Nonlinear mixed-effects modelling with nlmixr2

Matthew Fidler, Novartis Justin J. Wilkins, Occams

2024-12-10

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 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 what the body does to the drug (pharmacokinetics, PK) and what the drug does to the body (pharmacodynamics, 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.

I promised you beer! It's actually 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). Ethanol is a pretty interesting case, because it's 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.

1.2 `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.

1.3 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.

1.4 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.

1.5 What you won't learn

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

1.5.1 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).

1.5.2 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).

1.5.3 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.

1.5.4 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.

1.5.5 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.

1.6 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.

1.6.1 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.

1.6.2 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.

1.6.3 `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.

```
library(nlmixr2)
```

```
Loading required package: nlmixr2data
```

1.7 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.

1.8 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:

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

```
- Session info -------------------------------------------------------------
 setting  value
 version  R version 4.3.1 (2023-06-16)
 os       macOS Sonoma 14.0
 system   aarch64, darwin20
 ui       X11
 language (EN)
 collate  en_US.UTF-8
 ctype    en_US.UTF-8
 tz       Europe/Berlin
 date     2023-10-24
 pandoc   3.1.1 @ /Applications/RStudio.app/Contents/Resources/app/quarto/bin/tools/ (via rma

- Packages -----------------------------------------------------------------
 ! package       * version    date (UTC) lib source
   askpass         1.2.0      2023-09-03 [1] RSPM (R 4.3.0)
   assertthat      0.2.1      2019-03-21 [1] CRAN (R 4.3.0)
 P backports       1.4.1      2021-12-13 [?] CRAN (R 4.3.0)
   base64enc       0.1-3      2015-07-28 [1] CRAN (R 4.3.0)
   BH              1.81.0-1   2023-01-22 [1] CRAN (R 4.3.0)
   binom           1.1-1.1    2022-05-02 [1] CRAN (R 4.3.0)
   bit             4.0.5      2022-11-15 [1] CRAN (R 4.3.0)
   bit64           4.0.5      2020-08-30 [1] CRAN (R 4.3.0)
   bitops          1.0-7      2021-04-24 [1] CRAN (R 4.3.0)
```

```
  boot           1.3-28.1   2022-11-22 [1] CRAN (R 4.3.1)
  bslib          0.5.1      2023-08-11 [1] CRAN (R 4.3.0)
P cachem         1.0.8      2023-05-01 [?] CRAN (R 4.3.0)
  cellranger     1.1.0      2016-07-27 [1] CRAN (R 4.3.0)
P checkmate      2.2.0      2023-04-27 [?] CRAN (R 4.3.0)
  class          7.3-22     2023-05-03 [1] CRAN (R 4.3.1)
  classInt       0.4-10     2023-09-05 [1] RSPM (R 4.3.0)
P cli            3.6.1      2023-03-23 [?] CRAN (R 4.3.0)
  clipr          0.8.0      2022-02-22 [1] CRAN (R 4.3.0)
  cluster        2.1.4      2022-08-22 [1] CRAN (R 4.3.1)
P colorspace     2.1-0      2023-01-23 [?] CRAN (R 4.3.0)
  commonmark     1.9.0      2023-03-17 [1] CRAN (R 4.3.0)
  cpp11          0.4.6      2023-08-10 [1] CRAN (R 4.3.0)
  crayon         1.5.2      2022-09-29 [1] CRAN (R 4.3.0)
  curl           5.1.0      2023-10-02 [1] RSPM (R 4.3.0)
P data.table     1.14.8     2023-02-17 [?] CRAN (R 4.3.0)
  Deriv          4.1.3      2021-02-24 [1] CRAN (R 4.3.0)
  DescTools      0.99.50    2023-09-06 [1] RSPM (R 4.3.0)
P digest         0.6.33     2023-07-07 [?] CRAN (R 4.3.0)
P dparser        1.3.1-10   2023-03-16 [?] CRAN (R 4.3.0)
P dplyr          1.1.3      2023-09-03 [?] RSPM (R 4.3.0)
  e1071          1.7-13     2023-02-01 [1] CRAN (R 4.3.0)
  ellipsis       0.3.2      2021-04-29 [1] CRAN (R 4.3.0)
P evaluate       0.22       2023-09-29 [?] RSPM (R 4.3.0)
  Exact          3.2        2022-09-25 [1] CRAN (R 4.3.0)
  expm           0.999-7    2023-01-09 [1] CRAN (R 4.3.0)
P fansi          1.0.5      2023-10-08 [?] RSPM (R 4.3.0)
  farver         2.1.1      2022-07-06 [1] CRAN (R 4.3.0)
P fastmap        1.1.1      2023-02-24 [?] CRAN (R 4.3.0)
  fontawesome    0.5.2      2023-08-19 [1] CRAN (R 4.3.0)
  foreign        0.8-85     2023-09-09 [1] RSPM (R 4.3.0)
  Formula        1.2-5      2023-02-24 [1] CRAN (R 4.3.0)
  fs             1.6.3      2023-07-20 [1] CRAN (R 4.3.0)
P generics       0.1.3      2022-07-05 [?] CRAN (R 4.3.0)
P ggplot2        3.4.4      2023-10-12 [?] RSPM (R 4.3.0)
  ggtext         0.1.2      2022-09-16 [1] CRAN (R 4.3.0)
  gld            2.6.6      2022-10-23 [1] CRAN (R 4.3.0)
P glue           1.6.2      2022-02-24 [?] CRAN (R 4.3.0)
  gridExtra      2.3        2017-09-09 [1] CRAN (R 4.3.0)
  gridtext       0.1.5      2022-09-16 [1] CRAN (R 4.3.0)
P gtable         0.3.4      2023-08-21 [?] CRAN (R 4.3.0)
  highr          0.10       2022-12-22 [1] CRAN (R 4.3.0)
  Hmisc          5.1-1      2023-09-12 [1] RSPM (R 4.3.0)
```

```
  hms             1.1.3      2023-03-21 [1] CRAN (R 4.3.0)
  htmlTable       2.4.1      2022-07-07 [1] CRAN (R 4.3.0)
P htmltools       0.5.6.1    2023-10-06 [?] RSPM (R 4.3.0)
  htmlwidgets     1.6.2      2023-03-17 [1] CRAN (R 4.3.0)
  httr            1.4.7      2023-08-15 [1] CRAN (R 4.3.0)
  inline          0.3.19     2021-05-31 [1] CRAN (R 4.3.0)
  isoband         0.2.7      2022-12-20 [1] CRAN (R 4.3.0)
  jpeg            0.1-10     2022-11-29 [1] CRAN (R 4.3.0)
  jquerylib       0.1.4      2021-04-26 [1] CRAN (R 4.3.0)
P jsonlite        1.8.7      2023-06-29 [?] CRAN (R 4.3.0)
  KernSmooth      2.23-22    2023-07-10 [1] CRAN (R 4.3.0)
P knitr           1.44       2023-09-11 [?] RSPM (R 4.3.0)
  labeling        0.4.3      2023-08-29 [1] RSPM (R 4.3.0)
P lattice         0.21-9     2023-10-01 [?] RSPM (R 4.3.0)
  lazyeval        0.2.2      2019-03-15 [1] CRAN (R 4.3.0)
P lbfgsb3c        2020-3.2   2020-03-03 [?] CRAN (R 4.3.0)
P lifecycle       1.0.3      2022-10-07 [?] CRAN (R 4.3.0)
  lmom            3.0        2023-08-29 [1] RSPM (R 4.3.0)
P lotri           0.4.3      2023-03-20 [?] CRAN (R 4.3.0)
P magrittr        2.0.3      2022-03-30 [?] CRAN (R 4.3.0)
  markdown        1.10       2023-10-10 [1] RSPM (R 4.3.0)
  MASS            7.3-60     2023-05-04 [1] CRAN (R 4.3.1)
  Matrix          1.6-1.1    2023-09-18 [1] RSPM (R 4.3.0)
P memoise         2.0.1      2021-11-26 [?] CRAN (R 4.3.0)
  mgcv            1.9-0      2023-07-11 [1] CRAN (R 4.3.0)
  mime            0.12       2021-09-28 [1] CRAN (R 4.3.0)
  minpack.lm      1.2-4      2023-09-11 [1] RSPM (R 4.3.0)
  minqa           1.2.6      2023-09-11 [1] RSPM (R 4.3.0)
P munsell         0.5.0      2018-06-12 [?] CRAN (R 4.3.0)
  mvtnorm         1.2-3      2023-08-25 [1] CRAN (R 4.3.0)
P n1qn1           6.0.1-11   2022-10-18 [?] CRAN (R 4.3.0)
P nlme            3.1-163    2023-08-09 [?] CRAN (R 4.3.0)
P nlmixr2       * 2.0.9      2023-02-21 [?] CRAN (R 4.3.0)
P nlmixr2data   * 2.0.8      2023-08-30 [?] RSPM (R 4.3.0)
P nlmixr2est      2.1.8      2023-10-08 [?] RSPM (R 4.3.0)
  nlmixr2extra    2.0.8      2022-10-22 [1] CRAN (R 4.3.0)
P nlmixr2plot     2.0.7      2022-10-20 [?] CRAN (R 4.3.0)
  nnet            7.3-19     2023-05-03 [1] CRAN (R 4.3.1)
  numDeriv        2016.8-1.1 2019-06-06 [1] CRAN (R 4.3.0)
  openssl         2.1.1      2023-09-25 [1] RSPM (R 4.3.0)
  pander          0.6.5      2022-03-18 [1] CRAN (R 4.3.0)
P pillar          1.9.0      2023-03-22 [?] CRAN (R 4.3.0)
P pkgconfig       2.0.3      2019-09-22 [?] CRAN (R 4.3.0)
```

```
  png             0.1-8      2022-11-29 [1] CRAN (R 4.3.0)
P PreciseSums     0.6        2023-04-22 [?] CRAN (R 4.3.0)
  prettyunits     1.2.0      2023-09-24 [1] RSPM (R 4.3.0)
  progress        1.2.2      2019-05-16 [1] CRAN (R 4.3.0)
  proxy           0.4-27     2022-06-09 [1] CRAN (R 4.3.0)
  purrr           1.0.2      2023-08-10 [1] CRAN (R 4.3.0)
P qs              0.25.5     2023-02-22 [?] CRAN (R 4.3.0)
P R6              2.5.1      2021-08-19 [?] CRAN (R 4.3.0)
P RApiSerialize   0.1.2      2022-08-25 [?] CRAN (R 4.3.0)
  rappdirs        0.3.3      2021-01-31 [1] CRAN (R 4.3.0)
  RColorBrewer    1.1-3      2022-04-03 [1] CRAN (R 4.3.0)
P Rcpp            1.0.11     2023-07-06 [?] CRAN (R 4.3.0)
  RcppArmadillo   0.12.6.4.0 2023-09-10 [1] RSPM (R 4.3.0)
  RcppEigen       0.3.3.9.3  2022-11-05 [1] CRAN (R 4.3.0)
P RcppParallel    5.1.7      2023-02-27 [?] CRAN (R 4.3.0)
  RCurl           1.98-1.12  2023-03-27 [1] CRAN (R 4.3.0)
  readr           2.1.4      2023-02-10 [1] CRAN (R 4.3.0)
  readxl          1.4.3      2023-07-06 [1] CRAN (R 4.3.0)
  rematch         2.0.0      2023-08-30 [1] RSPM (R 4.3.0)
  rex             1.2.1      2021-11-26 [1] CRAN (R 4.3.0)
P rlang           1.1.1      2023-04-28 [?] CRAN (R 4.3.0)
P rmarkdown       2.25       2023-09-18 [?] RSPM (R 4.3.0)
  rootSolve       1.8.2.4    2023-09-21 [1] RSPM (R 4.3.0)
  rpart           4.1.21     2023-10-09 [1] RSPM (R 4.3.0)
P rstudioapi      0.15.0     2023-07-07 [?] CRAN (R 4.3.0)
P rxode2          2.0.14     2023-10-07 [?] CRAN (R 4.3.1)
P rxode2et        2.0.10     2023-03-17 [?] CRAN (R 4.3.0)
  rxode2ll        2.0.11     2023-03-17 [1] CRAN (R 4.3.0)
P rxode2parse     2.0.16     2023-03-28 [?] CRAN (R 4.3.0)
P rxode2random    2.0.11     2023-03-28 [?] CRAN (R 4.3.0)
  sass            0.4.7      2023-07-15 [1] CRAN (R 4.3.0)
P scales          1.2.1      2022-08-20 [?] CRAN (R 4.3.0)
  sitmo           2.0.2      2021-10-13 [1] CRAN (R 4.3.0)
  StanHeaders     2.26.28    2023-09-07 [1] RSPM (R 4.3.0)
P stringfish      0.15.8     2023-05-30 [?] CRAN (R 4.3.0)
  stringi         1.7.12     2023-01-11 [1] CRAN (R 4.3.0)
  stringr         1.5.0      2022-12-02 [1] CRAN (R 4.3.0)
  survival        3.5-7      2023-08-14 [1] CRAN (R 4.3.0)
P symengine       0.2.2      2022-10-23 [?] CRAN (R 4.3.0)
P sys             3.4.2      2023-05-23 [?] CRAN (R 4.3.0)
P tibble          3.2.1      2023-03-20 [?] CRAN (R 4.3.0)
  tidyr           1.3.0      2023-01-24 [1] CRAN (R 4.3.0)
P tidyselect      1.2.0      2022-10-10 [?] CRAN (R 4.3.0)
```

```
  tinytex         0.47       2023-09-29 [1] RSPM (R 4.3.0)
  tzdb            0.4.0      2023-05-12 [1] CRAN (R 4.3.0)
P utf8            1.2.3      2023-01-31 [?] CRAN (R 4.3.0)
P vctrs           0.6.4      2023-10-12 [?] RSPM (R 4.3.0)
  viridis         0.6.4      2023-07-22 [1] CRAN (R 4.3.0)
  viridisLite     0.4.2      2023-05-02 [1] CRAN (R 4.3.0)
P vpc             1.2.2      2021-01-11 [?] CRAN (R 4.3.0)
  vroom           1.6.4      2023-10-02 [1] RSPM (R 4.3.0)
  withr           2.5.1      2023-09-26 [1] RSPM (R 4.3.0)
P xfun            0.40       2023-08-09 [?] CRAN (R 4.3.0)
  xgxr            1.1.2      2023-03-22 [1] CRAN (R 4.3.0)
  xml2            1.3.5      2023-07-06 [1] CRAN (R 4.3.0)
P yaml            2.3.7      2023-01-23 [?] CRAN (R 4.3.0)

 [1] /Users/justin/Documents/GitHub/nlmixr2_book/renv/library/R-4.3/aarch64-apple-darwin20
 [2] /Users/justin/Library/Caches/org.R-project.R/R/renv/sandbox/R-4.3/aarch64-apple-darwin2

 P -- Loaded and on-disk path mismatch.

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

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 `nlmixr2` and 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.

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. This one is not yet on CRAN but can be downloaded from GitHub: `devtools::install_github("richardhooijmaijers/shinyMixR")`.
- `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)`

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.

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.

4 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 pharma. 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 (18).

4.1 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.

4.1.1 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 $\widehat{\beta}$. 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:

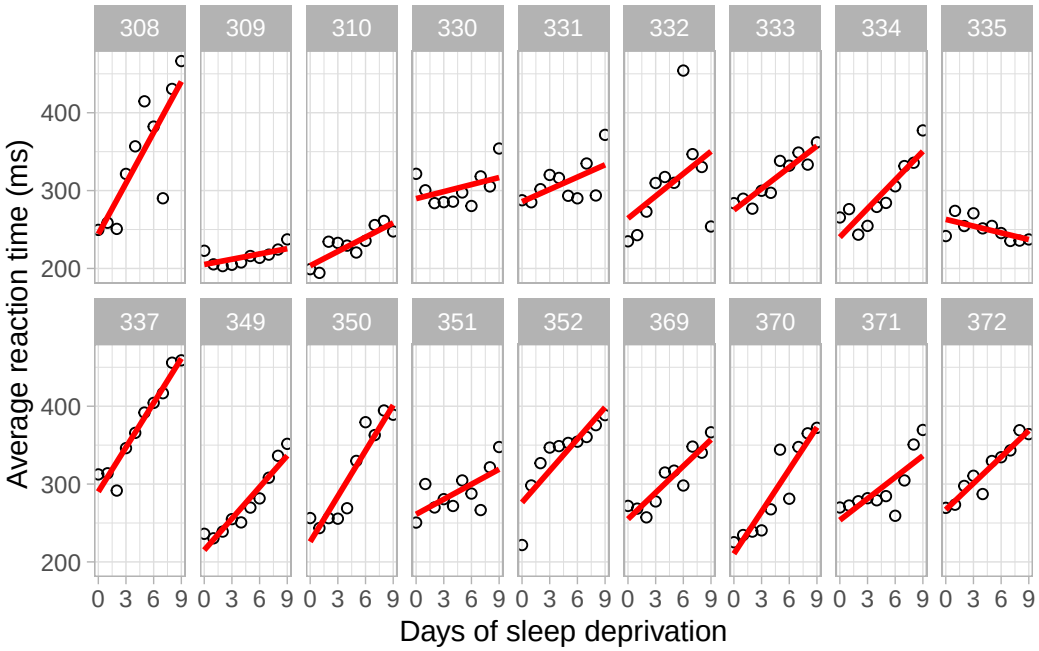
$$\beta \sim N(\mu, \sigma)$$

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* (19,20). 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:

```
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.

4.1.2 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.

One of the most prominent and impactful applications of nonlinear mixed-effects modeling is found in pharmacokinetics (PK) and pharmacodynamics (PD). PK concerns the study of drug absorption, distribution, metabolism, and excretion in the body, while PD explores the drug's effects on the body. Compartmental models, a subset of nonlinear mixed-effects models, use a series of interconnected compartments to represent drug movement through the body. These models facilitate the estimation of critical parameters such as clearance (CL), volume of distribution (V), and absorption rate (ka), which are crucial for optimizing drug dosing regimens.

In the realm of drug development, nonlinear mixed-effects 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:

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

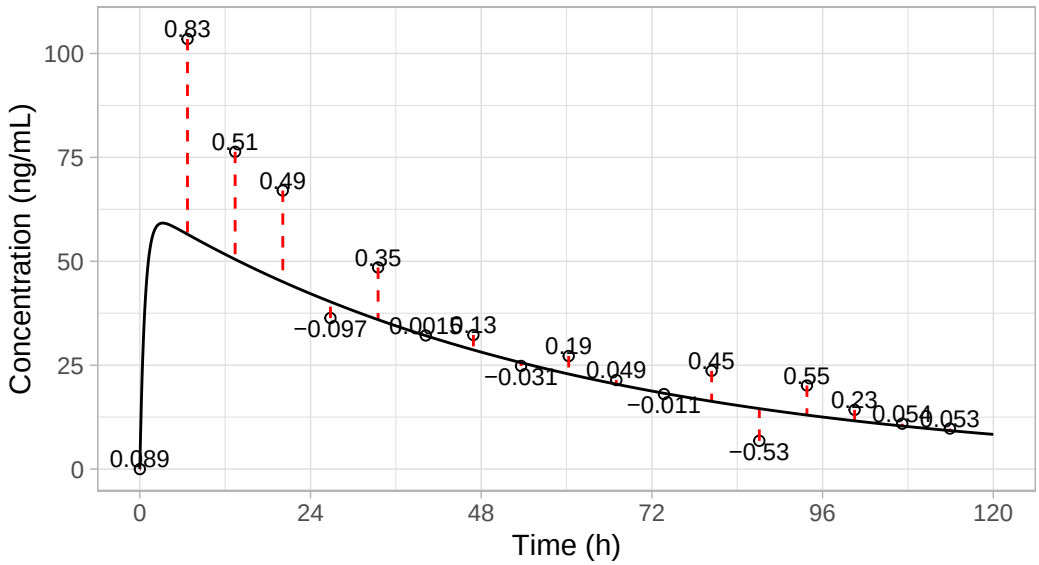
The functional form of f can of course vary depending on the model being considered (in this case, still the oral one-compartmental PK model).

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 (21). 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:

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

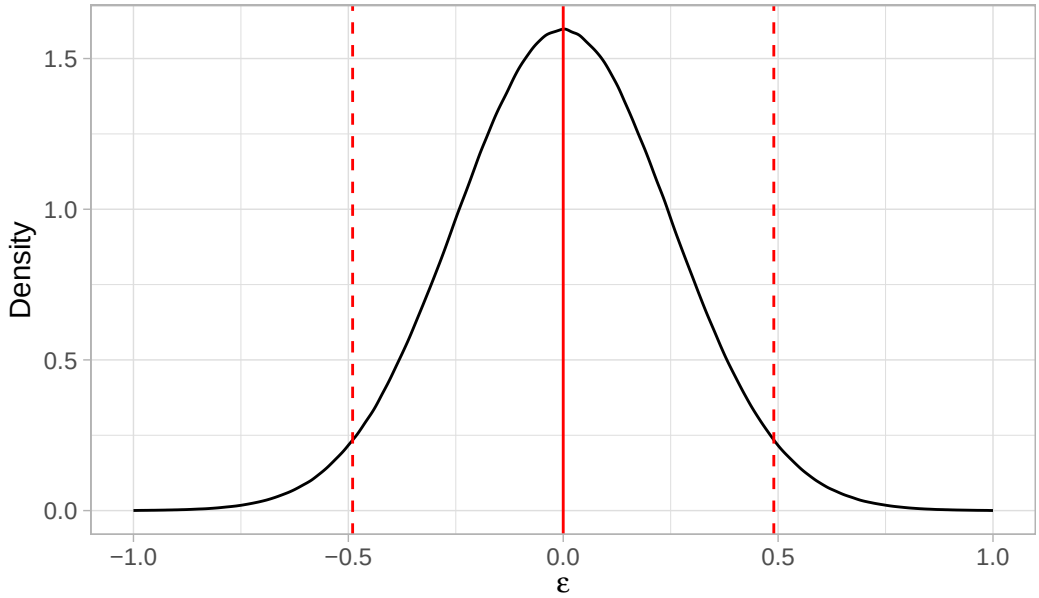
Here we have added the independently-distributed random effect $\kappa_{V,ij}$, representing between-occasion variability in individual i at measurement occasion j, assumed normally distributed with mean 0 and variance π^2.

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:

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

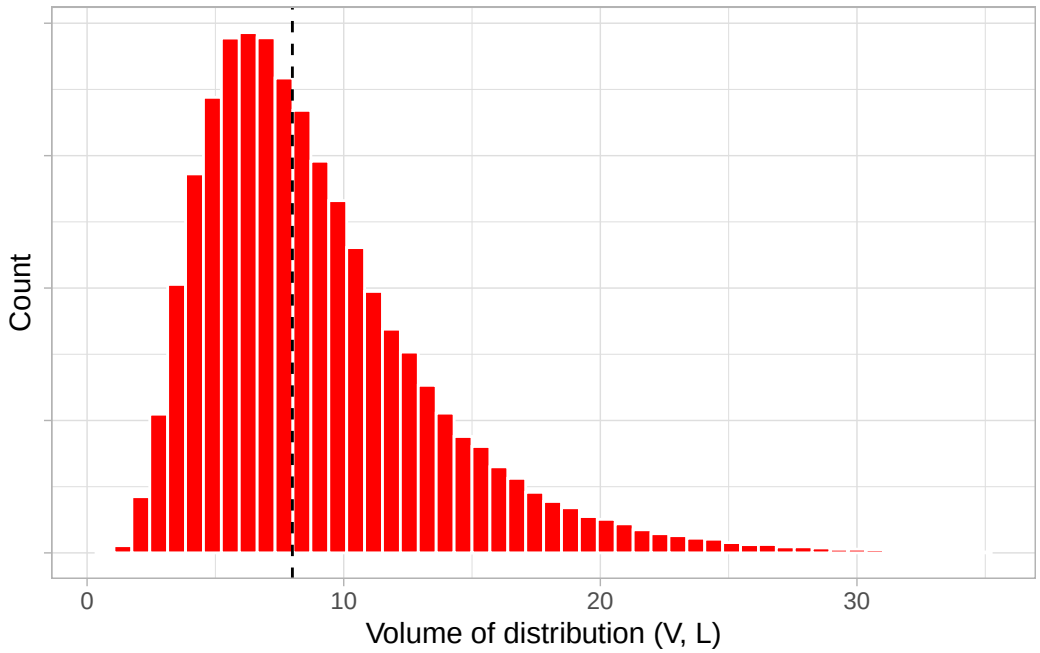
Or both:

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

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^2_V 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 II

Simulations with `rxode2`

5 Getting started

5.1 About `rxode2`

`rxode2` is a set of R packages for solving and simulating from models based on ordinary differential equations (ODEs). These 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

These should all have been installed automatically along with `nlmixr2`. We're going to start with `rxode2` because it forms the computational core of `nlmixr2`, and uses the same shorthand for specifying models.

5.2 Installing and loading `rxode2`

To install `rxode2`, you first need to download it from CRAN, if you haven't already...

```
install.packages("rxode2")
```

and then load it:

```
library("rxode2")
```

```
rxode2 2.0.14 using 1 threads (see ?getRxThreads)
  no cache: create with `rxCreateCache()`
```

```
======================================
rxode2 has not detected OpenMP support and will run in single-threaded mode
This is a Mac. Please read https://mac.r-project.org/openmp/
======================================
```

You now have `rxode2` available for use in your R environment. (We hope you're using Rstudio, because that makes everything easier.)

6 Events in `rxode2` and `nlmixr2`

6.1 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.

6.1.1 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…

6.1.1.1 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.

6.1.1.2 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.

6.1.1.3 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.

When returning the rxode2 solved dataset, there are a few additional event IDs (evid) that you may encounter, depending on the solving options you have used.

- evid = -1 indicates when a modeled rate ends (corresponds to rate = -1 in the event table).
- evid = -2 indicates when a modeled duration ends (corresponds to rate = -2 in the event table).
- evid = -10 indicates when a rate-specified zero-order infusion ends (corresponds to rate > 0 in the event table).
- evid = -20 indicates when a dur-specified zero-order infusion ends (corresponds to dur > 0 in the event table).
- evid = 101, 102, 103, … indicate modeled times (mtime) corresponding to the 1, 2, 3, … modeled times.

These can only be accessed when solving with the option combination addDosing=TRUE and subsetNonmem=FALSE. If you want to see the classic EVID equivalents you can use addDosing=NA.

6.1.1.4 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.

6.1.2 Dosing

6.1.2.1 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).

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.3     v readr     2.1.4
v forcats   1.0.0     v stringr   1.5.0
v ggplot2   3.4.4     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.0
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 beco
```

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

```
rxode2 2.0.14 using 1 threads (see ?getRxThreads)
  no cache: create with `rxCreateCache()`
========================================
rxode2 has not detected OpenMP support and will run in single-threaded mode
This is a Mac. Please read https://mac.r-project.org/openmp/
========================================
```

```
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

ev
```

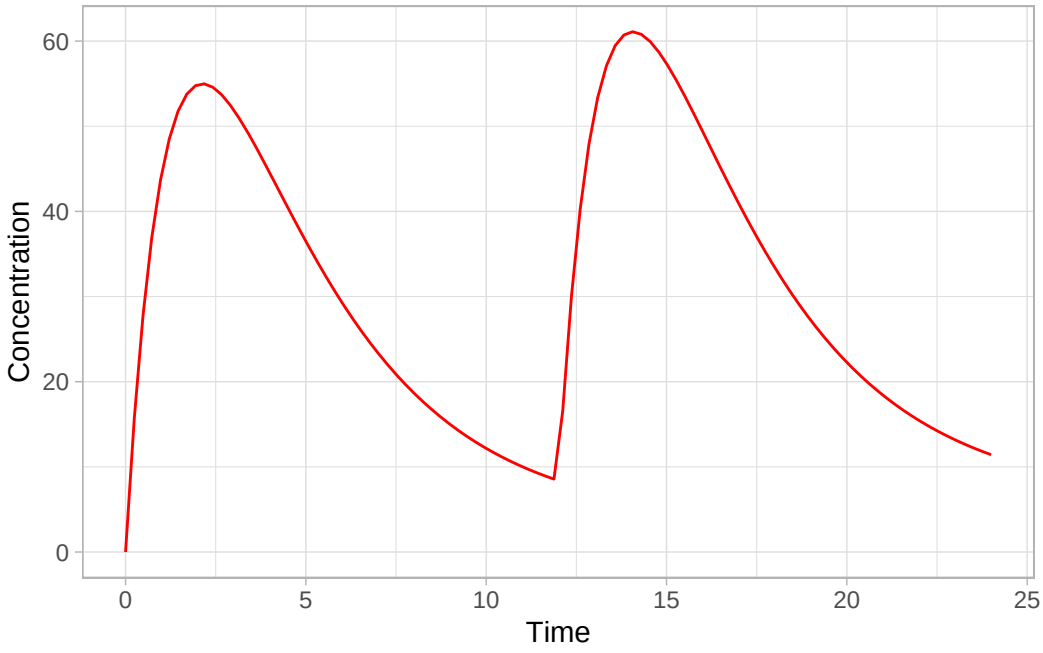
```
-- EventTable with 101 records --
1 dosing records (see x$get.dosing(); add with add.dosing or et)
100 observation times (see x$get.sampling(); add with add.sampling or et)
multiple doses in `addl` columns, expand with x$expand(); or etExpand(x)
-- First part of x: --
# A tibble: 101 x 5
    time   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 will be replaced by 'label()'
```

```
using C compiler: 'Apple clang version 15.0.0 (clang-1500.0.40.1)'
using SDK: ''
ld: warning: -single_module is obsolete
ld: warning: -multiply_defined is obsolete
```



6.1.2.2 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

6.1.2.2.1 Constant infusions

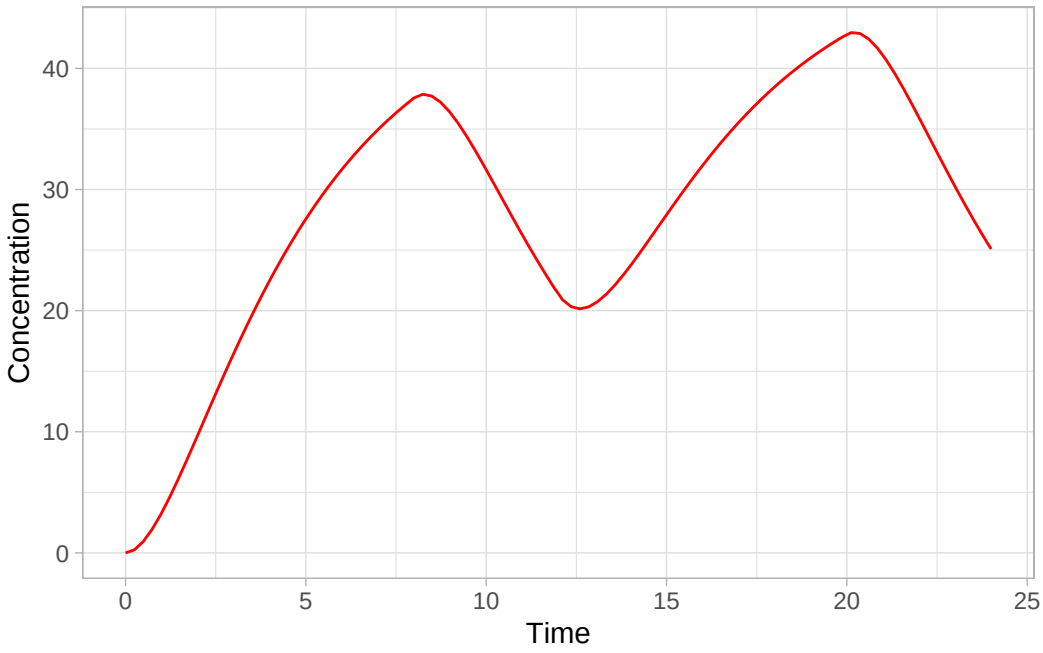
There are two ways to specify an infusion in `rxode2`. The first is to use the `dur` field. Here we specify an infusion given over 8 hours.

```
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
```

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

```
i parameter labels from comments will be replaced by 'label()'

using C compiler: ‘Apple clang version 15.0.0 (clang-1500.0.40.1)’
using SDK: ‘’
ld: warning: -single_module is obsolete
ld: warning: -multiply_defined is obsolete
```



We could, alternatively, specify `rate` instead.

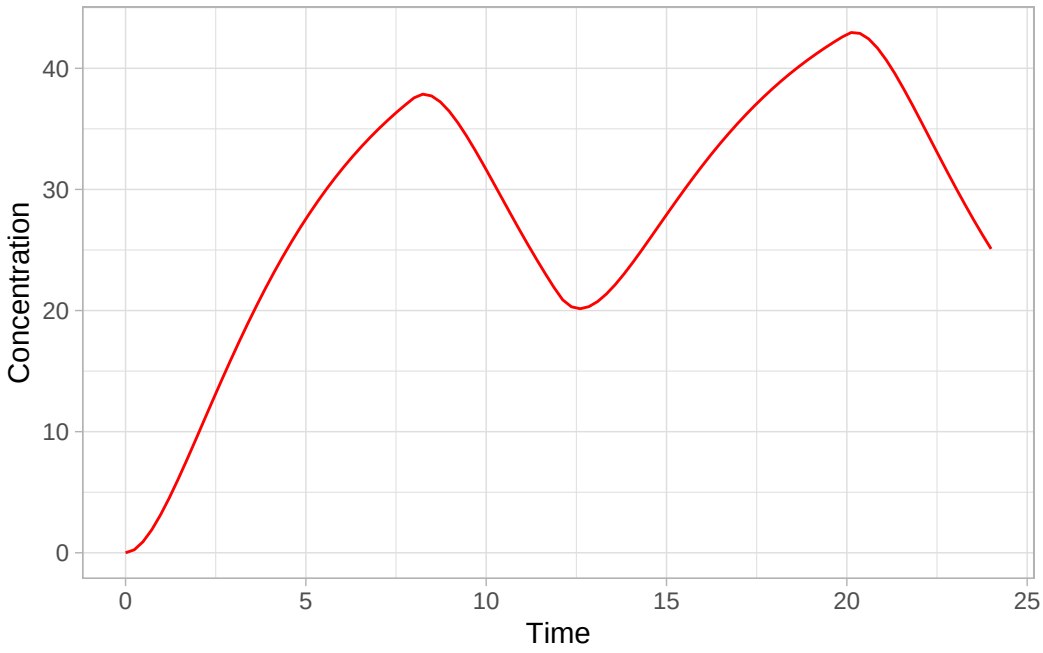
```
ev <- et(timeUnits="hr") %>%
    et(amt=10000, ii=12, until=24, rate=10000/8) %>%
    et(seq(0, 24, length.out=100))

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 will be replaced by 'label()'
```



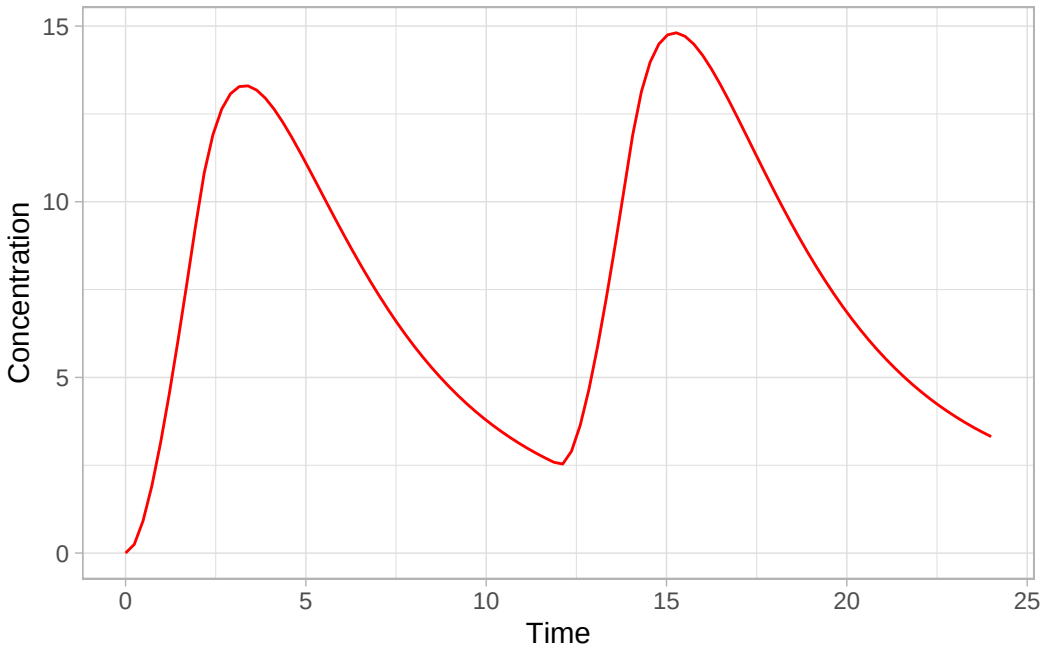
As you can see, we get the same result whichever way we do it. Where we need to pay attention is bioavailability (F). If we change F, the infusion is changed.

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()
```

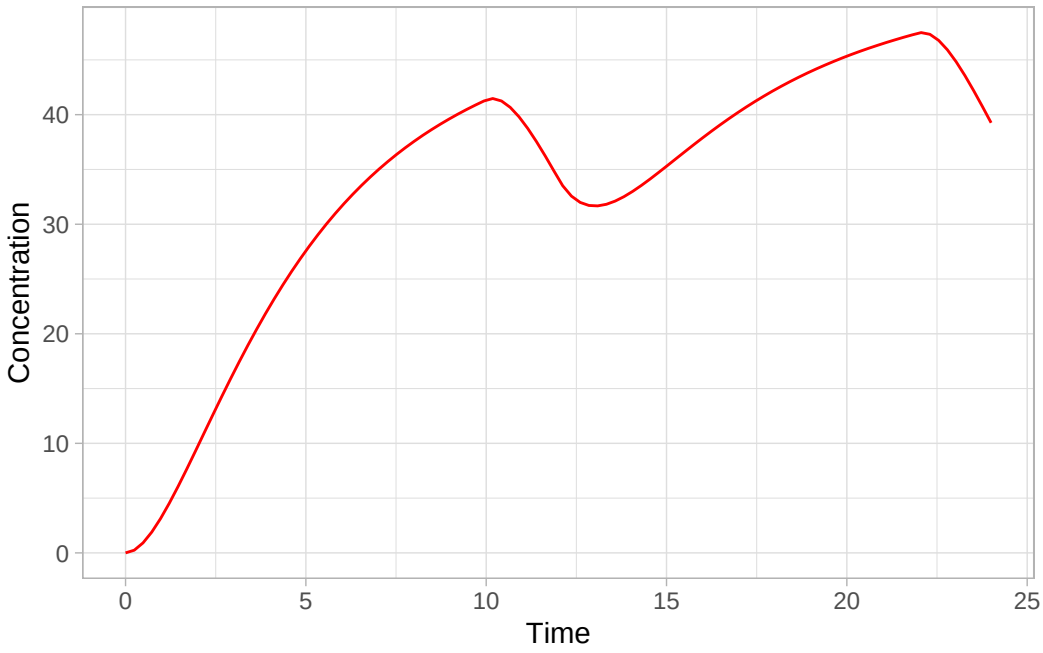
```
i parameter labels from comments will be replaced by 'label()'
```



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 will be replaced by 'label()'
```



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 will be replaced by 'label()'
```

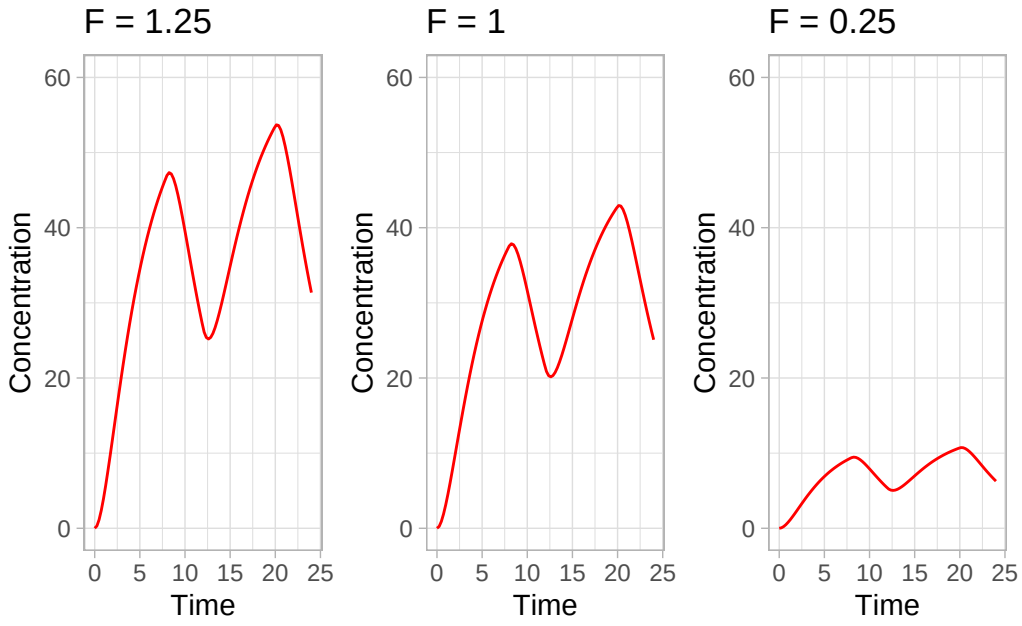
```
p1 <- rxSolve(mod1, ev, params = c(other_params, fdepot=1.25)) %>% ggplot(aes(time, C2)) +
    geom_line(col="red") +
  coord_cartesian(ylim=c(0,60)) +
  xlab("Time") + ylab("Concentration") +labs(title="F = 1.25")+
  theme_light()
```

```
i parameter labels from comments will be replaced by 'label()'
```

```
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 will be replaced by 'label()'
```

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



6.1.2.3 Steady state

6.1.2.3.1 Dosing to staedy state

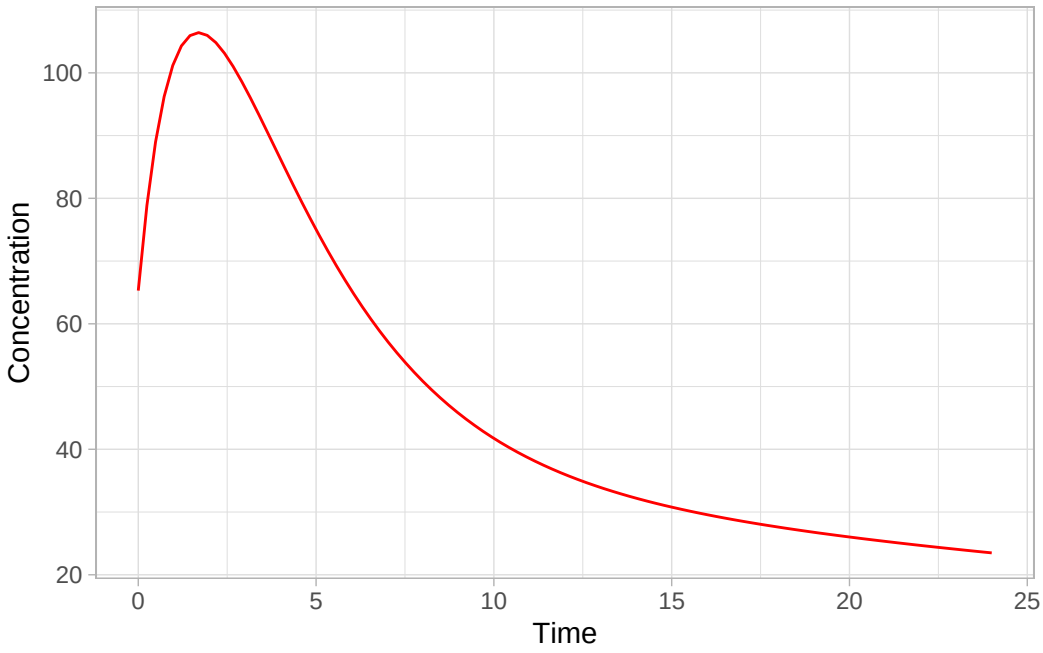
When the steady-state (`ss`) flag is set, doses are solved until steady state is reached with a constant inter-dose interval.

```
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
```

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

```
i parameter labels from comments will be replaced by 'label()'
```



6.1.2.3.2 Steady state for complex dosing

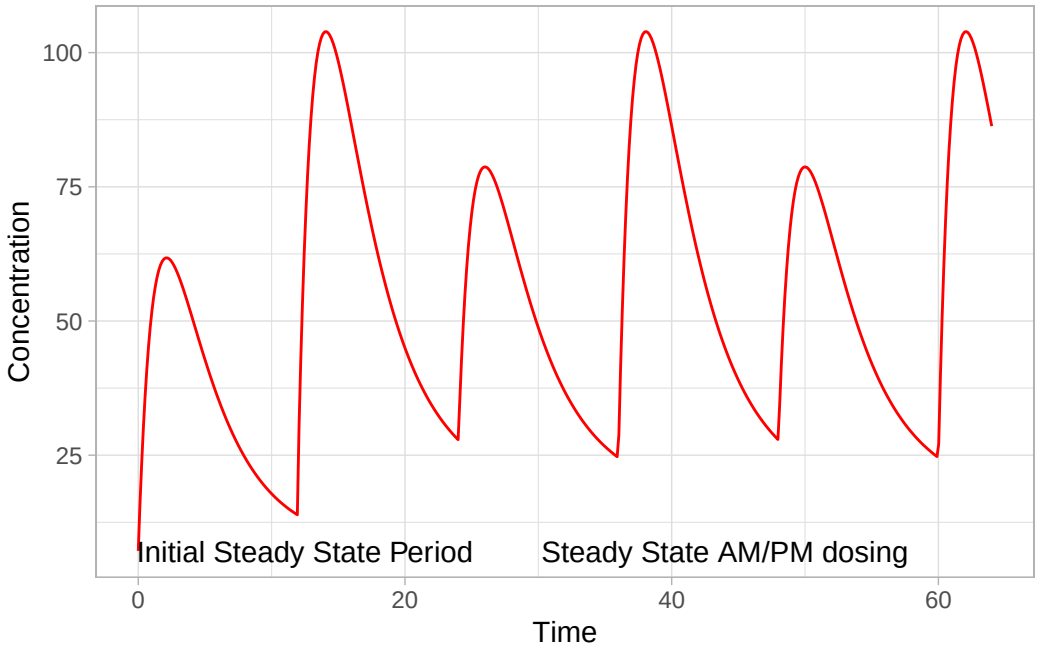
By using the `ss=2` flag, the super-positioning principle in linear kinetics can be applied to set up nonstandard dosing to steady state. In thise xample, we're giving different deses in mornings and evenings.

```
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
```

```
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 will be replaced by 'label()'
```



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

6.1.2.3.3 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:

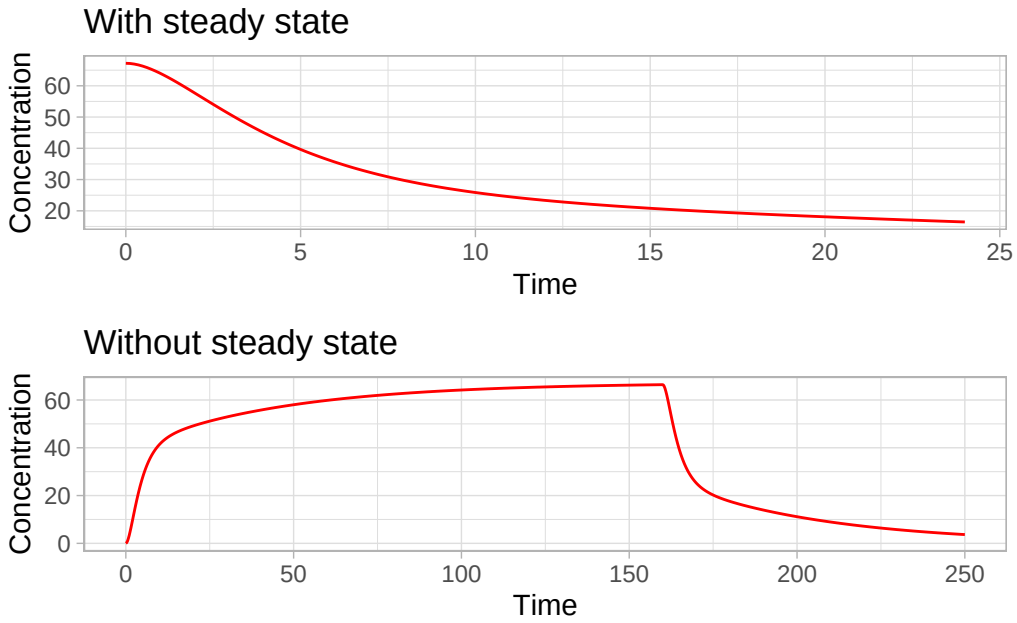
```
ev <- et(timeUnits="hr") %>%
    et(amt=0, ss=1,rate=10000/8)

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 will be replaced by 'label()'
```

```
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 will be replaced by 'label()'
```

```
library(patchwork)

p1 / p2
```



6.1.2.4 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.

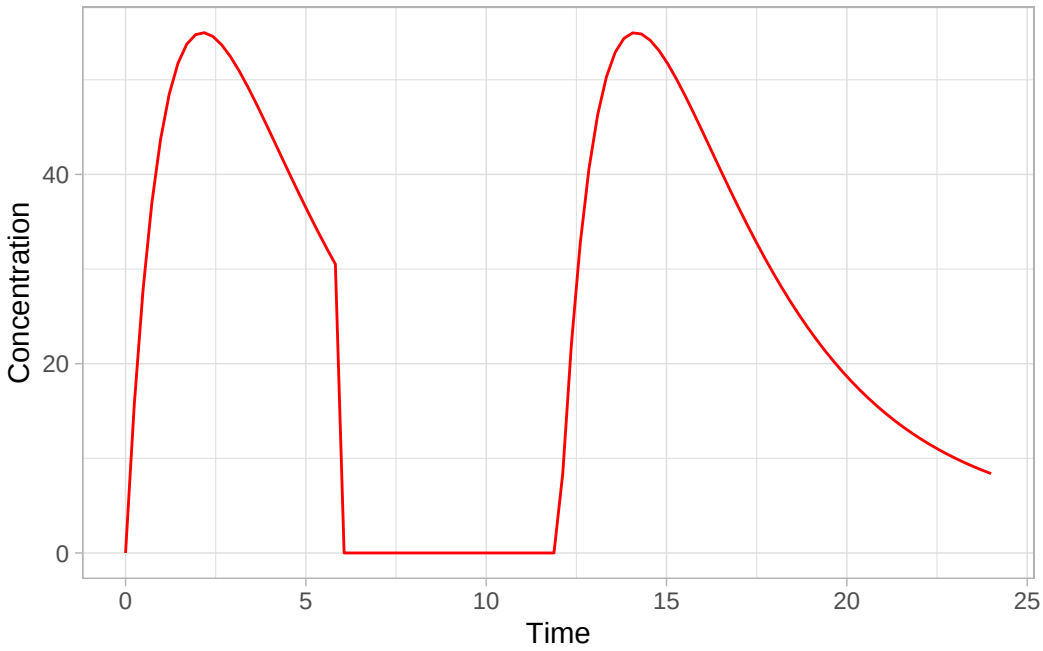
```
ev <- et(timeUnits="hr") %>%
    et(amt=10000, ii=12, addl=3) %>%
    et(time=6, evid=reset) %>%
    et(seq(0, 24, length.out=100))

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
```

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

```
i parameter labels from comments will be replaced by 'label()'
```



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

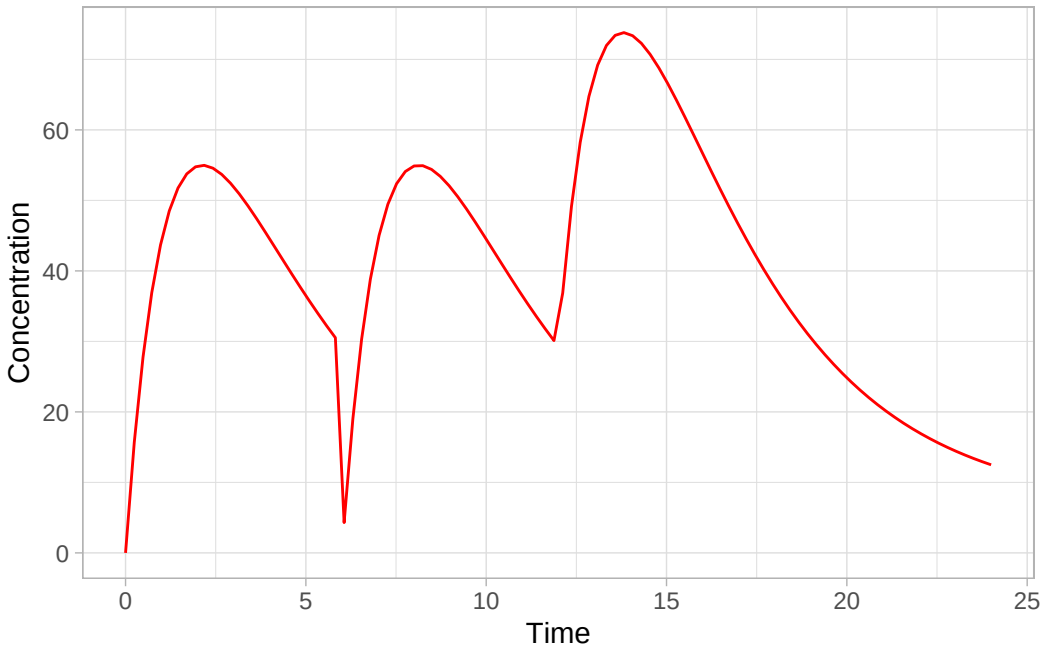
Now let's do a reset-and-dose.

```
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 will be replaced by 'label()'
```



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.

6.1.2.5 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
```

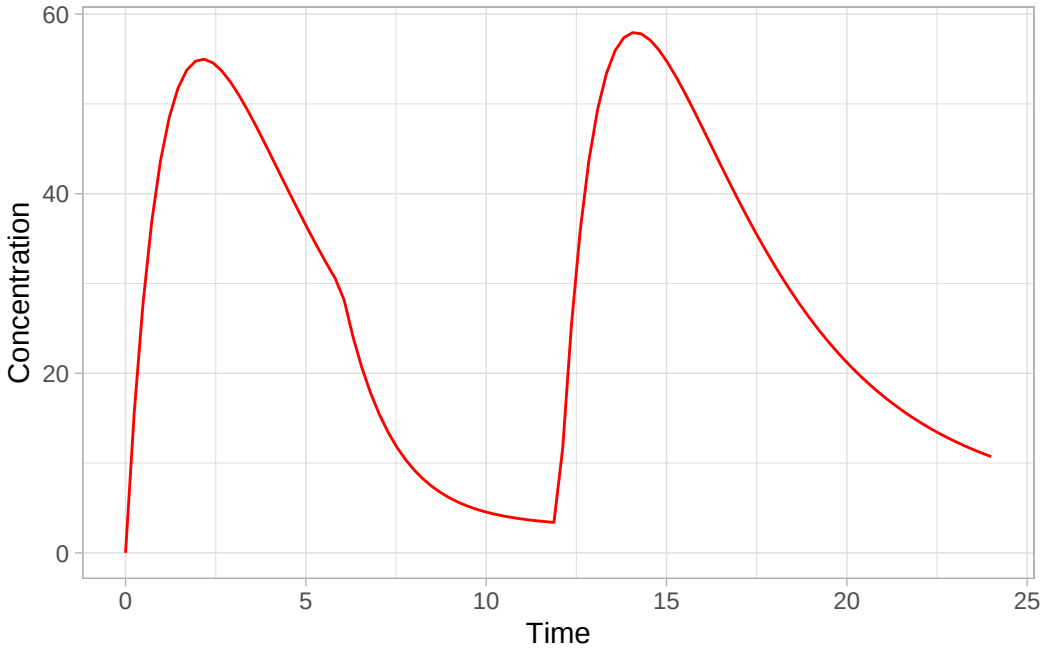
```
-- EventTable with 102 records --
2 dosing records (see x$get.dosing(); add with add.dosing or et)
100 observation times (see x$get.sampling(); add with add.sampling or et)
multiple doses in `addl` columns, expand with x$expand(); or etExpand(x)
-- First part of x: --
```

```
# 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 will be replaced by 'label()'
```



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.

7 Getting started with `rxode2`: simulating single subjects

7.1 Basic concepts of `rxode2` syntax

Now that we've got to grips with events, let's have a look at the models themselves!

7.1.1 Writing models for `rxode2`

Writing models for `rxode2` (and by extension `nlmixr2`) is relatively straightforward. Here's a simple example.

```
library(rxode2)
```

```
rxode2 2.0.14 using 1 threads (see ?getRxThreads)
  no cache: create with `rxCreateCache()`
```

```
========================================
rxode2 has not detected OpenMP support and will run in single-threaded mode
This is a Mac. Please read https://mac.r-project.org/openmp/
========================================
```

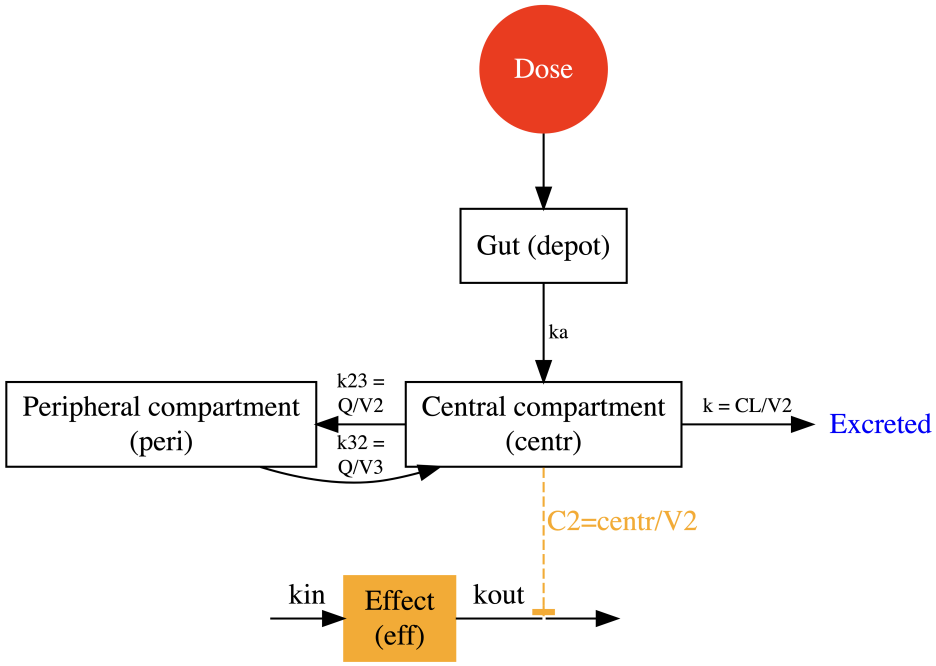
```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.3     v readr     2.1.4
v forcats   1.0.0     v stringr   1.5.0
v ggplot2   3.4.4     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.0
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
```

```
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

    # effects
    Kin  = 1      # Effect generation rate constant (/h)
    Kout = 1      # Effect elimination rate constant (/h)
    EC50 = 200    # EC50 (ug/ml)
  })
  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

    eff(0)      <- 1          # amount in the effect compartment at time 0
    d/dt(eff)   <- Kin - Kout*(1-C2/(EC50+C2))*eff    # effect compartment
  })
}
```

Yes, yes, I know we said simple. There's a lot to unpack here! This code represents a two-compartmental PK model with first-order absorption and linear elimination.



An `rxode2` model specification consists of a series of one or more statements, optionally terminated by semicolons, and comments (delimited by # and an end-of-line). Comments are also optional.

7.1.2 Model blocks

`rxode2` models are divided into two discrete blocks: `ini()`, which sets initial conditions, and `model()`, which defines the model and other aspects of the system that change with time. Blocks of statements are delimited by curly braces. The `ini()` section is pretty straightforward, as you see, with important variables in the system being assigned their starting values. You can also include a variety of other statements (such as conditionals, `while` loops and print commands) if you wish.

The `model()` block is where the action is. Here, as you might imagine, we define the model: the components of the system that change with the independent variable (often this is time). In our example, we are doing the following:

- Defining concentrations `C2` and `C3` in the central (`centr`) and peripheral (`peri`) PK compartments, respectively
- Defining the differential equations for the depot (`depot`), `centr`, `peri` and effect (`eff`) compartments
- Defining the starting amount in `eff` (1 at time 0)

7.1.3 Making statements: `rxode2` nomenclature and syntax

Before we go any further, it's probably useful to spend a bit of time talking about assignments, nomenclature and syntax in `rxode2`.

7.1.3.1 Assignments and operators

Assignment statements can be:

- *simple assignments*, in which the left-hand side is an identifier (a variable)
- *time-derivative assignments*, where the left-hand side specifies the change of the amount in the corresponding state variable (compartment) with respect to time, e.g. `d/dt(depot)`
- special *initial-condition assignments* in which the left-hand side specifies the compartment of the initial condition being specified, e.g. `depot(0) = 0`
- special *model event changes* such as bioavailability (e.g. `f(depot) = 1`), lag time (e.g. `alag(depot) = 0`), modeled rate (e.g. `rate(depot) = 2`) and modeled duration of infusion (e.g. `dur(depot) = 2`)
- special *change-point syntax*, or *modeled event times*, e.g `mtime(var) = time`
- *Jacobian-derivative assignments*, in which the left hand specifies the change in the compartment ODE with respect to a variable. For example, if `d/dt(y) = dy`, then a Jacobian for this compartment can be specified as `df(y)/dy(dy) = 1`. There may be some advantage to obtaining the solution or specifying the Jacobian for very stiff ODE systems. However, for the few stiff systems we tried with LSODA, this actually slowed things down.

Assignments can be made using `=`, `<-` or `~`. When using the `~` operator, simple assignments and time-derivative assignments will not be output.

Special statements can be:

- *Compartment declaration statements*, which can change the default dosing compartment and the assumed compartment number(s) as well as add extra compartment names at the end (useful for multiple-endpoint `nlmixr2` models); these can be specified using `cmt(compartmentName)`
- *Parameter declaration statements*, which can be used to ensure the input parameters are kept in a certain order instead of ordering the parameters by the order in which they are parsed. This is useful for keeping the parameter order the same when using different ODE models, for example (e.g. `param(par1, par2,...)`)

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

Numeric expressions can include the standard numeric operators (`+`, `-`, `*`, `/`, `^`) as well as mathematical functions defined in the C or the R math libraries (e.g. `fabs`, `exp`, `log`, `sin`, `abs`). You may also access R's math functions, like `lgammafn` for the log gamma function.

`rxode2` syntax is case-sensitive, like the rest of R: `ABC` is different from `abc`, `Abc`, `ABc`, and so forth.

7.1.3.2 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 underscore.

Identifiers in a model specification can take the following forms:

- State variables in the dynamic system (e.g. compartments in a pharmacokinetics model)
- Implied input variable, `t` (time), `tlast` (last time point), and `podo` (oral dose, in the case of absorption transit models)
- Special constants such as `pi` or R's predefined constants
- Model parameters (such as rate of absorption and clearance)
- Other left-hand side (LHS) variables created by assignments as part of the model specification

Currently, the `rxode2` modeling language only recognizes system state variables and parameters. Any values that need to be passed from R to the ODE model (such as covariates) must be passed in the `params` argument of the integrator function (`rxSolve()`) or be available in the supplied event dataset, which we'll get to in a bit.

Some variable names are reserved for use in `rxode2` event tables. The following items cannot be assigned, or used as a state, but can be accessed in the `rxode2` code:

- `cmt`: compartment
- `dvid`: dependent variable ID
- `addl`: number of additional doses
- `ss`: steady state
- `rate`: infusion rate
- `id`: unique subject identifier

The following variables, however, cannot be used in a model specification in any way:

- `evid`: event type
- `ii`: interdose interval

`rxode2` generates variables which are used internally. These variables start with an `rx` prefix. To avoid any problems, it is *strongly* suggested not to use an `rx` prefix when writing model code, since all kinds of unpleasant and unpredictable things may happen.

7.1.3.3 Logical Operators

Logical operators support the standard R operators (==, !=, >=, <=, > and <). As in R, these can be used in `if()`, `while()` and `ifelse()` expressions, as well as in standard assignments. For instance, the following is valid:

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

Notice that you can also use character expressions in comparisons. This convenience comes at a cost, however, since character comparisons are slower than numeric expressions. Unlike R, `as.numeric()` or `as.integer()` for logical statements are not only not needed, but not permitted - they will throw an error if you try to use them.

7.1.4 Interface and data handling between R and the generated C code

Users define the dynamic system's state variables via the `d/dt(identifier)` statements as part of the model specification, and model parameters via the `params` argument in the `rxode2` `solve()` method:

```
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)

solve(theta, event = qd1, inits = inits)
```

The values of these variables at pre-specified time points are saved during model fitting/integration and returned with the fitted values as part of the modeling output (see the function `eventTable()`, and in particular its member function `add.sampling()` for further information on defining a set of time points at which to capture the values of these variables).

The ODE specification mini-language is parsed with the help of the open source tool `DParser` (22).

7.1.5 Supported functions

All the supported functions in `rxode2` can be seen with the function `rxSupportedFuns()`. There are a lot of them.

7.2 Other model types

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

7.2.1 "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.

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

```
using C compiler: 'Apple clang version 15.0.0 (clang-1500.0.40.1)'
using SDK: ''
ld: warning: -single_module is obsolete
ld: warning: -multiply_defined is obsolete
```

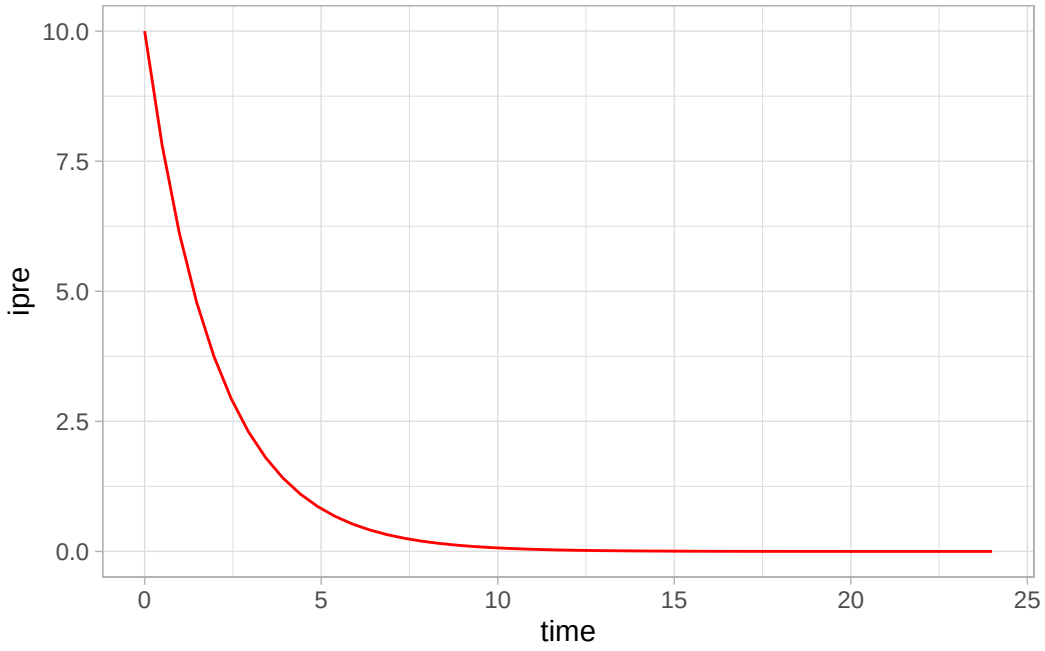
Solving prediction-only models is done just the same way as for ODE systems, but is faster.

```
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
```

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



7.2.2 Solved systems

`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();
})
```

```
using C compiler: 'Apple clang version 15.0.0 (clang-1500.0.40.1)'
using SDK: ''
ld: warning: -single_module is obsolete
ld: warning: -multiply_defined is obsolete
```

```
mod_solved
```

```
rxode2 2.0.14 model named rx_57aabcc3bef3848236fe152a3320cfae 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
```

7.2.3 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,
           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, bu
    eff(0)    = 1  ## The initial amount in the effect compartment is 1
    d/dt(eff) =  Kin - Kout*(1 - C2/(EC50 + C2))*eff;
})
```

```
mod2
```

```
rxode2 2.0.14 model named rx_7f1f252ab28eab7486f3b298b1d754d8 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.

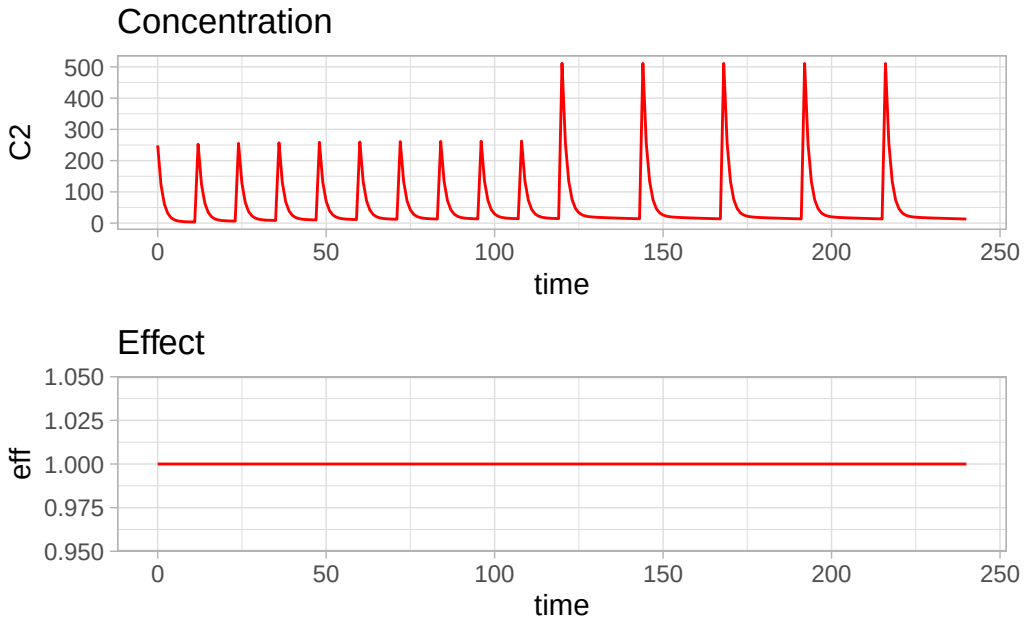
In this example, therefore, the effect compartment is compartment #1 while the PK dosing compartment for the depot is compartment #2.

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

```
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
```

```
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)
```



7.3 Compartment numbers

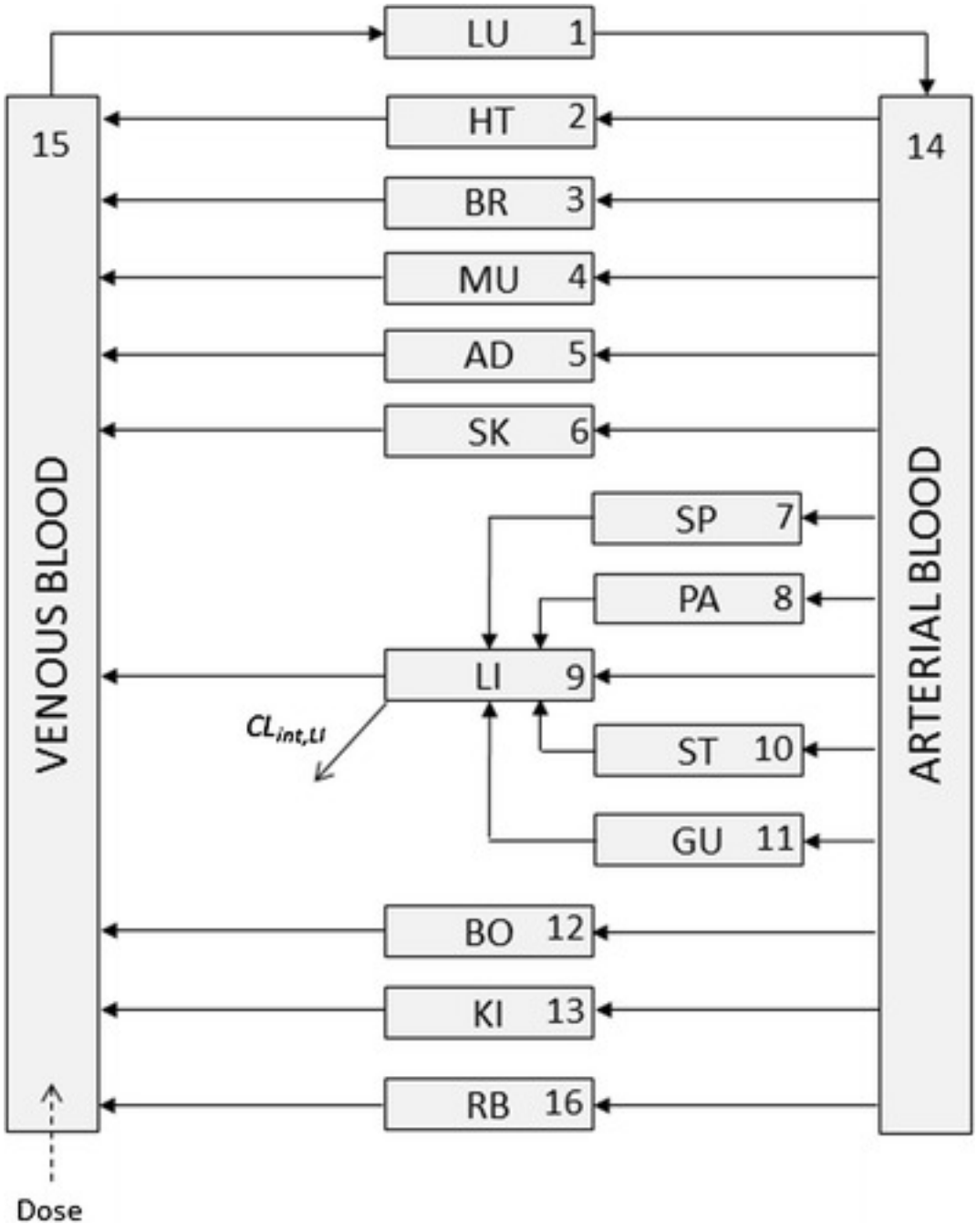
rxode2 automatically assigns compartment numbers when parsing. For illustrative purposes, elt's look at something a bit more compex than we have so far: a PBPK model for mavoglurant published by Wendling and colleagues (23).

```
library(rxode2)

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
14
VENOUS BLOOD
ARTERIAL BLOOD
$CL_{int,LI}$
Dose

7.3.1 How `rxode2` assigns compartment numbers

```
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.

7.3.2 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:

```
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.

7.3.3 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:

```
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:

```
mod2$simulationModel
```

```
using C compiler: 'Apple clang version 15.0.0 (clang-1500.0.40.1)'
using SDK: ''
ld: warning: -single_module is obsolete
ld: warning: -multiply_defined is obsolete

rxode2 2.0.14 model named rx_beefd30f33bb9cd4324bb242420aa74f 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
})
```

7.4 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 (24,25). `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).

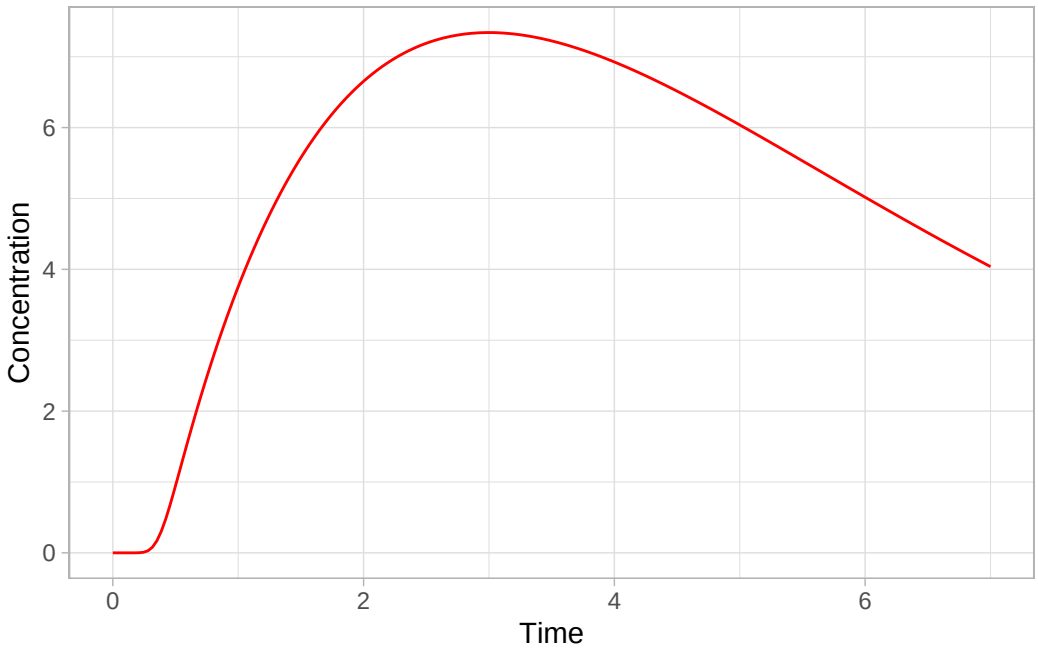
```
mod <- function() {
  ini({
    ## Table 3 from Savic 2007
    cl  <- 17.2 # (L/hr)
    vc  <- 45.1 # L
    ka  <- 0.38 # 1/hr
    mtt <- 0.37 # hr
    bio <- 1
    n   <- 20.1
  })
  model({
    k           <- cl/vc
    ktr         <- (n+1)/mtt
    d/dt(depot) <- transit(n,mtt,bio)-ka*depot
   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

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

```
i parameter labels from comments will be replaced by 'label()'
```

```
using C compiler: 'Apple clang version 15.0.0 (clang-1500.0.40.1)'
using SDK: ''
ld: warning: -single_module is obsolete
ld: warning: -multiply_defined is obsolete
```

```
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 (25).
- 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)

7.5 Covariates

7.6 Multiple subjects

7.7 Working with `rxode2` output

7.8 Piping

7.9 Special cases

7.9.1 Jacobian solving

7.9.2 `rxode2` models in `shiny`

7.9.3 Precompiled `rxode2` models in other R packages

8 Simulating populations

8.1 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:

```
library(rxode2)
```

```
rxode2 2.0.14 using 1 threads (see ?getRxThreads)
  no cache: create with `rxCreateCache()`
```

```
========================================
rxode2 has not detected OpenMP support and will run in single-threaded mode
This is a Mac. Please read https://mac.r-project.org/openmp/
========================================
```

```
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 4. 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 6, 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.

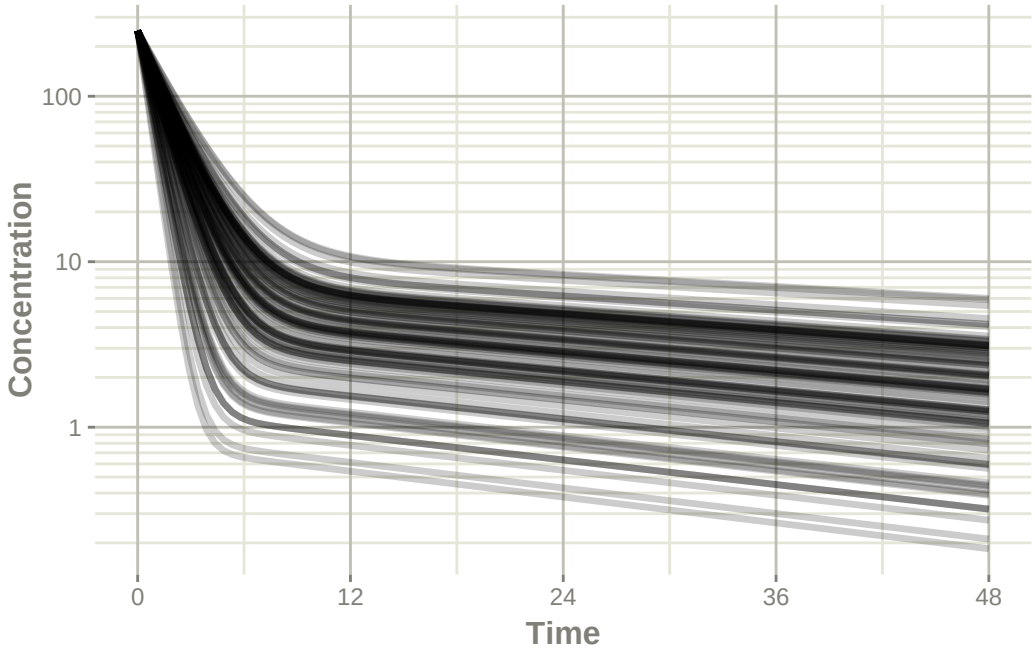
```
sim  <- rxSolve(mod, ev, nSub=100)
```

```
i parameter labels from comments will be replaced by 'label()'
```

```
using C compiler: 'Apple clang version 15.0.0 (clang-1500.0.40.1)'
using SDK: ''
ld: warning: -single_module is obsolete
ld: warning: -multiply_defined is obsolete
```

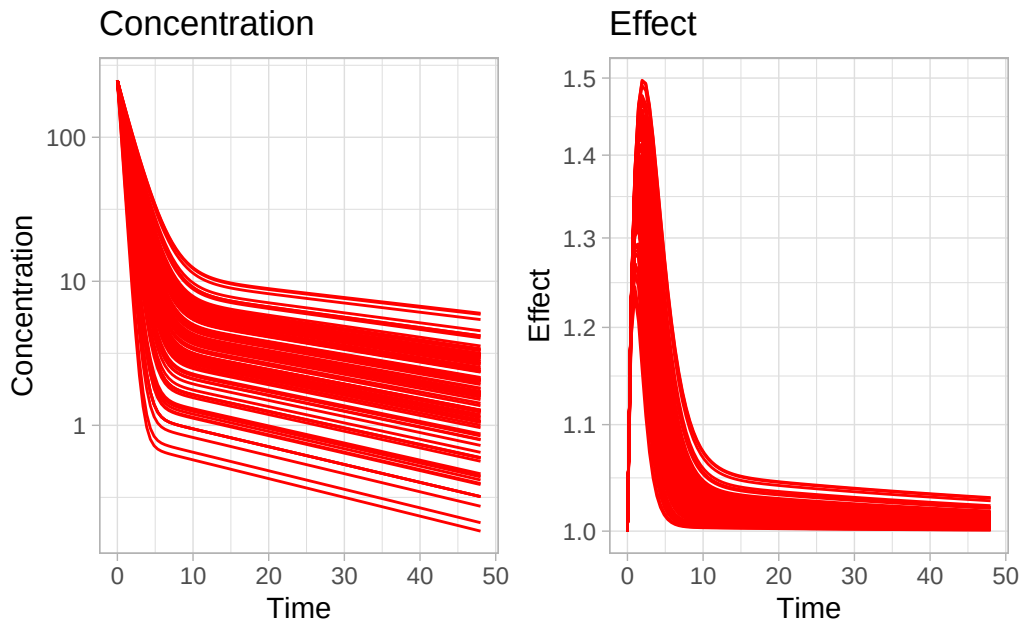
To look at the results quickly, you can use the built-in `plot` routine. This will create a `ggplot2` object that you can modify as you wish using standard `ggplot2` syntax. The extra parameter we've supplied to `plot` clarifies the piece of information we are interested in plotting. In this case, it's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

```
class(sim)
```

```
[1] "rxSolve"        "rxSolveParams"  "rxSolveCovs"    "rxSolveInits"
[5] "rxSolveSimType" "data.frame"
attr(,".rxode2.env")
<environment: 0x121631c30>
```

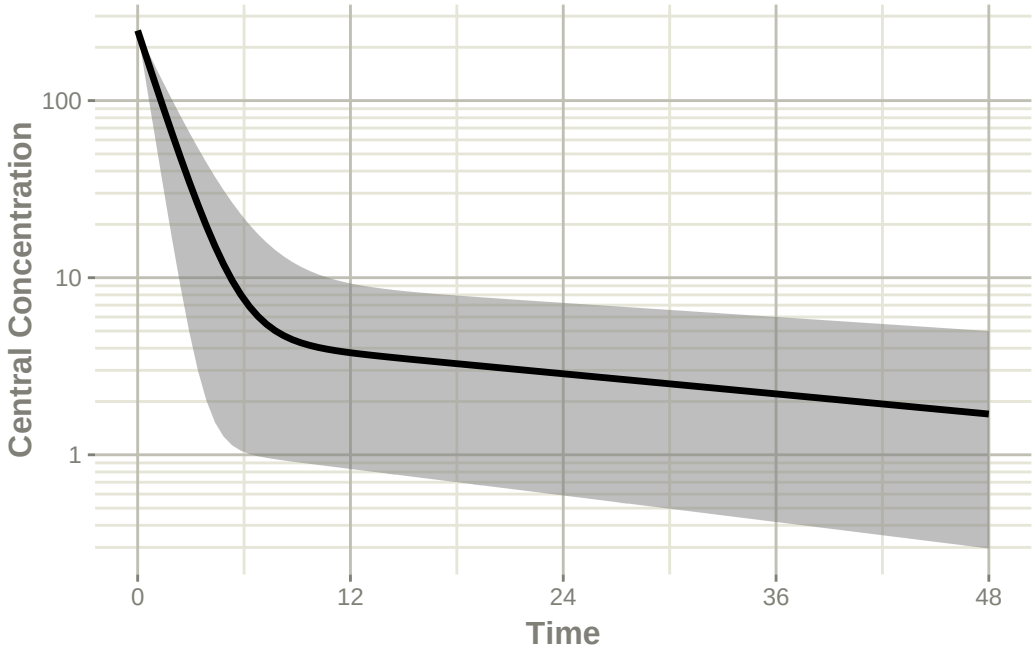
```
head(sim)
```

```
  sim.id      time        C2        C3       CL depot     centr     peri
1      1 0.0000000 248.75622  0.000000 13.54257     0 10000.000    0.000
2      1 0.4848485 186.36159  3.669665 13.54257     0  7491.736 1089.891
3      1 0.9696970 140.01713  6.360099 13.54257     0  5628.689 1888.949
4      1 1.4545455 105.59032  8.323759 13.54257     0  4244.731 2472.156
5      1 1.9393939  80.01268  9.748097 13.54257     0  3216.510 2895.185
6      1 2.4242424  61.00582 10.772297 13.54257     0  2452.434 3199.372
       eff
1 1.000000
2 1.222877
3 1.359259
4 1.422944
5 1.432400
6 1.406572
```

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.

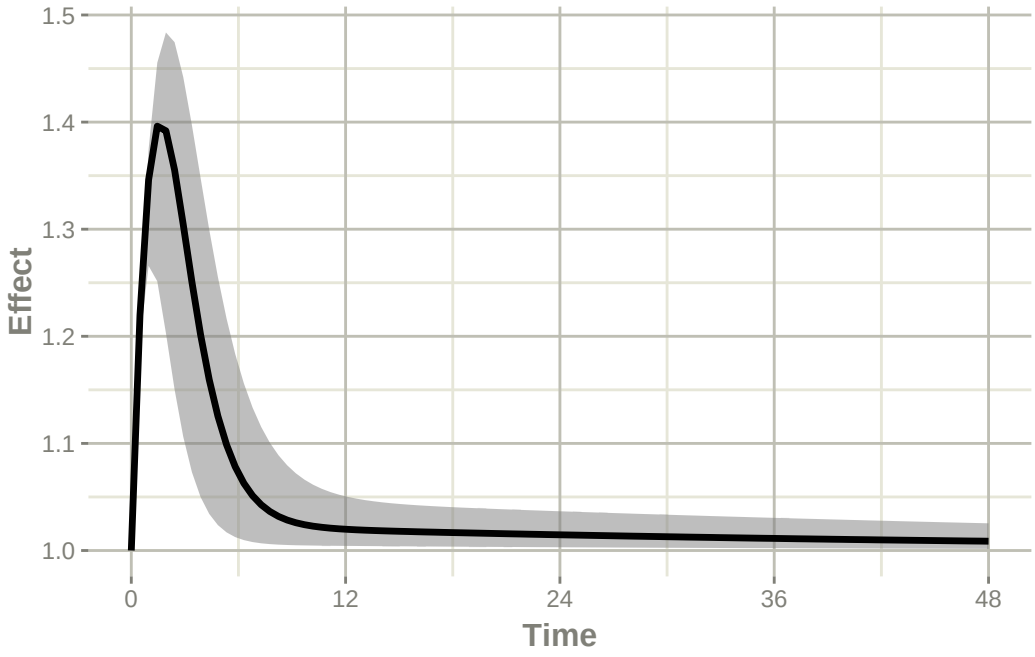
```
confint(sim, "C2", level=0.95) %>%
    plot(ylab="Central Concentration", log="y")
```

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

summarizing data...done
```




```
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
```



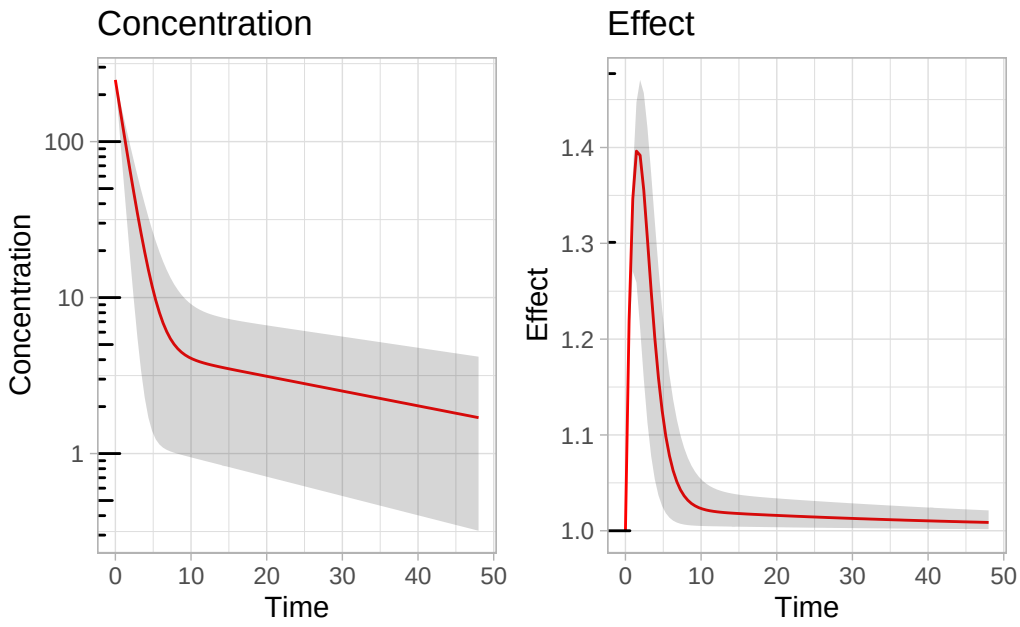
This is a shortcut for this slightly longer code:

```
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
```

```
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.

```
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 -0.3173233
2      2 0.294 18.6 40.2 10.5 297   1    1  200 -0.6963040
3      3 0.294 18.6 40.2 10.5 297   1    1  200 -0.1187198
4      4 0.294 18.6 40.2 10.5 297   1    1  200 -0.4707693
5      5 0.294 18.6 40.2 10.5 297   1    1  200  0.2856009
6      6 0.294 18.6 40.2 10.5 297   1    1  200 -0.2746254
```

8.2 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 concen
  ini(eff.sd=sqrt(0.1), prop.sd=sqrt(0.1))
```

```
i parameter labels from comments will be replaced by 'label()'

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)
```

```
using C compiler: 'Apple clang version 15.0.0 (clang-1500.0.40.1)'
using SDK: ''
ld: warning: -single_module is obsolete
ld: warning: -multiply_defined is obsolete
```

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$params): --
# A tibble: 100 x 12
   sim.id    KA   TCl    V2     Q    V3   Kin  Kout  EC50 eff.sd prop.sd  eta.Cl
    <int> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl>   <dbl>   <dbl>
 1      1 0.294  18.6  40.2  10.5   297     1     1   200  0.316   0.316 -0.265
 2      2 0.294  18.6  40.2  10.5   297     1     1   200  0.316   0.316 -0.198
 3      3 0.294  18.6  40.2  10.5   297     1     1   200  0.316   0.316 -0.228
 4      4 0.294  18.6  40.2  10.5   297     1     1   200  0.316   0.316  0.311
 5      5 0.294  18.6  40.2  10.5   297     1     1   200  0.316   0.316 -0.597
 6      6 0.294  18.6  40.2  10.5   297     1     1   200  0.316   0.316 -0.0645
 7      7 0.294  18.6  40.2  10.5   297     1     1   200  0.316   0.316  0.149
 8      8 0.294  18.6  40.2  10.5   297     1     1   200  0.316   0.316  0.138
 9      9 0.294  18.6  40.2  10.5   297     1     1   200  0.316   0.316 -0.923
10     10 0.294  18.6  40.2  10.5   297     1     1   200  0.316   0.316 -0.135
# i 90 more rows
-- Initial Conditions (x$inits): --
depot centr  peri   eff
    0     0     0     1

Simulation without uncertainty in parameters, omega, or sigma matricies
```

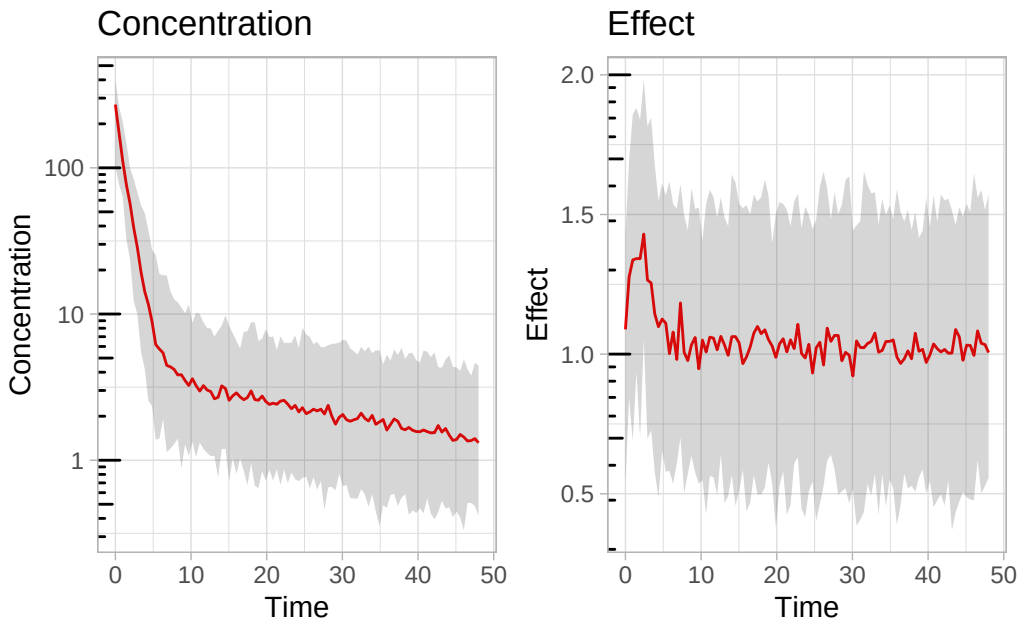
```
-- First part of data (object): --
# A tibble: 20,000 x 12
  sim.id  time    C2    C3    CL ipredSim     sim depot  centr  peri   eff   CMT
   <int> <dbl> <dbl> <dbl> <dbl>    <dbl>   <dbl> <dbl>  <dbl> <dbl> <dbl> <dbl>
1      1 0      249.  0     14.3     1      0.538     0 10000     0   1         4
2      1 0      249.  0     14.3   249.   138.        0 10000     0   1         5
3      1 0.485  185.  3.65  14.3     1.22   1.69      0  7426. 1085.  1.22      4
4      1 0.485  185.  3.65  14.3   185.   174.        0  7426. 1085.  1.22      5
5      1 0.970  138.  6.31  14.3     1.36   1.13      0  5530. 1874.  1.36      4
6      1 0.970  138.  6.31  14.3   138.   209.        0  5530. 1874.  1.36      5
# i 19,994 more rows
```

```
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.
```

```
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
```



8.3 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.

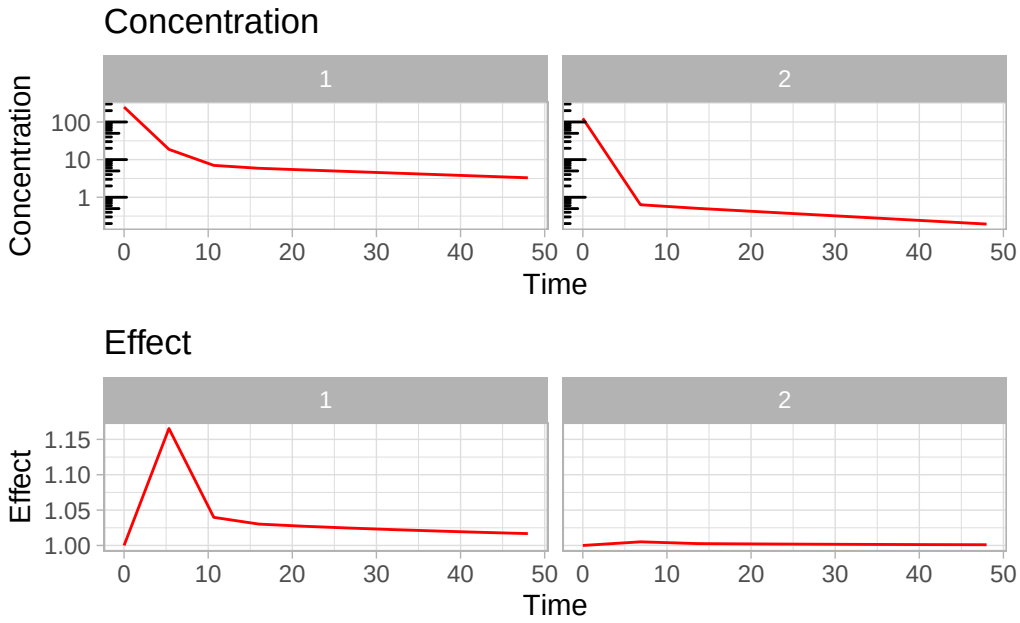
```
library(dplyr)
ev1 <- et(amountUnits="mg", timeUnits="hours") %>%
    et(amt=10000, cmt=2) %>%
    et(0,48,length.out=10)
```

```
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 will be replaced by 'label()'
```

```
using C compiler: ‘Apple clang version 15.0.0 (clang-1500.0.40.1)’
using SDK: ‘’
ld: warning: -single_module is obsolete
ld: warning: -multiply_defined is obsolete
```

```
#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.

8.4 Simulating clinical trials

By either using a simple single event table, or data from a clinical trial as described above, a complete clinical trial simulation can be performed.

Typically in clinical trial simulations you want to account for the uncertainty in the fixed parameter estimates, and even the uncertainty in both your between subject variability as well as the unexplained variability.

`rxode2` allows you to account for these uncertainties by simulating multiple virtual "studies," specified by the parameter `nStud`. Each of these studies samples a realization of fixed effect parameters and covariance matrices for the between subject variability(`omega`) and unexplained variabilities (`sigma`). Depending on the information you have from the models, there are a few strategies for simulating a realization of the `omega` and `sigma` matrices.

The first strategy occurs when either there is not any standard errors for standard deviations (or related parameters), or there is a modeled correlation in the model you are simulating from. In that case the suggested strategy is to use the inverse Wishart (parameterized to scale

to the conjugate prior)/scaled inverse chi distribution. this approach uses a single parameter to inform the variability of the covariance matrix sampled (the degrees of freedom).

The second strategy occurs if you have standard errors on the variance/standard deviation with no modeled correlations in the covariance matrix. In this approach you perform separate simulations for the standard deviations and the correlation matrix. First you simulate the variance/standard deviation components in the `thetaMat` multivariate normal simulation. After simulation and transformation to standard deviations, a correlation matrix is simulated using the degrees of freedom of your covariance matrix. Combining the simulated standard deviation with the simulated correlation matrix will give a simulated covariance matrix. For smaller dimension covariance matrices (dimension $<$ 10x10) it is recommended you use the `lkj` distribution to simulate the correlation matrix. For higher dimension covariance matrices it is suggested you use the inverse wishart distribution (transformed to a correlation matrix) for the simulations.

The covariance/variance prior is simulated from `rxode2`s `cvPost()` function.

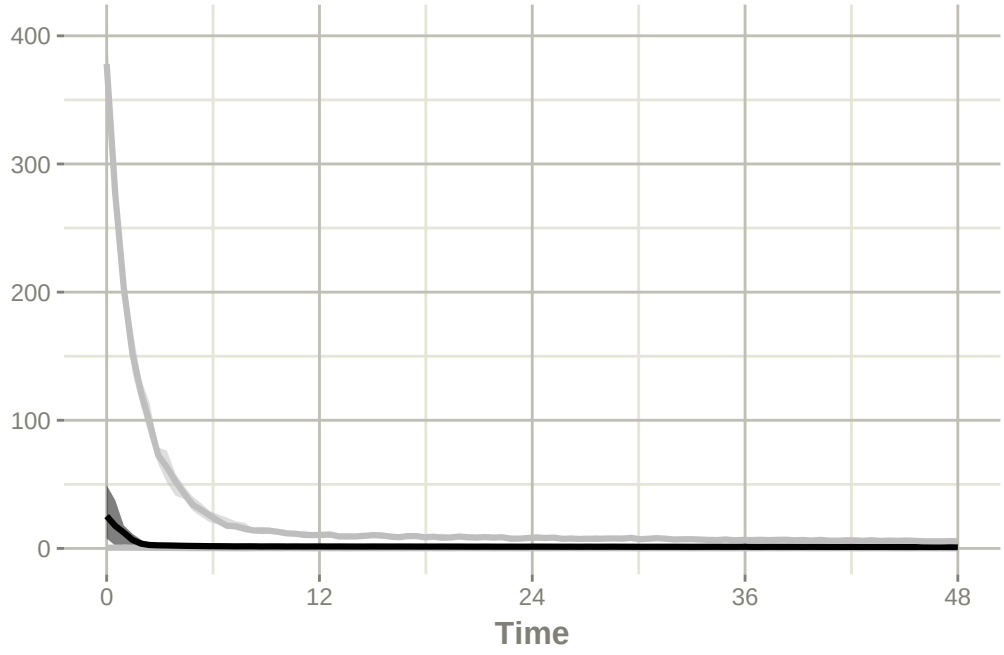
8.5 Simulation from inverse Wishart correlations

An example of this simulation is below:

```
## Creating covariance matrix
tmp <- matrix(rnorm(8^2), 8, 8)
tMat <- tcrossprod(tmp, tmp) / (8 ^ 2)
dimnames(tMat) <- list(NULL, names(mod2$theta)[1:8])

sim  <- rxSolve(mod2, ev, nSub=100, thetaMat=tMat, nStud=10,
                dfSub=10, dfObs=100)

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

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

```
plot(s)
```



If you wish you can see what `omega` and `sigma` was used for each virtual study by accessing them in the solved data object with `$omega.list` and `$sigma.list`:

```
head(sim$omegaList)
```

```
[[1]]
          eta.Cl
eta.Cl 0.2123854

[[2]]
          eta.Cl
eta.Cl 0.2439259

[[3]]
          eta.Cl
eta.Cl 0.3853127

[[4]]
          eta.Cl
eta.Cl 0.1724243

[[5]]
```

```
          eta.Cl
eta.Cl 0.2478496

[[6]]
          eta.Cl
eta.Cl 0.1835147
```

```
head(sim$sigmaList)
```

```
[[1]]
            err.eff      err.C2
err.eff  1.14300092 -0.02522627
err.C2  -0.02522627  0.91792638

[[2]]
            err.eff      err.C2
err.eff  1.03843335 -0.02825221
err.C2  -0.02825221  0.89033230

[[3]]
          err.eff    err.C2
err.eff 1.1376544 0.2073658
err.C2  0.2073658 1.1247929

[[4]]
           err.eff     err.C2
err.eff  1.0465015 -0.0832682
err.C2  -0.0832682  0.9850565

[[5]]
           err.eff     err.C2
err.eff  1.2164896 -0.1675439
err.C2  -0.1675439  1.4092104

[[6]]
           err.eff     err.C2
err.eff 0.86976132 0.01196649
err.C2  0.01196649 0.99729356
```

You can also see the parameter realizations from the `$params` data frame.

8.6 Simulate using variance/standard deviation standard errors

Lets assume we wish to simulate from the nonmem run included in xpose

First we setup the model; Since we are taking this from nonmem and would like to use the more free-form style from the classic `rxode2` model we start from the classic model:

```
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
})
```

```
using C compiler: 'Apple clang version 15.0.0 (clang-1500.0.40.1)'
using SDK: ''
ld: warning: -single_module is obsolete
ld: warning: -multiply_defined is obsolete
```

Next we input the estimated parameters:

```
theta <- c(tcl=2.63E+01, tv=1.35E+00, tka=4.20E+00, tlag=2.08E-01,
           prop.sd=2.05E-01, add.sd=1.06E-02, crcl.cl=7.17E-03,
           ## Note that since we are using the separation strategy the ETA variances are h
           eta.cl=7.30E-02,  eta.v=3.80E-02, eta.ka=1.91E+00)
```

And also their covariances; To me, the easiest way to create a named covariance matrix is to use `lotri()`:

```
thetaMat <- 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
          -1.23E-03, -1.27E-05, -1.33E-03, -1.47E-05, -1.03E-04, 1.02E-05, 1.67E-06, 6.68E
          7.69E-02, -7.23E-03, 3.74E-01, 1.79E-03, -2.85E-03, 1.18E-05, -2.54E-04, 1.61E-0
```

```
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)

## From the run we know that:
##   total number of observations is: 476
##    Total number of individuals:     74
sim  <- rxSolve(rx1, theta, evw,  nSub=100, nStud=10,
                thetaMat=thetaMat,
                ## Match boundaries of problem
                thetaLower=0,
                sigma=c("prop.sd", "add.sd"), ## Sigmas are standard deviations
                sigmaXform="identity", # default sigma xform="identity"
                omega=c("eta.cl", "eta.v", "eta.ka"), ## etas are variances
                omegaXform="variance", # default omega xform="variance"
                iCov=data.frame(WT=rnorm(1000, 70, 15), CLCR=rnorm(1000, 65, 25)),
                dfSub=74, dfObs=476);
```

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

```
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      26.5   0.162 -0.449   1.40  0.0562  4.64  1.09
 2      1 2      26.5   0.162 -0.586   1.40  0.557   4.64  0.754
 3      1 3      26.5   0.162 -1.13    1.40 -0.511   4.64  0.382
 4      1 4      26.5   0.162  0.393   1.40  0.275   4.64 -0.240
 5      1 5      26.5   0.162  2.61    1.40  1.01    4.64  1.30
 6      1 6      26.5   0.162 -0.0363  1.40  0.932   4.64  0.466
 7      1 7      26.5   0.162  0.487   1.40  0.976   4.64 -1.12
```

```
 8      1 8      26.5   0.162 -0.544   1.40  0.0459  4.64 -0.642
 9      1 9      26.5   0.162 -0.540   1.40 -0.209   4.64  0.861
10      1 10     26.5   0.162  2.47    1.40 -1.78    4.64 -0.496
# 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 0.118  10.1  118. 13.8  0.678 -0.371   79.7  62.5
2      1     1 0.579  10.1  118. 13.8  0.813  1.65    79.7  62.5
3      1     1 2.78   10.1  118. 13.8  0.673  3.16    79.7  62.5
4      1     1 5.09   10.1  118. 13.8  0.553  0.987   79.7  62.5
5      1     1 8.88   10.1  118. 13.8  0.399 -0.0103  79.7  62.5
6      1     2 0.232  21.8  179.  9.86 0.493  0.227   73.5  68.0
# i 49,994 more rows
```

```
## Notice that the simulation time-points change for the individual

## If you want the same sampling time-points you can do that as well:
evw <- et(amount.units="mg", time.units="hours") %>%
    et(amt=100) %>%
    et(0, 24, length.out=50) %>%
    et(id=1:100)

sim  <- rxSolve(rx1, theta, evw,  nSub=100, nStud=10,
                thetaMat=thetaMat,
                ## Match boundaries of problem
                thetaLower=0,
                sigma=c("prop.sd", "add.sd"), ## Sigmas are standard deviations
                sigmaXform="identity", # default sigma xform="identity"
                omega=c("eta.cl", "eta.v", "eta.ka"), ## etas are variances
                omegaXform="variance", # default omega xform="variance"
                iCov=data.frame(WT=rnorm(100, 70, 15), CLCR=rnorm(100, 65, 25)),
                dfSub=74, dfObs=476,
```

```
                  resample=TRUE)
```

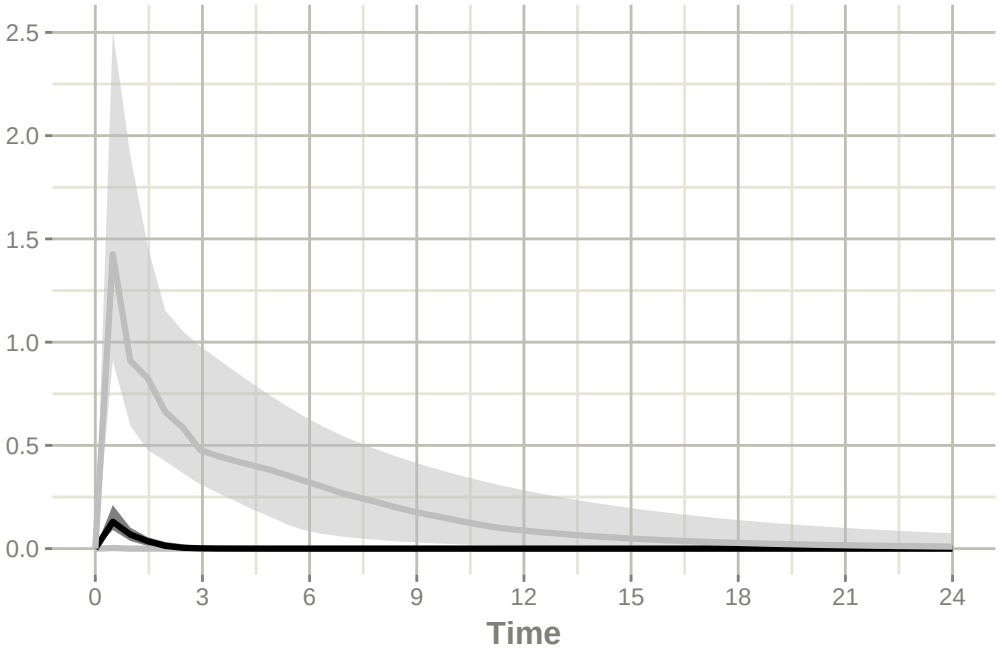
```
i thetaMat has too many items, ignored: 'tlag'
```

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

```
summarizing data...
```

```
done
```

```
  plot(s)
```



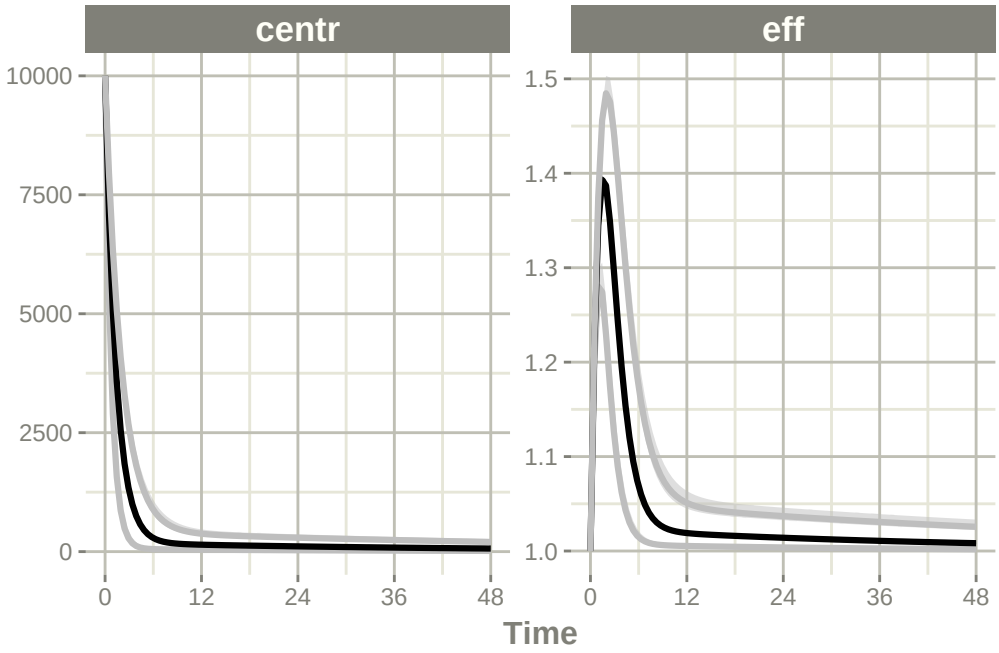
8.7 Simulate without uncertainty in `omega` or `sigma` parameters

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(mod2, ev, nSub=100, thetaMat=tMat, nStud=10,
                simVariability=FALSE)

s <-sim %>% confint(c("centr", "eff"))
```

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

```
plot(s)
```



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

9 More complex examples

Part III

Modeling with `nlmixr2`

10 Parameter estimation and model fitting

11 Model diagnostics and evaluation

12 Handling Covariate Effects

13 Model Selection and Comparison

14 Case Studies and Practical Applications

15 Tracking work with `shinyMixR`

Part IV

Advanced topics

16 Inside the mixrverse

17 Extending the mixrverse

Part V

The mixrverse

18 Qualifying the mixrverse

19 babelmixr

20 Into the mixrverse

Part VI

The future

21 Future Directions

22 Conclusion and Final Thoughts

23 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. 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.

19. 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

20. Bates D, Mächler M, Bolker B, Walker S. Fitting linear mixed-effects models using lme4. Journal of Statistical Software. 2015;67(1):1–48.

21. Karlsson MO, Sheiner LB. The importance of modeling interoccasion variability in population pharmacokinetic analyses. Journal of Pharmacokinetics and Biopharmaceutics. 1993;21(6):735–50.

22. Fidler M, Plevyak J. Dparser: Port of 'dparser' package. 2023.

23. 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

24. 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.

25. 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.