion duration will be `dur`=`amt`/`rate`. Special cases: when `rate` = -1, rate is modeled; when `rate` = -2, duration is modeled.
`dur`	Infusion duration	When specified, the infusion rate will be `rate` = `amt`/`dur`
`evid`	Event ID	Reserved codes are used in this field. `0`=observation; `1`=dose; `2`=other; `3`=reset; `4`=reset and dose; `5`=replace; `6`=multiply; `7`=transit
`cmt`	Compartment	Represents compartment number or name for dose or observation
`ss`	Steady state flag	Reserved codes are used in this field. 0=non-steady-state; 1=steady state; 2=steady state + prior states
`ii`	Inter-dose interval	Time between doses defined in `addl`.

Data Item	Meaning	Notes
`addl`	Number of additional doses	Number of doses identical to the current dose, separated by `ii`.

Those with experience using NONMEM will immediately recognize the convesntions used here. There are some differences, however…

7.1.1.1 Compartments (`cmt`)

- The compartment data item (`cmt`) can be a string or factor using actual compartment names, as well as a number
- You may turn off a compartment using a negative compartment number, or "`-cmtName`" where `cmtName` is the compartment name as a string.
- The compartment data item (`cmt`) can be a number. The number of the compartment is defined by the order of appearance of the compartment name in the model. This can be tedious to keep track of, so you can specify compartment numbers easier by listing compartments via `cmt(cmtName)` in the order you would like at the beginning of the model, where `cmtName` is the name of the compartment.

7.1.1.2 Duration (`dur`) and rate (`rate`) of infusion

- The duration data item, `dur`, specifies the duration of infusions.
- Bioavailability changes will change the rate of infusion, since `dur` and `amt` are fixed in the input data.
- Similarly, when specifying `rate`/`amt` for an infusion, changing the bioavailability will change the infusion duration, since `rate` and `amt` are fixed in the input data.

7.1.1.3 Event IDs (`evid`)

- NONMEM-style events are supported (0: Observation, 1: Dose, 2: Other, 3: Reset, 4: Reset and dose).

- A number of additional event types are also included:
  - Replace events (`evid`=5): This replaces the amount in a compartment with the value specified in the `amt` column. This is equivalent to `deSolve`=`replace`.
  - Multiply events (`evid`=6): This multiplies the value in the compartment with the value specified by the `amt` column. This is equivalent to `deSolve`=`multiply`.

    - Transit or phantom event (`evid`=7): This puts the dose in the `dose()` function and calculates time since last dose (`tad()`) but doesn't actually put the dose in the compartment. This allows the `transit()` function to be applied to the compartment easily.

Simulated datasets generated by `rxode2` may contain records with a few additional event types, depending on the options used for solving:

- `EVID = -1` denotes when a modeled rate ends (corresponds to `rate` = -1)
- `EVID = -2` denotes when a modeled duration ends (corresponds to `rate` = -2)
- `EVID = -10` denotes when a rate-specified zero-order infusion ends (corresponds to `rate` > 0)
- `EVID = -20` denotes when a duration specified zero-order infusion ends (corresponds to `dur` > 0)
- `EVID = 101, 102, 103,...` correspond to modeled times `1`, `2`, `3`, … (`mtime`).

These are provided when solving with the option combination `addDosing=TRUE` and `subsetNonmem=FALSE`. If you want to see the classic `EVID` equivalents, `addDosing=NA` can be specified.

7.1.1.4 Other notes

- `evid` can be the classic `RxODE` style (described here) or the `NONMEM`-style `evid` we have described above.
- Dependent variable (`DV`) is not required; `rxode2` is a ODE solving framework and doesn't need it.
- A flag for missing dependent variable (`MDV`) is not required; it is captured in `evid`.
- Instead of `NONMEM`-compatible data, `deSolve`-compatible data-frames can also be accepted.

7.1.2 Dosing

7.1.2.1 Bolus and additive doses

A bolus dose is the default type of dose in `rxode2` and only requires `amt`. For setting up event tables, we use the convenience function `et()`, as in the example below. Notice as well how we are using piping from the `magrittr` package (you don't need to, but it makes things a lot easier to type and read).

Here's an example: we'll use a simple PK model for illustrative purposes (we'll explain how it works in detail later on).

```r
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.1     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.1
v purrr     1.0.2     
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becom
```

```r
library(ggplot2)
library(rxode2)
```

```
rxode2 3.0.2 using 16 threads (see ?getRxThreads)
```

```r
mod1 <- function() {
  ini({
    # central compartment
    KA   = 0.294  # /h
    CL   = 18.6   # L/h
    V2   = 40.2   # L

    # peripheral compartment
    Q    = 10.5   # L/h
    V3   = 297    # L
    fdepot <- 1
  })
  model({
    C2          <- centr/V2   # concentration in the central compartment
    C3          <- peri/V3    # concentration in the peripheral compartment

    d/dt(depot) <- -KA*depot                        # depot compartment
    d/dt(centr) <-  KA*depot - CL*C2 - Q*C2 + Q*C3   # central compartment
    d/dt(peri)  <-                     Q*C2 - Q*C3   # peripheral compartment
    f(depot)    <- fdepot # set bioavailability
  })
}
```

We'll give 3 doses of 10000 mg, every 12 hours until 24 h, and 100 equally-spaced observations between 0 and 24 hours. We also specify time units of "hours", which adds this to the output object specification.

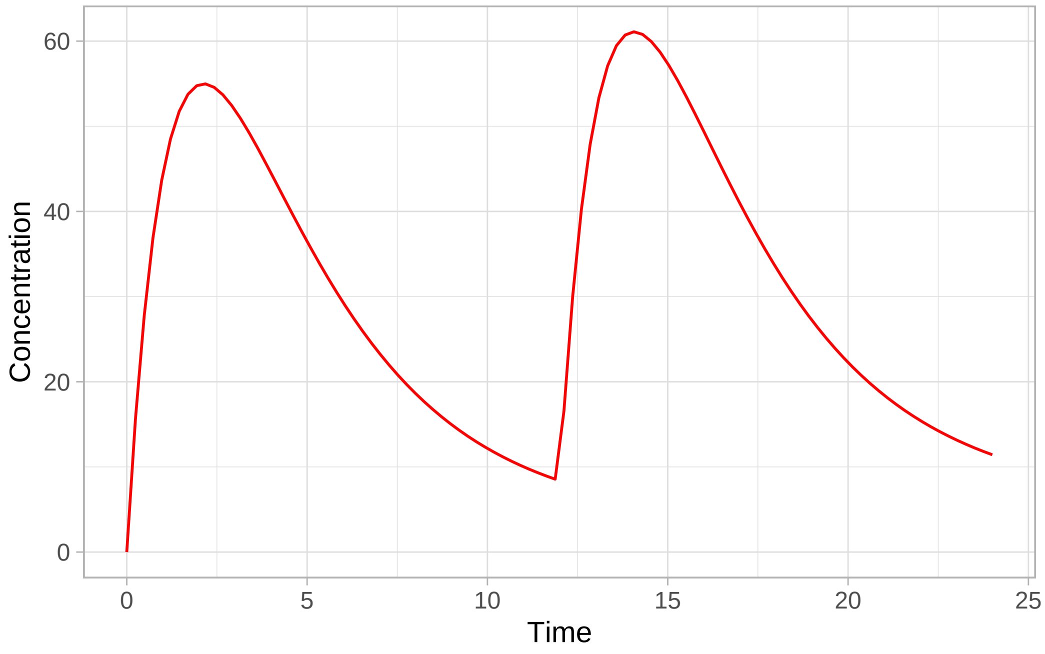
```
ev <- et(timeUnits="hr") %>%                # using hours
    et(amt=10000, ii=12, until=24) %>%      # 10000 mg, 2 doses spaced by 12 hours
    et(seq(0, 24, length.out=100))          # 100 observations between 0 and 24 h, equally spa

ev
```

```
-- EventTable with 101 records --
1 dosing records (see x$get.dosing(); add with add.dosing or et)
100 observation times (see x$get.sampling(); add with add.sampling or et)
multiple doses in `addl` columns, expand with x$expand(); or etExpand(x)
-- First part of x: --
# A tibble: 101 x 5
    time   amt    ii  addl evid
   <dbl> <dbl> <dbl> <int> <evid>
 1 0        NA    NA    NA 0:Observation
 2 0     10000    12     2 1:Dose (Add)
 3 0.242    NA    NA    NA 0:Observation
 4 0.485    NA    NA    NA 0:Observation
 5 0.727    NA    NA    NA 0:Observation
 6 0.970    NA    NA    NA 0:Observation
 7 1.21     NA    NA    NA 0:Observation
 8 1.45     NA    NA    NA 0:Observation
 9 1.70     NA    NA    NA 0:Observation
10 1.94     NA    NA    NA 0:Observation
# i 91 more rows
```

```
rxSolve(mod1, ev) %>% ggplot(aes(time, C2)) +
    geom_line(col="red") +
  xlab("Time") + ylab("Concentration") +
  theme_light()
```

```
i parameter labels from comments are typically ignored in non-interactive mode
```

```
i Need to run with the source intact to parse comments
```



7.1.2.2 Infusions

`rxode2` supports several different kinds of infusion.

- Constant rate infusion (using `rate`)
- Constant duration infusion (using `dur`)
- Estimated rate of infusion
- Estimated duration of infusion

7.1.2.2.1 Constant infusions

There are two ways to specify an infusion in `rxode2`. The first is to use the `dur` field. Here we specify an infusion given over 8 hours.

```r
ev <- et(timeUnits="hr") %>%
    et(amt=10000, ii=12, until=24, dur=8) %>%
    et(seq(0, 24, length.out=100))

ev
```

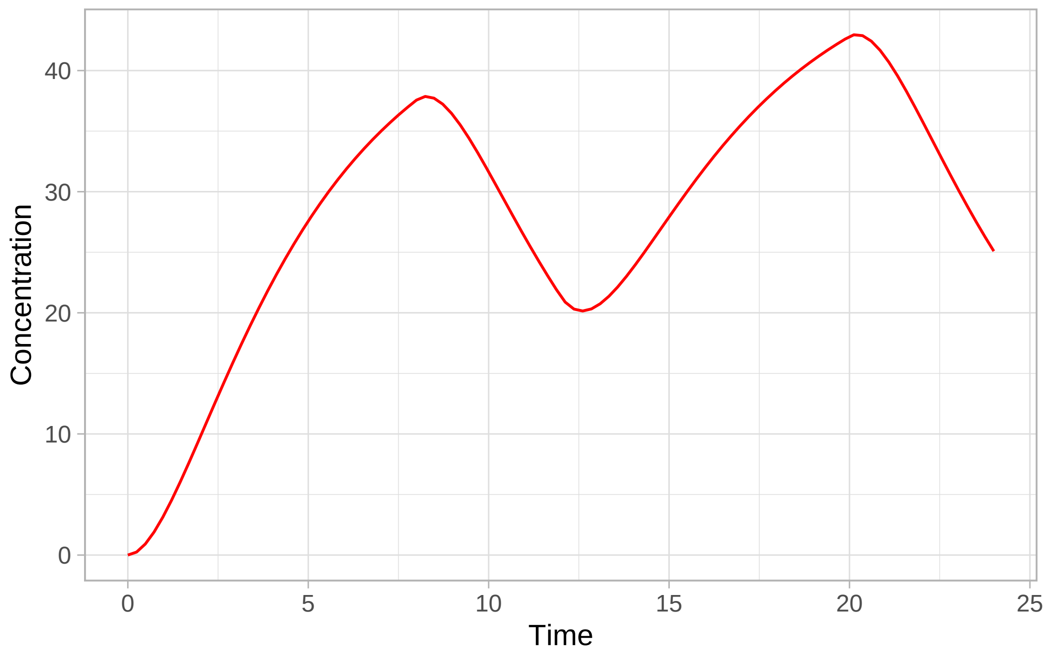
```
-- EventTable with 101 records --
1 dosing records (see x$get.dosing(); add with add.dosing or et)
100 observation times (see x$get.sampling(); add with add.sampling or et)
```

```
multiple doses in `addl` columns, expand with x$expand(); or etExpand(x)
-- First part of x: --
# A tibble: 101 x 6
    time   amt    ii  addl evid          dur
   <dbl> <dbl> <dbl> <int> <evid>        <rate/dur>
 1 0        NA    NA    NA 0:Observation NA
 2 0     10000    12     2 1:Dose (Add)   8
 3 0.242    NA    NA    NA 0:Observation NA
 4 0.485    NA    NA    NA 0:Observation NA
 5 0.727    NA    NA    NA 0:Observation NA
 6 0.970    NA    NA    NA 0:Observation NA
 7 1.21     NA    NA    NA 0:Observation NA
 8 1.45     NA    NA    NA 0:Observation NA
 9 1.70     NA    NA    NA 0:Observation NA
10 1.94     NA    NA    NA 0:Observation NA
# i 91 more rows
```

```r
rxSolve(mod1, ev) %>% ggplot(aes(time, C2)) +
    geom_line(col="red") +
  xlab("Time") + ylab("Concentration") +
  theme_light()
```

```
i parameter labels from comments are typically ignored in non-interactive mode
```

```
i Need to run with the source intact to parse comments
```



We could, alternatively, specify `rate` instead.

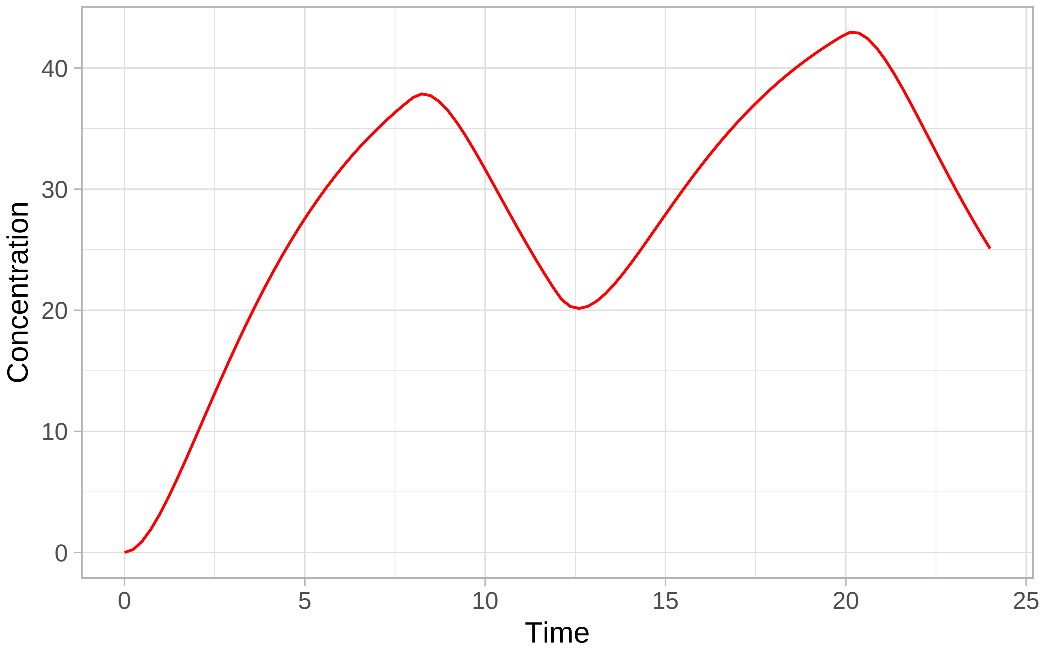
```
ev <- et(timeUnits="hr") %>%
    et(amt=10000, ii=12, until=24, rate=10000/8) %>%
    et(seq(0, 24, length.out=100))

ev
```

```
-- EventTable with 101 records --
1 dosing records (see x$get.dosing(); add with add.dosing or et)
100 observation times (see x$get.sampling(); add with add.sampling or et)
multiple doses in `addl` columns, expand with x$expand(); or etExpand(x)
-- First part of x: --
# A tibble: 101 x 6
    time   amt rate          ii  addl evid
   <dbl> <dbl> <rate/dur> <dbl> <int> <evid>
 1 0        NA NA            NA    NA 0:Observation
 2 0     10000  1250         12     2 1:Dose (Add)
 3 0.242    NA NA            NA    NA 0:Observation
 4 0.485    NA NA            NA    NA 0:Observation
 5 0.727    NA NA            NA    NA 0:Observation
 6 0.970    NA NA            NA    NA 0:Observation
 7 1.21     NA NA            NA    NA 0:Observation
 8 1.45     NA NA            NA    NA 0:Observation
 9 1.70     NA NA            NA     NA 0:Observation
10 1.94     NA NA            NA     NA 0:Observation
# i 91 more rows
```

```
rxSolve(mod1, ev) %>% ggplot(aes(time, C2)) +
    geom_line(col="red") +
  xlab("Time") + ylab("Concentration") +
  theme_light()
```

```
i parameter labels from comments are typically ignored in non-interactive mode
```

```
i Need to run with the source intact to parse comments
```

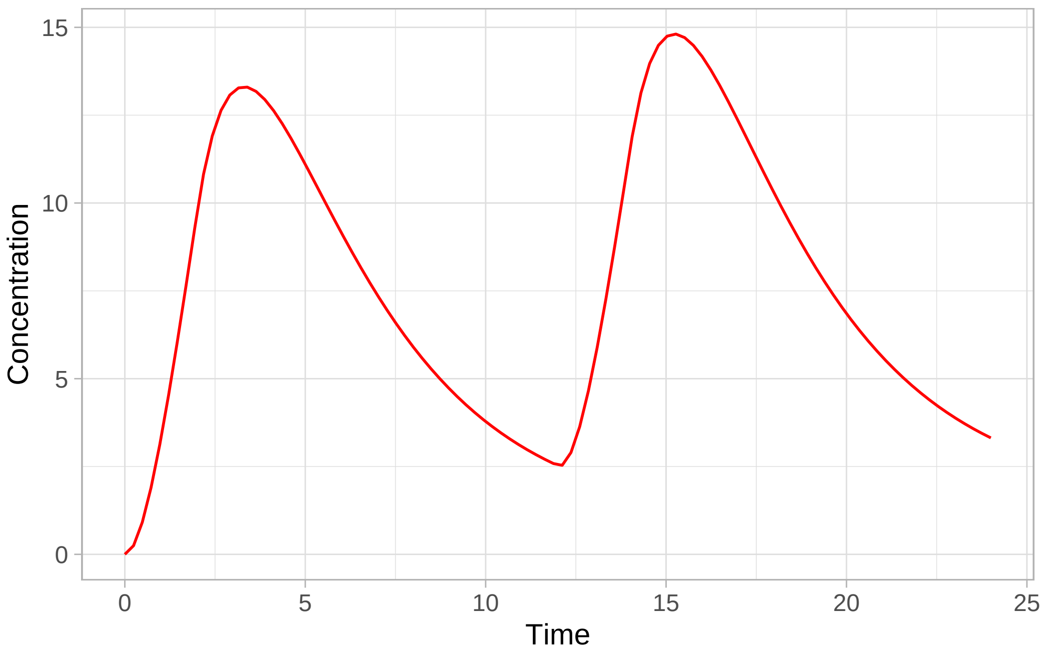


As you can see, we get the same result whichever way we do it. Where we need to pay attention is bioavailability (F). If we change F, the infusion is changed.

When we apply `rate`, a bioavailability decrease decreases the infusion duration. For instance:

```
# other parameters are as before
other_params <- c(
    KA = 0.294,
    CL = 18.6,
    V2 = 40.2,
    Q = 10.5,
    V3 = 297)

rxSolve(mod1, ev, params = c(other_params, fdepot=0.25)) %>% ggplot(aes(time, C2)) +
    geom_line(col="red") +
  xlab("Time") + ylab("Concentration") +
  theme_light()
```

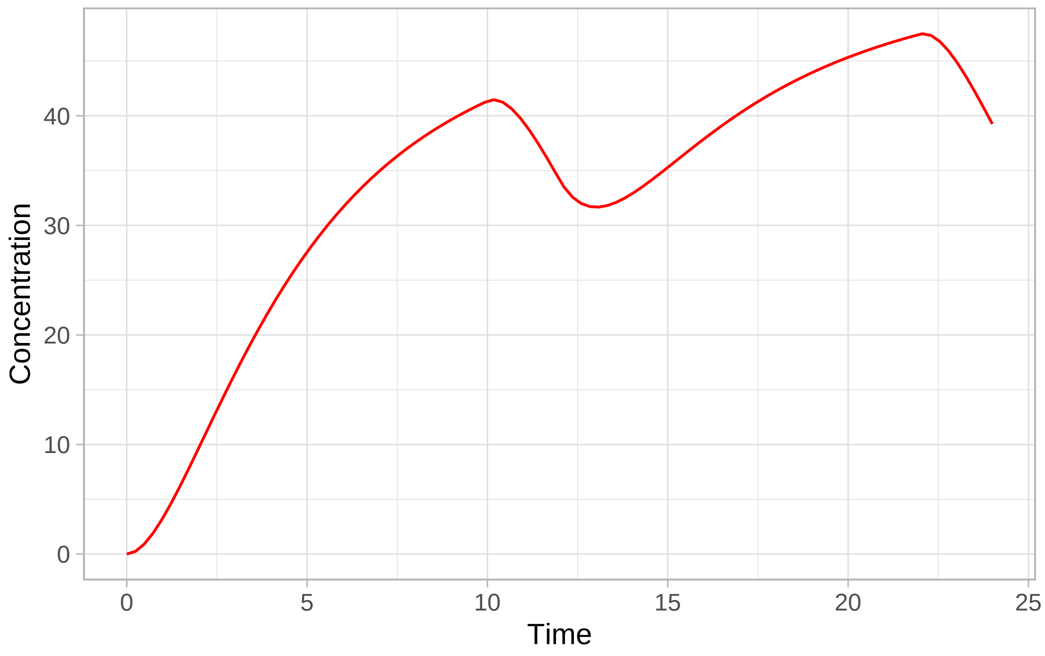
```
i parameter labels from comments are typically ignored in non-interactive mode

i Need to run with the source intact to parse comments
```



Similarly, increasing the bioavailability increases the infusion duration.

```
rxSolve(mod1, ev, params = c(other_params, fdepot=1.25)) %>% ggplot(aes(time, C2)) +
    geom_line(col="red") +
  xlab("Time") + ylab("Concentration") +
  theme_light()
```

```
i parameter labels from comments are typically ignored in non-interactive mode

i Need to run with the source intact to parse comments
```



The rationale for this behavior is that the `rate` and `amt` variables are specified by the event table, so the only thing that can change with a bioavailability increase is the duration of the infusion. Similarly, when specifying the `amt` and `dur` components in the event table, bioavailability changes affect the `rate` of infusion.

```
ev <- et(timeUnits="hr") %>%
    et(amt=10000, ii=12,until=24, dur=8) %>%
    et(seq(0, 24, length.out=100))
```

Looking at the two approaches side by side, we can clearly see the differences in the `depot` compartment.

```
library(ggplot2)
library(patchwork)

p0 <- rxSolve(mod1, ev) %>% ggplot(aes(time, C2)) +
    geom_line(col="red") +
  coord_cartesian(ylim=c(0,60)) +
  xlab("Time") + ylab("Concentration") +labs(title="F = 1")+
  theme_light()
```

```
i parameter labels from comments are typically ignored in non-interactive mode
```

```
i Need to run with the source intact to parse comments
```

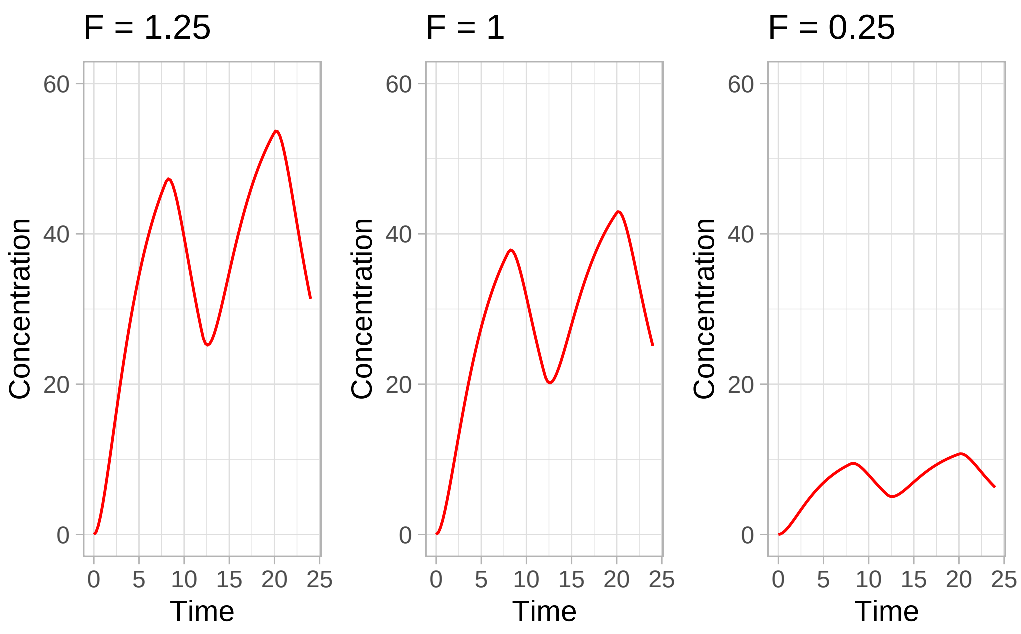
```
p1 <- rxSolve(mod1, ev, params = c(other_params, fdepot=1.25)) %>% ggplot(aes(time, C2)) +
    geom_line(col="red") +
  coord_cartesian(ylim=c(0,60)) +
  xlab("Time") + ylab("Concentration") +labs(title="F = 1.25")+
  theme_light()
```

```
i parameter labels from comments are typically ignored in non-interactive mode
i Need to run with the source intact to parse comments
```

```
p2 <- rxSolve(mod1, ev, params = c(other_params, fdepot=0.25)) %>% ggplot(aes(time, C2)) +
    geom_line(col="red") +
  coord_cartesian(ylim=c(0,60)) +
  xlab("Time") + ylab("Concentration") + labs(title="F = 0.25")+
  theme_light()
```

```
i parameter labels from comments are typically ignored in non-interactive mode
i Need to run with the source intact to parse comments
```

```
## Use patchwork syntax to combine plots
p1 + p0 + p2
```



7.1.2.3 Steady state

7.1.2.3.1 Dosing to staedy state

When the steady-state (`ss`) flag is set, doses are solved until steady state is reached with a constant inter-dose interval.

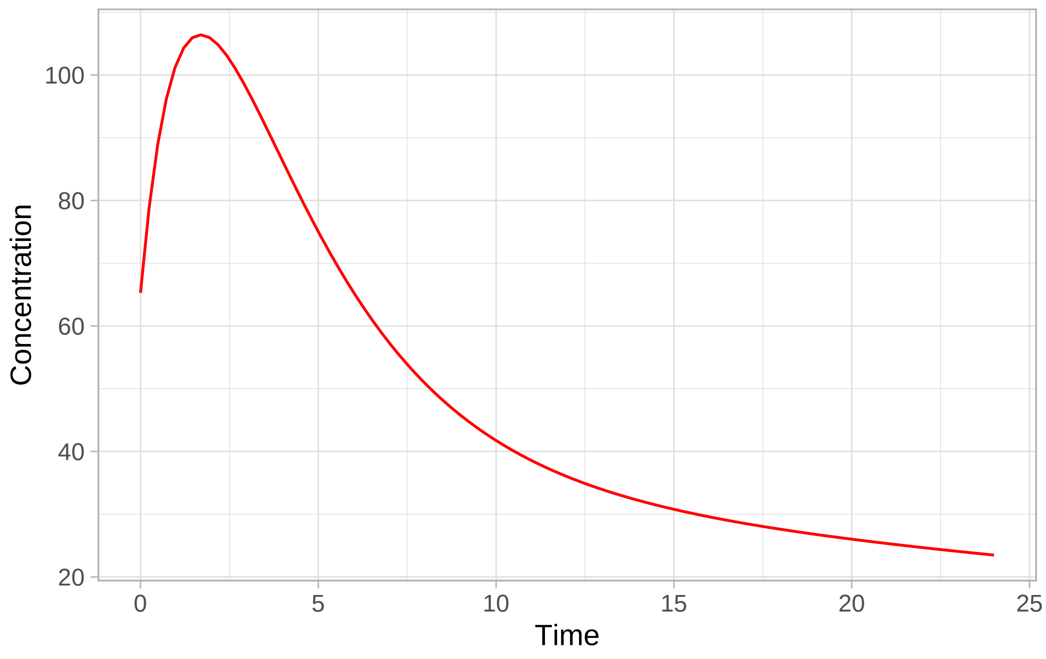
```r
ev <- et(timeUnits="hr") %>%
    et(amt=10000, ii=6, ss=1) %>%
    et(seq(0, 24, length.out=100))

ev
```

```
-- EventTable with 101 records --
1 dosing records (see x$get.dosing(); add with add.dosing or et)
100 observation times (see x$get.sampling(); add with add.sampling or et)
-- First part of x: --
# A tibble: 101 x 5
    time   amt    ii evid               ss
   <dbl> <dbl> <dbl> <evid>          <int>
 1 0        NA    NA 0:Observation    NA
 2 0     10000     6 1:Dose (Add)      1
 3 0.242    NA    NA 0:Observation    NA
 4 0.485    NA    NA 0:Observation    NA
 5 0.727    NA    NA 0:Observation    NA
 6 0.970    NA    NA 0:Observation    NA
 7 1.21     NA    NA 0:Observation    NA
 8 1.45     NA    NA 0:Observation    NA
 9 1.70     NA    NA 0:Observation    NA
10 1.94     NA    NA 0:Observation    NA
# i 91 more rows
```

```r
rxSolve(mod1, ev) %>% ggplot(aes(time, C2)) +
    geom_line(col="red") +
  xlab("Time") + ylab("Concentration") +
  theme_light()
```

```
i parameter labels from comments are typically ignored in non-interactive mode
```

```
i Need to run with the source intact to parse comments
```



7.1.2.3.2 Steady state for complex dosing

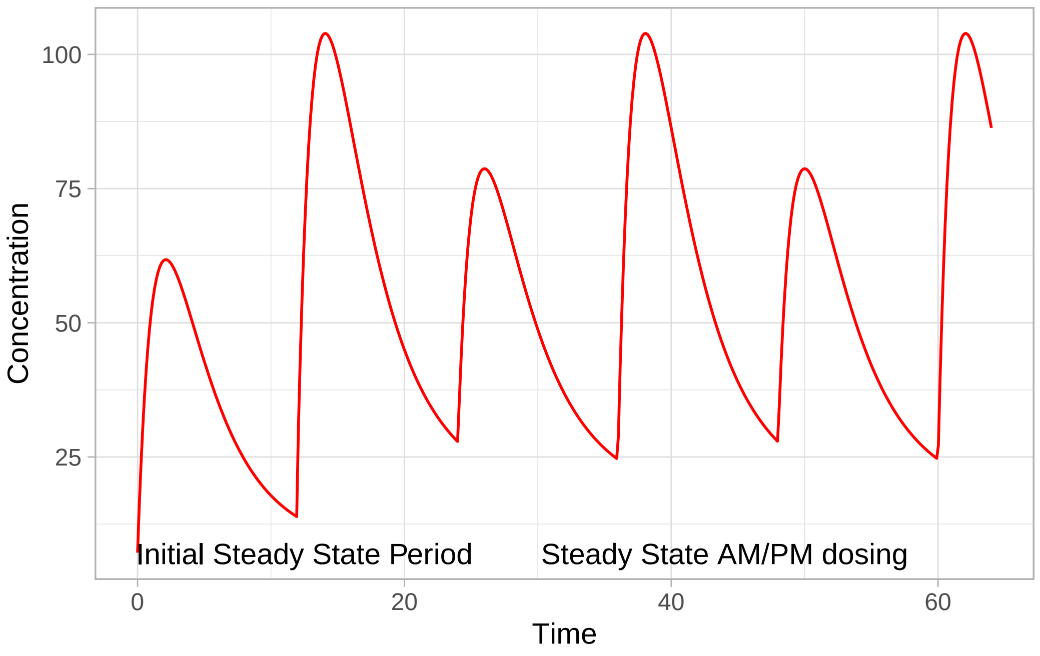
By using the `ss=2` flag, the super-positioning principle in linear kinetics can be applied to set up nonstandard dosing to steady state. In thise xample, we're giving different deses in mornings and evenings.

```
ev <- et(timeUnits="hr") %>%
    et(amt=10000, ii=24, ss=1) %>%
    et(time=12, amt=15000, ii=24, ss=2) %>%
    et(time=24, amt=10000, ii=24, addl=3) %>%
    et(time=36, amt=15000, ii=24, addl=3) %>%
    et(seq(0, 64, length.out=500))

ev$get.dosing()
```

```
  id low time high       cmt   amt rate ii addl evid ss dur
1  1  NA    0   NA (default) 10000    0 24    0    1  1   0
2  1  NA   12   NA (default) 15000    0 24    0    1  2   0
3  1  NA   24   NA (default) 10000    0 24    3    1  0   0
4  1  NA   36   NA (default) 15000    0 24    3    1  0   0
```

```r
rxSolve(mod1, ev,maxsteps=10000) %>% ggplot(aes(time, C2)) +
    geom_line(col="red") +
  xlab("Time") + ylab("Concentration") +
  theme_light() +
  annotate("text", x=12.5, y=7,
           label="Initial Steady State Period") +
  annotate("text", x=44,   y=7,
           label="Steady State AM/PM dosing")
```

```
i parameter labels from comments are typically ignored in non-interactive mode

i Need to run with the source intact to parse comments
```



As you can see, it takes a full dose cycle to reach true steady state.

7.1.2.3.3 Steady state for constant infusion or zero order processes

The last type of steady state dosing that `rxode2` supports is constant infusion rate. This can be specified as follows:

- No inter-dose interval: `ii`=`0`
- A steady state dose: `ss`=`1`
- Either a positive rate (`rate`>0) or an estimated rate (`rate`=`-1`)

- A zero dose: `amt=0`

Once the steady-state constant infusion is achieved, the infusion is turned off.

Note that `rate=-2`, in which we model the duration of infusion, doesn't really make much sense in this situation, since we are solving the infusion until steady state is reached. The duration is specified by the steady state solution.

Also note that bioavailability changes on this steady state infusion also do not make sense, because they neither change the `rate`, nor the duration of the steady state infusion. Modeled bioavailability on this type of dosing event is therefore ignored by `rxode2`.

Here is an example:

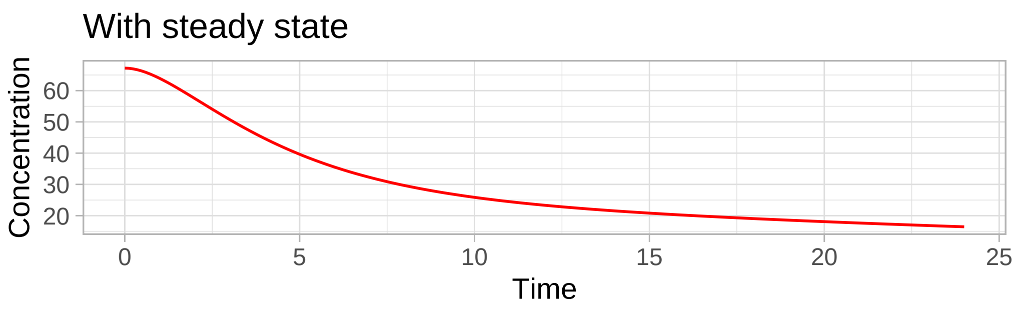
```r
ev <- et(timeUnits="hr") %>%
    et(amt=0, ss=1,rate=10000/8)

p1 <- rxSolve(mod1, ev) %>% ggplot(aes(time, C2)) +
    geom_line(col="red") +
  xlab("Time") + ylab("Concentration") +
  theme_light()+
  labs(title="With steady state")
```

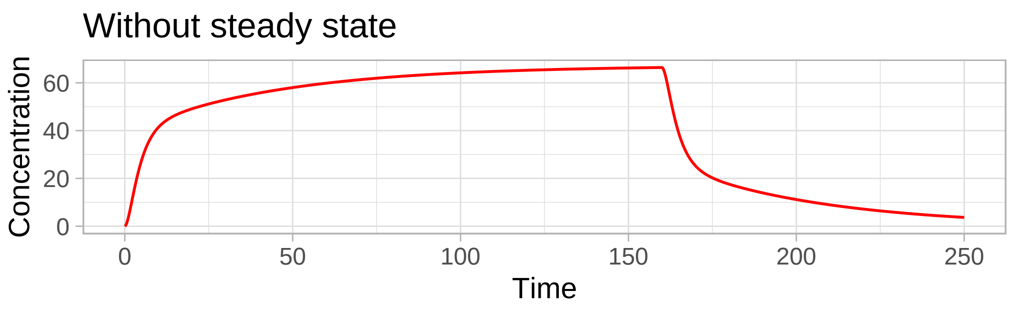
```
i parameter labels from comments are typically ignored in non-interactive mode

i Need to run with the source intact to parse comments
```

```r
ev <- et(timeUnits="hr") %>%
    et(amt=200000, rate=10000/8) %>%
    et(0, 250, length.out=1000)

p2 <- rxSolve(mod1, ev) %>% ggplot(aes(time, C2)) +
    geom_line(col="red") +
  xlab("Time") + ylab("Concentration") +
  theme_light()+
  labs(title="Without steady state")
```

```
i parameter labels from comments are typically ignored in non-interactive mode
i Need to run with the source intact to parse comments
```

```r
library(patchwork)

p1 / p2
```





7.1.2.4 Reset Events

Reset events are implemented by specifying `evid=3` or `evid=reset`, for reset, or `evid=4` for reset-and-dose.

Let's see what happens in this system when we reset it at 6 hours post-dose.

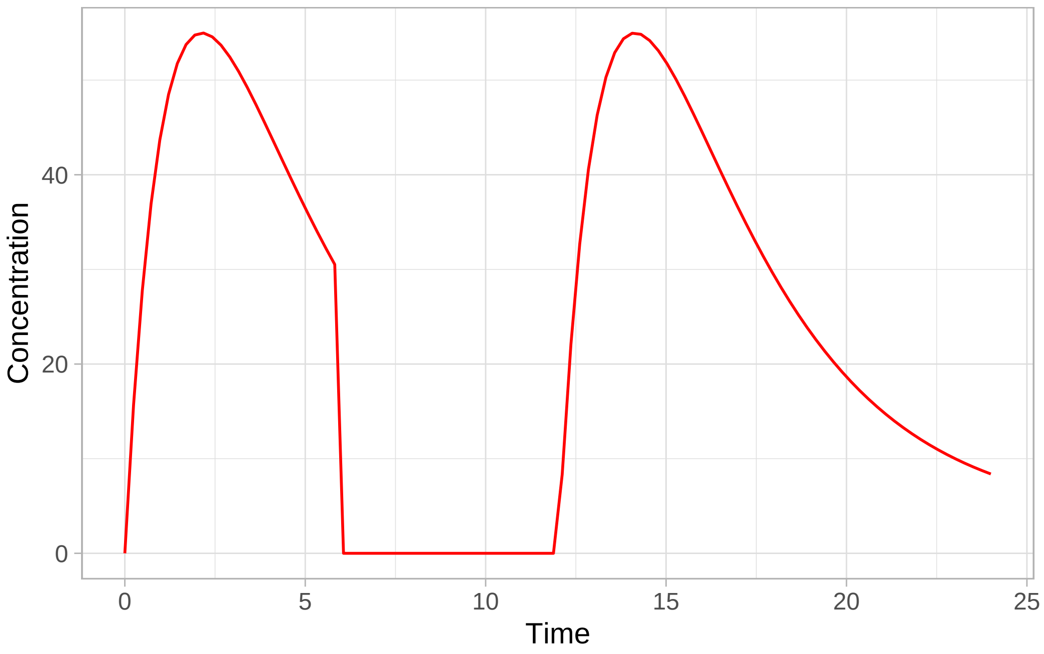
```
ev <- et(timeUnits="hr") %>%
    et(amt=10000, ii=12, addl=3) %>%
    et(time=6, evid=reset) %>%
    et(seq(0, 24, length.out=100))

ev
```

```
-- EventTable with 102 records --
2 dosing records (see x$get.dosing(); add with add.dosing or et)
100 observation times (see x$get.sampling(); add with add.sampling or et)
multiple doses in `addl` columns, expand with x$expand(); or etExpand(x)
-- First part of x: --
# A tibble: 102 x 5
    time   amt    ii  addl evid
   <dbl> <dbl> <dbl> <int> <evid>
 1 0        NA    NA    NA 0:Observation
 2 0     10000    12     3 1:Dose (Add)
 3 0.242    NA    NA    NA 0:Observation
 4 0.485    NA    NA    NA 0:Observation
 5 0.727    NA    NA    NA 0:Observation
 6 0.970    NA    NA    NA 0:Observation
 7 1.21     NA    NA    NA 0:Observation
 8 1.45     NA    NA    NA 0:Observation
 9 1.70     NA    NA    NA 0:Observation
10 1.94     NA    NA    NA 0:Observation
# i 92 more rows
```

```r
rxSolve(mod1, ev) %>% ggplot(aes(time, C2)) +
    geom_line(col="red") +
  xlab("Time") + ylab("Concentration") +
  theme_light()
```

```
i parameter labels from comments are typically ignored in non-interactive mode
```

```
i Need to run with the source intact to parse comments
```



All the compartments in the system are reset to their initial values. The next dose starts the dosing cycle at the beginning again.

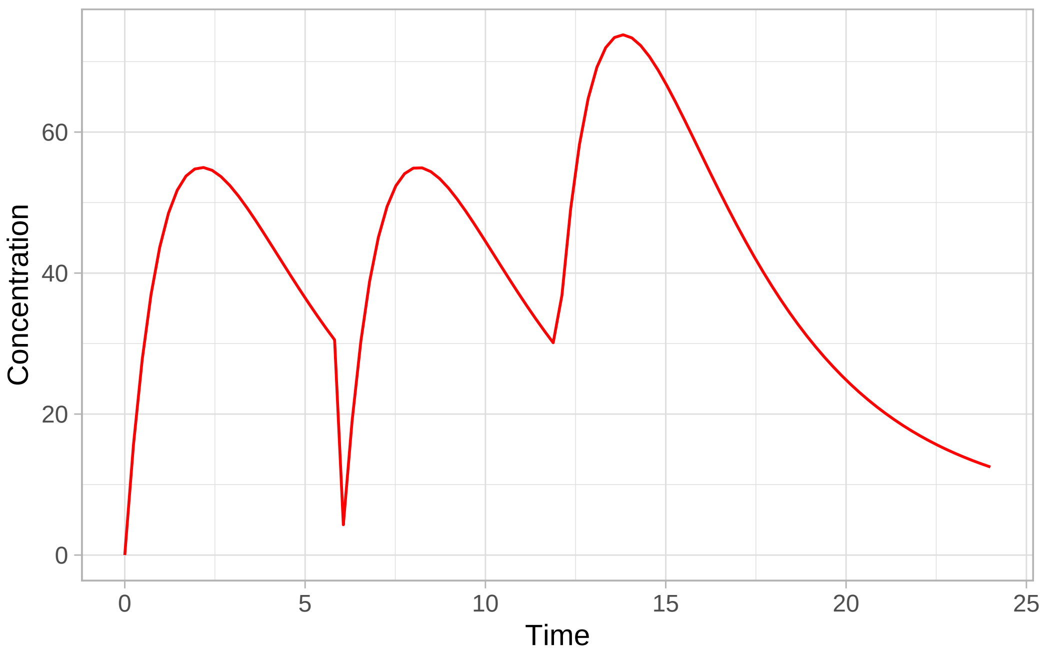
Now let's do a reset-and-dose.

```
ev <- et(timeUnits="hr") %>%
    et(amt=10000, ii=12, addl=3) %>%
    et(time=6, amt=10000, evid=4) %>%
    et(seq(0, 24, length.out=100))

ev
```

```
-- EventTable with 102 records --
2 dosing records (see x$get.dosing(); add with add.dosing or et)
100 observation times (see x$get.sampling(); add with add.sampling or et)
multiple doses in `addl` columns, expand with x$expand(); or etExpand(x)
-- First part of x: --
# A tibble: 102 x 5
    time   amt    ii  addl evid
   <dbl> <dbl> <dbl> <int> <evid>
 1 0        NA    NA    NA 0:Observation
 2 0     10000    12     3 1:Dose (Add)
 3 0.242    NA    NA    NA 0:Observation
 4 0.485    NA    NA    NA 0:Observation
 5 0.727    NA    NA    NA 0:Observation
 6 0.970    NA    NA    NA 0:Observation
 7 1.21     NA    NA    NA 0:Observation
 8 1.45     NA    NA    NA 0:Observation
 9 1.70     NA    NA    NA 0:Observation
10 1.94     NA    NA    NA 0:Observation
# i 92 more rows
```

```
rxSolve(mod1, ev) %>% ggplot(aes(time, C2)) +
    geom_line(col="red") +
  xlab("Time") + ylab("Concentration") +
  theme_light()
```

```
i parameter labels from comments are typically ignored in non-interactive mode

i Need to run with the source intact to parse comments
```



Here, we give the most recent dose again when we reset the system. Admittedly, these types of events have limited applicability in most circumstances but can occasionally be useful.

7.1.2.5 Turning off compartments

You may also turn off a compartment, which gives the following kind of result (in this example, we're turning off the `depot` compartment at 6 hours post-dose). This can be useful when one needs to model things like gastric emptying.

```
ev <- et(timeUnits="hr") %>%
    et(amt=10000, ii=12, addl=3) %>%
    et(time=6, cmt="-depot", evid=2) %>%
    et(seq(0, 24, length.out=100))

ev
```

```
-- EventTable with 102 records --
2 dosing records (see x$get.dosing(); add with add.dosing or et)
100 observation times (see x$get.sampling(); add with add.sampling or et)
multiple doses in `addl` columns, expand with x$expand(); or etExpand(x)
-- First part of x: --
# A tibble: 102 x 6
    time cmt         amt    ii  addl evid
```

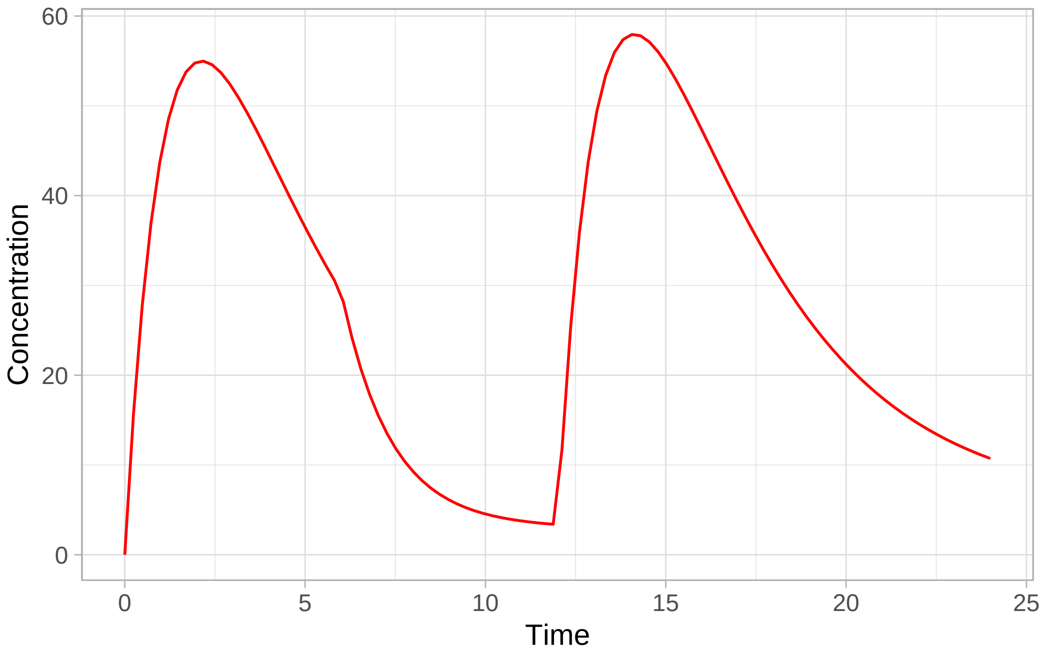
```
   <dbl> <chr>     <dbl> <dbl> <int> <evid>
 1 0     (obs)        NA    NA    NA 0:Observation
 2 0     (default) 10000    12     3 1:Dose (Add)
 3 0.242 (obs)        NA    NA    NA 0:Observation
 4 0.485 (obs)        NA    NA    NA 0:Observation
 5 0.727 (obs)        NA    NA    NA 0:Observation
 6 0.970 (obs)        NA    NA    NA 0:Observation
 7 1.21  (obs)        NA    NA    NA 0:Observation
 8 1.45  (obs)        NA    NA    NA 0:Observation
 9 1.70  (obs)        NA    NA    NA 0:Observation
10 1.94  (obs)        NA    NA    NA 0:Observation
# i 92 more rows
```

Solving shows what this does in the system:

```
rxSolve(mod1, ev) %>% ggplot(aes(time, C2)) +
  geom_line(col="red") +
  xlab("Time") + ylab("Concentration") +
  theme_light()
```

```
i parameter labels from comments are typically ignored in non-interactive mode
```

```
i Need to run with the source intact to parse comments
```



In this case, the depot is turned off, and the depot compartment concentrations are set to the initial values, but the other compartments are not. When another dose is administered to the depot, it is turned back on.

Note that a dose to a compartment only turns the compartment that was dosed back on. Any others that might have been turned off, stay off. Turning compartments back on can be achieved by administering a dose of zero or by supplying an `evid`=2 record for that compartment.

8 Simulating single subjects

8.1 Compartment numbers

rxode2 automatically assigns compartment numbers when parsing. For illustrative purposes, let's look at something a bit more compex than we have so far: a PBPK model for mavoglurant published by Wendling and colleagues (30).

```
library(rxode2)
```

```
rxode2 3.0.2 using 16 threads (see ?getRxThreads)
```

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.1     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.1
v purrr     1.0.2
```

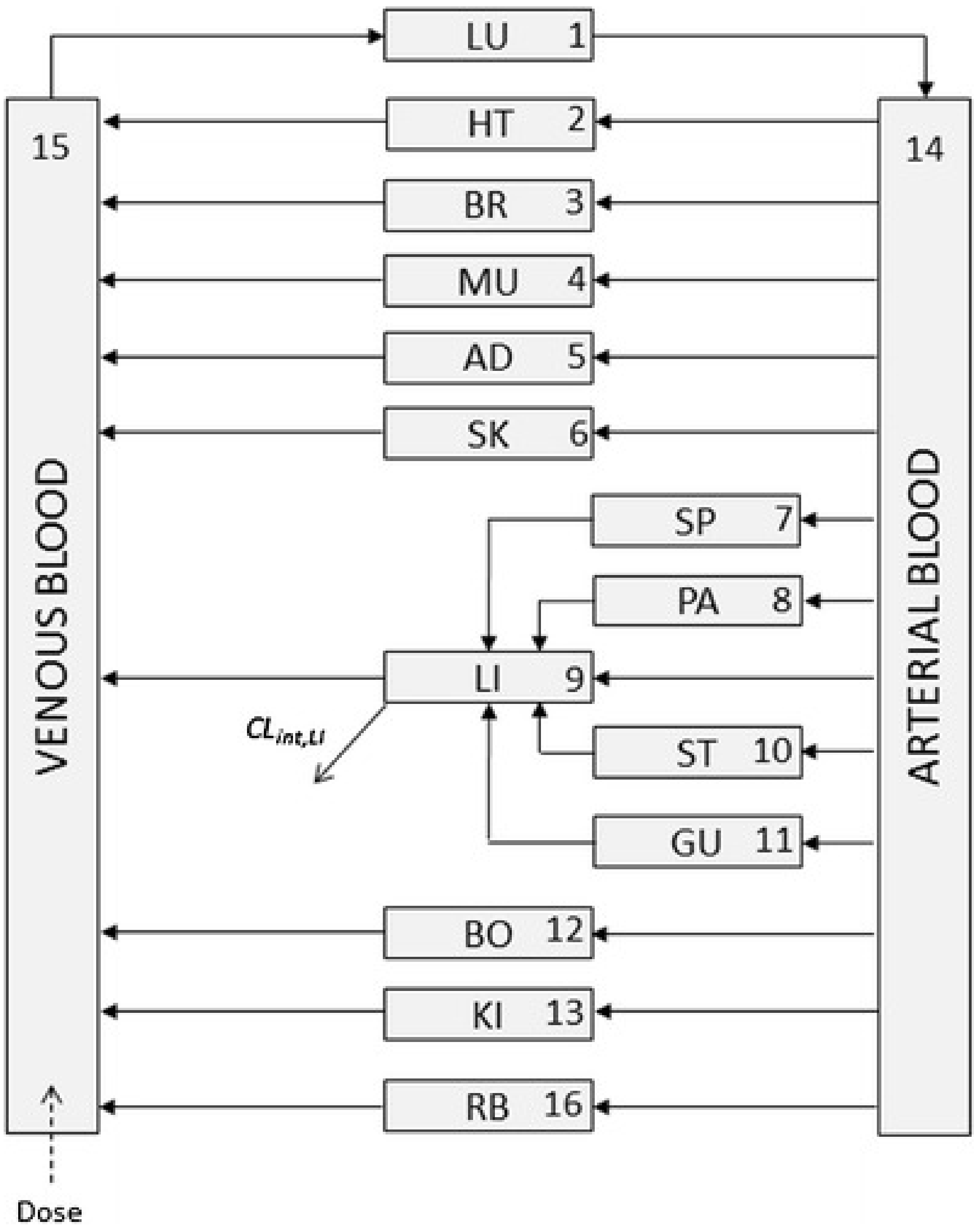
```
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becom
```

```
pbpk <- function() {
  model({
    KbBR = exp(lKbBR)
    KbMU = exp(lKbMU)
    KbAD = exp(lKbAD)
    CLint= exp(lCLint + eta.LClint)
    KbBO = exp(lKbBO)
    KbRB = exp(lKbRB)
```

```
## Regional blood flows
# Cardiac output (L/h) from White et al (1968)
CO  = (187.00*WT^0.81)*60/1000
QHT = 4.0 *CO/100
QBR = 12.0*CO/100
QMU = 17.0*CO/100
QAD = 5.0 *CO/100
QSK = 5.0 *CO/100
QSP = 3.0 *CO/100
QPA = 1.0 *CO/100
QLI = 25.5*CO/100
QST = 1.0 *CO/100
QGU = 14.0*CO/100
# Hepatic artery blood flow
QHA = QLI - (QSP + QPA + QST + QGU)
QBO = 5.0 *CO/100
QKI = 19.0*CO/100
QRB = CO - (QHT + QBR + QMU + QAD + QSK + QLI + QBO + QKI)
QLU = QHT + QBR + QMU + QAD + QSK + QLI + QBO + QKI + QRB

## Organs' volumes = organs' weights / organs' density
VLU = (0.76 *WT/100)/1.051
VHT = (0.47 *WT/100)/1.030
VBR = (2.00 *WT/100)/1.036
VMU = (40.00*WT/100)/1.041
VAD = (21.42*WT/100)/0.916
VSK = (3.71 *WT/100)/1.116
VSP = (0.26 *WT/100)/1.054
VPA = (0.14 *WT/100)/1.045
VLI = (2.57 *WT/100)/1.040
VST = (0.21 *WT/100)/1.050
VGU = (1.44 *WT/100)/1.043
VBO = (14.29*WT/100)/1.990
VKI = (0.44 *WT/100)/1.050
VAB = (2.81 *WT/100)/1.040
VVB = (5.62 *WT/100)/1.040
VRB = (3.86 *WT/100)/1.040

## Fixed parameters
BP = 0.61       # Blood:plasma partition coefficient
fup = 0.028     # Fraction unbound in plasma
fub = fup/BP    # Fraction unbound in blood
```

```
    KbLU = exp(0.8334)
    KbHT = exp(1.1205)
    KbSK = exp(-.5238)
    KbSP = exp(0.3224)
    KbPA = exp(0.3224)
    KbLI = exp(1.7604)
    KbST = exp(0.3224)
    KbGU = exp(1.2026)
    KbKI = exp(1.3171)

    ##-----------------------------------------
    S15 = VVB*BP/1000
    C15 = Venous_Blood/S15

    ##-----------------------------------------
    d/dt(Lungs) = QLU*(Venous_Blood/VVB - Lungs/KbLU/VLU)
    d/dt(Heart) = QHT*(Arterial_Blood/VAB - Heart/KbHT/VHT)
    d/dt(Brain) = QBR*(Arterial_Blood/VAB - Brain/KbBR/VBR)
    d/dt(Muscles) = QMU*(Arterial_Blood/VAB - Muscles/KbMU/VMU)
    d/dt(Adipose) = QAD*(Arterial_Blood/VAB - Adipose/KbAD/VAD)
    d/dt(Skin) = QSK*(Arterial_Blood/VAB - Skin/KbSK/VSK)
    d/dt(Spleen) = QSP*(Arterial_Blood/VAB - Spleen/KbSP/VSP)
    d/dt(Pancreas) = QPA*(Arterial_Blood/VAB - Pancreas/KbPA/VPA)
    d/dt(Liver) = QHA*Arterial_Blood/VAB + QSP*Spleen/KbSP/VSP +
      QPA*Pancreas/KbPA/VPA + QST*Stomach/KbST/VST +
      QGU*Gut/KbGU/VGU - CLint*fub*Liver/KbLI/VLI - QLI*Liver/KbLI/VLI
    d/dt(Stomach) = QST*(Arterial_Blood/VAB - Stomach/KbST/VST)
    d/dt(Gut) = QGU*(Arterial_Blood/VAB - Gut/KbGU/VGU)
    d/dt(Bones) = QBO*(Arterial_Blood/VAB - Bones/KbBO/VBO)
    d/dt(Kidneys) = QKI*(Arterial_Blood/VAB - Kidneys/KbKI/VKI)
    d/dt(Arterial_Blood) = QLU*(Lungs/KbLU/VLU - Arterial_Blood/VAB)
    d/dt(Venous_Blood) = QHT*Heart/KbHT/VHT + QBR*Brain/KbBR/VBR +
      QMU*Muscles/KbMU/VMU + QAD*Adipose/KbAD/VAD + QSK*Skin/KbSK/VSK +
      QLI*Liver/KbLI/VLI + QBO*Bones/KbBO/VBO + QKI*Kidneys/KbKI/VKI +
      QRB*Rest_of_Body/KbRB/VRB - QLU*Venous_Blood/VVB
    d/dt(Rest_of_Body) = QRB*(Arterial_Blood/VAB - Rest_of_Body/KbRB/VRB)
  })
}
```

This is quite a meaty model, with 16 compartments linked by ODEs.



LU 1
HT 2
BR 3
MU 4
AD 5
SK 6
SP 7
PA 8
LI 9
ST 10
GU 11
BO 12
KI 13
RB 16
15
VENOUS BLOOD
14
ARTERIAL BLOOD
$CL_{int,LI}$
Dose

8.1.1 How rxode2 assigns compartment numbers

```r
pbpk <- pbpk()
print(pbpk)
```

```
 -- rxode2-based free-form 16-cmt ODE model -------------------------------------

States ($state or $stateDf):
   Compartment Number Compartment Name
1                   1            Lungs
2                   2            Heart
3                   3            Brain
4                   4          Muscles
5                   5          Adipose
6                   6             Skin
7                   7           Spleen
8                   8         Pancreas
9                   9            Liver
10                 10          Stomach
11                 11              Gut
12                 12            Bones
13                 13          Kidneys
14                 14   Arterial_Blood
15                 15     Venous_Blood
16                 16     Rest_of_Body
 -- Model (Normalized Syntax): --
function() {
    model({
        KbBR = exp(lKbBR)
        KbMU = exp(lKbMU)
        KbAD = exp(lKbAD)
        CLint = exp(lCLint + eta.LClint)
        KbBO = exp(lKbBO)
        KbRB = exp(lKbRB)
        CO = (187 * WT^0.81) * 60/1000
        QHT = 4 * CO/100
        QBR = 12 * CO/100
        QMU = 17 * CO/100
        QAD = 5 * CO/100
        QSK = 5 * CO/100
        QSP = 3 * CO/100
```

```
QPA = 1 * CO/100
QLI = 25.5 * CO/100
QST = 1 * CO/100
QGU = 14 * CO/100
QHA = QLI - (QSP + QPA + QST + QGU)
QBO = 5 * CO/100
QKI = 19 * CO/100
QRB = CO - (QHT + QBR + QMU + QAD + QSK + QLI + QBO +
    QKI)
QLU = QHT + QBR + QMU + QAD + QSK + QLI + QBO + QKI +
    QRB
VLU = (0.76 * WT/100)/1.051
VHT = (0.47 * WT/100)/1.03
VBR = (2 * WT/100)/1.036
VMU = (40 * WT/100)/1.041
VAD = (21.42 * WT/100)/0.916
VSK = (3.71 * WT/100)/1.116
VSP = (0.26 * WT/100)/1.054
VPA = (0.14 * WT/100)/1.045
VLI = (2.57 * WT/100)/1.04
VST = (0.21 * WT/100)/1.05
VGU = (1.44 * WT/100)/1.043
VBO = (14.29 * WT/100)/1.99
VKI = (0.44 * WT/100)/1.05
VAB = (2.81 * WT/100)/1.04
VVB = (5.62 * WT/100)/1.04
VRB = (3.86 * WT/100)/1.04
BP = 0.61
fup = 0.028
fub = fup/BP
KbLU = exp(0.8334)
KbHT = exp(1.1205)
KbSK = exp(-0.5238)
KbSP = exp(0.3224)
KbPA = exp(0.3224)
KbLI = exp(1.7604)
KbST = exp(0.3224)
KbGU = exp(1.2026)
KbKI = exp(1.3171)
S15 = VVB * BP/1000
C15 = Venous_Blood/S15
d/dt(Lungs) = QLU * (Venous_Blood/VVB - Lungs/KbLU/VLU)
d/dt(Heart) = QHT * (Arterial_Blood/VAB - Heart/KbHT/VHT)
```

```
        d/dt(Brain) = QBR * (Arterial_Blood/VAB - Brain/KbBR/VBR)
        d/dt(Muscles) = QMU * (Arterial_Blood/VAB - Muscles/KbMU/VMU)
        d/dt(Adipose) = QAD * (Arterial_Blood/VAB - Adipose/KbAD/VAD)
        d/dt(Skin) = QSK * (Arterial_Blood/VAB - Skin/KbSK/VSK)
        d/dt(Spleen) = QSP * (Arterial_Blood/VAB - Spleen/KbSP/VSP)
        d/dt(Pancreas) = QPA * (Arterial_Blood/VAB - Pancreas/KbPA/VPA)
        d/dt(Liver) = QHA * Arterial_Blood/VAB + QSP * Spleen/KbSP/VSP +
            QPA * Pancreas/KbPA/VPA + QST * Stomach/KbST/VST +
            QGU * Gut/KbGU/VGU - CLint * fub * Liver/KbLI/VLI -
            QLI * Liver/KbLI/VLI
        d/dt(Stomach) = QST * (Arterial_Blood/VAB - Stomach/KbST/VST)
        d/dt(Gut) = QGU * (Arterial_Blood/VAB - Gut/KbGU/VGU)
        d/dt(Bones) = QBO * (Arterial_Blood/VAB - Bones/KbBO/VBO)
        d/dt(Kidneys) = QKI * (Arterial_Blood/VAB - Kidneys/KbKI/VKI)
        d/dt(Arterial_Blood) = QLU * (Lungs/KbLU/VLU - Arterial_Blood/VAB)
        d/dt(Venous_Blood) = QHT * Heart/KbHT/VHT + QBR * Brain/KbBR/VBR +
            QMU * Muscles/KbMU/VMU + QAD * Adipose/KbAD/VAD +
            QSK * Skin/KbSK/VSK + QLI * Liver/KbLI/VLI + QBO *
            Bones/KbBO/VBO + QKI * Kidneys/KbKI/VKI + QRB * Rest_of_Body/KbRB/VRB -
            QLU * Venous_Blood/VVB
        d/dt(Rest_of_Body) = QRB * (Arterial_Blood/VAB - Rest_of_Body/KbRB/VRB)
    })
}
```

Here, `Venous_Blood` is assigned to compartment 15. Keeping track of compartment numbers in large models like this can be inconvenient and challenging, and more importantly, can lead to mistakes. While it is easy, and probably clearer, to specify the compartments by name rather than number, other pharmacometric software in common use only supports compartment numbers. Having a way to number compartments easily can therefore be handy when moving between different tools.

8.1.2 Pre-declaring the compartments

Pre-assigning compartment numbers can be helpful in this situation.

To add the compartments to the model in the order desired, we can pre-declare them with `cmt`. For example, specifying `Venous_Blood` and `Skin` as the first and second compartments, respectively, is pretty straightforward:

```r
pbpk2 <- function() {
  model({
```

```
cmt(Venous_Blood)  ## Now this is the first compartment, ie cmt=1
cmt(Skin)          ## Now this is the second compartment, ie cmt=2

KbBR = exp(lKbBR)
KbMU = exp(lKbMU)
KbAD = exp(lKbAD)
CLint= exp(lCLint + eta.LClint)
KbBO = exp(lKbBO)
KbRB = exp(lKbRB)

## Regional blood flows
# Cardiac output (L/h) from White et al (1968)m
CO  = (187.00*WT^0.81)*60/1000;
QHT = 4.0 *CO/100;
QBR = 12.0*CO/100;
QMU = 17.0*CO/100;
QAD = 5.0 *CO/100;
QSK = 5.0 *CO/100;
QSP = 3.0 *CO/100;
QPA = 1.0 *CO/100;
QLI = 25.5*CO/100;
QST = 1.0 *CO/100;
QGU = 14.0*CO/100;
QHA = QLI - (QSP + QPA + QST + QGU); # Hepatic artery blood flow
QBO = 5.0 *CO/100;
QKI = 19.0*CO/100;
QRB = CO - (QHT + QBR + QMU + QAD + QSK + QLI + QBO + QKI);
QLU = QHT + QBR + QMU + QAD + QSK + QLI + QBO + QKI + QRB;

## Organs' volumes = organs' weights / organs' density
VLU = (0.76 *WT/100)/1.051;
VHT = (0.47 *WT/100)/1.030;
VBR = (2.00 *WT/100)/1.036;
VMU = (40.00*WT/100)/1.041;
VAD = (21.42*WT/100)/0.916;
VSK = (3.71 *WT/100)/1.116;
VSP = (0.26 *WT/100)/1.054;
VPA = (0.14 *WT/100)/1.045;
VLI = (2.57 *WT/100)/1.040;
VST = (0.21 *WT/100)/1.050;
VGU = (1.44 *WT/100)/1.043;
VBO = (14.29*WT/100)/1.990;
```

```
VKI = (0.44 *WT/100)/1.050;
VAB = (2.81 *WT/100)/1.040;
VVB = (5.62 *WT/100)/1.040;
VRB = (3.86 *WT/100)/1.040;

## Fixed parameters
BP = 0.61;      # Blood:plasma partition coefficient
fup = 0.028;    # Fraction unbound in plasma
fub = fup/BP;   # Fraction unbound in blood

KbLU = exp(0.8334);
KbHT = exp(1.1205);
KbSK = exp(-.5238);
KbSP = exp(0.3224);
KbPA = exp(0.3224);
KbLI = exp(1.7604);
KbST = exp(0.3224);
KbGU = exp(1.2026);
KbKI = exp(1.3171);

##-----------------------------------------
S15 = VVB*BP/1000;
C15 = Venous_Blood/S15

##-----------------------------------------
d/dt(Lungs) = QLU*(Venous_Blood/VVB - Lungs/KbLU/VLU);
d/dt(Heart) = QHT*(Arterial_Blood/VAB - Heart/KbHT/VHT);
d/dt(Brain) = QBR*(Arterial_Blood/VAB - Brain/KbBR/VBR);
d/dt(Muscles) = QMU*(Arterial_Blood/VAB - Muscles/KbMU/VMU);
d/dt(Adipose) = QAD*(Arterial_Blood/VAB - Adipose/KbAD/VAD);
d/dt(Skin) = QSK*(Arterial_Blood/VAB - Skin/KbSK/VSK);
d/dt(Spleen) = QSP*(Arterial_Blood/VAB - Spleen/KbSP/VSP);
d/dt(Pancreas) = QPA*(Arterial_Blood/VAB - Pancreas/KbPA/VPA);
d/dt(Liver) = QHA*Arterial_Blood/VAB + QSP*Spleen/KbSP/VSP +
  QPA*Pancreas/KbPA/VPA + QST*Stomach/KbST/VST + QGU*Gut/KbGU/VGU -
  CLint*fub*Liver/KbLI/VLI - QLI*Liver/KbLI/VLI;
  d/dt(Stomach) = QST*(Arterial_Blood/VAB - Stomach/KbST/VST);
  d/dt(Gut) = QGU*(Arterial_Blood/VAB - Gut/KbGU/VGU);
  d/dt(Bones) = QBO*(Arterial_Blood/VAB - Bones/KbBO/VBO);
  d/dt(Kidneys) = QKI*(Arterial_Blood/VAB - Kidneys/KbKI/VKI);
  d/dt(Arterial_Blood) = QLU*(Lungs/KbLU/VLU - Arterial_Blood/VAB);
  d/dt(Venous_Blood) = QHT*Heart/KbHT/VHT + QBR*Brain/KbBR/VBR +
        QMU*Muscles/KbMU/VMU + QAD*Adipose/KbAD/VAD + QSK*Skin/KbSK/VSK +
        QLI*Liver/KbLI/VLI + QBO*Bones/KbBO/VBO + QKI*Kidneys/KbKI/VKI +
        QRB*Rest_of_Body/KbRB/VRB - QLU*Venous_Blood/VVB;
        d/dt(Rest_of_Body) = QRB*(Arterial_Blood/VAB - Rest_of_Body/KbRB/VRB);
  })
}
```

```
pbpk2 <- pbpk2()
pbpk2
```

```
 -- rxode2-based free-form 16-cmt ODE model -------------------------------------

States ($state or $stateDf):
   Compartment Number Compartment Name
1                   1     Venous_Blood
2                   2             Skin
3                   3            Lungs
4                   4            Heart
5                   5            Brain
6                   6          Muscles
7                   7          Adipose
8                   8           Spleen
9                   9         Pancreas
10                 10            Liver
11                 11          Stomach
12                 12              Gut
13                 13            Bones
14                 14          Kidneys
15                 15   Arterial_Blood
16                 16     Rest_of_Body
 -- Model (Normalized Syntax): --
function() {
    model({
        cmt(Venous_Blood)
        cmt(Skin)
        KbBR = exp(lKbBR)
        KbMU = exp(lKbMU)
        KbAD = exp(lKbAD)
        CLint = exp(lCLint + eta.LClint)
        KbBO = exp(lKbBO)
        KbRB = exp(lKbRB)
        CO = (187 * WT^0.81) * 60/1000
```

```
QHT = 4 * CO/100
QBR = 12 * CO/100
QMU = 17 * CO/100
QAD = 5 * CO/100
QSK = 5 * CO/100
QSP = 3 * CO/100
QPA = 1 * CO/100
QLI = 25.5 * CO/100
QST = 1 * CO/100
QGU = 14 * CO/100
QHA = QLI - (QSP + QPA + QST + QGU)
QBO = 5 * CO/100
QKI = 19 * CO/100
QRB = CO - (QHT + QBR + QMU + QAD + QSK + QLI + QBO +
    QKI)
QLU = QHT + QBR + QMU + QAD + QSK + QLI + QBO + QKI +
    QRB
VLU = (0.76 * WT/100)/1.051
VHT = (0.47 * WT/100)/1.03
VBR = (2 * WT/100)/1.036
VMU = (40 * WT/100)/1.041
VAD = (21.42 * WT/100)/0.916
VSK = (3.71 * WT/100)/1.116
VSP = (0.26 * WT/100)/1.054
VPA = (0.14 * WT/100)/1.045
VLI = (2.57 * WT/100)/1.04
VST = (0.21 * WT/100)/1.05
VGU = (1.44 * WT/100)/1.043
VBO = (14.29 * WT/100)/1.99
VKI = (0.44 * WT/100)/1.05
VAB = (2.81 * WT/100)/1.04
VVB = (5.62 * WT/100)/1.04
VRB = (3.86 * WT/100)/1.04
BP = 0.61
fup = 0.028
fub = fup/BP
KbLU = exp(0.8334)
KbHT = exp(1.1205)
KbSK = exp(-0.5238)
KbSP = exp(0.3224)
KbPA = exp(0.3224)
KbLI = exp(1.7604)
KbST = exp(0.3224)
```

```
        KbGU = exp(1.2026)
        KbKI = exp(1.3171)
        S15 = VVB * BP/1000
        C15 = Venous_Blood/S15
        d/dt(Lungs) = QLU * (Venous_Blood/VVB - Lungs/KbLU/VLU)
        d/dt(Heart) = QHT * (Arterial_Blood/VAB - Heart/KbHT/VHT)
        d/dt(Brain) = QBR * (Arterial_Blood/VAB - Brain/KbBR/VBR)
        d/dt(Muscles) = QMU * (Arterial_Blood/VAB - Muscles/KbMU/VMU)
        d/dt(Adipose) = QAD * (Arterial_Blood/VAB - Adipose/KbAD/VAD)
        d/dt(Skin) = QSK * (Arterial_Blood/VAB - Skin/KbSK/VSK)
        d/dt(Spleen) = QSP * (Arterial_Blood/VAB - Spleen/KbSP/VSP)
        d/dt(Pancreas) = QPA * (Arterial_Blood/VAB - Pancreas/KbPA/VPA)
        d/dt(Liver) = QHA * Arterial_Blood/VAB + QSP * Spleen/KbSP/VSP +
            QPA * Pancreas/KbPA/VPA + QST * Stomach/KbST/VST +
            QGU * Gut/KbGU/VGU - CLint * fub * Liver/KbLI/VLI -
            QLI * Liver/KbLI/VLI
        d/dt(Stomach) = QST * (Arterial_Blood/VAB - Stomach/KbST/VST)
        d/dt(Gut) = QGU * (Arterial_Blood/VAB - Gut/KbGU/VGU)
        d/dt(Bones) = QBO * (Arterial_Blood/VAB - Bones/KbBO/VBO)
        d/dt(Kidneys) = QKI * (Arterial_Blood/VAB - Kidneys/KbKI/VKI)
        d/dt(Arterial_Blood) = QLU * (Lungs/KbLU/VLU - Arterial_Blood/VAB)
        d/dt(Venous_Blood) = QHT * Heart/KbHT/VHT + QBR * Brain/KbBR/VBR +
            QMU * Muscles/KbMU/VMU + QAD * Adipose/KbAD/VAD +
            QSK * Skin/KbSK/VSK + QLI * Liver/KbLI/VLI + QBO *
            Bones/KbBO/VBO + QKI * Kidneys/KbKI/VKI + QRB * Rest_of_Body/KbRB/VRB -
            QLU * Venous_Blood/VVB
        d/dt(Rest_of_Body) = QRB * (Arterial_Blood/VAB - Rest_of_Body/KbRB/VRB)
    })
}
```

Now `Venous_Blood` and `Skin` are where we want them.

8.1.3 Appending compartments

You can also append "compartments" to the model. Because of the ODE solving internals, you cannot add fake compartments to the model until after all the differential equations are defined.

For example this is legal:

```r
mod2 <- function(){
  model({
    C2 = center/V
    d / dt(depot) = -KA * depot
    d/dt(center) = KA * depot - CL*C2
    cmt(eff)
  })
}

mod2 <- mod2()
print(mod2)
```

```
 -- rxode2-based free-form 2-cmt ODE model --------------------------------------

States ($state or $stateDf):
  Compartment Number Compartment Name
1                  1            depot
2                  2           center
 -- Model (Normalized Syntax): --
function() {
    model({
        C2 = center/V
        d/dt(depot) = -KA * depot
        d/dt(center) = KA * depot - CL * C2
        cmt(eff)
    })
}
```

You can see this more clearly by querying the `simulationModel` property:

```r
mod2$simulationModel
```

```
rxode2 3.0.2 model named rx_ba83b0b12edfe2f775e3ed5670eb91ea_x6 model (ready).
x$state: depot, center
x$stateExtra: eff
x$params: V, KA, CL
x$lhs: C2
```

Defining "extra" compartments before the differential equations is not supported. The model below will throw an error if executed.

```
mod2 <- rxode2({
    cmt(eff)
    C2 = center/V;
    d / dt(depot) = -KA * depot
    d/dt(center) = KA * depot - CL*C2
})
```

8.2 Transit compartments

Transit compartments provide a useful way to better approximate absorption lag times, without the numerical problems and stiffness associated with the conventional way of modeling these (31,32). `rxode2` has them built in.

The transit compartment function (`transit`) can be used to specify the model without having to write it out in full (although you could do that too, if you wanted to). `transit()` takes parameters corresponding to the number of transit compartments (`n` in the code below), the mean transit time (`mtt`), and bioavailability (`bio`, which is optional).

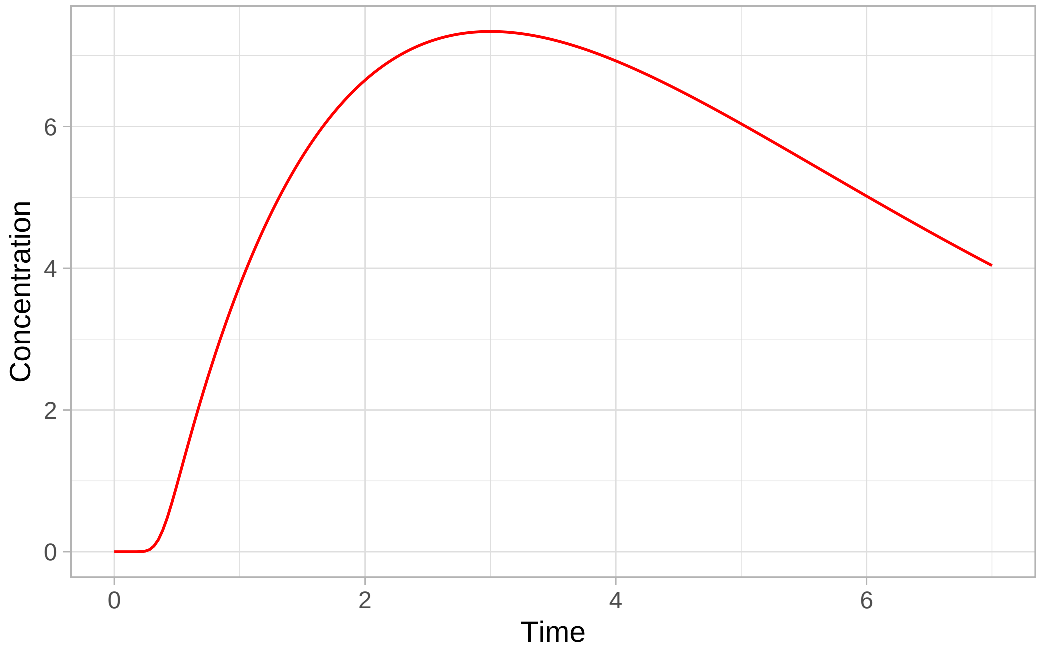
```
mod <- function() {
  ini({
    ## Table 3 from Savic 2007
    cl  <- 17.2 # (L/hr)
    vc  <- 45.1 # L
    ka  <- 0.38 # 1/hr
    mtt <- 0.37 # hr
    bio <- 1
    n   <- 20.1
  })
  model({
    k            <- cl/vc
    ktr          <- (n+1)/mtt
    d/dt(depot) <- transit(n,mtt,bio)-ka*depot
    # or alternately -
    # d/dt(depot) <- exp(log(bio*podo(depot))+log(ktr)+n*log(ktr*tad(depot))-
    #                    ktr*tad(depot)-lgammafn(n+1))-ka*depot
    d/dt(cen)   <- ka*depot-k*cen
  })
}

et <- et(0, 7, length.out=200) %>%
  et(amt=20, evid=7)
```

```r
transit <- rxSolve(mod, et)
```

```
i parameter labels from comments are typically ignored in non-interactive mode
```

```
i Need to run with the source intact to parse comments
```

```r
ggplot(transit, aes(time, cen)) +
  geom_line(col="red") +
  xlab("Time") + ylab("Concentration") +
  theme_light()
```



A couple of things to keep in mind when using this approach:

- This approach implicitly assumes that the absorption through the transit compartment is completed before the next dose is given If this isn't the case, some additional code is needed to correct for this (32).
- Different dose types (bolus or infusion) to the `depot` compartment affect the time after dose calculation (`tad`) which is used in the transit compartment code. Direct doses into compartments are therefore not currently supported. The most stable way around this is to use `tad(cmt)` and `podo(cmt)` - this way, doses to other compartments do not affect the transit compartment machinery.

- Internally, the `transit` syntax uses either the currently defined compartment, `d/dt(cmt)=transit(...)`, or `cmt`. If the transit compartment is used outside of a `d/dt()` (not recommended), the `cmt` that is used is the last `d/dt(cmt)` defined it the model. This also means compartments do not affect one another (ie a oral, transit compartment drug dosed immediately with an IV infusion)

8.3 Covariates

8.4 Multiple subjects

8.5 Working with `rxode2` output

8.6 Piping

8.7 Special cases

8.7.1 Jacobian solving

8.7.2 `rxode2` models in `shiny`

8.7.3 Precompiled `rxode2` models in other R packages

9 Simulating populations

9.1 Between-subject variability

Simulating single profiles is fun and all, and can be very helpful in explaining what the model is doing, but for this kind of thing to be really useful, we need to be able simulate populations of individuals, not just single patients.

Let's revisit our two-compartment indirect response model:

```r
library(rxode2)
```

```
rxode2 3.0.2 using 16 threads (see ?getRxThreads)
```

```r
library(patchwork)

set.seed(740727)
rxSetSeed(740727)

mod <- function() {
  ini({
    KA   <- 0.294
    TCl  <- 18.6
    eta.Cl ~ 0.4^2  # between-subject variability; variance is 0.16
    V2   <- 40.2
    Q    <- 10.5
    V3   <- 297
    Kin  <- 1
    Kout <- 1
    EC50 <- 200
  })
  model({
    C2 <- centr/V2
    C3 <- peri/V3
    CL <-  TCl*exp(eta.Cl)  ## coded as a variable in the model
    d/dt(depot) <- -KA*depot
    d/dt(centr) <- KA*depot - CL*C2 - Q*C2 + Q*C3
    d/dt(peri)  <-                    Q*C2 - Q*C3
    d/dt(eff)   <- Kin - Kout*(1-C2/(EC50+C2))*eff
    eff(0) <- 1
  })
}
```

You'll notice we've added something new: between-subject variability, `eta.Cl`, to which we have assigned a value of 0.16 (a variance, corresponding to a standard deviation of 0.4). Notice also our convention of using the tilde (`~`) to indicate that this is a random variable. We define it in the `ini` block and use it in the `mod` block - here, it provides for a log-normal distribution of clearance values. "Eta" is a commonly-used term for between-subject variability in pharmacometrics, and is derived from this original expression, which you might remember from Chapter 5. Here we're using CL rather than V.

$$CL_i = CL \cdot \exp(\eta_{CL,i}) \eta_{CL,i} \sim N(0, \omega_{CL})$$

So here, variability around CL (η_{CL}) is normally distributed with a mean of 0 and a variance of 0.16 (corresponding to an ω_{CL} value of 0.4). CL itself will be log-normally distributed.

The next step is to create the dosing regimen, which every simulated subject will share:

```
ev <- et(amountUnits="mg", timeUnits="hours") %>%
  et(amt=10000, cmt="centr")
```

We can add sampling times as well (although `rxode2` will fill these in for you if you don't do this now).

```
ev <- ev %>% et(0,48, length.out=100)
```

Notice as well that `et` takes similar arguments to `seq` when adding sampling times. As you'll remember from Chapter 7, many methods for adding sampling times and events are available in case we want to set up something complex. Now that we have a dosing and sampling scheme set up, we can simulate from the model. Here we create 100 subjects using the `nSub` argument.

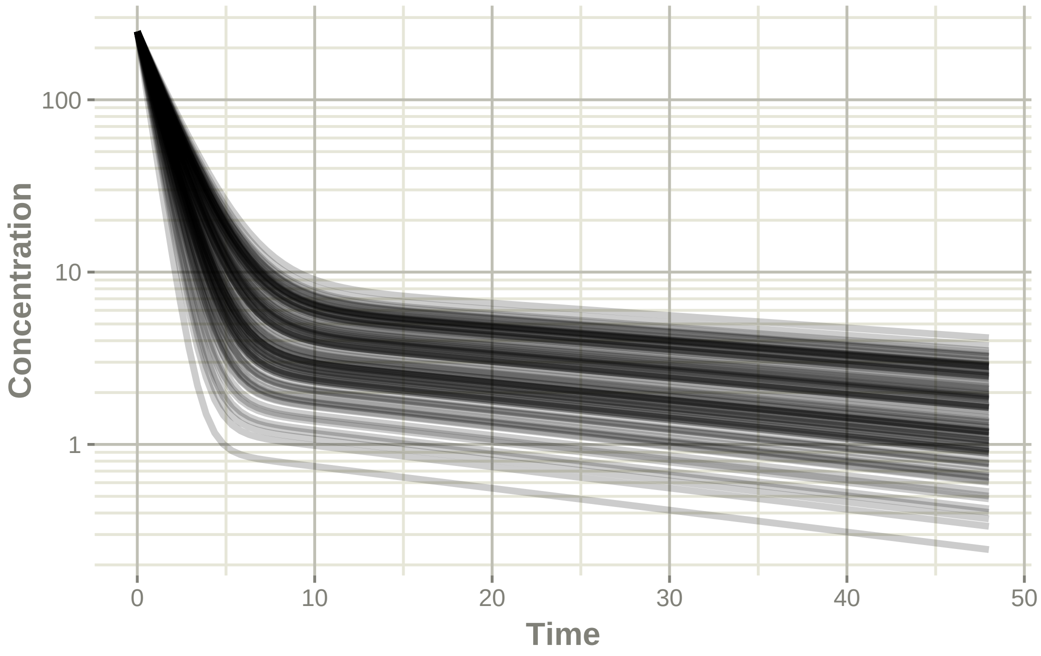
```
sim  <- rxSolve(mod, ev, nSub=100)
```

```
i parameter labels from comments are typically ignored in non-interactive mode
```

```
i Need to run with the source intact to parse comments
```

To look at the results quickly, you can use the built-in `plot` routine. This will create a `ggplot2` object that you can modify as you wish using standard `ggplot2` syntax. The extra parameter we've supplied to `plot` clarifies the piece of information we are interested in plotting. In this case, it's the the derived parameter `C2`, concentration.

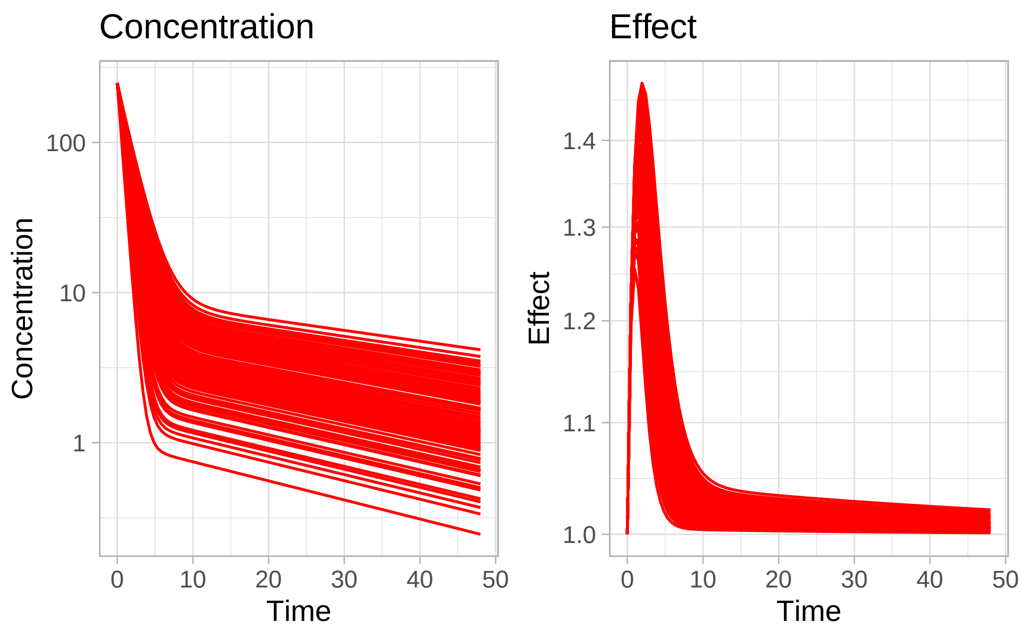
```
library(ggplot2)

plot(sim, C2, ylab="Concentration", log="y")
```



Once we have results we like, we can get a bit more creative with `ggplot2` and `patchwork`, which lets us arrange plots the way we want them.

```
p1 <- ggplot(sim, aes(time, C2, group=sim.id)) +
  geom_line(col="red") +
  scale_y_log10("Concentration") +
  scale_x_continuous("Time") +
  theme_light() +
  labs(title="Concentration")

p2 <- ggplot(sim, aes(time, eff, group=sim.id)) +
  geom_line(col="red") +
  scale_y_log10("Effect") +
  scale_x_continuous("Time") +
  theme_light() +
```

```r
  labs(title="Effect")

p1 + p2
```



Usually, simply simulating the system isn't enough. There's too much information, and it can be difficult to see trends easily. We need to summarize it.

The `rxode2` object is a type of data frame, which means we can get at the simulated data quite easily.

```r
class(sim)
```

```
[1] "rxSolve"         "rxSolveParams"   "rxSolveCovs"     "rxSolveInits"
[5] "rxSolveSimType" "data.frame"
attr(,".rxode2.env")
<environment: 0x0000014cf0191380>
```

```r
head(sim)
```

```
  sim.id      time        C2       C3       CL depot     centr     peri
1      1 0.0000000 248.75622 0.000000 25.55412     0 10000.000    0.000
2      1 0.4848485 161.23794 3.427477 25.55412     0  6481.765 1017.961
3      1 0.9696970 104.85954 5.594035 25.55412     0  4215.353 1661.428
```

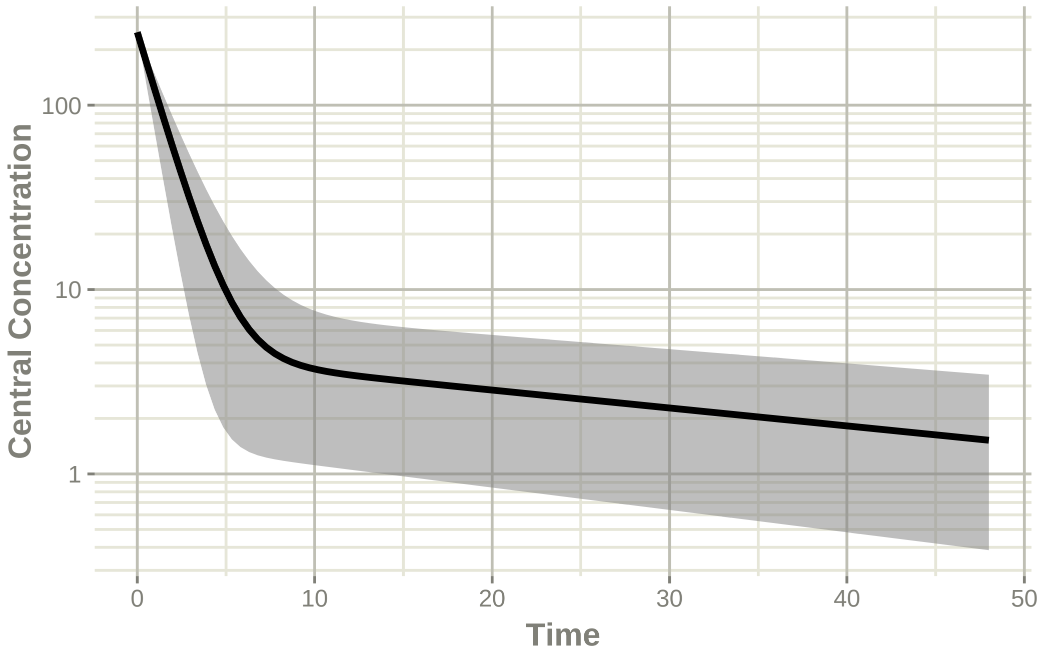
```
4       1 1.4545455  68.53693 6.948985 25.55412     0  2755.185 2063.849
5       1 1.9393939  45.13142 7.781702 25.55412     0  1814.283 2311.166
6       1 2.4242424  30.04527 8.278552 25.55412     0  1207.820 2458.730
        eff
1 1.000000
2 1.213538
3 1.323126
4 1.351026
5 1.327152
6 1.278489
```

`rxode2` includes some helpful shortcuts for summarizing the data. For example, we can extract the 5th, 50th, and 95th percentiles of the simulated data for each time point and plot them quite easily.

```r
confint(sim, "C2", level=0.95) %>%
    plot(ylab="Central Concentration", log="y")
```

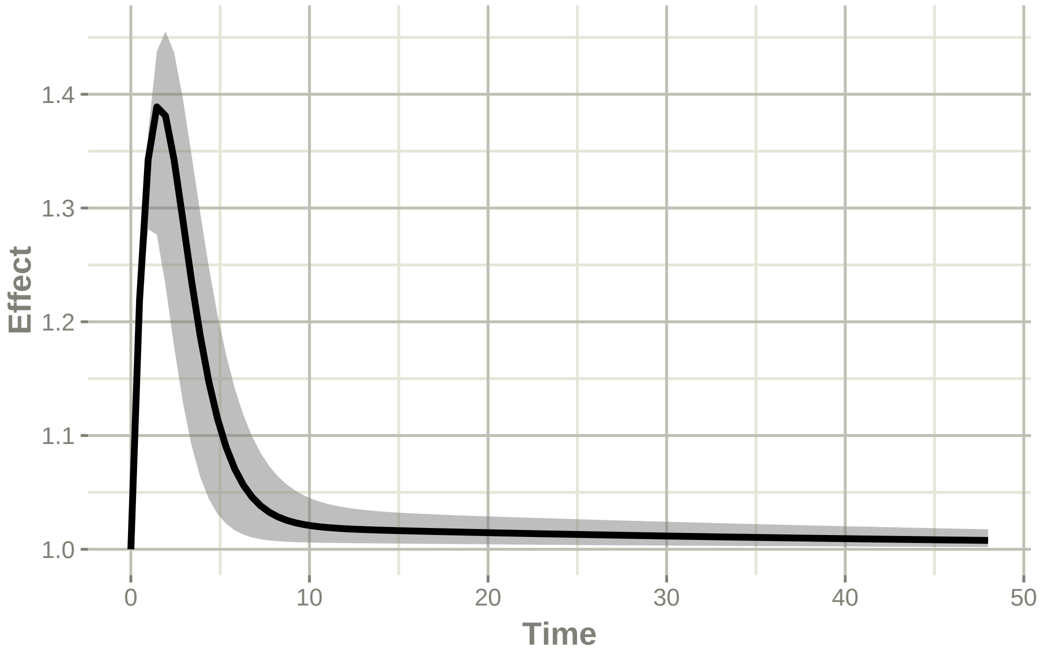
```
! in order to put confidence bands around the intervals, you need at least 2500 simulations
```

```
summarizing data...done
```



```r
confint(sim, "eff", level=0.95) %>%
    plot(ylab="Effect")
```

```
! in order to put confidence bands around the intervals, you need at least 2500 simulations

summarizing data...done
```

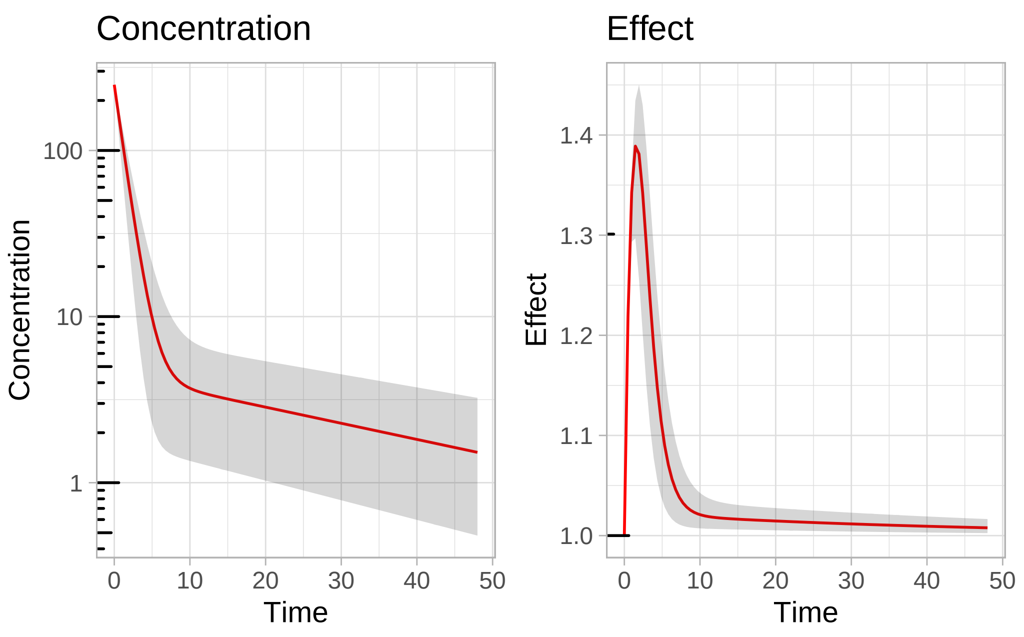


This is a shortcut for this slightly longer code:

```r
library(dplyr)
```

```

Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```r
summary <- sim %>%
  group_by(time) %>%
  summarize(C2.5=quantile(C2, 0.05),
            C2.50=quantile(C2, 0.50),
            C2.95=quantile(C2, 0.95),
            eff.5=quantile(eff, 0.05),
            eff.50=quantile(eff, 0.50),
            eff.95=quantile(eff, 0.95))

p1 <- ggplot(summary, aes(time, C2.50)) +
  geom_line(col="red") +
  geom_ribbon(aes(ymin=C2.5, ymax=C2.95), alpha=0.2) +
  scale_y_log10("Concentration") +
  scale_x_continuous("Time") +
  annotation_logticks(sides="l")+
  theme_light() +
  labs(title="Concentration")

p2 <- ggplot(summary, aes(time, eff.50)) +
  geom_line(col="red") +
  geom_ribbon(aes(ymin=eff.5, ymax=eff.95), alpha=0.2) +
  scale_y_continuous("Effect") +
  scale_x_continuous("Time") +
  annotation_logticks(sides="l")+
  theme_light() +
  labs(title="Effect")

p1 + p2
```



The parameters that were simulated for this example can also be extracted relatively easily.

```r
head(sim$param)
```

```
  sim.id    KA  TCl   V2    Q  V3 Kin Kout EC50      eta.Cl
1      1 0.294 18.6 40.2 10.5 297   1    1  200  0.31763688
2      2 0.294 18.6 40.2 10.5 297   1    1  200 -0.08874718
3      3 0.294 18.6 40.2 10.5 297   1    1  200  0.66198306
4      4 0.294 18.6 40.2 10.5 297   1    1  200 -0.48947347
5      5 0.294 18.6 40.2 10.5 297   1    1  200  0.09676769
6      6 0.294 18.6 40.2 10.5 297   1    1  200 -0.31990182
```

9.2 Random unexplained variability

In addition to simulating between-subject variability, it's often important to simulate unexplained variability. This is variability that is not explained by differences between subjects, such as laboratory assay error, for example.

Recall that random unexplained variability can be defined in a number of ways. The first, in which an additive relationship is assumed, is defined as:

$$DV_{obs,i,j} = DV_{pred,i,j} + \sigma_{add,i,j}\sigma_{add} \sim N(0, \epsilon_{add})$$

Residual error can also be modelled to be proportional:

$$DV_{obs,i,j} = DV_{pred,i,j} \cdot (1 + \sigma_{prop,i,j}) \sigma_{prop} \sim N(0, \epsilon_{prop})$$

Or both:

$$DV_{obs,i,j} = DV_{pred,i,j} \cdot (1 + \sigma_{prop,i,j}) + \sigma_{add,i,j}$$

Without rewriting our model from scratch, we can simply add residual error to our concentration and effect compartments using model piping, as follows.

```
mod2 <- mod %>%
  model(eff ~ add(eff.sd), append=TRUE) %>%    # add additive residual error to effect
  model(C2 ~ prop(prop.sd), append=TRUE) %>%   # add proportional residual error to concentra
  ini(eff.sd=sqrt(0.1), prop.sd=sqrt(0.1))
```

```
i parameter labels from comments are typically ignored in non-interactive mode

i Need to run with the source intact to parse comments

i add residual parameter `eff.sd` and set estimate to 1

i add residual parameter `prop.sd` and set estimate to 1

i change initial estimate of `eff.sd` to `0.316227766016838`

i change initial estimate of `prop.sd` to `0.316227766016838`
```

You can see how the dataset should be defined with `$multipleEndpoint`:

```
mod2$multipleEndpoint
```

```
   variable                cmt                dvid*
1 eff ~ ... cmt='eff' or cmt=4 dvid='eff' or dvid=1
2  C2 ~ ...  cmt='C2' or cmt=5  dvid='C2' or dvid=2
```

We can set up an event table like this…

```
ev <- et(amountUnits="mg", timeUnits="hours") %>%
  et(amt=10000, cmt="centr") %>%
  et(seq(0,48, length.out=100), cmt="eff") %>%
  et(seq(0,48, length.out=100), cmt="C2")
```

And now we can solve the system.

```
sim  <- rxSolve(mod2, ev, nSub=100)
```

The results here are presented by compartment number, so we'll need to do a bit of filtering to generate our summary plots with residual error. The values of `C2` and `eff` with residual error are found in `sim`.

```
sim
```

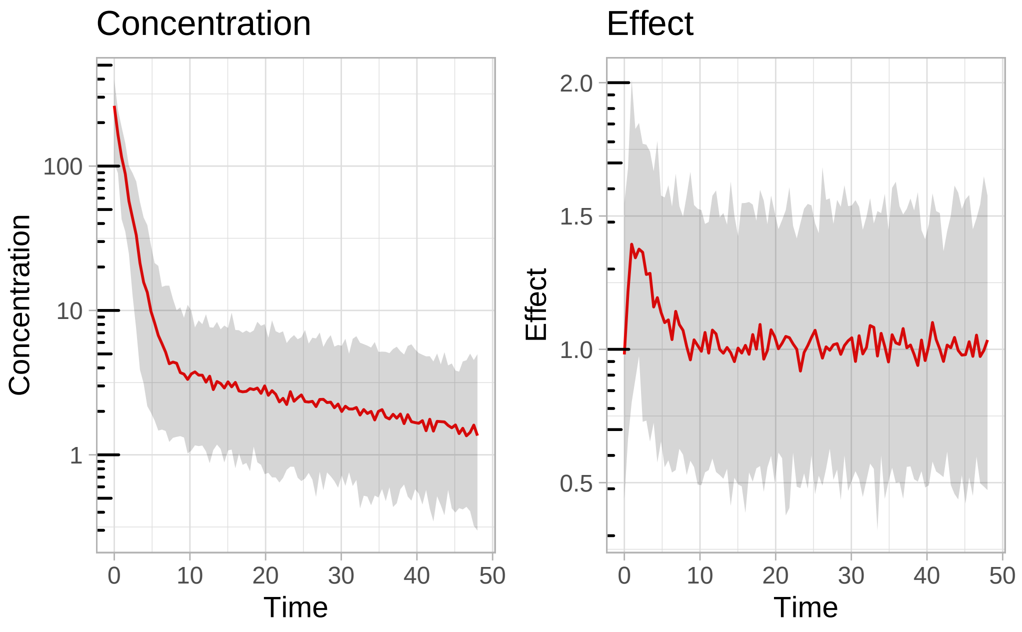
```
-- Solved rxode2 object --
-- Parameters (x$params): --
# A tibble: 100 x 12
   sim.id    KA   TCl    V2     Q    V3   Kin  Kout  EC50 eff.sd prop.sd  eta.Cl
    <int> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl>   <dbl>   <dbl>
 1      1 0.294  18.6  40.2  10.5   297     1     1   200  0.316   0.316  0.294
 2      2 0.294  18.6  40.2  10.5   297     1     1   200  0.316   0.316 -0.430
 3      3 0.294  18.6  40.2  10.5   297     1     1   200  0.316   0.316 -0.378
 4      4 0.294  18.6  40.2  10.5   297     1     1   200  0.316   0.316  0.0313
 5      5 0.294  18.6  40.2  10.5   297     1     1   200  0.316   0.316  0.519
 6      6 0.294  18.6  40.2  10.5   297     1     1   200  0.316   0.316 -0.0466
 7      7 0.294  18.6  40.2  10.5   297     1     1   200  0.316   0.316 -0.421
 8      8 0.294  18.6  40.2  10.5   297     1     1   200  0.316   0.316 -0.129
 9      9 0.294  18.6  40.2  10.5   297     1     1   200  0.316   0.316  0.242
10     10 0.294  18.6  40.2  10.5   297     1     1   200  0.316   0.316 -0.0400
# i 90 more rows
-- Initial Conditions (x$inits): --
depot centr  peri   eff
    0     0     0     1

Simulation without uncertainty in parameters, omega, or sigma matricies

-- First part of data (object): --
# A tibble: 20,000 x 12
  sim.id  time    C2    C3    CL ipredSim    sim depot  centr  peri   eff   CMT
   <int> <dbl> <dbl> <dbl> <dbl>    <dbl>  <dbl> <dbl>  <dbl> <dbl> <dbl> <dbl>
```

```
1       1 0       249.  0      24.9      1       1.36      0 10000      0   1        4
2       1 0       249.  0      24.9    249.    198.        0 10000      0   1        5
3       1 0.485   162.  3.44   24.9      1.21    0.982     0  6529. 1021.  1.21     4
4       1 0.485   162.  3.44   24.9    162.    218.        0  6529. 1021.  1.21     5
5       1 0.970   106.  5.63   24.9      1.32    1.20      0  4277. 1672.  1.32     4
6       1 0.970   106.  5.63   24.9    106.    113.        0  4277. 1672.  1.32     5
# i 19,994 more rows
```

```r
summary <- sim %>%
  group_by(time,CMT) %>%
  summarize(C2.5=quantile(sim, 0.05),
            C2.50=quantile(sim, 0.50),
            C2.95=quantile(sim, 0.95),
            eff.5=quantile(sim, 0.05),
            eff.50=quantile(sim, 0.50),
            eff.95=quantile(sim, 0.95))
```

```
`summarise()` has grouped output by 'time'. You can override using the
`.groups` argument.
```

```r
p1 <- ggplot(subset(summary, CMT==5), aes(time, C2.50)) +
  geom_line(col="red") +
  geom_ribbon(aes(ymin=C2.5, ymax=C2.95), alpha=0.2) +
  scale_y_log10("Concentration") +
  scale_x_continuous("Time") +
  annotation_logticks(sides="l")+
  theme_light() +
  labs(title="Concentration")

p2 <- ggplot(subset(summary, CMT==4), aes(time, eff.50)) +
  geom_line(col="red") +
  geom_ribbon(aes(ymin=eff.5, ymax=eff.95), alpha=0.2) +
  scale_y_continuous("Effect") +
  scale_x_continuous("Time") +
  annotation_logticks(sides="l")+
  theme_light() +
  labs(title="Effect")

p1 + p2
```



9.3 Simulating a population of individuals with different dosing regimens

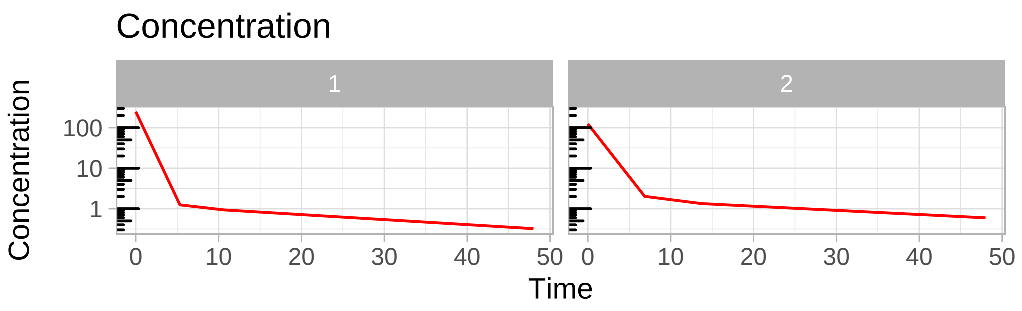
It's always nice to have a fixed dosing schedule in which everyone gets the right dose at precisely the right time, but in clinical practice this is something that doesn't often happen. Sometimes, therefore, you might want to set up the dosing and observations in your simulations to match those of particular individuals in a clinical trial. To do this, you'll have to create a data frame using the `rxode2` event specification, as well as an `ID` column to indicate which individual the doses and events refer to.

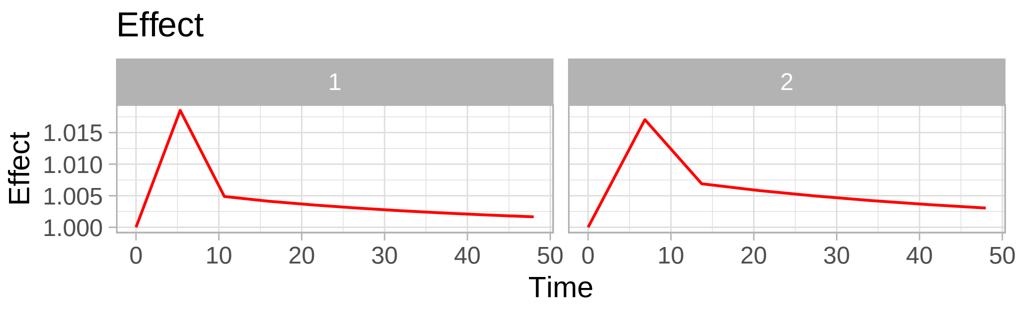
```
library(dplyr)
ev1 <- et(amountUnits="mg", timeUnits="hours") %>%
    et(amt=10000, cmt=2) %>%
    et(0,48,length.out=10)

ev2 <- et(amountUnits="mg", timeUnits="hours") %>%
    et(amt=5000, cmt=2) %>%
    et(0,48,length.out=8)

dat <- rbind(data.frame(ID=1, ev1$get.EventTable()),
             data.frame(ID=2, ev2$get.EventTable()))
```

```
## Note the number of subject is not needed since it is determined by the data
sim  <- rxSolve(mod, dat)
```

```
i parameter labels from comments are typically ignored in non-interactive mode

i Need to run with the source intact to parse comments
```

```
#sim %>% select(id, time, eff, C2)

p1 <- ggplot(sim, aes(time, C2)) +
  geom_line(col="red") +
  scale_y_log10("Concentration") +
  scale_x_continuous("Time") +
  facet_grid(~id) +
  annotation_logticks(sides="l")+
  theme_light() +
  labs(title="Concentration")

p2 <- ggplot(sim, aes(time, eff)) +
  geom_line(col="red") +
  scale_y_continuous("Effect") +
  scale_x_continuous("Time") +
  facet_grid(~id) +
  theme_light() +
  labs(title="Effect")

p1 / p2
```





This can, however, start getting a bit slow and unwieldy if you have a lot of patients. In this situation, a split-apply-combine strategy is often more efficient. We could split the data frame by `ID`, generate an event table for and apply the `rxSolve` function to each patient, and then recombine the results into a single data frame at the end.

9.4 Simulating clinical trials

Simulating clinical trials is a gigantic topic and has had entire books written about it - see the excellent one edited by Peck and Kimko for a particularly useful example (33). A complete clinical trial simulation can be performed in `rxode2` either by using a simple single event table, or data from a clinical trial as described above.

9.4.1 Parameter uncertainty

In simulations of clinical trials, one should typically consider not only variability between subjects and unexplained residual variability, but also the uncertainty of the parameter estimates in the model to be used. Parameter uncertainty can be accounted for by simulating multiple virtual "studies," specified in `rxode2` by the parameter `nStud`. The best way to do this is to sample the fixed effect parameters and covariance matrices for the between subject variability (`omega`) and unexplained variabilities (`sigma`) from the multivariate normal distribution defined by the model's variance-covariance matrix. Depending on the information you have from the models, there are some alternative ways to do this as well, but it's beyond the scope of this book to go into them in detail - the interested reader is referred to the `rxode2` documentation (particularly relating to `cvPost()`) for more information.

Lets assume we'd like to simulate from a simple one-compartment model with oral absorption and a lag time as well as first-order elimination, with an effect of creatinine clearance (CrCL) on CL and weight on V, and combined additive and proportional residual error. The model is as follows:

```
rx1 <- rxode2({
  cl    <- tcl*(1+crcl.cl*(CLCR-65)) * exp(eta.cl)
  v     <- tv * WT * exp(eta.v)
  ka    <- tka * exp(eta.ka)
  ipred <- linCmt()
  obs   <- ipred * (1 + prop.sd) + add.sd
})
```

Next we need to provide the model parameters:

```
theta <- c(tcl     = 26.3,
           tv      = 1.35,
           tka     = 4.20,
           tlag    = 0.208,
           prop.sd = 0.205,
           add.sd  = 0.0106,
           crcl.cl = 0.00717,
           eta.cl  = 0.0730,
           eta.v   = 0.0380,
           eta.ka  = 1.91)
```

And the covariances - the easiest way to create a named covariance matrix is to use `lotri()`. The numbers themselves can be obtained from a covariance matrix from a previous model fit, or from a prior distribution. Here we'll use the values from a previous fit.

```
vcovMat <- lotri(
    tcl + tv + tka + tlag + prop.sd + add.sd + crcl.cl + eta.cl + eta.v + eta.ka ~
        c(7.95E-01,
          2.05E-02, 1.92E-03,
          7.22E-02, -8.30E-03, 6.55E-01,
          -3.45E-03, -6.42E-05, 3.22E-03, 2.47E-04,
          8.71E-04, 2.53E-04, -4.71E-03, -5.79E-05, 5.04E-04,
          6.30E-04, -3.17E-06, -6.52E-04, -1.53E-05, -3.14E-05, 1.34E-05,
          -3.30E-04, 5.46E-06, -3.15E-04, 2.46E-06, 3.15E-06, -1.58E-06, 2.88E-06,
          -1.29E-03, -7.97E-05, 1.68E-03, -2.75E-05, -8.26E-05, 1.13E-05, -1.66E-06, 1.58E-0
          -1.23E-03, -1.27E-05, -1.33E-03, -1.47E-05, -1.03E-04, 1.02E-05, 1.67E-06, 6.68E-0
          7.69E-02, -7.23E-03, 3.74E-01, 1.79E-03, -2.85E-03, 1.18E-05, -2.54E-04, 1.61E-03,
```

An event table, as always, is needed:

```
evw <- et(amount.units="mg", time.units="hours") %>%
    et(amt=100) %>%
    ## For this problem we will simulate with sampling windows
    et(list(c(0, 0.5),
            c(0.5, 1),
            c(1, 3),
            c(3, 6),
            c(6, 12))) %>%
    et(id=1:1000)

evw
```

```
-- EventTable with 6000 records --
1000 dosing records (see x$get.dosing(); add with add.dosing or et)
5000 observation times (see x$get.sampling(); add with add.sampling or et)
-- First part of x: --
# A tibble: 6,000 x 6
      id   low  time  high   amt evid
   <int> <dbl> <dbl> <dbl> <dbl> <evid>
 1     1  NA   0      NA     100 1:Dose (Add)
 2     1   0   0.340   0.5    NA 0:Observation
 3     1   0.5 0.948   1      NA 0:Observation
 4     1   1   1.51    3      NA 0:Observation
 5     1   3   5.51    6      NA 0:Observation
 6     1   6   7.63   12      NA 0:Observation
 7     2  NA   0      NA     100 1:Dose (Add)
 8     2   0   0.497   0.5    NA 0:Observation
 9     2   0.5 0.948   1      NA 0:Observation
10     2   1   2.74    3      NA 0:Observation
# i 5,990 more rows
```

Now we can simulate some data. There are some technical parameters that need to be set - `sigmaXform`, `omegaXform`, `dfSub` and `dfObs`, as well as the distributions of covariates to use (weight will have a mean of 70 and a standard deviation of 15, while CrCL will have a mean of 100 and a standard deviation of 65). One could also provide a set of 1000 values for both of these covariates by sampling from previously-observed data.

```r
## Total number of observations is: 476
## Total number of individuals:     74
sim  <- rxSolve(rx1, params = theta, events = evw,
                nSub = 100, nStud = 10,  # 100 subjects, 10 trials
                thetaMat = vcovMat,      # covariance matrix
                thetaLower = 0,          # lower bound for structural parameters
                sigma = c("prop.sd", "add.sd"),
                sigmaXform = "identity", # directly model the sigmas
                omega = c("eta.cl", "eta.v", "eta.ka"), # etas are variances
                omegaXform = "variance", # etas are variances
                iCov = data.frame(WT=rnorm(1000, 70, 15),     # normally-distributed weight
                                  CLCR=rnorm(1000, 65, 25)),  # normally-distributed CrCL
                dfSub = 74,     # degrees of freedom for sampling subjects (n)
                dfObs = 476)    # degrees of freedom for sampling observations (n obs)
```

```
i thetaMat has too many items, ignored: 'tlag'
```

```r
print(sim)
```

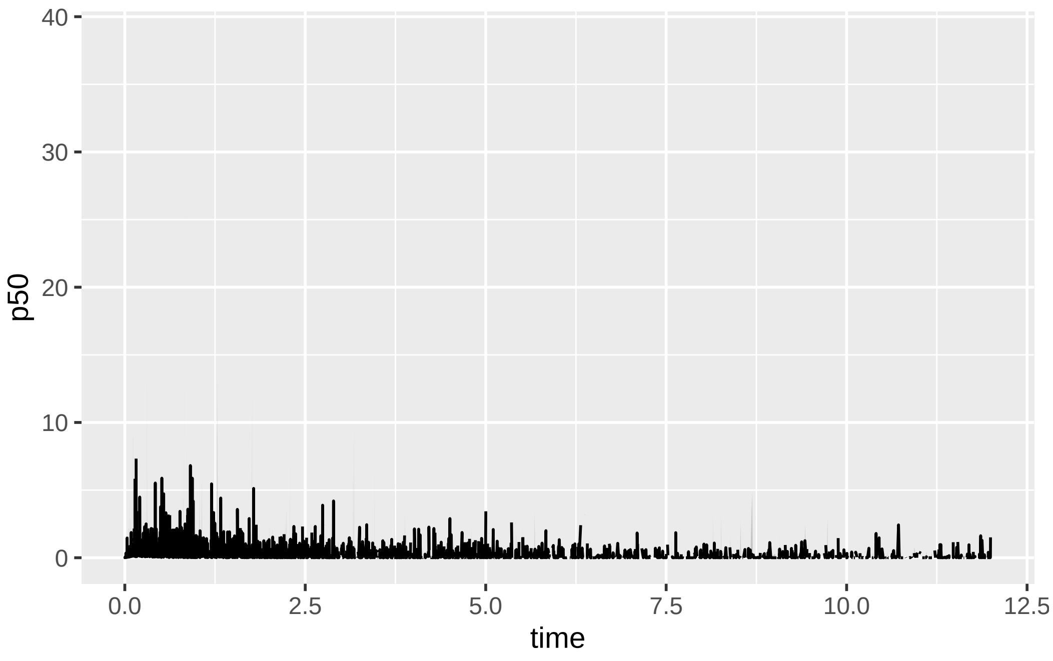
```
-- Solved rxode2 object --
-- Parameters ($params): --
# A tibble: 10,000 x 9
   sim.id id      tcl crcl.cl  eta.cl    tv  eta.v   tka   eta.ka
    <int> <fct> <dbl>   <dbl>   <dbl> <dbl>  <dbl> <dbl>    <dbl>
 1      1 1      26.7   0.775 -1.02    1.44  1.38   5.06  0.123
 2      1 2      26.7   0.775 -1.05    1.44  0.675  5.06 -0.0523
 3      1 3      26.7   0.775 -0.945   1.44 -3.83   5.06  1.36
 4      1 4      26.7   0.775  0.213   1.44 -1.10   5.06  1.74
 5      1 5      26.7   0.775 -0.609   1.44  2.35   5.06 -0.383
 6      1 6      26.7   0.775 -0.589   1.44  0.658  5.06  0.338
 7      1 7      26.7   0.775  0.179   1.44  1.53   5.06  0.832
 8      1 8      26.7   0.775 -1.20    1.44 -1.80   5.06 -1.21
 9      1 9      26.7   0.775 -0.0883  1.44 -0.299  5.06 -0.0965
10      1 10     26.7   0.775  0.417   1.44 -1.85   5.06  1.11
# i 9,990 more rows
-- Initial Conditions ($inits): --
named numeric(0)

Simulation with uncertainty in:
* parameters ($thetaMat for changes)
* omega matrix ($omegaList)
* sigma matrix ($sigmaList)

-- First part of data (object): --
# A tibble: 50,000 x 10
  sim.id    id  time    cl     v    ka ipred   obs    WT  CLCR
   <int> <int> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1      1     1 0.340 -188.  346.  5.73    NA    NA  60.9  38.5
2      1     1 0.948 -188.  346.  5.73    NA    NA  60.9  38.5
3      1     1 1.51  -188.  346.  5.73    NA    NA  60.9  38.5
4      1     1 5.51  -188.  346.  5.73    NA    NA  60.9  38.5
5      1     1 7.63  -188.  346.  5.73    NA    NA  60.9  38.5
6      1     2 0.497 -131.  150.  4.80    NA    NA  53.3  45.5
# i 49,994 more rows
```

```
s <- sim %>% confint(c("ipred"))
```

```
summarizing data...


done
```

```
ggplot(s[s$p1==0.5,], aes(time, p50)) +
  geom_ribbon(aes(ymin=p2.5, ymax=p97.5), alpha=0.2) +
  geom_line()
```

```
Warning: Removed 2 rows containing missing values or values outside the scale range
(`geom_line()`).
```



9.5 Simulate without uncertainty in `omega` or `sigma` parameters

If you do not wish to sample from the prior distributions of either the `omega` or `sigma` matrices, you can turn off this feature by specifying the `simVariability = FALSE` option when solving:

```
sim  <- rxSolve(rx1, evw, nSub=1000, thetaMat=vcovMat, nStud=10, params=theta,
                iCov = data.frame(WT=rnorm(1000, 70, 15),      # normally-distributed weight
                                  CLCR=rnorm(1000, 65, 25)),  # normally-distributed CrCL
                simVariability=FALSE)
```

```
i thetaMat has too many items, ignored: 'tlag'
```

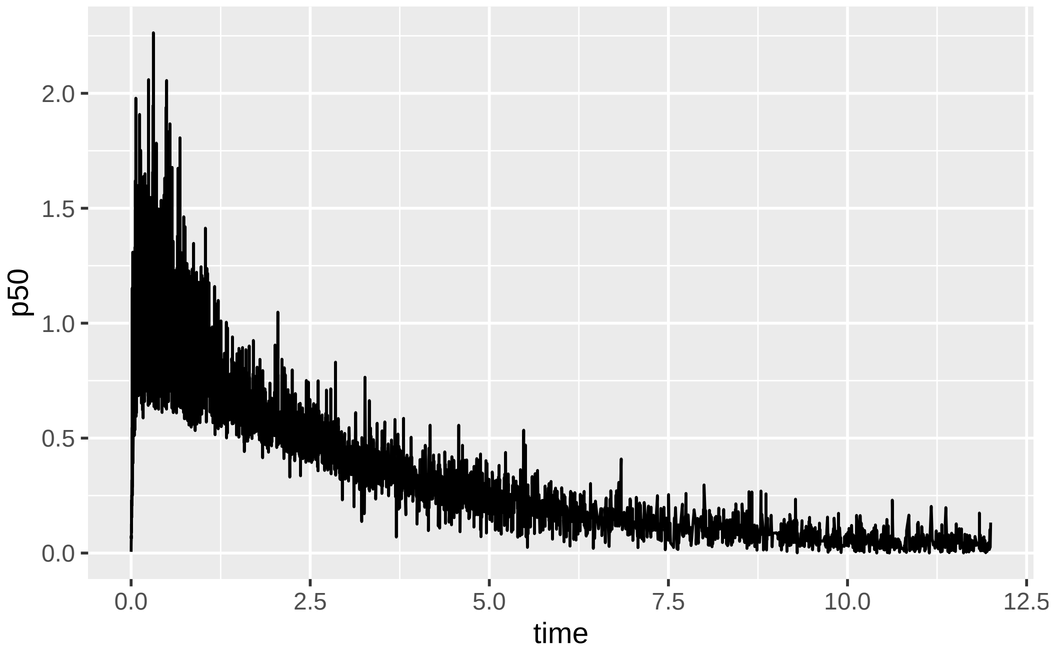
```
Warning: multi-subject simulation without without 'omega'
```

```
s <-sim %>% confint(c("ipred"))
```

```
summarizing data...
```

```
done
```

```r
ggplot(s[s$p1==0.5,], aes(time, p50)) +
  geom_ribbon(aes(ymin=p2.5, ymax=p97.5), alpha=0.2) +
  geom_line()
```



Note since realizations of `omega` and `sigma` were not simulated, `$omegaList` and `$sigmaList` both return `NULL`.

10 More complex examples

To be added.

Part IV

Data Preparation and Management

11 Data structure

`nlmixr2` and `rxode2` use the same record-based dataset structure as NONMEM, the industry gold standard, for its inputs. Doses, observations and other types of events are encoded as a "stream" of records, each of which is encoded as a row in a data frame.

11.1 Default data items

Data frames used with `nlmixr2` and `rxode2` support the following data items:

Data Item	Meaning	Notes
`id`	Unique identifier	Can be integer, factor, character, or numeric.
`time`	Time	Numeric.
`dv`	Dependent variable	Numeric. Not used for simulations.
`mdv`	Flag for missing dependent variable	0=not missing; 1=missing. Not used for simulations.
`amt`	Dose amount	Positive for doses, zero or `NA` for observations.
`rate`	Infusion rate	When specified, infusion duration will be `dur`=`amt`/`rate`. Specified for doses, zero or `NA` for observations. Alternatively, `rate` can take the following values: -1=rate modeled; -2=duration modeled.
`dur`	Infusion duration	When specified, infusion rate will be `rate`=`amt`/`dur`. Specified for doses, zero or `NA` for observations.
`evid`	Event identifier	0=observation; 1=dose; 2=other; 3=reset; 4=reset+dose; 5=replace; 6=multiply; 7=transit.
`cmt`	Compartment	Can be integer, factor or character.
`ss`	Steady state flag	0=non-steady-state; 1=steady state; 2=steady state+prior states. Specified for doses, zero or `NA` for observations.
`ii`	Interdose interval	Time between successive doses. Requires `addl`. Specified for doses, zero or `NA` for observations.
`addl`	Number of additional doses	Number of times to repeat current dose. Requires `ii`. Specified for doses, zero or `NA` for observations.

We've talked a lot about event types already - see Chapter 7.

`deSolve`-compatible data frames can also be used, but an extensive discussion of this functionality is beyond the scope of this book.

11.2 Additional columns

Any number of additional columns may be used, as and where necessary. Covariates are often specified this way - continuous covariates should be numeric, but categorical covariates can be integers, strings or factors.

12 Exploratory analysis

12.1 Overview

This document provides a template for exploring Single or Multiple Ascending Dose PK data.

12.2 Load Packages

```
library(xgxr)
library(ggplot2)
library(dplyr)

# setting ggplot theme
xgx_theme_set()
```

12.3 Load the dataset and assign columns. Take subsets of data that are needed

Multiplying dose by weight to get a mg dosing.

```
# units of dataset
time_units_dataset <- "hours"
time_units_plot    <- "days"
dose_label         <- "Dose (mg)"
conc_label         <- "Concentration (ug/ml)"
concnorm_label     <- "Normalized Concentration (ug/ml)/mg"

# covariates in the dataset
covariates <- c("WT")

# load dataset
data <- nlmixr_theo_sd
```

```r
# make sure that the necessary columns are assigned
# five columns are required: TIME, LIDV, CMT, DOSE, DOSEREG
data <- data %>%
  mutate(ID = ID) %>%   #ID column
  group_by(ID) %>%
  mutate(TIME = TIME,     #TIME column name
         NOMTIME = as.numeric(as.character(cut(TIME,
                                 breaks = c(-Inf, 0.1, 0.7, 1.5, 4, 8, 10.5, 15, Inf),
                                 labels = c( 0, 0.5, 1, 2, 7, 9, 12, 24)))),
         EVID    = EVID,                   # EVENT ID >=1 is dose, 0 otherwise
         CYCLE   = 1,                      # CYCLE of PK data
         LIDV    = DV,                     # DEPENDENT VARIABLE column name
         CENS    = 0,                      # CENSORING column name
         CMT     = CMT,                    # COMPARTMENT column here (e.g. CMT or YTYPE)
         DOSE    = signif(max(AMT) * WT, 2), # DOSE column here (numeric value)
                                           # convert mg/kg (in dataset) to mg
         DOSEREG = DOSE) %>% # DOSE REGIMEN column here
  ungroup()

# convert DOSEREG to factor for proper ordering in the plotting
# add LIDVNORM dose normalized concentration for plotting
data <- data %>%
  arrange(DOSE) %>%
  mutate(LIDVNORM    = LIDV / DOSE,
         DOSEREG     = factor(DOSEREG, levels = unique(DOSEREG)),
         DOSEREG_REV = factor(DOSEREG, levels = rev(unique(DOSEREG))))
                       # define order of treatment factor

# for plotting the PK data
data_pk <- filter(data, CMT == 2, TIME > 0)

# NCA
NCA <- data %>%
  filter(CMT == 2, NOMTIME > 0, NOMTIME <= 24) %>%
  group_by(ID) %>%
  summarize(AUC_0_24     = caTools::trapz(TIME, LIDV),
            Cmax_0_24    = max(LIDV),
            Ctrough_0_24 = LIDV[length(LIDV)],
            DOSE         = DOSE[1],
            WT           = WT[1]) %>%
  tidyr::gather(PARAM, VALUE, -c(ID, DOSE, WT)) %>%
  mutate(VALUE_NORM = VALUE / DOSE) %>%
  ungroup()
```

12.4 Summary of the data issues and the covariates

```
check <- xgx_check_data(data,covariates)
#> Warning in xgx_check_data(data, covariates): Setting YTYPE column equal to CMT
#> Warning in xgx_check_data(data, covariates): Setting MDV column equal to as.numeric(EVID!=
#>
#> DATA SUMMARY
#> CONTINUOUS COVARIATES
#> NO CATEGORICAL COVARIATES
#> POSSIBLE DATA ISSUES - FIRST 6 RECORDS
#> The following columns contained missing values
#> :

knitr::kable(check$summary)
```

Category	Description	YTYPE	Statistic	Value
Patients	Number of Patients	-	12	12
MDV	Number of patients with zero PK or PD observations	all	0	0
MDV	Number of Missing Data Points (MDV==1 and EVID==0)	all	0	0
Dose	Number of non-zero doses	-	12	12
Dose	Number of zero doses (AMT==0)	-	0	0
Dose	Number of patients that never received drug	-	0	0
DV	Number of Data Points	1	12	12
DV	Number of Data Points	2	132	132
DV	Number of Data Points per Individual	1	min = 1, median = 1, max = 1	1
DV	Number of Data Points per Individual	2	min = 11, median = 11, max = 11	11
DV	Number of Data Points with zero value (DV==0)	1	0	0
DV	Number of Data Points with zero value (DV==0)	2	9	9

Category	Description	YTYPE	Statistic	Value
DV	Number of Data Points with NA (is.na(DV))	1	0	0
DV	Number of Data Points with NA (is.na(DV))	2	0	0
DV+TIME	Multiple measurements at same time	1	0	0
DV+TIME	Multiple measurements at same time	2	0	0
CENS	Number of Censored Data Points	1	0 (0%)	0
CENS	Number of Censored Data Points	2	0 (0%)	0
All Columns	Negative Values (number)	-	:	0
All Columns	Missing Values (number)	-	:	0

```
knitr::kable(head(check$data_subset))
```

Data_Check_Issue	ID	TIME	DV	CENS	YTYPE
DV == 0	9	0	0	0	2
DV == 0	2	0	0	0	2
DV == 0	3	0	0	0	2
DV == 0	4	0	0	0	2
DV == 0	5	0	0	0	2
DV == 0	6	0	0	0	2

```
knitr::kable(check$cts_covariates)
```

Covariate	Nmissing	min	25th	median	75th	max
WT	0	54.6	63.575	70.5	74.425	86.4

```
knitr::kable(check$cat_covariates)
```

12.5 Provide an overview of the data

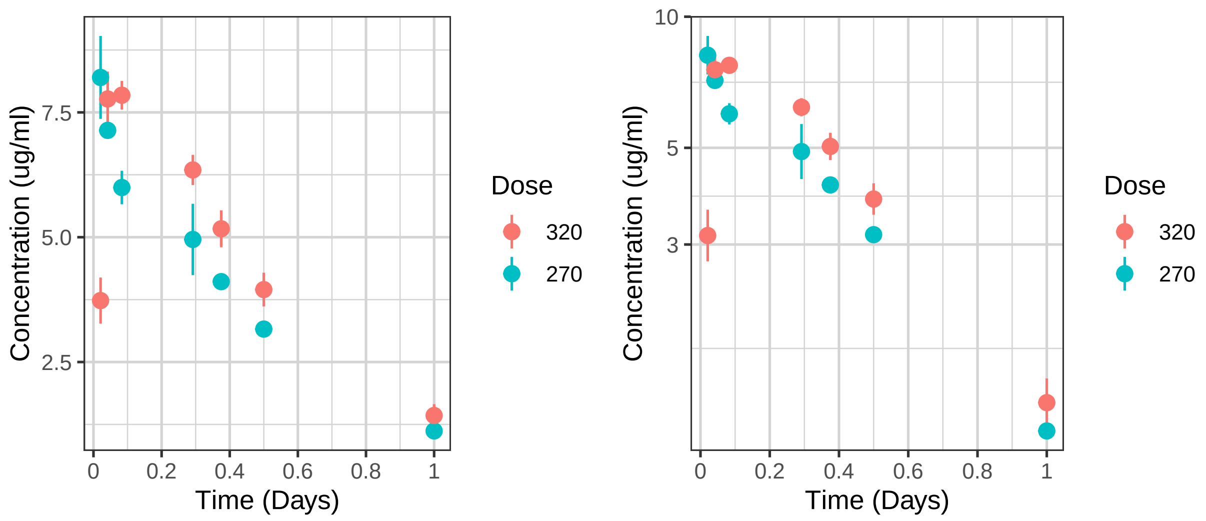
Summarize the data in a way that is easy to visualize the general trend of PK over time and between doses. Using summary statistics can be helpful, e.g. Mean +/- SE, or median, 5th

& 95th percentiles. Consider either coloring by dose or faceting by dose. Depending on the amount of data one graph may be better than the other.

When looking at summaries of PK over time, there are several things to observe. Note the number of doses and number of time points or sampling schedule. Observe the overall shape of the average profiles. What is the average Cmax per dose? Tmax? Does the elimination phase appear to be parallel across the different doses? Is there separation between the profiles for different doses? Can you make a visual estimate of the number of compartments that would be needed in a PK model?

12.5.1 Concentration over Time, colored by dose, mean +/- 95% CI

```r
glin <- ggplot(data = data_pk, aes(x = NOMTIME,
                                   y = LIDV,
                                   group = DOSE,
                                   color = DOSEREG_REV)) +
  stat_summary() +
  xgx_scale_x_time_units(time_units_dataset, time_units_plot) +
  labs(y = conc_label, color = "Dose")

glog <- glin + scale_y_log10()
gridExtra::grid.arrange(gridExtra::arrangeGrob(glin, glog, nrow = 1))
```



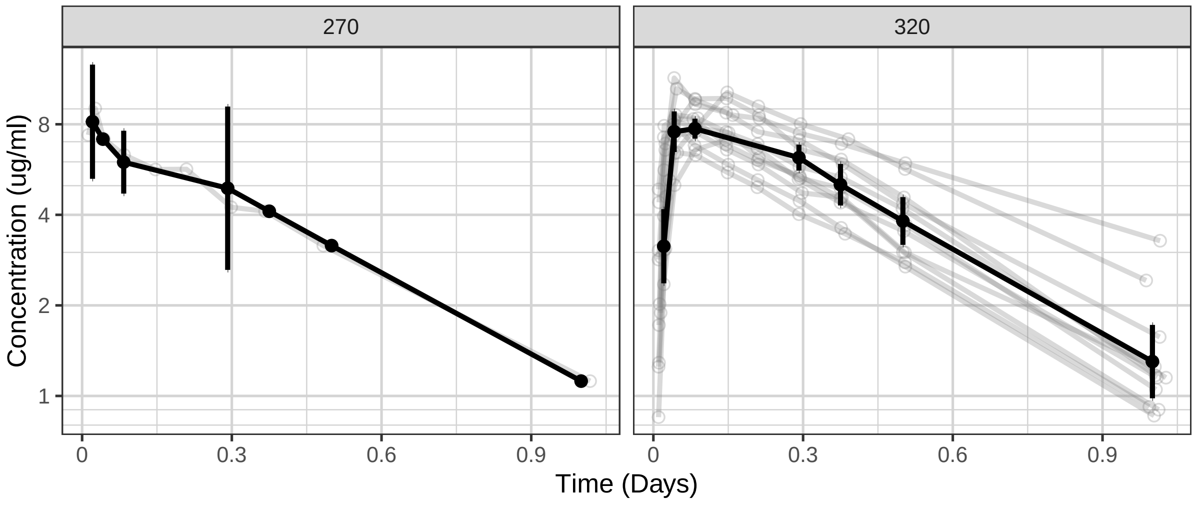
12.5.2 Side-by-side comparison of first administered dose and steady state

For multiple dose studies, zoom in on key visits for a clearer picture of the profiles. Look for accumulation (if any) between first administered dose and steady state.

```
if (exists("data_pk_rich")) {
  ggplot(data_pk_rich, aes(x = PROFTIME,
                           y = LIDV,
                           group = interaction(CYCLE,DOSE),
                           color = DOSEREG_REV)) +
    facet_grid(~DAY_label, scales = "free_x") +
    xgx_stat_ci() +
    xgx_scale_x_time_units(time_units_dataset, time_units_plot) +
    xgx_scale_y_log10() +
    labs(y = conc_label, color = "Dose")
}
```

12.5.3 Concentration over Time, faceted by dose, mean +/- 95% CI, overlaid on gray spaghetti plots

```
ggplot(data = data_pk, aes(x = TIME,
                           y = LIDV,
                           group = interaction(ID, CYCLE))) +
  geom_line(size = 1, color = rgb(0.5, 0.5, 0.5), alpha = 0.3) +
  geom_point(aes(color = factor(CENS), shape = factor(CENS)),
             size = 2, alpha = 0.3) +
  xgx_stat_ci(aes(x = NOMTIME, group = NULL, color = NULL)) +
  facet_grid(.~DOSEREG) +
  xgx_scale_x_time_units(time_units_dataset, time_units_plot) +
  xgx_scale_y_log10() +
  ylab(conc_label) +
  theme(legend.position = "none") +
  scale_shape_manual(values = c(1, 8)) +
  scale_color_manual(values = c("grey50", "red"))
```

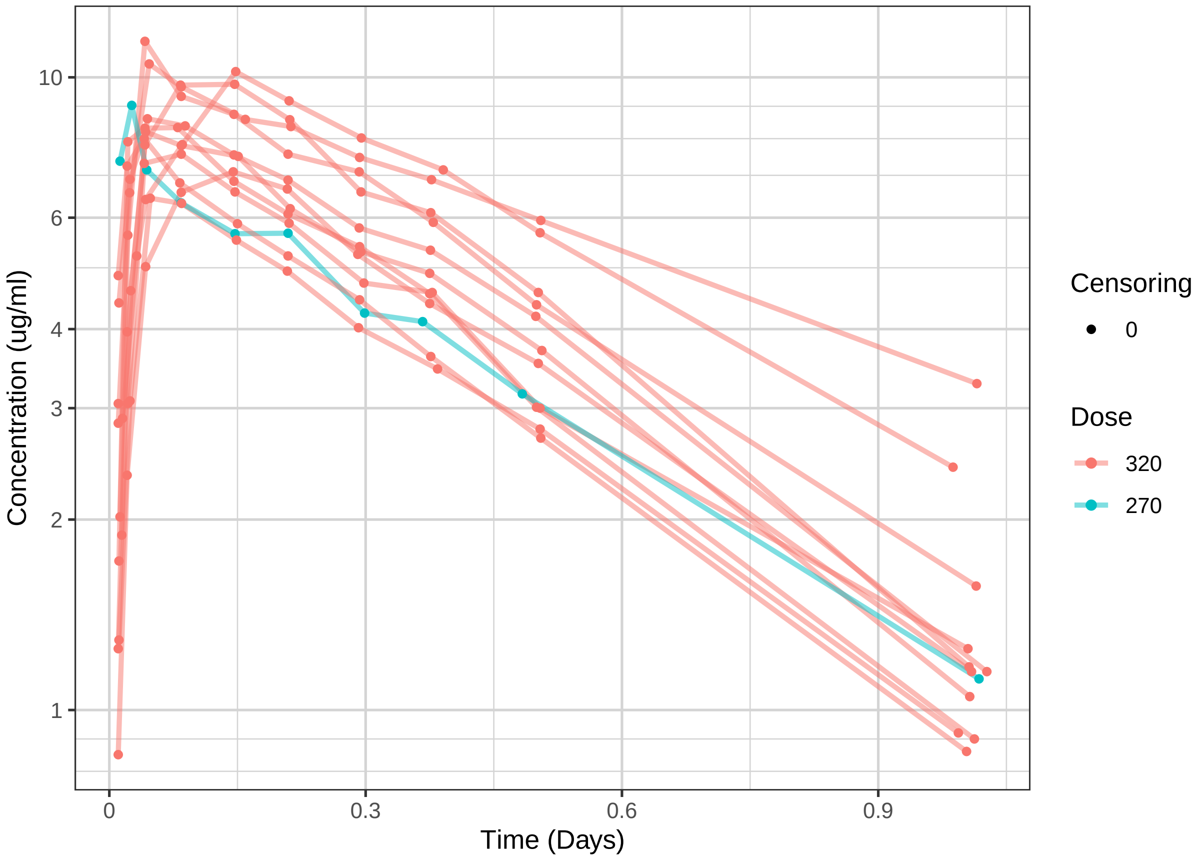


12.6 Explore variability

Use spaghetti plots to visualize the extent of variability between individuals. The wider the spread of the profiles, the higher the between subject variability. Distinguish different doses by color, or separate into different panels. If coloring by dose, do the individuals in the different dose groups overlap across doses? Dose there seem to be more variability at higher or lower concentrations?

12.6.1 Concentration over Time, colored by dose, dots and lines grouped by individual

```
ggplot(data = data_pk, aes(x = TIME,
                           y = LIDV,
                           group = interaction(ID, CYCLE),
                           color = factor(DOSEREG_REV),
                           shape = factor(CENS))) +
  geom_line(size = 1, alpha = 0.5) +
  geom_point() +
  xgx_scale_x_time_units(time_units_dataset, time_units_plot) +
  xgx_scale_y_log10() +
  labs(y = conc_label, color = "Dose", shape = "Censoring")
```

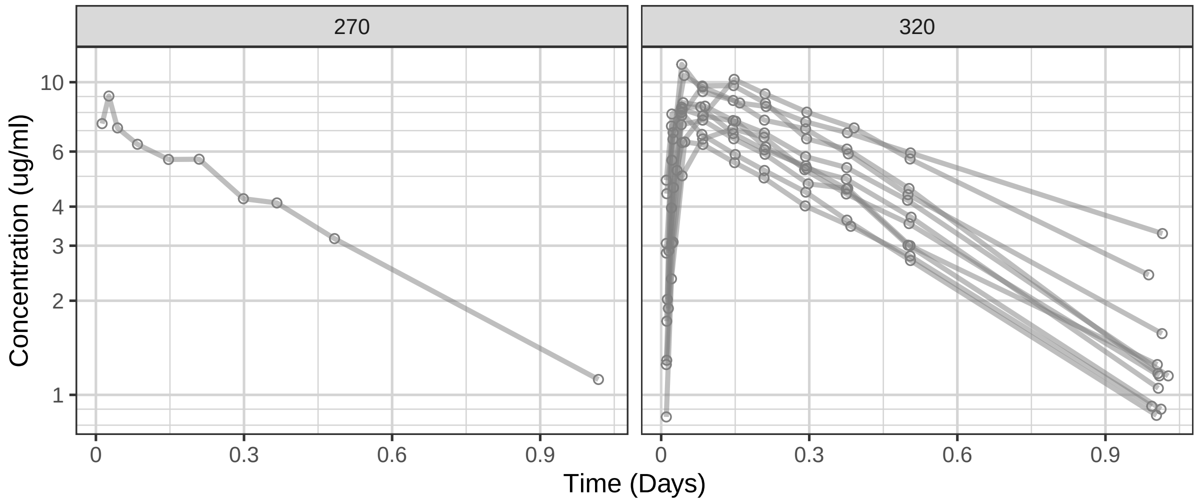


12.6.2 Side-by-side comparison of first administered dose and steady state

```
if (exists("data_pk_rich")) {
  ggplot(data = data_pk_rich, aes(x = TIME,
                                  y = LIDV,
                                  group = interaction(ID, CYCLE),
                                  color = DOSEREG_REV,
                                  shape = factor(CENS))) +
    geom_line(size = 1, alpha = 0.5) +
    geom_point() +
    facet_grid(~DAY_label, scales = "free_x") +
    xgx_scale_x_time_units(time_units_dataset, time_units_plot) +
    xgx_scale_y_log10() +
    labs(y = conc_label, color = "Dose", shape = "Censoring")
}
```

12.6.3 Concentration over Time, faceted by dose, lines grouped by individual

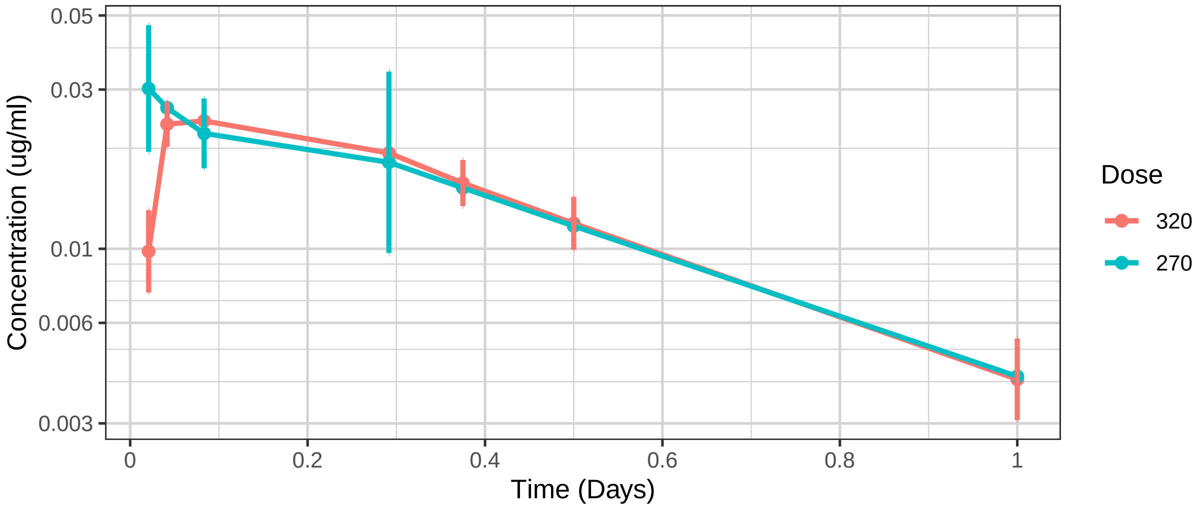
```r
ggplot(data = data_pk, aes(x = TIME,
                           y = LIDV,
                           group = interaction(ID, CYCLE),
                           color = factor(CENS),
                           shape = factor(CENS))) +
  geom_line(size = 1, alpha = 0.5) +
  geom_point() +
  facet_grid(.~DOSEREG) +
  xgx_scale_x_time_units(time_units_dataset, time_units_plot) +
  xgx_scale_y_log10() +
  ylab(conc_label) +
  scale_shape_manual(values = c(1, 8)) +
  scale_color_manual(values = c("grey50", "red")) +
  theme(legend.position = "none")
```



12.7 Assess the dose linearity of exposure

12.7.1 Dose Normalized Concentration over Time, colored by dose, mean +/- 95% CI

```r
ggplot(data = data_pk, aes(x = NOMTIME,
                           y = LIDVNORM,
                            group = DOSEREG_REV,
                            color = DOSEREG_REV)) +
  xgx_stat_ci() +
  xgx_scale_x_time_units(time_units_dataset, time_units_plot) +
  xgx_scale_y_log10() +
  labs(y = conc_label, color = "Dose")
```



12.7.2 Side-by-side comparison of first administered dose and steady state

```
if (exists("data_pk_rich")) {
  ggplot(data_pk_rich, aes(x = NOMTIME,
                            y = LIDVNORM,
                            group = interaction(DOSE, CYCLE),
                            color = DOSEREG_REV)) +
    xgx_stat_ci() +
    facet_grid(~DAY_label,scales = "free_x") +
    xgx_scale_x_time_units(time_units_dataset, time_units_plot) +
    xgx_scale_y_log10() +
    labs(y = conc_label, color = "Dose")
}
```

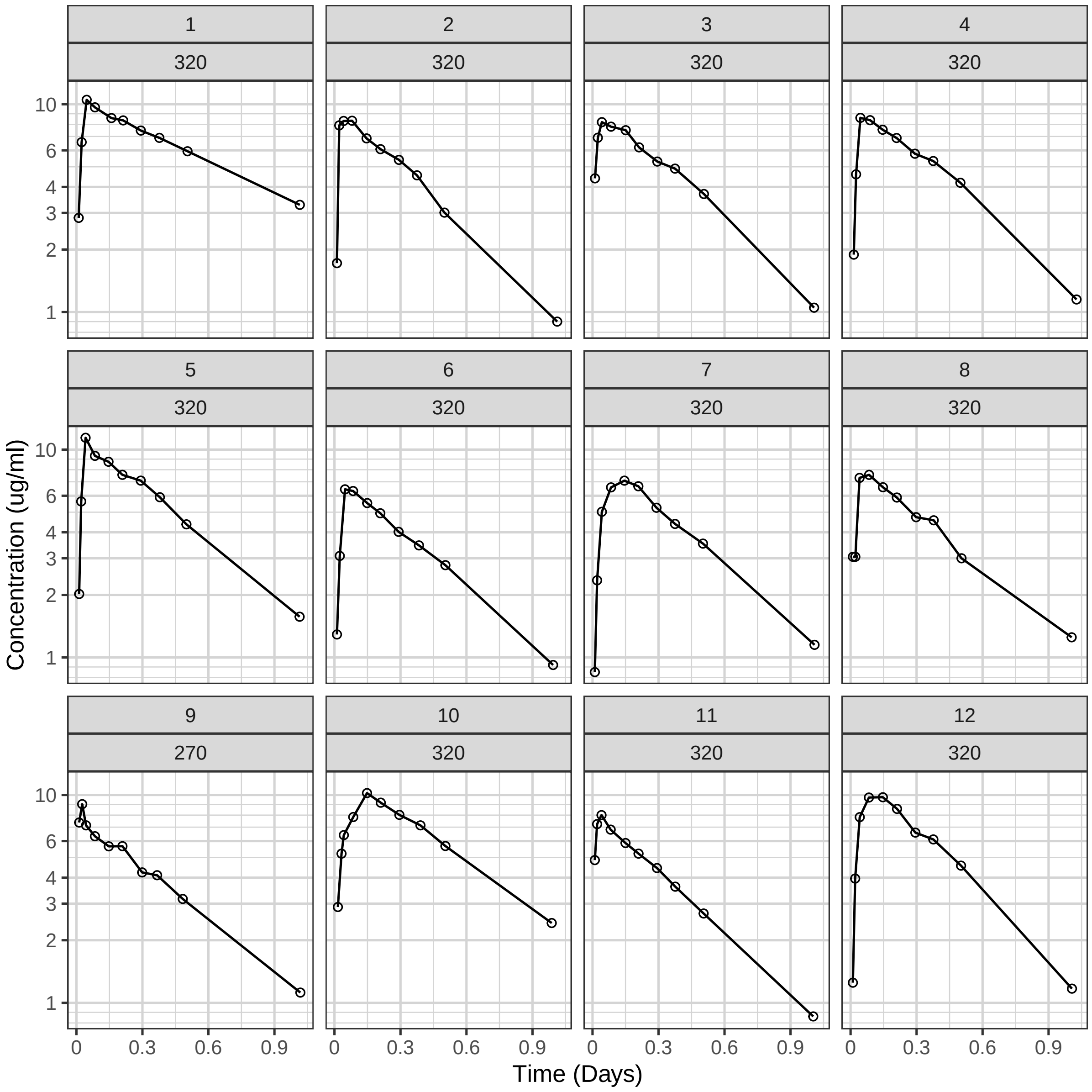
12.8 Explore irregularities in profiles

Plot individual profiles in order to inspect them for any irregularities. Inspect the profiles for outlying data points that may skew results or bias conclusions. Looking at the shapes of the individual profiles now, do they support your observations made about the mean profile (e.g. number of compartments, typical Cmax, Tmax)?

Plotting individual profiles on top of gray spaghetti plots puts individual profiles into context, and may help identify outlying individuals for further inspection. Are there any individuals that appear to have very high or low Cmax compared to others within the same dose group? What about the timing of Cmax? What about the slope of the elimination phase? Does it appear that any subjects could have received an incorrect dose?

12.8.1 Concentration over Time, faceted by individual, individual line plots overlaid on gray spaghetti plots for that dose group

```r
ggplot(data = data_pk, aes(x = TIME, y = LIDV)) +
  geom_line() +
  geom_point(aes(color = factor(CENS), shape = factor(CENS))) +
  facet_wrap(~ID + DOSEREG) +
  xgx_scale_x_time_units(time_units_dataset, time_units_plot) +
  xgx_scale_y_log10() +
  ylab(conc_label) +
  theme(legend.position = "none") +
  scale_shape_manual(values = c(1, 8)) +
  scale_color_manual(values = c("black", "red"))
```

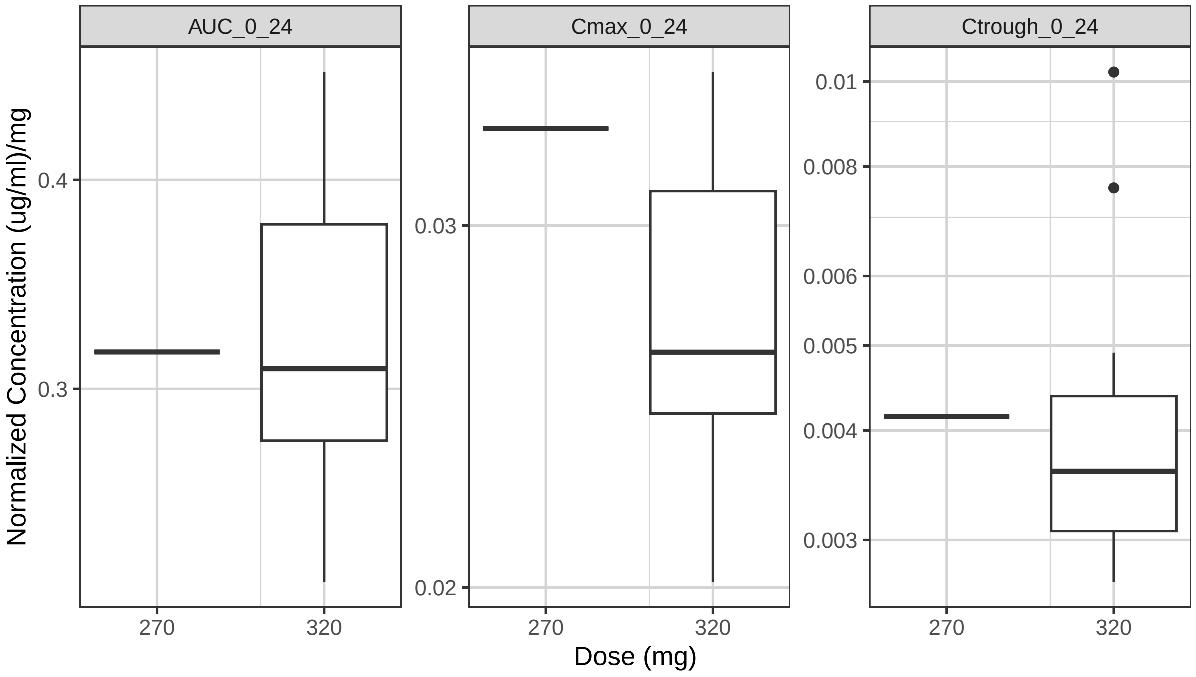


12.9 NCA

12.9.1 NCA of dose normalized AUC vs Dose

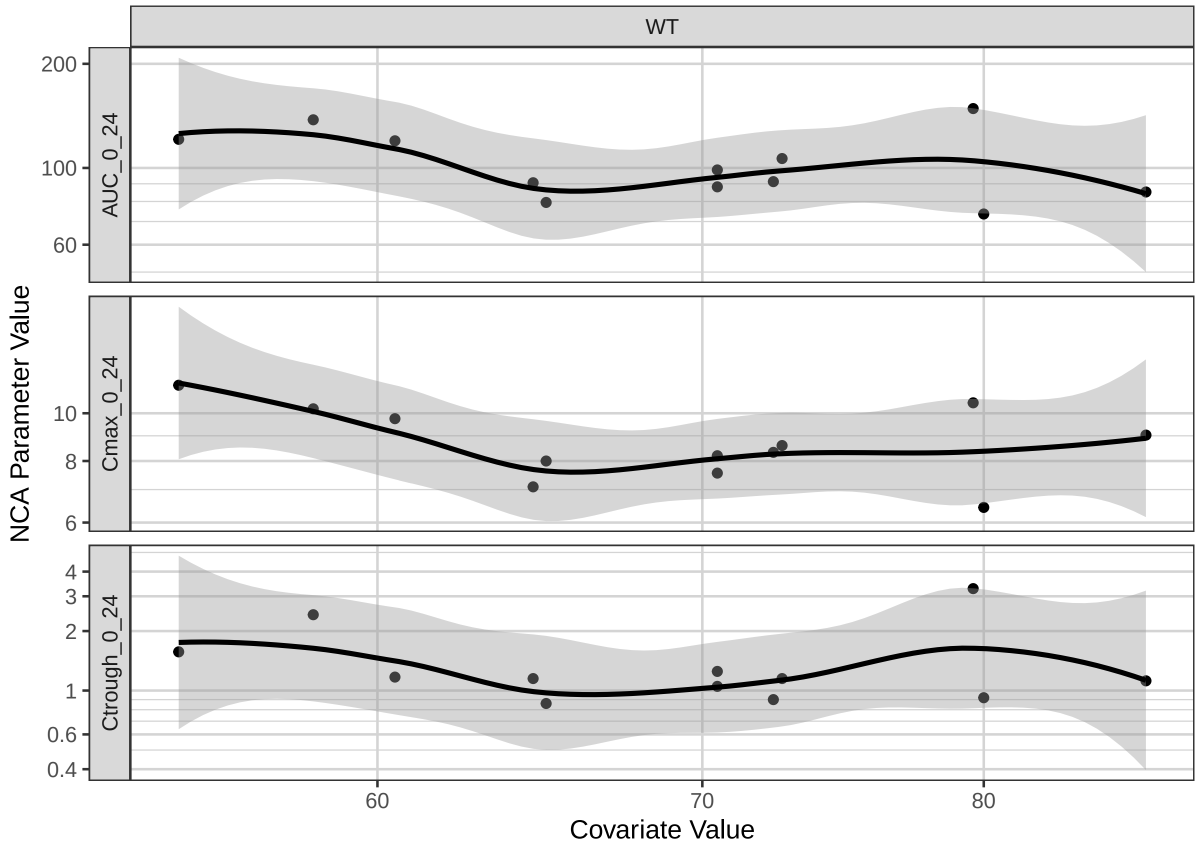
Observe the dose normalized AUC over different doses. Does the relationship appear to be constant across doses or do some doses stand out from the rest? Can you think of reasons why some would stand out? For example, the lowest dose may have dose normalized AUC much higher than the rest, could this be due to CENS observations? If the highest doses have dose normalized AUC much higher than the others, could this be due to nonlinear clearance, with clearance saturating at higher doses? If the highest doses have dose normalized AUC much lower than the others, could there be saturation of bioavailability, reaching the maximum absorbable dose?

```r
if (!exists("NCA")) {
  warning("For PK data exploration, it is highly recommended to perform an NCA")
} else {
  ggplot(data = NCA, aes(x = DOSE, y = VALUE_NORM)) +
    geom_boxplot(aes(group = DOSE)) +
    geom_smooth(method = "loess", color = "black") +
    facet_wrap(~PARAM, scales = "free_y") +
    xgx_scale_x_log10(breaks = unique(NCA$DOSE)) +
    xgx_scale_y_log10() +
    labs(x = dose_label, y = concnorm_label)
}
```



12.10 Covariate Effects

```r
if (!exists("NCA")) {
  warning("For covariate exploration, it is highly recommended to perform an NCA")
} else {
  NCA_cts <- NCA[, c("PARAM", "VALUE", check$cts_covariates$Covariate)] %>%
    tidyr::gather(COV, COV_VALUE, -c(PARAM, VALUE))

  NCA_cat <- NCA[, c("PARAM", "VALUE", check$cat_covariates$Covariate)] %>%
    tidyr::gather(COV, COV_VALUE, -c(PARAM, VALUE))

  if (nrow(check$cts_covariates) >= 1) {
    gg <- ggplot(data = NCA_cts, aes(x = COV_VALUE, y = VALUE)) +
      geom_point() +
      geom_smooth(method = "loess", color = "black") +
      facet_grid(PARAM~COV,switch = "y", scales = "free_y") +
      xgx_scale_x_log10() +
      xgx_scale_y_log10() +
      labs(x = "Covariate Value", y = "NCA Parameter Value")
    print(gg)
  }

  if (nrow(check$cat_covariates) >= 1) {
    gg <- ggplot(data = NCA_cat, aes(x = COV_VALUE, y = VALUE)) +
      geom_boxplot() +
      facet_grid(PARAM~COV, switch = "y", scales = "free_y") +
      xgx_scale_y_log10() +
      labs(x = "Covariate Value", y = "NCA Parameter Value")
    print(gg)
  }
}
```


WT
AUC_0_24
Cmax_0_24
Ctrough_0_24
NCA Parameter Value
Covariate Value
200
100
60
10
8
6
4
3
2
1
0.6
0.4
60
70
80

Part V

Modeling with `nlmixr2`

13 Parameter estimation and model fitting

Fitting models has taken us 12 sections to get to!

Let's start with a very simple PK example, using the single-dose theophylline dataset generously provided by Dr. Robert A. Upton of the University of California, San Francisco:

```r
## Load libraries
library(nlmixr2)
```

```
Loading required package: nlmixr2data
```

```r
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.1     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.1
v purrr     1.0.2     

-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becor
```

```r
library(xgxr)

str(theo_sd)
```

```
'data.frame':	144 obs. of  7 variables:
 $ ID  : int  1 1 1 1 1 1 1 1 1 1 ...
 $ TIME: num  0 0 0.25 0.57 1.12 2.02 3.82 5.1 7.03 9.05 ...
 $ DV  : num  0 0.74 2.84 6.57 10.5 9.66 8.58 8.36 7.47 6.89 ...
 $ AMT : num  320 0 0 0 0 ...
 $ EVID: int  101 0 0 0 0 0 0 0 0 0 ...
 $ CMT : int  1 2 2 2 2 2 2 2 2 2 ...
 $ WT  : num  79.6 79.6 79.6 79.6 79.6 79.6 79.6 79.6 79.6 79.6 ...
```

We can try fitting a simple one-compartment PK model to this small dataset. We write the model as follows:

```
one.cmt <- function() {
  ini({
    ## You may label each parameter with a comment
    tka <- 0.45                # Log Ka
    tcl <- log(c(0, 2.7, 100)) # Log Cl
    ## This works with interactive models
    ## You may also label the preceding line with label("label text")
    tv <- 3.45; label("log V")
    ## the label("Label name") works with all models
    eta.ka ~ 0.6
    eta.cl ~ 0.3
    eta.v  ~ 0.1
    add.sd <- 0.7
  })
  model({
    ka <- exp(tka + eta.ka)
    cl <- exp(tcl + eta.cl)
    v <- exp(tv + eta.v)
    linCmt() ~ add(add.sd)
  })
}

f <- nlmixr(one.cmt)
```

```
i parameter labels from comments are typically ignored in non-interactive mode
```

```
i Need to run with the source intact to parse comments
```

Let's work through this step by step.

We can now run the model...

```
fit <- nlmixr(one.cmt, theo_sd, est="focei",
              control=list(print=0))
```

```
i parameter labels from comments are typically ignored in non-interactive mode

i Need to run with the source intact to parse comments

rxode2 3.0.2 using 16 threads (see ?getRxThreads)

calculating covariance matrix
done

> Calculating residuals/tables

v done

> compress origData in nlmixr2 object, save 5952

> compress parHistData in nlmixr2 object, save 5128
```

```
print(fit)
```

```
-- nlmixr2 FOCEi (outer: nlminb) --

         OBJF      AIC      BIC Log-likelihood Condition#(Cov) Condition#(Cor)
FOCEi 116.807 373.4068 393.5864      -179.7034         68.7202        9.387667

-- Time (sec $time): --

        setup optimize covariance table compress other
elapsed 0.026    0.125      0.125  0.11     0.06 0.984

-- Population Parameters ($parFixed or $parFixedDf): --

       Parameter  Est.     SE %RSE Back-transformed(95%CI) BSV(CV%) Shrink(SD)%
tka              0.466  0.195 41.9       1.59 (1.09, 2.34)     70.7      1.98%
tcl               1.01 0.0751 7.42       2.75 (2.38, 3.19)     26.7      3.70%
tv         log V  3.46 0.0436 1.26       31.7 (29.1, 34.6)     13.9      10.5%
add.sd           0.695                               0.695

  Covariance Type ($covMethod): r,s
  No correlations in between subject variability (BSV) matrix
  Full BSV covariance ($omega) or correlation ($omegaR; diagonals=SDs)
  Distribution stats (mean/skewness/kurtosis/p-value) available in $shrink
  Information about run found ($runInfo):
   * gradient problems with initial estimate and covariance; see $scaleInfo
   * ETAs were reset to zero during optimization; (Can control by foceiControl(resetEtaP=.))
   * initial ETAs were nudged; (can control by foceiControl(etaNudge=., etaNudge2=))
  Censoring ($censInformation): No censoring
  Minimization message ($message):
    relative convergence (4)

-- Fit Data (object is a modified tibble): --
# A tibble: 132 x 20
  ID     TIME    DV      PRED    RES   WRES IPRED   IRES  IWRES     CPRED   CRES
  <fct> <dbl> <dbl>     <dbl>  <dbl>  <dbl> <dbl>  <dbl>  <dbl>     <dbl>  <dbl>
1 1      0     0.74 -1.79e-15  0.740  1.07   0      0.74   1.07  -7.12e-16  0.740
2 1      0.25  2.84  3.28e+ 0 -0.437 -0.232  3.85 -1.01   -1.45   3.24e+ 0 -0.395
3 1      0.57  6.57  5.85e+ 0  0.718  0.287  6.79 -0.215 -0.310   5.80e+ 0  0.771
# i 129 more rows
# i 9 more variables: CWRES <dbl>, eta.ka <dbl>, eta.cl <dbl>, eta.v <dbl>,
#   ka <dbl>, cl <dbl>, v <dbl>, tad <dbl>, dosenum <dbl>
```

We can alternatively express the same model by ordinary differential equations (ODEs):

```r
one.compartment <- function() {
  ini({
    tka <- 0.45    # Log Ka
    tcl <- 1       # Log Cl
    tv  <- 3.45    # Log V
    eta.ka ~ 0.6
    eta.cl ~ 0.3
    eta.v ~ 0.1
    add.sd <- 0.7
  })
  model({
    ka <- exp(tka + eta.ka)
    cl <- exp(tcl + eta.cl)
    v <- exp(tv + eta.v)
    d/dt(depot) = -ka * depot
    d/dt(center) = ka * depot - cl / v * center
    cp = center / v
    cp ~ add(add.sd)
  })
}
```

We can try the Stochastic Approximation EM (SAEM) method to this model:

```r
fit2 <- nlmixr(one.compartment, theo_sd,  est="saem",
               control=list(print=0))
```

```
i parameter labels from comments are typically ignored in non-interactive mode

i Need to run with the source intact to parse comments

> loading into symengine environment...

> pruning branches (`if`/`else`) of saem model...

v done

> finding duplicate expressions in saem model...

> optimizing duplicate expressions in saem model...

v done

using C compiler: 'gcc.exe (GCC) 13.3.0'

i calculate uninformed etas

i done

Calculating covariance matrix

> loading into symengine environment...

> pruning branches (`if`/`else`) of saem model...

v done

> finding duplicate expressions in saem predOnly model 0...
> finding duplicate expressions in saem predOnly model 1...

> optimizing duplicate expressions in saem predOnly model 1...

> finding duplicate expressions in saem predOnly model 2...

v done

> Calculating residuals/tables

v done

> compress origData in nlmixr2 object, save 5952

> compress phiM in nlmixr2 object, save 63504

> compress parHistData in nlmixr2 object, save 13928

> compress saem0 in nlmixr2 object, save 30440
```

```r
print(fit2)
```

```
-- nlmixr2 SAEM OBJF by FOCEi approximation --

 Gaussian/Laplacian Likelihoods: AIC() or $objf etc.
 FOCEi CWRES & Likelihoods: addCwres()

-- Time (sec $time): --

        setup covariance saem table compress other
elapsed 0.001       0.03  5.5  0.09     0.14 1.879

-- Population Parameters ($parFixed or $parFixedDf): --

        Est.     SE %RSE Back-transformed(95%CI) BSV(CV%) Shrink(SD)%
tka    0.464  0.195   42       1.59 (1.09, 2.33)     71.1   -0.0900%
tcl     1.01  0.085 8.43       2.74 (2.32, 3.24)     27.4      4.80%
tv      3.46 0.0447 1.29         31.7 (29, 34.6)     13.1      8.77%
add.sd 0.696                               0.696
```

```
  Covariance Type ($covMethod): linFim
  No correlations in between subject variability (BSV) matrix
  Full BSV covariance ($omega) or correlation ($omegaR; diagonals=SDs)
  Distribution stats (mean/skewness/kurtosis/p-value) available in $shrink
  Censoring ($censInformation): No censoring

-- Fit Data (object is a modified tibble): --
# A tibble: 132 x 19
  ID     TIME    DV  PRED    RES IPRED   IRES  IWRES eta.ka eta.cl   eta.v    cp
  <fct> <dbl> <dbl> <dbl>  <dbl> <dbl>  <dbl>  <dbl>  <dbl>  <dbl>   <dbl> <dbl>
1 1      0     0.74  0      0.74  0      0.74   1.06  0.0839 -0.477 -0.0849  0
2 1      0.25  2.84  3.28 -0.437  3.83 -0.991 -1.42  0.0839 -0.477 -0.0849  3.83
3 1      0.57  6.57  5.86  0.715  6.76 -0.194 -0.278 0.0839 -0.477 -0.0849  6.76
# i 129 more rows
# i 7 more variables: depot <dbl>, center <dbl>, ka <dbl>, cl <dbl>, v <dbl>,
#   tad <dbl>, dosenum <dbl>
```

And if we wanted to, we could even apply the traditional R method nlme method to this model:

```
fitN <- nlmixr(one.compartment, theo_sd, list(pnlsTol=0.5), est="nlme")
```

```
i parameter labels from comments are typically ignored in non-interactive mode

i Need to run with the source intact to parse comments

> loading into symengine environment...

> pruning branches (`if`/`else`) of nlme model...

v done

> finding duplicate expressions in nlme model...

> optimizing duplicate expressions in nlme model...

v done
```

```
**Iteration 1
LME step: Loglik: -183.2083, nlminb iterations: 1
reStruct  parameters:
      ID1       ID2       ID3
0.2195819 0.9924330 1.6502972
 Beginning PNLS step: ..  completed fit_nlme() step.
PNLS step: RSS =  64.12613
 fixed effects: 0.4669548  1.021641  3.45082
 iterations: 3
Convergence crit. (must all become <= tolerance = 1e-05):
     fixed   reStruct
0.03630921 1.13594019

**Iteration 2
LME step: Loglik: -183.985, nlminb iterations: 1
reStruct  parameters:
      ID1       ID2       ID3
0.1028034 0.9611973 1.6475974
 Beginning PNLS step: ..  completed fit_nlme() step.
PNLS step: RSS =  64.12613
 fixed effects: 0.4669548  1.021641  3.45082
 iterations: 1
Convergence crit. (must all become <= tolerance = 1e-05):
       fixed     reStruct
0.000000e+00 9.838286e-07

> loading into symengine environment...


> pruning branches (`if`/`else`) of full model...


v done


> finding duplicate expressions in EBE model...


> optimizing duplicate expressions in EBE model...


> compiling EBE model...


v done
```

```
> Calculating residuals/tables

v done

> compress origData in nlmixr2 object, save 5952
```

```r
print(fitN)
```

```
-- nlmixr2 nlme by maximum likelihood --

         OBJF      AIC      BIC Log-likelihood Condition#(Cov) Condition#(Cor)
nlme 125.3701 381.9699 402.1495       -183.985        25.57385               1

-- Time (sec $time): --

        setup table compress other
elapsed 0.001  0.11     0.04 3.419

-- Population Parameters ($parFixed or $parFixedDf): --

         Est.      SE  %RSE Back-transformed(95%CI) BSV(CV%) Shrink(SD)%
tka     0.467  0.1873 40.12    1.595 (1.105, 2.303)     69.5      2.16%
tcl     1.022 0.08496 8.316    2.778 (2.352, 3.281)     27.1      7.54%
tv      3.451 0.03704 1.073     31.53 (29.32, 33.9)     13.5      8.13%
add.sd 0.6958                                0.6958

  Covariance Type ($covMethod): nlme
  No correlations in between subject variability (BSV) matrix
  Full BSV covariance ($omega) or correlation ($omegaR; diagonals=SDs)
  Distribution stats (mean/skewness/kurtosis/p-value) available in $shrink
  Censoring ($censInformation): No censoring

-- Fit Data (object is a modified tibble): --
# A tibble: 132 x 19
  ID     TIME    DV  PRED    RES IPRED   IRES  IWRES eta.ka eta.cl   eta.v    cp
  <fct> <dbl> <dbl> <dbl>  <dbl> <dbl>  <dbl>  <dbl>  <dbl>  <dbl>   <dbl> <dbl>
1 1      0     0.74  0     0.74   0     0.74   1.06  0.0573 -0.472 -0.0886  0
2 1      0.25  2.84  3.30 -0.459  3.79 -0.950 -1.36  0.0573 -0.472 -0.0886  3.79
3 1      0.57  6.57  5.89  0.681  6.72 -0.152 -0.218 0.0573 -0.472 -0.0886  6.72
# i 129 more rows
# i 7 more variables: depot <dbl>, center <dbl>, ka <dbl>, cl <dbl>, v <dbl>,
#   tad <dbl>, dosenum <dbl>
```

This example delivers a complete model fit as the `fit` object, including parameter history, a set of fixed effect estimates, and random effects for all included subjects.

14 The UI

The nlmixr modeling dialect, inspired by R and NONMEM, can be used to fit models using all current and future estimation algorithms within nlmixr. Using these widely-used tools as inspiration has the advantage of delivering a model specification syntax that is instantly familiar to the majority of analysts working in pharmacometrics and related fields.

14.1 Overall model structure

Model specifications for nlmixr are written using functions containing `ini` and `model` blocks. These functions can be called anything, but often contain these two components. Let's look at a very simple one-compartment model with no covariates.

```r
f <- function() {
  ini({   # Initial conditions/variables
    # are specified here
  })
  model({ # The model is specified
    # here
  })
}
```

14.1.1 The ini block

The `ini` block specifies initial conditions, including initial estimates and boundaries for those algorithms which support them (currently, the built-in `nlme` and `saem` methods do not). Nomenclature is similar to that used in NONMEM, Monolix and other similar packages. In the NONMEM world, the `ini` block is analogous to `$THETA`, `$OMEGA` and `$SIGMA` blocks.

```r
f <- function() { # Note that arguments to the function are currently
  # ignored by nlmixr
  ini({
    # Initial conditions for population parameters (sometimes
    # called THETA parameters) are defined by either '<-' or '='
    lCl <- 1.6      # log Cl (L/hr)

    # Note that simple expressions that evaluate to a number are
    # OK for defining initial conditions (like in R)
    lVc = log(90)  # log V (L)

    ## Also, note that a comment on a parameter is captured as a parameter label
    lKa <- 1       # log Ka (1/hr)

    # Bounds may be specified by c(lower, est, upper), like NONMEM:
    # Residuals errors are assumed to be population parameters
    prop.err <- c(0, 0.2, 1)

    # IIV terms will be discussed in the next example
  })

  # The model block will be discussed later
  model({})
}
```

As shown in the above example:

- Simple parameter values are specified using an R-compatible assignment
- Boundaries may be specified by `c(lower, est, upper)`.
- Like NONMEM, `c(lower,est)` is equivalent to `c(lower,est,Inf)`
- Also like NONMEM, `c(est)` does not specify a lower bound, and is equivalent to specifying the parameter without using R's `c()` function.

These parameters can be named using almost any R-compatible name. Please note that:

- Residual error estimates should be coded as population estimates (i.e. using `=` or `<-`, not `~`).
- Variable names that start with `_` are not supported. Note that R does not allow variable starting with `_` to be assigned without quoting them.
- Naming variables that start with `rx` or `nlmixr` is not suggested, since `rxode2()` and nlmixr use these prefixes internally for certain estimation routines and for calculating residuals.
- Variable names are case-sensitive, just like they are in R. `CL` is not the same as `Cl`.

In mixture models, multivariate normal individual deviations from the normal population and parameters are estimated (in NONMEM these are called "ETA" parameters). Additionally, the variance/covariance matrix of these deviations are is also estimated (in NONMEM this is the "OMEGA" matrix). These also take initial estimates. In nlmixr, these are specified by the ~ operator. This that is typically used in statistics R for “modeled by”, and was chosen to distinguish these estimates from the population and residual error parameters.

Continuing from the prior example, we can annotate the estimates for the between-subject error distribution…

```r
f <- function() {
  ini({
    lCl <- 1.6      ; label("log Cl (L/hr)")
    lVc = log(90)   ; label("log V (L)")
    lKa <- 1        ; label("log Ka (1/hr)")
    prop.err <- c(0, 0.2, 1)

    # Initial estimate for ka IIV variance
    # Labels work for single parameters
    eta.ka ~ 0.1    ## BSV Ka

    # For correlated parameters, you specify the names of each
    # correlated parameter separated by a addition operator `+`
    # and the left handed side specifies the lower triangular
    # matrix initial of the covariance matrix.
    eta.cl + eta.vc ~ c(0.1,
                        0.005, 0.1)

    # Note that labels do not currently work for correlated
    # parameters.  Also, do not put comments inside the lower
    # triangular matrix as this will currently break the model.
  })

  # The model block will be discussed later
  model({})
}
```

As shown in the above example:

- Simple variances are specified by the variable name and the estimate separated by ~
- Correlated parameters are specified by the sum of the variable labels and then the lower triangular matrix of the covariance is specified on the left handed side of the equation. This is also separated by ~.
- The initial estimates are specified on the variance scale, and in analogy with NONMEM, the square roots of the diagonal elements correspond to coefficients of variation when used in the exponential IIV implementation.

14.1.2 The model block

The `model` block specifies the model, and is analogous to the `$PK`, `$PRED` and `$ERROR` blocks in NONMEM.

Once the initialization block has been defined, you can define a model in terms of the variables defined in the `ini` block. You can also mix `rxode2()` blocks into the model if needed.

The current method of defining a nlmixr model is to specify the parameters, and then any required `rxode2()` lines. Continuing the annotated example:

```
f <- function() {
  ini({
    lCl <- 1.6       # log Cl (L/hr)
    lVc <- log(90)   # log Vc (L)
    lKA <- 0.1       # log Ka (1/hr)
    prop.err <- c(0, 0.2, 1)

    eta.Cl ~ 0.1     # BSV Cl
    eta.Vc ~ 0.1     # BSV Vc
    eta.KA ~ 0.1     # BSV Ka
  })
  model({
    # Parameters are defined in terms of the previously-defined
    # parameter names:
    Cl <- exp(lCl + eta.Cl)
    Vc =  exp(lVc + eta.Vc)
    KA <- exp(lKA + eta.KA)

    # Next, the differential equations are defined:
    kel <- Cl / Vc;

    d/dt(depot)  = -KA*depot;
    d/dt(centr)  =  KA*depot-kel*centr;

    # And the concentration is then calculated
    cp = centr / Vc;
    # Finally, we specify that the plasma concentration follows
    # a proportional error distribution (estimated by the parameter
    # prop.err)
    cp ~ prop(prop.err)
  })
}
```

A few points to note:

- Parameters are defined before the differential equations. Currently directly defining the differential equations in terms of the population parameters is not supported.
- The differential equations, parameters and error terms are in a single block, instead of multiple sections.
- Additionally state names, calculated variables, also cannot start with either `rx_` or `nlmixr_` since these are used internally in some estimation routines.
- Errors are specified using the tilde, `~`. Currently you can use either `add(parameter)` for additive error, `prop(parameter)` for proportional error or `add(parameter1) + prop(parameter2)` for combined additive and proportional error. You can also specify `norm(parameter)` for additive error, since it follows a normal distribution.
- Some routines, like `saem`, require parameters expressed in terms of `Pop.Parameter + Individual.Deviation.Parameter + Covariate*Covariate.Parameter`. The order of these parameters does not matter. This is similar to NONMEM's mu-referencing, though not as restrictive. This means that for `saem`, a parameterization of the form `Cl <- Cl*exp(eta.Cl)` is not allowed.
- The type of parameter in the model is determined by the `ini` block; covariates used in the model are not included in the `ini` block. These variables need to be present in the modeling dataset for the model to run.

14.2 Running models

Models can be fitted several ways, including via the [magrittr] forward-pipe operator.

```
fit <- nlmixr(one.compartment) %>% saem.fit(data=theo_sd)
```

```
fit2 <- nlmixr(one.compartment, data=theo_sd, est="saem")
```

```
fit3 <- one.compartment %>% saem.fit(data=theo_sd)
```

Options to the estimation routines can be specified using nlmeControl for nlme estimation:

```
fit4 <- nlmixr(one.compartment, theo_sd,est="nlme",control = nlmeControl(pnlsTol = .5))
```

where options are specified in the `nlme` documentation. Options for saem can be specified using `saemControl`:

```
fit5 <- nlmixr(one.compartment,theo_sd,est="saem",control=saemControl(n.burn=250,n.em=350,pr
```

this example specifies 250 burn-in iterations, 350 em iterations and a print progress every 50 runs.

14.3 Model Syntax for solved PK systems

Solved PK systems are also currently supported by nlmixr with the 'linCmt()' pseudo-function. An annotated example of a solved system is below:

```
f <- function(){
  ini({
    lCl <- 1.6      ; label("log Cl (L/hr)")
    lVc <- log(90)  ; label("log Vc (L)")
    lKA <- 0.1      ; label("log Ka (1/hr)")
    prop.err <- c(0, 0.2, 1)
    eta.Cl ~ 0.1   # BSV Cl
    eta.Vc ~ 0.1   # BSV Vc
    eta.KA ~ 0.1   # BSV Ka
  })
  model({
    Cl <- exp(lCl + eta.Cl)
    Vc = exp(lVc + eta.Vc)
    KA <- exp(lKA + eta.KA)
    ## Instead of specifying the ODEs, you can use
    ## the linCmt() function to use the solved system.
    ##
    ## This function determines the type of PK solved system
    ## to use by the parameters that are defined.  In this case
    ## it knows that this is a one-compartment model with first-order
    ## absorption.
    linCmt() ~ prop(prop.err)
  })
}
```

A few things to keep in mind:

- The solved systems implemented are the one, two and three compartment models with or without first-order absorption. Each of the models support a lag time with a tlag parameter.
- In general the linear compartment model figures out the model by the parameter names. `nlmixr2` currently knows about numbered volumes, `Vc`/`Vp`, Clearances in terms of both `Cl` and `Q`/`CLD`. Additionally nlmixr knows about elimination micro-constants (ie `K12`). Mixing of these parameters for these models is currently not supported.

For the most up-to-date information about `linCmt()` models see the rxode2 documentation.

14.4 Checking model syntax

After specifying the model syntax you can check that nlmixr is interpreting it correctly by using the nlmixr function on it. Using the above function we can get:

```
nlmixr(f)
```

```
i parameter labels from comments are typically ignored in non-interactive mode


i Need to run with the source intact to parse comments


 -- rxode2-based solved PK 1-compartment model with first-order absorption ------
 -- Initalization: --
Fixed Effects ($theta):
     lCl      lVc      lKA prop.err
 1.60000  4.49981  0.10000  0.20000

Omega ($omega):
       eta.Cl eta.Vc eta.KA
eta.Cl    0.1    0.0    0.0
eta.Vc    0.0    0.1    0.0
eta.KA    0.0    0.0    0.1
 -- mu-referencing ($muRefTable): --
  theta    eta level
1   lCl eta.Cl    id
2   lVc eta.Vc    id
3   lKA eta.KA    id

 -- Model (Normalized Syntax): --
function() {
    ini({
        lCl <- 1.6
        label("log Cl (L/hr)")
        lVc <- 4.49980967033027
        label("log Vc (L)")
        lKA <- 0.1
        label("log Ka (1/hr)")
        prop.err <- c(0, 0.2, 1)
        eta.Cl ~ 0.1
        eta.Vc ~ 0.1
        eta.KA ~ 0.1
    })
    model({
        Cl <- exp(lCl + eta.Cl)
        Vc = exp(lVc + eta.Vc)
        KA <- exp(lKA + eta.KA)
        linCmt() ~ prop(prop.err)
    })
}
```

In general this gives you information about the model (what type of solved system/`rxode2()`), initial estimates as well as the code for the model block.

15 Model diagnostics and evaluation

16 Handling Covariate Effects

17 Model Selection and Comparison

18 Case Studies and Practical Applications

19 Tracking work with `shinyMixR`

Part VI

Advanced topics

20 Inside the mixrverse

21 Extending the mixrverse

Part VII

The mixrverse

22 Qualifying the mixrverse

23 babelmixr

24 Into the mixrverse

Part VIII

The future

25 Future Directions

26 Conclusion and Final Thoughts

27 Function reference

References

1. Derendorf H, Schmidt S. Rowland and Tozer's Clinical Pharmacokinetics and Pharmacodynamics: Concepts and Applications. Wolters Kluwer; 2019.

2. Mould DR, Upton RN. Basic Concepts in Population Modeling, Simulation, and Model-Based Drug Development. CPT: Pharmacometrics & Systems Pharmacology. 2012;1(9):e6.

3. Mould DR, Upton RN. Basic concepts in population modeling, simulation, and model-based drug development - Part 2: Introduction to pharmacokinetic modeling methods. CPT: Pharmacometrics and Systems Pharmacology. 2013 Apr;2(4).

4. Upton RN, Mould DR. Basic concepts in population modeling, simulation, and model-based drug development: Part 3-introduction to pharmacodynamic modeling methods. CPT: Pharmacometrics and Systems Pharmacology. 2014 Jan;3(1).

5. Ette EI, Williams PJ. Pharmacometrics: The science of quantitative pharmacology. Wiley; 2007.

6. Gabrielsson J, Weiner D. Pharmacokinetic and Pharmacodynamic Data Analysis: Concepts and Applications, Fourth Edition. Taylor & Francis; 2007.

7. Wang W, Hallow KM, James DA. A tutorial on RxODE: Simulating differential equation pharmacometric models in R. CPT: Pharmacometrics and Systems Pharmacology. 2016;5(1):3–10.

8. Hallow KM, James DA, Wang W. Interactive evaluation of dosing regimens for a novel anti-diabetic agent: a case-study in the application of RxODE. In: PAGE 24 [Internet]. 2015. p. Abstr 3542. Available from: https://www.page-meeting.org/?abstract=3542

9. Pinheiro JC, Bates DM. Mixed-effects models in s and s-PLUS. New York: Springer; 2000.

10. Delyon BYB, Lavielle M, Moulines E. Convergence of a stochastic approximation version of the EM algorithm. Annals of Statistics [Internet]. 1999;27(1):94–128. Available from: https://arxiv.org/abs/arXiv:1011.1669v3

11. Xiong Y, James D, He F, Wang W. PMXstan: An R Library to Facilitate PKPD Modeling with Stan (M-01). Journal of Pharmacokinetics and Pharmacodynamics. 2015 Oct;42(S1):S11.

12. Carpenter B, Gelman A, Hoffman MD, Lee D, Goodrich B, Betancourt M, et al. Stan: A Probabilistic Programming Language. Journal of Statistical Software [Internet]. 2017;76(1):1–32. Available from: https://www.jstatsoft.org/index.php/jss/article/view/v076i01

13. Hoffman MD, Gelman A. The No-U-turn sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo. The Journal of Machine Learning Research. 2014 Jan;15(1):1593–623.

14. Almquist J, Leander J, Jirstrand M. Using sensitivity equations for computing gradients of the FOCE and FOCEI approximations to the population likelihood. Journal of Pharmacokinetics and Pharmacodynamics. 2015;42(3):191–209.

15. Fidler M, Wilkins JJ, Hooijmaijers R, Post TM, Schoemaker R, Trame MN, et al. Nonlinear Mixed-Effects Model Development and Simulation Using nlmixr and Related R Open-Source Packages. CPT: Pharmacometrics and Systems Pharmacology. 2019 Sep;8(9):621–33.

16. Schoemaker R, Fidler M, Laveille C, Wilkins JJ, Hooijmaijers R, Post TM, et al. Performance of the SAEM and FOCEI Algorithms in the Open-Source, Nonlinear Mixed Effect Modeling Tool nlmixr. CPT: Pharmacometrics and Systems Pharmacology. 2019;8(12):923–30.

17. Fidler M, Hooijmaijers R, Schoemaker R, Wilkins JJ, Xiong Y, Wang W. R and nlmixr as a gateway between statistics and pharmacometrics. CPT: Pharmacometrics and Systems Pharmacology. 2021 Apr;10(4):283–5.

18. Ariens EJ. Affinity and intrinsic activity in the theory of competitive inhibition. I. Problems and theory. Arch Int Pharmacodyn Ther. 1954 Sep;99(1):32–49.

19. Dayneka NL, Garg V, Jusko WJ. Comparison of four basic models of indirect pharmacodynamic responses. Journal of Pharmacokinetics and Biopharmaceutics. 1993 Aug 15;21(4):457–78.

20. Ezzet F, Pinheiro JC. Linear, Generalized Linear, and Nonlinear Mixed Effects Models. In: Ette EI, Williams PJ, editors. Pharmacometrics: The Science of Quantitative Pharmacology. New Jersey: John Wiley & Sons; 2007. p. 103–35.

21. Sheiner LB, Rosenberg B, Melmon KL. Modelling of individual pharmacokinetics for computer-aided drug dosage. Computers and Biomedical Research [Internet]. 1972 Oct 1 [cited 2024 Oct 14];5(5):441–59. Available from: https://www.sciencedirect.com/science/article/pii/0010480972900511

22. Belenky G, Wesensten NJ, Thorne DR, Thomas ML, Sing HC, Redmond DP, et al. Patterns of performance degradation and restoration during sleep restriction and subsequent recovery: A sleep dose-response study. Journal of Sleep Research [Internet]. 2003 [cited 2023 Sep 17];12(1):1–12. Available from: https://onlinelibrary.wiley.com/doi/abs/10.1046/j.1365-2869.2003.00337.x

23. Bates D, Mächler M, Bolker B, Walker S. Fitting linear mixed-effects models using lme4. Journal of Statistical Software. 2015;67(1):1–48.

24. Karlsson MO, Sheiner LB. The importance of modeling interoccasion variability in population pharmacokinetic analyses. Journal of Pharmacokinetics and Biopharmaceutics. 1993;21(6):735–50.

25. Box GEP, Cox DR. An analysis of transformations. Journal of the Royal Statistical Society: Series B (Methodological) [Internet]. 1964 Jul 1 [cited 2024 Nov 28];26(2):211–43. Available from: https://doi.org/10.1111/j.2517-6161.1964.tb00553.x

26. Yeo IK, Johnson RA. A new family of power transformations to improve normality or symmetry. Biometrika [Internet]. 2000 [cited 2024 Nov 28];87(4):954–9. Available from: https://www.jstor.org/stable/2673623

27. Aitchison J, Shen SM. Logistic-normal distributions: Some properties and uses. Biometrika [Internet]. 1980 [cited 2024 Nov 28];67(2):261–72. Available from: https://www.jstor.org/stable/2335470

28. Bliss CI. The method of probits. Science [Internet]. 1934 Jan 12 [cited 2024 Nov 28];79(2037):38–9. Available from: https://www.science.org/doi/10.1126/science.79.2037.38

29. West RM. Best practice in statistics: The use of log transformation. Ann Clin Biochem [Internet]. 2022 May 1 [cited 2024 Nov 28];59(3):162–5. Available from: https://doi.org/10.1177/00045632211050531

30. Wendling T, Tsamandouras N, Dumitras S, Pigeolet E, Ogungbenro K, Aarons L. Reduction of a Whole-Body Physiologically Based Pharmacokinetic Model to Stabilise the Bayesian Analysis of Clinical Data. The AAPS Journal [Internet]. 2016;18(1):196–209. Available from: http://link.springer.com/10.1208/s12248-015-9840-7

31. Savic RM, Jonker DM, Kerbusch T, Karlsson MO. Implementation of a transit compartment model for describing drug absorption in pharmacokinetic studies. Journal of Pharmacokinetics and Pharmacodynamics. 2007;34(5):711–26.

32. Wilkins JJ, Savic RM, Karlsson MO, Langdon G, McIlleron H, Pillai G, et al. Population pharmacokinetics of rifampin in pulmonary tuberculosis patients, including a semimechanistic model to describe variable absorption. Antimicrobial Agents and Chemotherapy. 2008;52(6):2138–48.

33. Peck CC, Kimko HHC. Clinical trial simulations : Applications and trends. 1. Aufl. New York, NY: Springer New York; 2011. (AAPS advances in the pharmaceutical sciences series; vol. 1).