Introduction to shinyMixR

Justin J. Wilkins
Occams, Amstelveen, The Netherlands

Richard Hooijmaaijers
LAP&P, Tilburg, The Netherlands

and the nlmixr Development Team

Introduction

- The **shinyMixR** package is a graphical user interface (GUI) tool for managing population PKPD projects with **nlmixr2** as the estimation engine
- The package is mainly intended to view, edit, run, compare, analyse and report nlmixr2 models
- The interface is created using the **shiny** and **shinydasboard** packages
- The interface operates within a specific project folder
- Most functions within the package can also be used in an interactive R session
- The package is open source and is available at: https://github.com/RichardHooijmaijers/shinyMixR

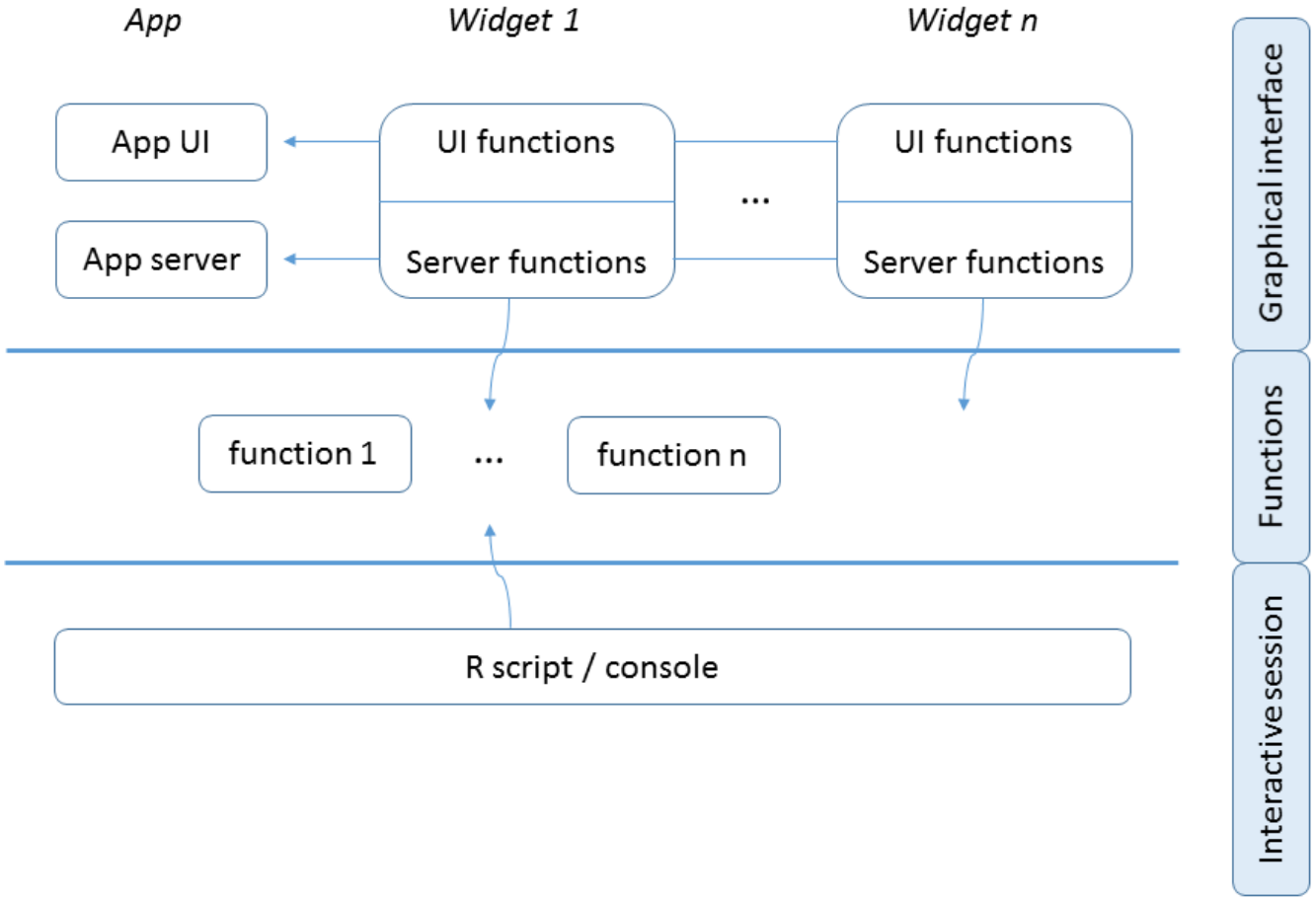
Introduction

shinyMixR is only a shell around nlmixr2 and needs the package to fully operate:

- Running models is done using the **`nlmixr2`** package (indirectly)
- Plotting is done using the **`xpose.nlmixr2`** package (or **`ggplot2`**)

- Managing is done using the **`DT`** and **`collapsibleTree`** packages
- Editing is done using the **`shinyAce`** package
- Reporting is done using the **`R3port`** package

Package structure



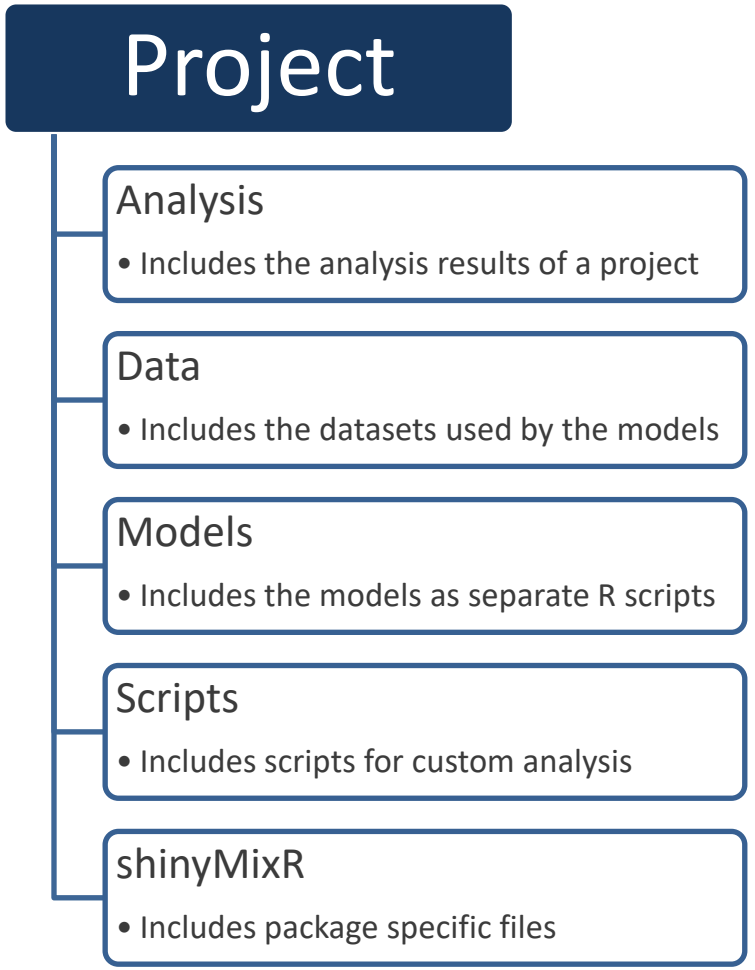
- The interface is built using ui/server scripts as done in other **shiny** apps
- Due to the size of the dashboard, widget *modules* were created that are used by ui/server
- More generic functions are available that are used by the interface as well as interactive usage

Project structure

The folder structure of a **shinyMixR** project is fixed and should be followed to enable to work with the package*:

- The main script(s) to start the application or work interactively is located in the project root
- This structure is important because:
  1. The package needs to monitor changes in specific folder and keep track of this in a project object
  2. Files are read and saved from specific locations
  3. When working with many models an organized folder structure is key

*functionality to build the folder structure is present in the package



Project Object

To manage the information within the project structure, a project object is maintained:

- This object has information regarding the available models, meta data, high level results, etc.
- All changes within a project are monitored/saved in this object:
  - Model is changed
  - New results are generated
  - Data is deleted
- Within the interface this is done automatically, or using refresh buttons
  (e.g. the interface does not “know” when a model is finished and new results are present)
- For an interactive session, in some cases updating of this object can be done using the `get_proj` function

Differences between interactive session and interface

Functionality available in <u>both</u>

- View overview of available models
- View hierarchical overview
- Run models externally
- Create parameter table
- Create GOF plots
- Create fit plots

Functionality only available in <u>interface</u>

- Export overview
- Adapt model meta data
- Edit, duplicate, create model
- Show progress of model runs
- Combine analysis results in report
- Run user created R template script

- Functionality only available in interface can be done in many cases indirectly in an interactive session as well
- It is more user-friendly/quicker to perform certain tasks using the interface
- It is easy to switch between interface and interactive session

Workflow – run nlmixr

```
run1 <- function() {
  ini({
    tka <- .5
    tcl <- -3.2
    tv  <- -1
    eta.ka ~ 1
    eta.cl ~ 2
    eta.v ~ 1
    add.err <- 0.1
  })
  model({
    ka <- exp(tka + eta.ka)
    cl <- exp(tcl + eta.cl)
    v  <- exp(tv + eta.v)
    linCmt() ~ add(add.err)
  })
}

dat <- read.csv("data/data.csv")

fit <- nlmixr(run1, dat, est="saem")
```

A model can be directly run in nlmixr2:

1. Define model.
2. Import, create and/or adapt the required data.
3. Run the model.

Workflow – model diagnostics

```
print(fit)
plot(fit)



xpdb <- xpose_data_nlmixr(fit)

dv_vs_idv(xpdb)
ipred_vs_idv(xpdb)
pred_vs_idv(xpdb)
dv_preds_vs_idv(xpdb)
dv_vs_pred(xpdb)
dv_vs_ipred(xpdb)
res_vs_idv(xpdb, res = 'CWRES')
res_vs_pred(xpdb, res = 'CWRES')
```

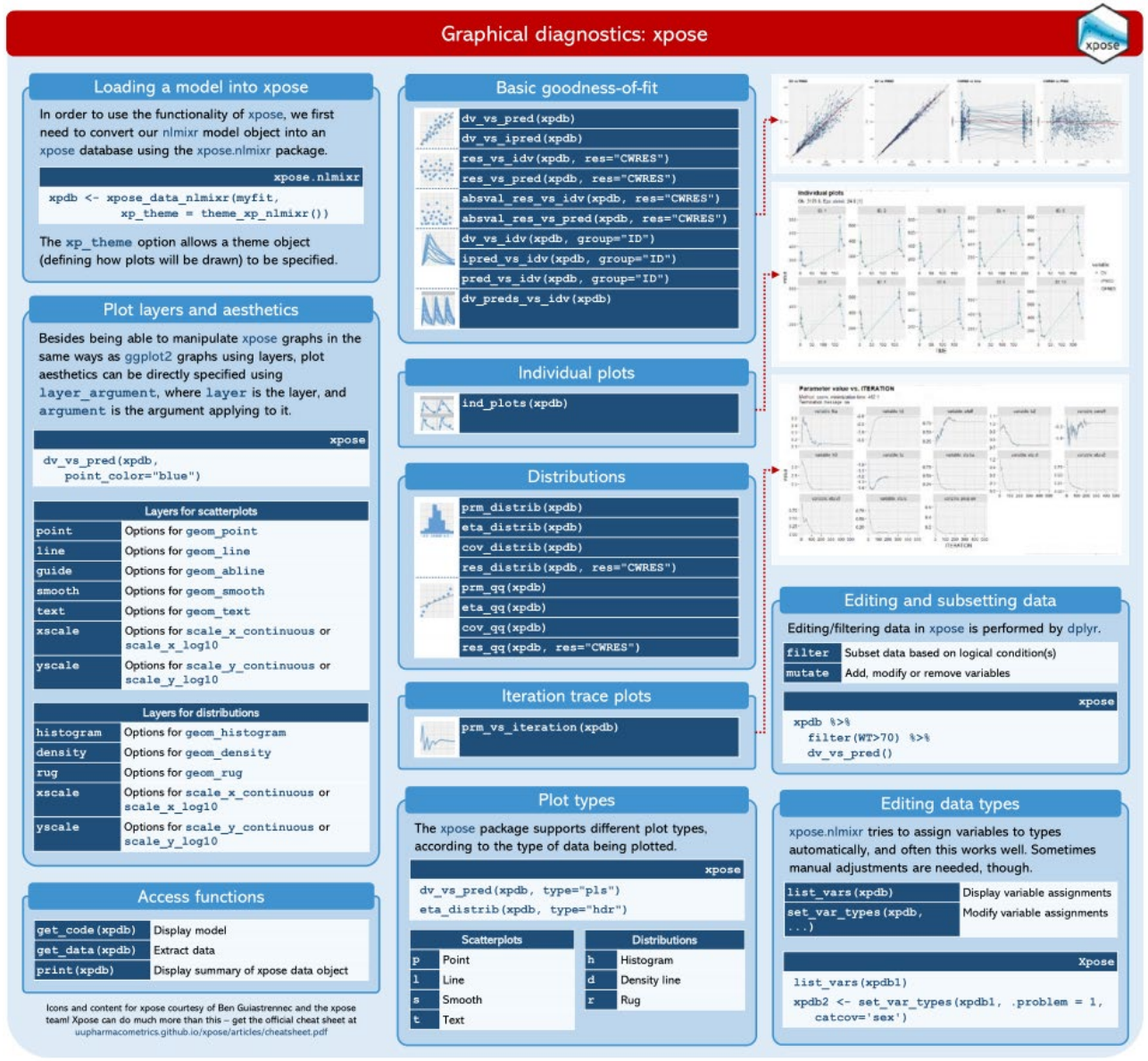
High level results can be printed in console and default plots can be created using **nlmixr2**.

More elaborate plots can be generated using the **xpose.nlmixr2** package

Most important function is **xpose_data_nlmixr** which transforms the **nlmixr** output to **xpose** format

Subsequently almost all **xpose** functions can be used to create results for **nlmixr** output
Only a few functions are shown here, for more examples see:
https://uupharmacometrics.github.io/xpose/index.html

Workflow – model diagnostics



Graphical diagnostics: xpose

Loading a model into xpose

In order to use the functionality of xpose, we first need to convert our nlmixr model object into an xpose database using the xpose.nlmixr package.

xpose.nlmixr

```
xpdb <- xpose_data_nlmixr(myfit,
          xp_theme = theme_xp_nlmixr())
```

The xp_theme option allows a theme object (defining how plots will be drawn) to be specified.

Plot layers and aesthetics

Besides being able to manipulate xpose graphs in the same ways as ggplot2 graphs using layers, plot aesthetics can be directly specified using layer_argument, where layer is the layer, and argument is the argument applying to it.

xpose

```
dv_vs_pred(xpdb,
  point_color="blue")
```

Layers for scatterplots	
point	Options for geom_point
line	Options for geom_line
guide	Options for geom_abline
smooth	Options for geom_smooth
text	Options for geom_text
xscale	Options for scale_x_continuous or scale_x_log10
yscale	Options for scale_y_continuous or scale_y_log10

Layers for distributions	
histogram	Options for geom_histogram
density	Options for geom_density
rug	Options for geom_rug
xscale	Options for scale_x_continuous or scale_x_log10
yscale	Options for scale_y_continuous or scale_y_log10

Access functions

get_code(xpdb)	Display model
get_data(xpdb)	Extract data
print(xpdb)	Display summary of xpose data object

Icons and content for xpose courtesy of Ben Guiastrennec and the xpose team! Xpose can do much more than this – get the official cheat sheet at uupharmacometrics.github.io/xpose/articles/cheatsheet.pdf

Basic goodness-of-fit

```
dv_vs_pred(xpdb)
dv_vs_ipred(xpdb)
res_vs_idv(xpdb, res="CWRES")
res_vs_pred(xpdb, res="CWRES")
absval_res_vs_idv(xpdb, res="CWRES")
absval_res_vs_pred(xpdb, res="CWRES")
dv_vs_idv(xpdb, group="ID")
ipred_vs_idv(xpdb, group="ID")
pred_vs_idv(xpdb, group="ID")
dv_preds_vs_idv(xpdb)
```

Individual plots

```
ind_plots(xpdb)
```

Distributions

```
prm_distrib(xpdb)
eta_distrib(xpdb)
cov_distrib(xpdb)
res_distrib(xpdb, res="CWRES")
prm_qq(xpdb)
eta_qq(xpdb)
cov_qq(xpdb)
res_qq(xpdb, res="CWRES")
```

Iteration trace plots

```
prm_vs_iteration(xpdb)
```

Plot types

The xpose package supports different plot types, according to the type of data being plotted.

xpose

```
dv_vs_pred(xpdb, type="pls")
eta_distrib(xpdb, type="hdr")
```

Scatterplots	
p	Point
l	Line
s	Smooth
t	Text

Distributions	
h	Histogram
d	Density line
r	Rug

Editing and subsetting data

Editing/filtering data in xpose is performed by dplyr.

filter	Subset data based on logical condition(s)
mutate	Add, modify or remove variables

xpose

```
xpdb %>%
  filter(WT>70) %>%
  dv_vs_pred()
```

Editing data types

xpose.nlmixr tries to assign variables to types automatically, and often this works well. Sometimes manual adjustments are needed, though.

list_vars(xpdb)	Display variable assignments
set_var_types(xpdb, ...)	Modify variable assignments

Xpose

```
list_vars(xpdb1)
xpdb2 <- set_var_types(xpdb1, .problem = 1,
  catcov='sex')
```

Elaborate information on how to use the **`xpose.nlmixr2`** package is available in the cheat sheet…

Workflow – shinyMixR

```
# show first part of model
cat(readLines("models/run1.r")[1:7],sep="\n")

run1 <- function() {
  data = "theo_sd"
  desc = "base model"
  ref  = ""
  imp  = 1
  est  = "nlme"
  control<-list()
  ini({
    tka <- .5
    tcl <- -3.2  ...

# command line
run_nmx("run1")
res <- readRDS("shinyMixR/run1.res.rds")
gof_plot(fit)
fit_plot(res,type="user")


# interface
run_shinymixr()
```

The model is defined in a separate file and includes meta data used by the package

A model is submitted by default in a separate R session. Plot functions are available to create **xpose.nlmixr2** or **ggplot2** type plots

The interface can be started using a single function

shinyMixR

Cheat Sheet

github.com/richardhooijmaijers/shinyMixR

Installation

Before installation, make sure you have R 3.4.1 or higher and at least the `nlmixr2` package installed and tested.

Installation of the package should be done using devtools:

```
install.packages('richardhooijmaijers/shinyMixR')
```

The package is based on the `shiny` and `bs4Dash` packages. Additional packages are off course `nlmixr2`, and for `xpose` type plotting `xpose.nlmixr2`.

Introduction

The shinyMixR package is developed as a model management tool for the nlmixr package. The package include a shiny (dashboard) interface but can also be used in an interactive R session. The package aims to help in managing, running, editing and analysing nlmixr2 models.

Folder Structure

A specific folder structure for a project is required and used by the package:

- **Project**
  - Analysis
    - Includes the analysis results of a project
  - Data
    - Includes the datasets used by the models
  - Models
    - Includes the models as separate R scripts
  - Scripts
    - Includes scripts for custom analysis
  - shinyMixR
    - Includes package specific files

A folder structure can be created including example data, models and scripts using:

```
create_proj()
```

This is the starting point for a project – you have to do this once per project.

Project Structure

Information regarding a project is maintained in the *project object*. This is a list within R with the following structure:

```
Object
|
|--- run 1
|     |--- model location
|     |--- model metadata
|     |--- model high level results
|
|--- run n
|
|--- general meta information
```

In case new models or results are available this is added to the object using the function `get_proj()`. When running interactively this function might be submitted to reflect the latest changes/results. The function will only search for newly created files. Within the interface this is done automatically or using the refresh button.

Interactive usage

For interactive usage, the most important functions are:

`create_proj()`	Create a folder structure for a `shinyMixR` project.
`run_nmx()`	Run a `nlmixr` model, possibly in a separate R session to overcome "freezing" of current session.
`overview()`	Create overview of all models in a project.
`tree_overview()`	Create a collapsible tree overview for visualizing relationship between models.
`par_table()`	Create dense parameter table for one or multiple models.
`gof_plot()`	Create a combination of most important goodness of fit plots.
`fit_plot()`	Create individual fit plots.
`get_proj()`	Get project information with available models and high level results.

Interface usage

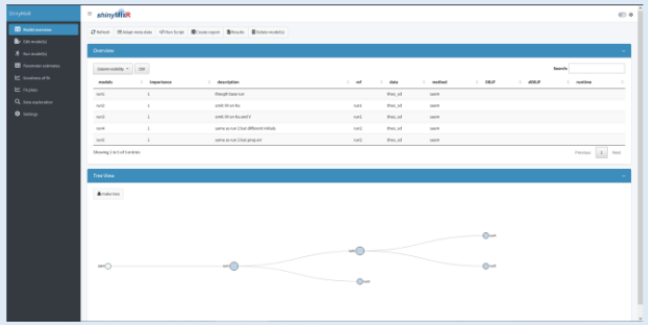
The interface can be started from the projects root folder:

```
run_shinymixr()
```

The app can be opened in an Rstudio window or web browser (using the launch.browser argument). The start window displays a dashboard with in the main window a (tree) overview of the models in the project structure.
The interface can be started at all times – even if the project was initially started in an interactive way; and *vice versa*.

Usage

There are various widgets available on the left side of the screen to run and analyse your models.



Model overview	Starting point, contains overview of models including most important information. Possibility to add/adapt meta information, change views, run scripts/reports and visualize relationship between models.
Edit model	Create, save, duplicate and edit models. Possibility to use template models. Syntax highlighted editor using `shinyAce` editor.
Run model	Run one or multiple nlmixr models side-by-side. Perform model runs in separate R session(s). Keep track of progress of runs.
Parameter estimates	Create dense parameter table. Quickly observe parameter estimates and compare estimates between models. . Export to HTML or PDF output*.
Goodness of fit	Create a combination of most important goodness of fit plots. Export plots to HTML or PDF output*.
Fit plots	Create individual fit plots. Export to HTML or PDF output*.
Data exploration	Create exploratory data plots for input or model results. Includes option to make interactive plots and table view of your data
Settings	Adapt the settings of the app, like type of output and look of editor.

** Results are by default saved in the analysis subfolder, which makes them available for the interface. For creation of PDF, LaTeX including various packages is required.*

shinyMixR

nlmixr²

Design evaluation and optimization with PopED, babelmixr2 and nlmixr2

Justin J. Wilkins, PhD
Occams, Amstelveen, The Netherlands

Theo Papathanasiou, MPharm, PhD
GSK, Zug, Switzerland

Matt Fidler, PhD
Novartis, Fort Worth, Texas, United States

and the nlmixr Development Team

Segment outline

- Brief introduction to optimal design theory
- Brief introduction to optimal design software **PopED**
- Introduction to **babelmixr2** interface with **PopED**
- Hands-on for design evaluation and optimization
  - Hands-on 1: Solved and ODE PK model
  - Hands-on 2: Solved PKPD model
  - *Hands-on 3: More complex ODE TMDD model (bonus)*

babelmixr2

- **babelmixr2** is a pharmacometric tool that aims to convert **nlmixr2** syntax to other commonly used tools
- **babelmixr2** can help you by:
  - Converting your **nlmixr2** model into the appropriate format for a commercial nonlinear mixed effects modeling tool like NONMEM or Monolix
  - Convert your NONMEM model to a **nlmixr2** model (in conjunction with **nonmem2rx**)
  - Convert you Monolix model to a **nlmixr2** model (in conjunction with **monolix2rx**)
  - Calculate scaling factors and automatically add initial conditions based on non-compartmental analysis (using **PKNCA**)
  - Perform Optimal Design using **nlmixr2** as an interface to **PopED**

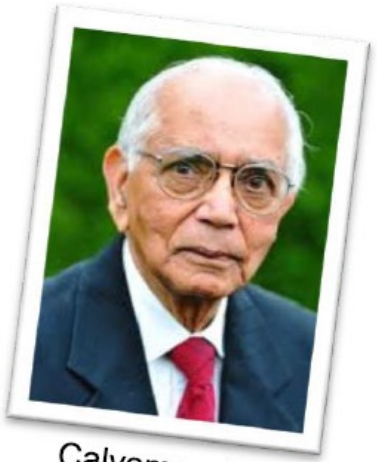
[1] CRAN: https://cran.r-project.org/web/packages/rxode2
[2] GitHub: https://github.com/nlmixr2/rxode2
[3] Wang W et al. CPT:PSP (2016) 5, 3–10.
[4] rxode2 packagedown: https://nlmixr2.github.io/rxode2

Background: Optimal design

- Clinical trial simulation has limitations…
  - Computationally expensive
  - Often intuition-based
- Optimal design (OD)
  - Can be used to produce experimental designs that are optimal with respect to some statistical criterion
  - Allow parameters to be estimated while **minimizing bias** and with **minimum variance**

Background: Cramer-Rao inequality

Harald Cramér
1893 - 1985



Calyampudi
Radhakrishna Rao
1920 -

$$\mathbf{COV}(\xi, \Theta) \geq \mathbf{FIM}(\xi, \Theta)^{-1}$$

- The variance of any unbiased estimator is at least as high as the inverse of the Fisher information.
- If Fisher information is **low**, parameter precision **cannot** be high
- Poorly designed experiments can be identified in advance
  - How? Because the **lower bounds** for the parameter's relative standard errors (RSEs) can be calculated by the diagonals of the inverse of the Fisher Information Matrix (FIM)!

$$\mathbf{FIM}(\xi, \Theta) = -\mathrm{E}\left[\frac{\partial^2 \mathrm{LL}(\xi, \Theta)}{\partial \Theta \partial \Theta^{\mathrm{T}}}\right]$$

nlmixr²

Background: Optimality criteria

- **D-optimality**
  - The determinant of the FIM is maximized*
  - The uncertainty of **all** model parameters is minimized

$$D_{crit}(\xi) = |\mathbf{FIM}(\xi, \Theta)|$$

- **Ds-optimality**
  - Designs focus on the imprecision of the "interesting" parameters

$$Ds_{crit}(\xi) = \frac{|\mathbf{FIM}(\xi, \Theta)|}{|\mathbf{FIM}_{Rdc}(\xi, \Theta)|}$$

- Many, many more… (A-, C- ED-, V-, compound…)

* Equivalent to minimizing the inverse of the determinant of the FIM

nlmixr²

Optimality in non-linear models – A somewhat counterintuitive exercise

- **Considerations and limitations**
  - For **linear** models, design optimization is independent of the model parameters (this is good!)
  - For nonlinear model, **there is** a dependency on 1) the **model structure** and 2) the **model parameters**
  - Counter-intuitive thinking: To optimize a design we first need to know the model and the parameters!
  - Usually we have some prior knowledge of the model parameters and model structure
    - There are methods to mitigate this (e.g. E-optimality for parameter uncertainty, model averaging for model structure uncertainty etc.)
- Do not take the optimal design RSEs at face value
  - Optimal design is based on calculating the **lower bounds** of the FIM – they will always appear smaller than in reality
  - Always perform a SSE exercise to better understand the truly expected RSEs

Quite a few limitations – why is optimal design interesting?

- Very! quick calculation of expected RSEs! No need for time consuming simulation!
- **Design evaluation** is a fast way to understand the information content for designs that are interesting to the clinical team!
  - No need to do optimization for all designs you are working with!
- Discard non-informative designs quickly
- Perform SSE only for the most interesting design!

Software for Optimal Design

- Many OD software available - PopED, PFIM, PopDes, NONMEM ($DESIGN) etc.
  - Pros and cons were explored by Nyberg et al.
- **Steep learning curve**: need to learn a “new” way to code models for each software of choice
- **Burdensome**: modeler needs to “translate” each model from the simulation/estimation code of choice to the appropriate optimal design software language
- **Repetitive**: Translation needs to happen for every new modeling exercise
- NONMEM $DESIGN helps (a bit) by implementing optimal design and estimation/simulation in a single tool
- **babelmixr2** is now able to interact seamlessly with **PopED**

Joakim Nyberg, Caroline Bazzoli, Kay Ogungbenro, Alexander Aliev, Sergei Leonov, Stephen Duffull, Andrew C. Hooker, France Mentré. Methods and software tools for design evaluation in population pharmacokinetics–pharmacodynamics studies. BJCP 2014. https://doi.org/10.1111/bcp.12352

PopED syntax

- PopED applies 3 functions in order to define a model:
  - `ff()`, the structural model
  - `fg()`, the parameter model (fixed effects and variability)
  - `feps()`, the residual error model
- `create.poped.database()` collects together these model functions, parameter values, and everything related to study design

PopED syntax

Structural model function

Hide

```
# Model: One comp first order absorption
# Analytic solution for both multiple and single dosing
ff <- function(model_switch,xt,parameters,poped.db){
  with(as.list(parameters),{
    y=xt
    N = floor(xt/TAU)+1
    y=(DOSE*Favail/V)*(KA/(KA - CL/V)) *
      (exp(-CL/V * (xt - (N - 1) * TAU)) * (1 - exp(-N * CL/V * TAU))/(1 - exp(-CL/V *
TAU)) -
         exp(-KA * (xt - (N - 1) * TAU)) * (1 - exp(-N * KA * TAU))/(1 - exp(-KA * TA
U)))
    return(list( y=y,poped.db=poped.db))
  })
}
```


nlmixr²

PopED syntax

Parameter function

Hide

```
# parameter definition function
# names match parameters in function ff
# PopED expects the bpop[] notation for fixed effects, the b[] notation for random effe
cts and the a[] notation for any parameter that we are interested to evaluate or optimi
ze over.
sfg <- function(x,a,bpop,b,bocc){
  parameters=c( V=bpop[1]*exp(b[1]),
                KA=bpop[2]*exp(b[2]),
                CL=bpop[3]*exp(b[3]),
                Favail=bpop[4],
                DOSE=a[1],
                TAU=a[2])
  return( parameters )
}
```


nlmixr²

PopED syntax

Residual unexplained variablity (RUV) function

Hide

```
# Residual unexplained variablity (RUV) function
# Additive + Proportional
# In this example, the additive and proportional residual errors are defined explicitel
y, but there are some shorthand functions available, such as:

feps <- function(model_switch,xt,parameters,epsi,poped.db){
  returnArgs <- do.call(poped.db$model$ff_pointer,list(model_switch,xt,parameters,pope
d.db))
  y <- returnArgs[[1]]
  poped.db <- returnArgs[[2]]

  y = y*(1+epsi[,1])+epsi[,2]

  return(list( y= y,poped.db =poped.db ))
}
```


nlmixr²

PopED syntax

Design and design space definition

Hide

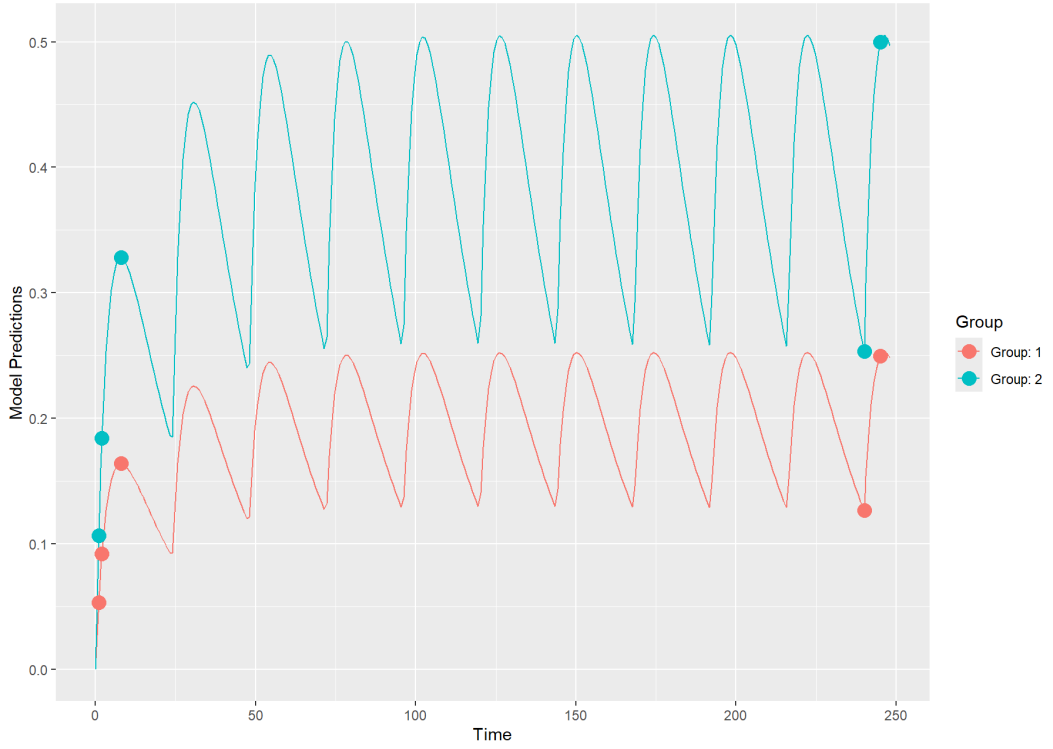
```
# Define design and design space
poped.db <- create.poped.database(
      # Definition of the user defined functions
      ff_fun="ff",
      fg_fun="sfg",
      fError_fun="feps",

      # Definition of the fixed effect, random effects and error parameters,
      # and if they are fixed or not
      bpop=c(V=72.8,KA=0.25,CL=3.75,Favail=0.9),
      notfixed_bpop=c(1,1,1,0),
      d=c(V=0.09,KA=0.09,CL=0.25^2),
      sigma=c(0.04,5e-6),
      notfixed_sigma=c(0,0),
```

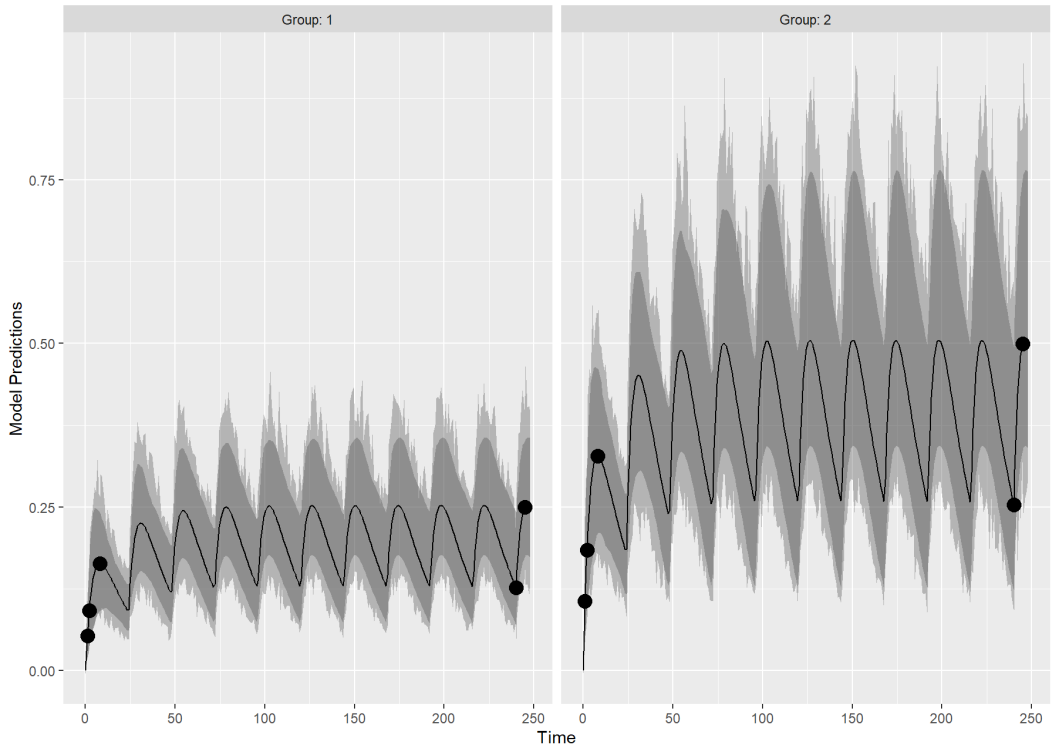
PopED syntax

```
# Definition of the design group number, and the number or subjects per group
m=2,
groupsize=20,

# Definition of the design space
xt=c( 1,2,8,240,245),
minxt=c(0,0,0,240,240),
maxxt=c(10,10,10,248,248),
bUseGrouped_xt=1,
a=list(c(DOSE=20,TAU=24),c(DOSE=40, TAU=24)),
maxa=c(DOSE=200,TAU=24),
mina=c(DOSE=0,TAU=24))
```


nlmixr²

Always helpful to plot the model and the explored design
Easy to simulate with or w/o variability





nlmixr²

Design evaluation

Design evaluation

Hide

```
## evaluate initial design
evaluate_design(poped.db)
```

```
## $ofv
## [1] 28.9197
##
## $fim
##                 V          KA           CL        d_V       d_KA        d_CL
## V     0.05336692   -8.683963  -0.05863412   0.000000   0.000000    0.000000
## KA   -8.68396266 2999.851007 -14.43058560   0.000000   0.000000    0.000000
## CL   -0.05863412  -14.430586  37.15243290   0.000000   0.000000    0.000000
## d_V   0.00000000    0.000000   0.00000000 999.953587 312.240246    3.202847
## d_KA  0.00000000    0.000000   0.00000000 312.240246 439.412556    2.287838
## d_CL  0.00000000    0.000000   0.00000000   3.202847   2.287838 3412.005199
##
## $rse
##         V        KA        CL       d_V      d_KA      d_CL
##  8.215338 10.090955  4.400304 39.833230 60.089601 27.391518
```


nlmixr²

Design Optimization

Optimization of sample times

Hide

```
# Optimization of sample times
output <- poped_optim(poped.db, opt_xt =TRUE, parallel=FALSE)
```

Hide

```
# Evaluate optimization results
summary(output)
```

```
## ==============================================================================
## FINAL RESULTS
## Optimized Sampling Schedule
## Group 1:  0.437     10     10    240  240.9
## Group 2:  0.437     10     10    240  240.9
##
## OFV = 30.2698
##
## Efficiency:
##   ((exp(ofv_final) / exp(ofv_init))^(1/n_parameters)) = 1.2523
##
## Expected relative standard error
## (%RSE, rounded to nearest integer):
##    Parameter   Values   RSE_0   RSE
##            V     72.8       8     6
##           KA     0.25      10     8
##           CL     3.75       4     4
##          d_V     0.09      40    32
##         d_KA     0.09      60    49
##         d_CL   0.0625      27    26
##
## Total running time: 22.6 seconds
```


nlmixr²

babelmixr2/PopED syntax

1. Define the model using **nlmixr2** language, as usual
2. Create an event table that represents the design
3. Construct the PopED database using the `nlmixr2()` command, together with the **"poped"** option
4. Evaluate and optimize designs using the PopED functions!

nlmixr²

babelmixr2/PopED syntax – Step 1 – The model definition

```
f <- function() {
  ini({
    tV <- 72.8
    tKa <- 0.25
    tCl <- 3.75
    tF <- fix(0.9)

    eta.v ~ 0.09
    eta.ka ~ 0.09
    eta.cl ~0.25^2

    prop.sd <- fix(sqrt(0.04))
    add.sd <- fix(sqrt(5e-6))

  })
  model({

    V<-tV*exp(eta.v)
    KA<-tKa*exp(eta.ka)
    CL<-tCl*exp(eta.cl)
    Favail <- tF
    N <-  floor(time/TAU)+1
    y <- (DOSE*Favail/V)*(KA/(KA - CL/V)) *
      (exp(-CL/V * (time - (N - 1) * TAU)) *
         (1 - exp(-N * CL/V * TAU))/(1 - exp(-CL/V * TAU)) -
         exp(-KA * (time - (N - 1) * TAU)) * (1 - exp(-N * KA * TAU))/(1 - exp(-KA * TAU)))

    y ~ prop(prop.sd) + add(add.sd)
  })
}
```

Parameter definition for :
1) fixed effects
2) random effects
3) residual variability

Standard nlmixr2 model definition

Note the DOSE and TAU variables
These are both “design variables”
We will define them in Step 3

nlmixr²

babelmixr2/PopED syntax – Step 2 – The event table

Event table definition

This is one point where the babelmixr2/PopED workflow diverges from the classical PopED workflow. Here, the user provides a rxode2 style event table for the observation times. this event table is essentially defining the minimal design that we will be exploring in the example.

Hide

```
# Event table definition. notice that for each design time point, we define a "low" and a "high"
time. These will be the observation windows that we allow our samples to be optimized over. This
is very helpful when there are logistical constraints that may dictate the design of our trial.

# PopED minxt and maxxt equivalent
e <- et(list(c(0, 10),
             c(0, 10),
             c(0, 10),
             c(240, 248),
             c(240, 248))) %>%
  as.data.frame()

# PopED xt equivalent
e$time <-  c(1,2,8,240,245)

e %>% knitr::kable()
```

low	time	high	evid
0	1	10	0
0	2	10	0
0	8	10	0
240	240	248	0
240	245	248	0

nlmixr²

babelmixr2/PopED syntax – Step 3 – The interface

babelmixr2/poped database

The final step is to create a PopED style database. This is done via the nlmixr2() function, together with the "poped" argument. This function call is equivalent to create.poped.database() from the classical PopED workflow. Notice that many of the design definition options defined within popedControl(), are similar to the ones showing in the classical PopED workflow.

Hide

```
babel.db <- nlmixr2(f, e, "poped",
                        popedControl(groupsize=20,
                                     bUseGrouped_xt=TRUE,
                                     a=list(c(DOSE=20,TAU=24),
                                            c(DOSE=40, TAU=24)),
                                     maxa=c(DOSE=200,TAU=24),
                                     mina=c(DOSE=0,TAU=24)))
```

PopED like Options

- **groupsize**: The number of subject per group (20 for in this example)
- **bUseGrouped_xt**: Do we want the design points to stay consistent between the two groups during optimization?
- **a**: a list of all the design variables of interest. **DOSE** and dosing interval (**TAU**) in this example

nlmixr²

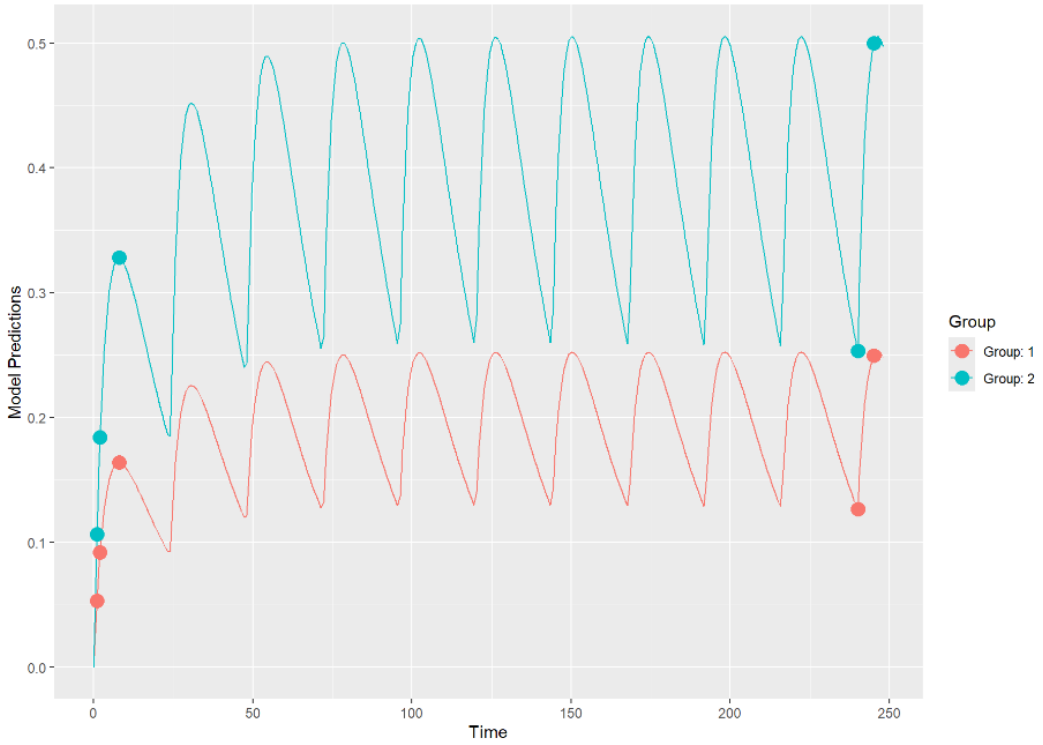
Downstream PopED function compatibility

Plot model prediction

The plot_model_prediction function from PopED work seamlessly with the babelmixr2/PopED database

Hide

```
##  create plot of model without variability
plot_model_prediction(babel.db, model_num_points = 300)
```



Design evaluation

The design evaluation function from PopED works seamlessly with the babelmixr2/PopED database. Notice that we are receiving the same results as in the classical PopED workflow.

Hide

```
evaluate_design(babel.db)
```

```
## $ofv
## [1] 28.9197
##
## $fim
##                  tV         tKa          tCl    d_eta.v   d_eta.ka    d_eta.cl
## tV        0.05336692   -8.683963  -0.05863412   0.000000   0.000000    0.000000
## tKa      -8.68396268 2999.851009 -14.43058543   0.000000   0.000000    0.000000
## tCl      -0.05863412  -14.430585  37.15243290   0.000000   0.000000    0.000000
## d_eta.v   0.00000000    0.000000   0.00000000 999.953586 312.240246    3.202847
## d_eta.ka  0.00000000    0.000000   0.00000000 312.240246 439.412557    2.287837
## d_eta.cl  0.00000000    0.000000   0.00000000   3.202847   2.287837 3412.005200
##
## $rse
##        tV       tKa       tCl   d_eta.v  d_eta.ka  d_eta.cl
##  8.215338 10.090955  4.400304 39.833230 60.089601 27.391518
```

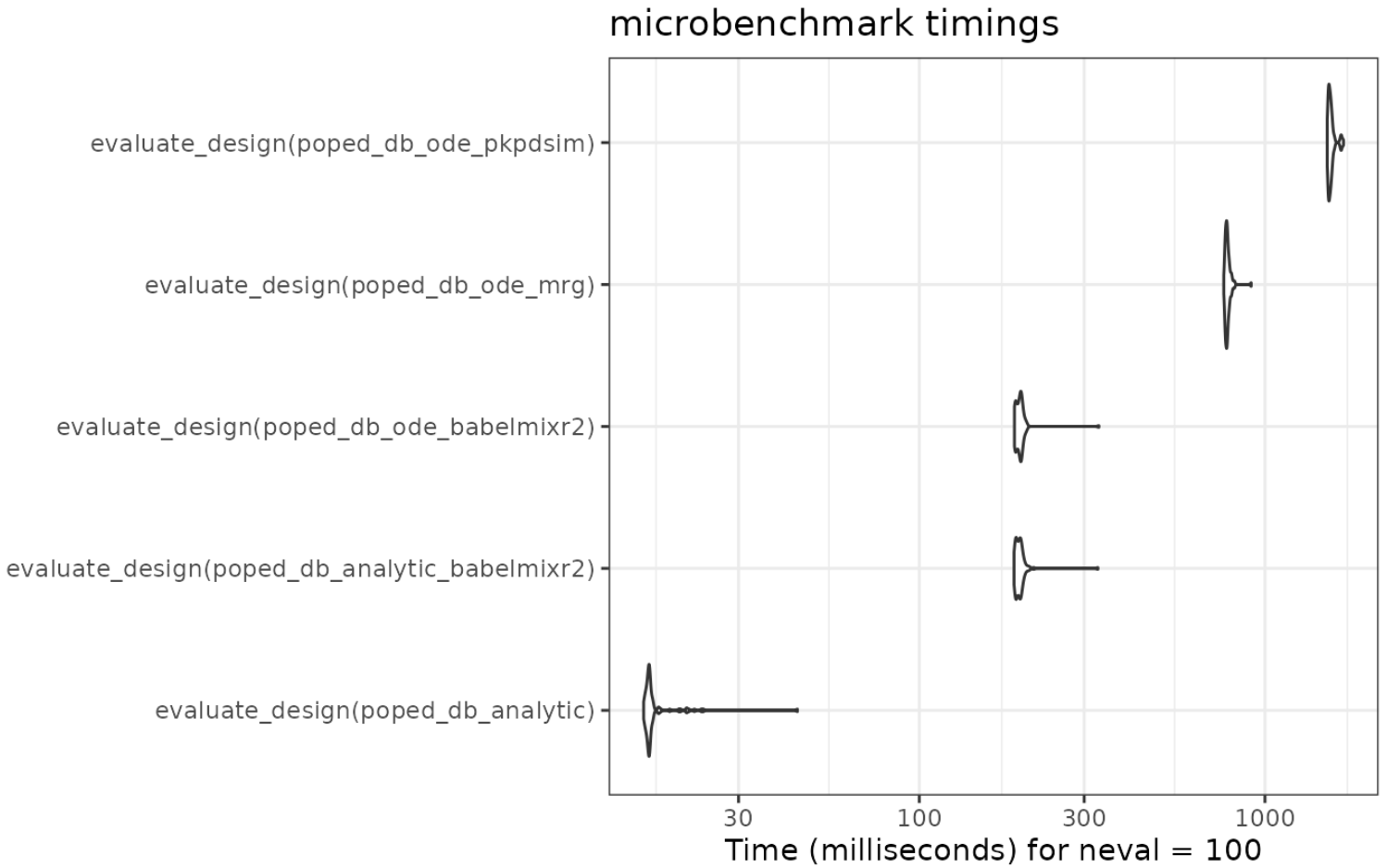

nlmixr²

Stochastic Simulation and Estimation (SSE)

- RSE calculation based on the determinant of the FIM is only a lower bound.
- This means that the RSEs obtained by the FIM are **expected** to be lower compared to what we might see in a real design
- SSEs are needed to understand the truly expected RSE for a specific design
  - Additional benefit of the SSE, is that this allows calculation of the parameter bias. This is not possible with FIM based approaches
- Best ways of working: Explore various designs with FIM evaluations. Compare designs to either the original design, or the D-optimal design, as both can help to set an anchor for comparison
- Run computationally expensive SSEs only for the best candidate designs
- Not covered here, but fairly straightforward to run SSEs with a combination of **rxode2** and **nlmixr2**

Benchmarking against other solvers

- **babelmixr2** ODE solver: the fastest in this comparison.
- Among other things, this is because the model is loaded into memory and does not need to be setup each time.
- Results may be problem depended
- As benchmarks: the **mrgsolve**, and **PKPDsim** implementations on the **PopED** website

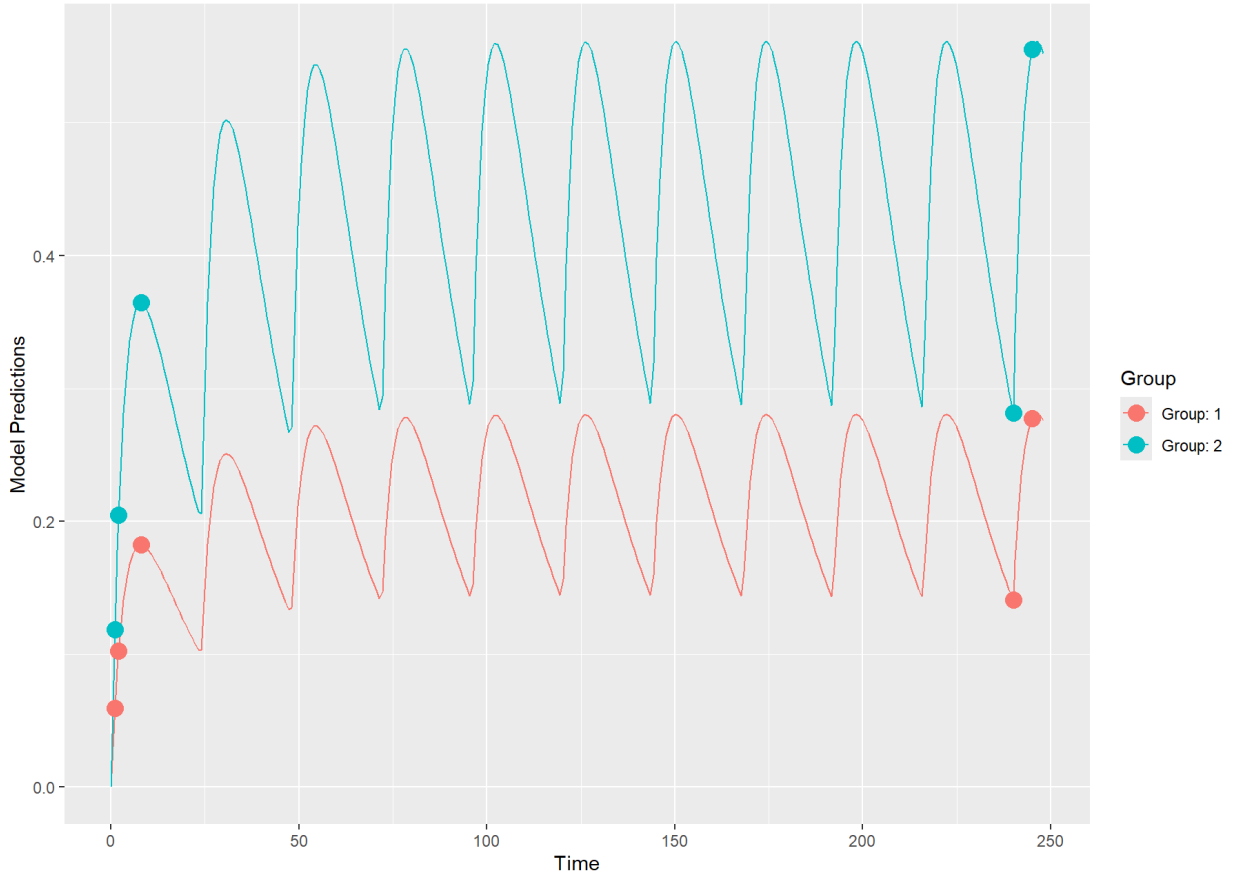


https://andrewhooker.github.io/PopED/articles/model_def_other_pkgs.html

nlmixr²

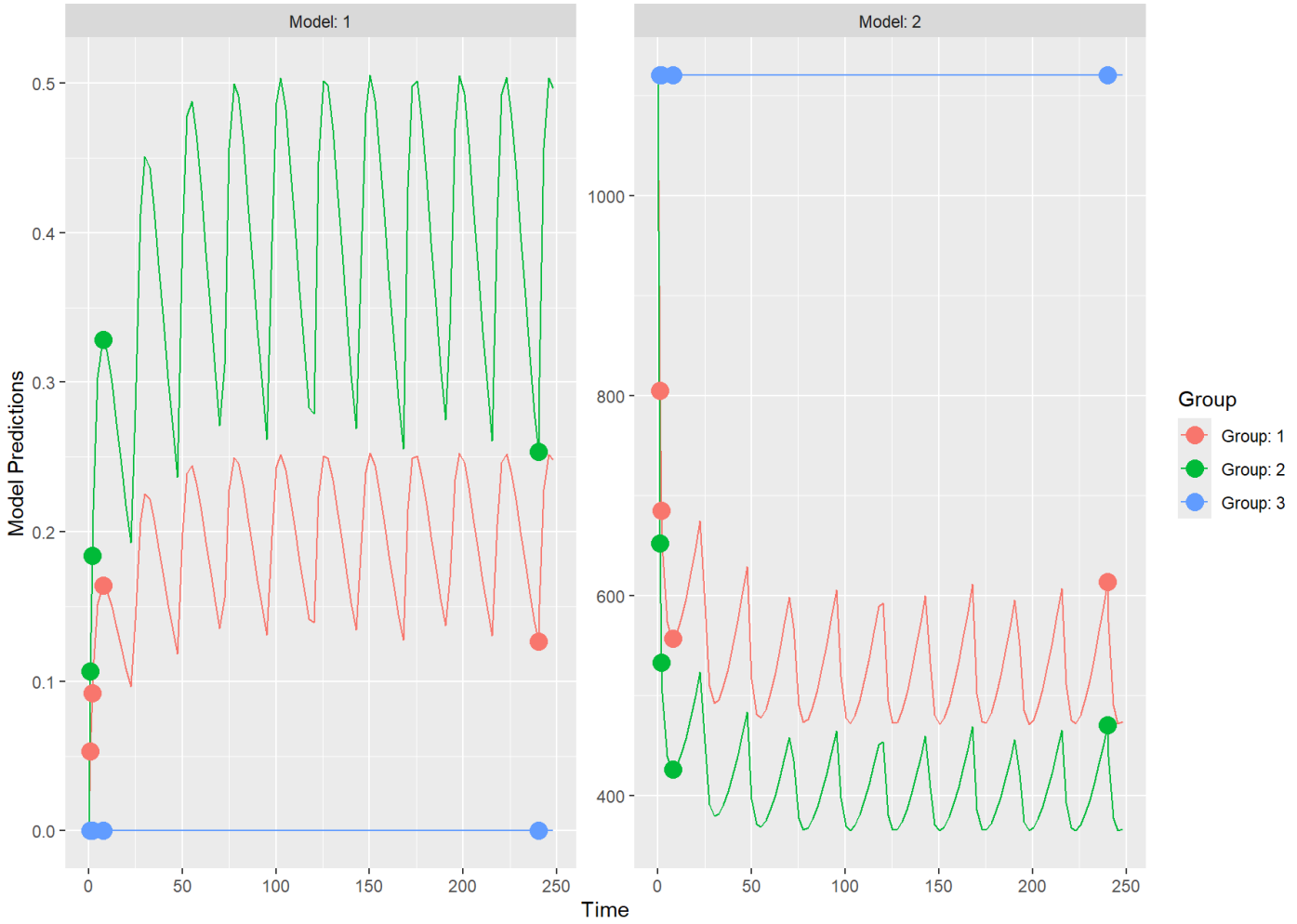
Hand-on 1: Closed form and ODE PK model

- Multiple doses
- Two dosing groups
- 20 subjects per group



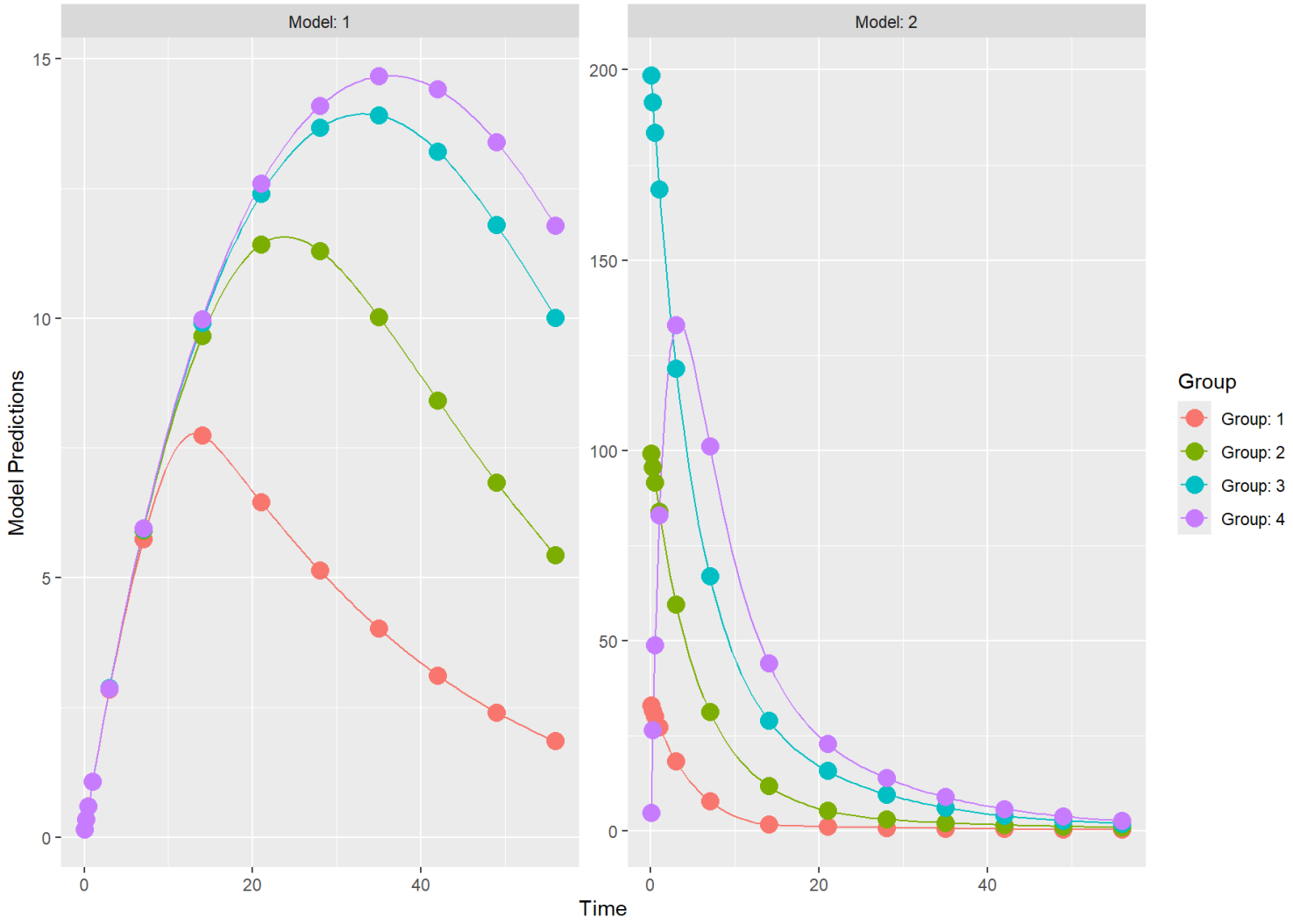
Hand-on 2: Closed form PKPD model

- Multiple doses
- Two dosing groups and one placebo group
- PK linked to PD by a direct relationship



Hand-on 3: ODE TMDD model

- Single dose
- Four dosing groups
- TMMD kinetics



In conclusion

- Optimal design: a helpful tool to explore and optimize study design options
  - Always run confirmatory simulations (SSE)
- **Friendly user interface** (UI) between **babelmixr2** and **PopED**
  - This allows for an entire workflow to be part of a “single” R script
- Small differences to traditional PopED interface due to the introduction of the event table

Key benefits of the babelmixr2 interface to PopED

- The simplicity of using **nlmixr2**/**rxode2** models directly with **PopED** without tedious and error-prone model conversions
- The ease of translating from NONMEM or Monolix to **nlmixr2** and then **PopED** through **babelmixr2**
- Our hope is that the new UI will alleviate part of the entry burden to optimal design applications

Personalized drug regimens with posologyr and rxode2

Justin J. Wilkins, PhD
Occams, Amstelveen, The Netherlands

Cyril Leven, PhD
Brest University Hospital, Brest, France

and the nlmixr Development Team

nlmixr²

Introduction

- **posologyr** is an R package developed (primarily) by Cyril Leven at Brest University Hospital to help with Bayesian dose individualization
  - By combining therapeutic drug monitoring (TDM) data with a population model, **posologyr** offers accurate posterior estimates and helps compute optimal individualized dosing regimens
  - **rxode2** provides the prediction/simulation engine

pharmaceutics

MDPI

Article

Free and Open-Source Posologyr Software for Bayesian Dose Individualization: An Extensive Validation on Simulated Data

Cyril Leven [1,2,*], Anne Coste [3] and Camille Mané [1]

[1] Department of Biochemistry and Pharmaco-Toxicology, Brest University Hospital, 29200 Brest, France; camille.mane@chu-brest.fr
[2] Univ Brest, EA 3878, GETBO, 29200 Brest, France
[3] Infectious Diseases Department, Brest University Hospital, 29200 Brest, France; anne.coste@chu-brest.fr
* Correspondence: cyril.leven@chu-brest.fr

Abstract: Model-informed precision dosing is being increasingly used to improve therapeutic drug monitoring. To meet this need, several tools have been developed, but open-source software remains uncommon. Posologyr is a free and open-source R package developed to enable Bayesian individual parameter estimation and dose individualization. Before using it for clinical practice, performance validation is mandatory. The estimation functions implemented in posologyr were benchmarked against reference software products on a wide variety of models and pharmacokinetic profiles: 35 population pharmacokinetic models, with 4.000 simulated subjects by model. Maximum A Posteriori (MAP) estimates were compared to NONMEM post hoc estimates, and full posterior distributions were compared to Monolix conditional distribution estimates. The performance of MAP estimation was excellent in 98.7% of the cases. Considering the full posterior distributions of individual parameters, the bias on dosage adjustment proposals was acceptable in 97% of cases with a median bias of 0.65%. These results confirmed the ability of posologyr to serve as a basis for the development of future Bayesian dose individualization tools.

check for updates

Citation: Leven, C.; Coste, A.; Mané, C. Free and Open-Source Posologyr Software for Bayesian Dose Individualization: An Extensive Validation on Simulated Data.

Keywords: clinical pharmacokinetics; dosage individualization; Bayesian dosing; therapeutic drug monitoring; Maximum A Posteriori

Leven C, Coste A, Mané C (2022). “Free and Open-Source Posologyr Software for Bayesian Dose Individualization: An Extensive Validation on Simulated Data.” Pharmaceutics, 14(2), 442. doi:10.3390/pharmaceutics14020442, https://europepmc.org/article/pmc/8879752.

What we need

- **posologyr** (from CRAN)

```
install.packages("posologyr")
```

- A **model**

- Some individual **data** (TDM PK samples, for instance)… or no data at all

- A **question**!

```
rx ← function() {
  ini({
    Cl_pop ← -0.439
    Q_pop  ← 1.29
    V1_pop ← 1.62
    V2_pop ← 1.63
    beta_Cl_SEX_1    ← -0.282
    beta_Cl_logtClCr ← 0.272
    beta_V2_logtWT   ← 1.32
    a    ← 2.77
    omega_Cl ~ 0.0917
    omega_Q  ~ 0.650
    omega_V1 ~ 0.231
    omega_V2 ~ 0.0347
  })
  model({
    Cl ← exp(Cl_pop + beta_Cl_SEX_1 * SEX + beta_Cl_logtClCr *
            log(ClCr/37.8) + omega_Cl)
    Q  ← exp(Q_pop + omega_Q)
    V1 ← exp(V1_pop + omega_V1)
    V2 ← exp(V2_pop + beta_V2_logtWT * log(WT/69.8) + omega_V2)

    k12 ← Q/V1
    k21 ← Q/V2

    d/dt(central) ← -k12 * central + k21 * cmt2 - Cl/V1 * central
    d/dt(cmt2)    ←  k12 * central - k21 * cmt2
    Cc ← central/V1
    DV ← Cc
    DV ~ add(a)
  })
}
```

This is a two-compartment IV infusion model for daptomycin imported from Monolix using **monolix2rx**!

Covariates include sex (SEX, 0 or 1) and creatinine clearance (ClCr) on Cl, and weight (WT) on V2.

Daptomycin

- Daptomycin is a cyclic lipopeptide antibacterial agent with in vitro activity against gram-positive bacteria, including methicillin-resistant *Staphylococcus aureus* (MRSA)
  - $AUC_{0\text{-}24}/MIC < 666$ associated with excess mortality in hospitalized patients (Falcone et al); others reported a target C_{max} range of 63.45-76.29 mg/L (Galar et al)
  - C_{min} associated with creatine phosphokinase (CPK) elevation above 24.3 mg/L (Bhavnani et al)
  - We have a handy population PK model (Dvorchik et al)

Bhavnani SM, Rubino CM, Ambrose PG, Drusano GL. Daptomycin Exposure and the Probability of Elevations in the Creatine Phosphokinase Level: Data from a Randomized Trial of Patients with Bacteremia and Endocarditis. Clinical Infectious Diseases. 2010 Jun 15;50(12):1568–74

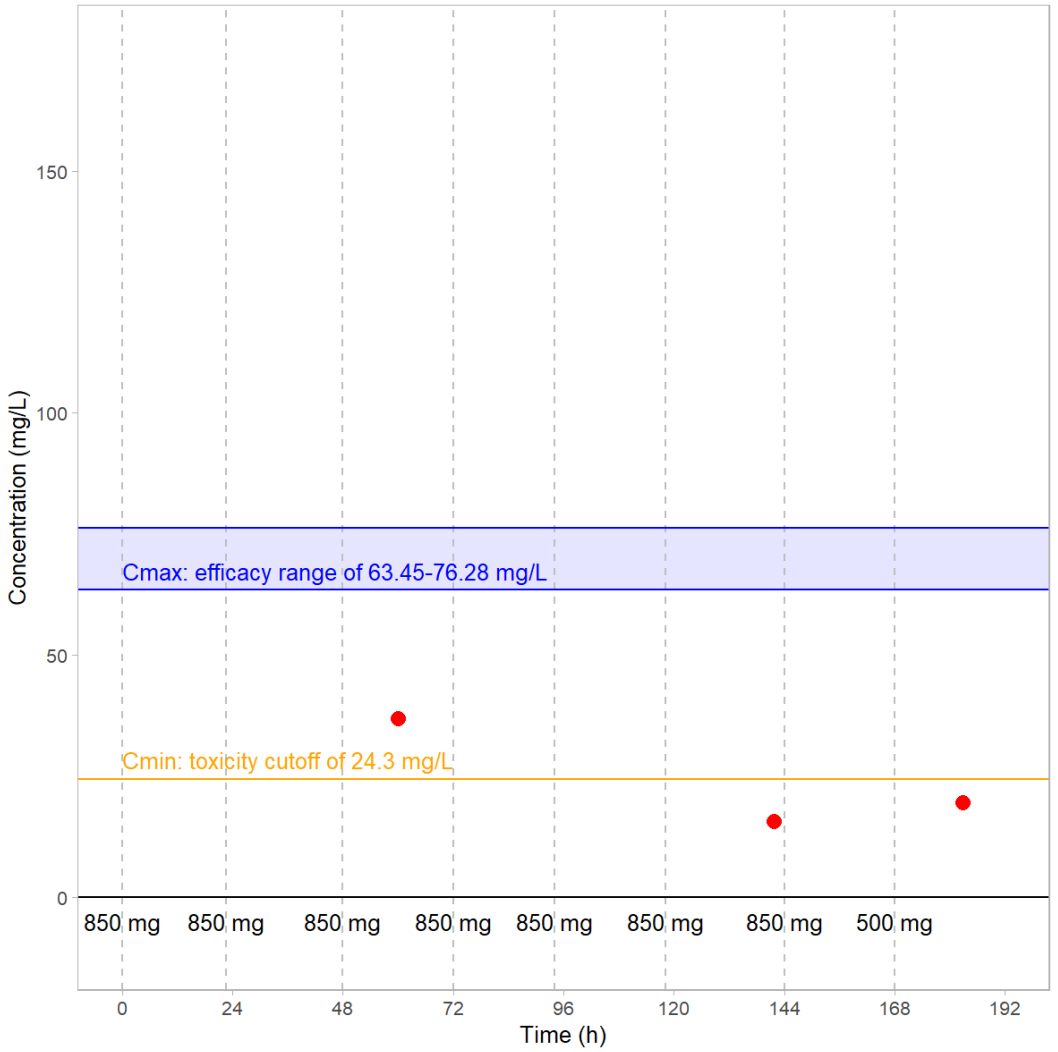
Dvorchik B, Arbeit RD, Chung J, Liu S, Knebel W, Kastrissios H. Population Pharmacokinetics of Daptomycin. Antimicrobial Agents and Chemotherapy. 2004 Aug;48(8):2799–807

Falcone M, Russo A, Cassetta MI, Lappa A, Tritapepe L, d'Ettorre G, et al. Variability of pharmacokinetic parameters in patients receiving different dosages of daptomycin: is therapeutic drug monitoring necessary? Journal of Infection and Chemotherapy. 2013 Jan 1;19(4):732–9

Galar A, Muñoz P, Valerio M, Cercenado E, García-González X, Burillo A, et al. Current use of daptomycin and systematic therapeutic drug monitoring: Clinical experience in a tertiary care institution. International Journal of Antimicrobial Agents. 2019 Jan 1;53(1):40–8.

Consider patient X…

- Patient X
  - Weight: 82.8 kg
  - Creatinine clearance: 78.6 mL/min
  - Female (SEX=1)
  - Daily infused doses of daptomycin 10 mg/kg (later lowered to 6 mg/kg) to treat MRSA
  - MRSA MIC is 1 mg/L
- Three observations, at 60 h, 142 h, 183 h after the first dose
- Question: what is the minimum dose needed to ensure that the AUC_{24}/minimum inhibitory concentration (MIC) at steady state exceeds 666?



What does this patient's concentration-time profile actually look like?

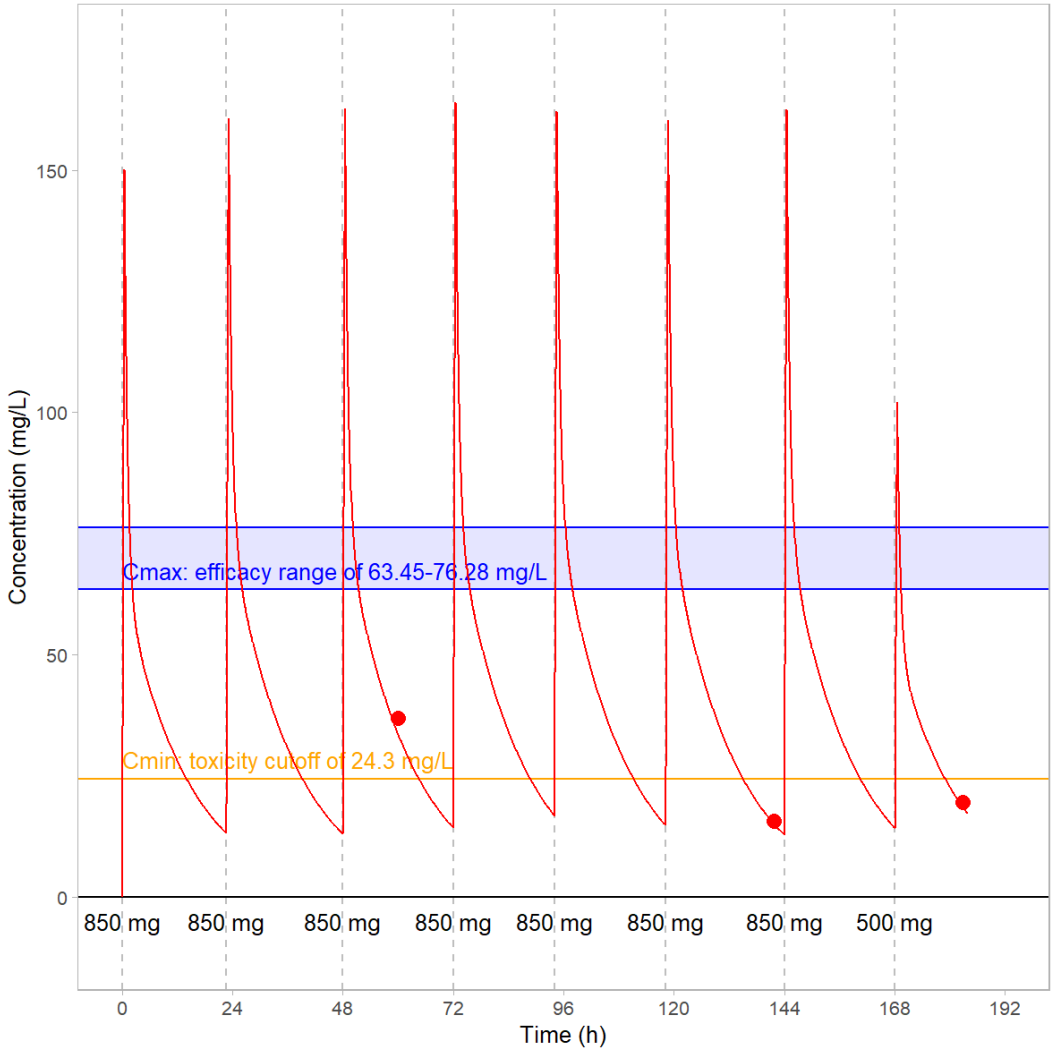
- Let's find out…

This patient's data

```
pt17fit ← poso_estim_map(dat = subset(dat, ID==17),
prior_model = rx)
```

The rxode2 model

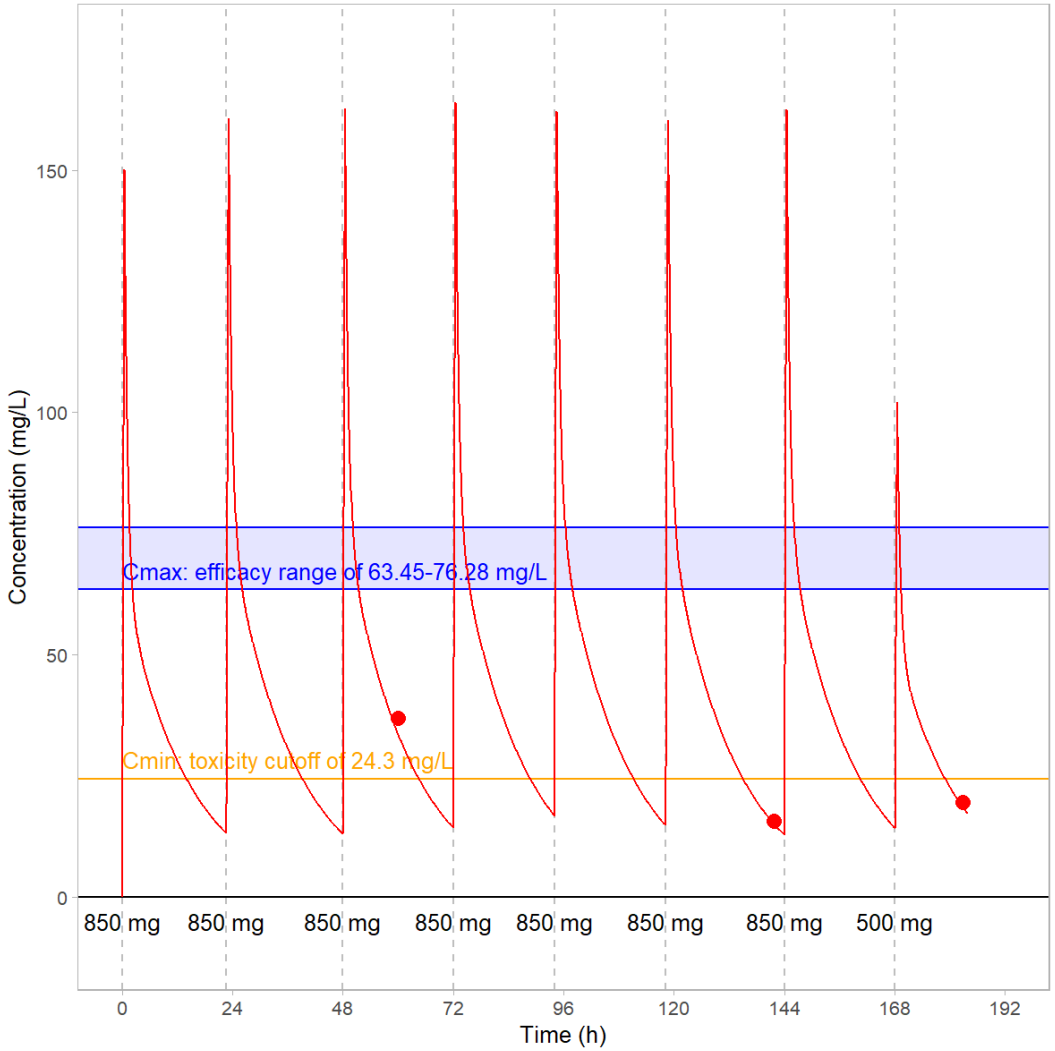
- `poso_estim_map()` provides:
  - Maximum *a posteriori* (MAP) individual parameters for individual patients based upon their data and the model…
  - Predicted concentration-time curve based on their dosing history
  - Rolling AUC



The dose seems to be OK, but…

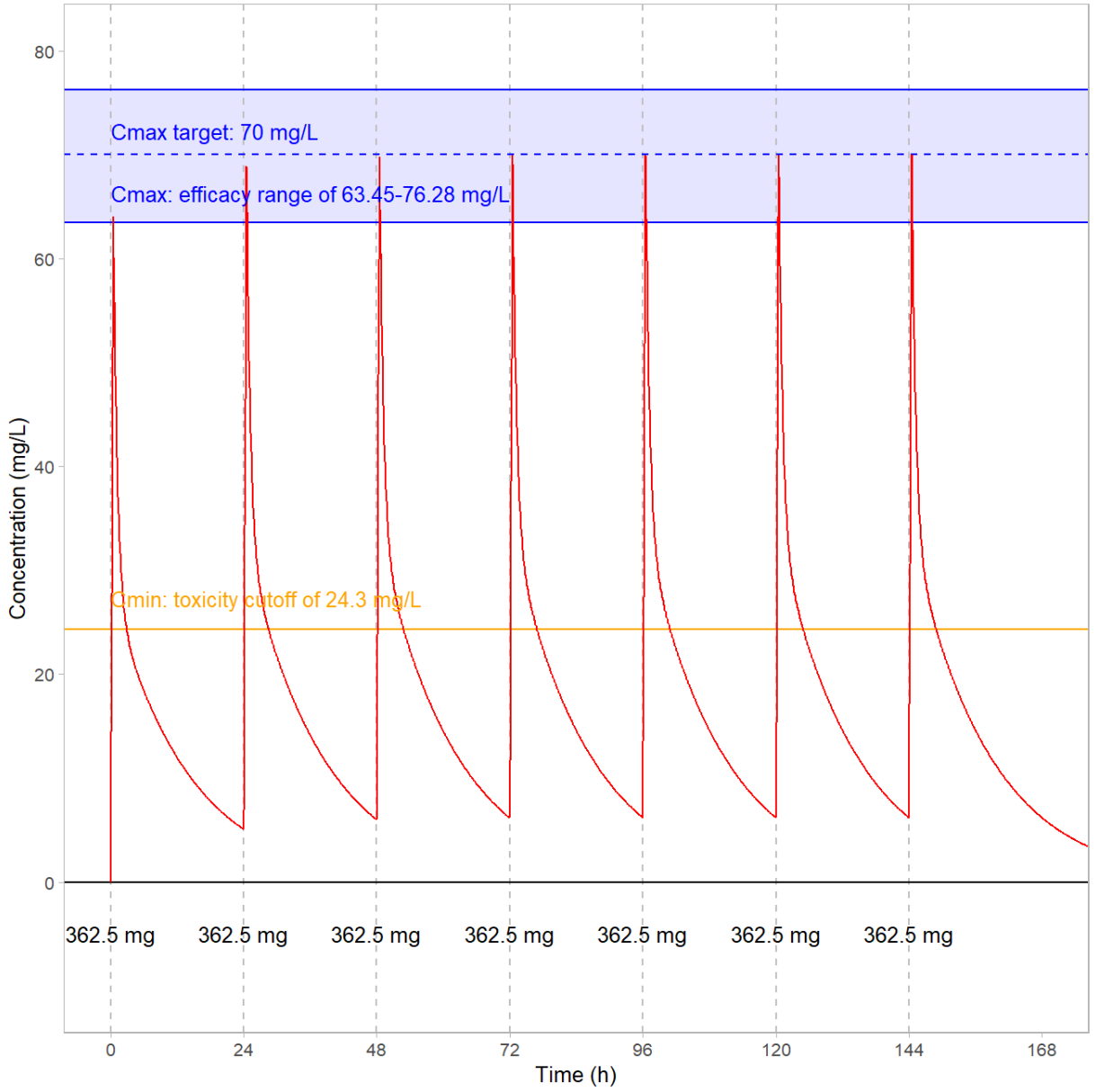
- C_{min} appears to be on target, but C_{max} might be higher than it needs to be…
- AUC between the 48 h and 72 h doses is 1003.7 mg.h/L
- Since MIC is 1 mg/L, AUC/MIC=1003.7, which is >> 666

- So for this patient, at least, a dose of 10 mg/kg/d is OK – efficacious and not overly toxic
- However, we can see from the dose reduction at 168 h that 6 mg/kg/d might be good enough…



What dose do we need for C_{max} of 70x the MIC (1 mg/L)?

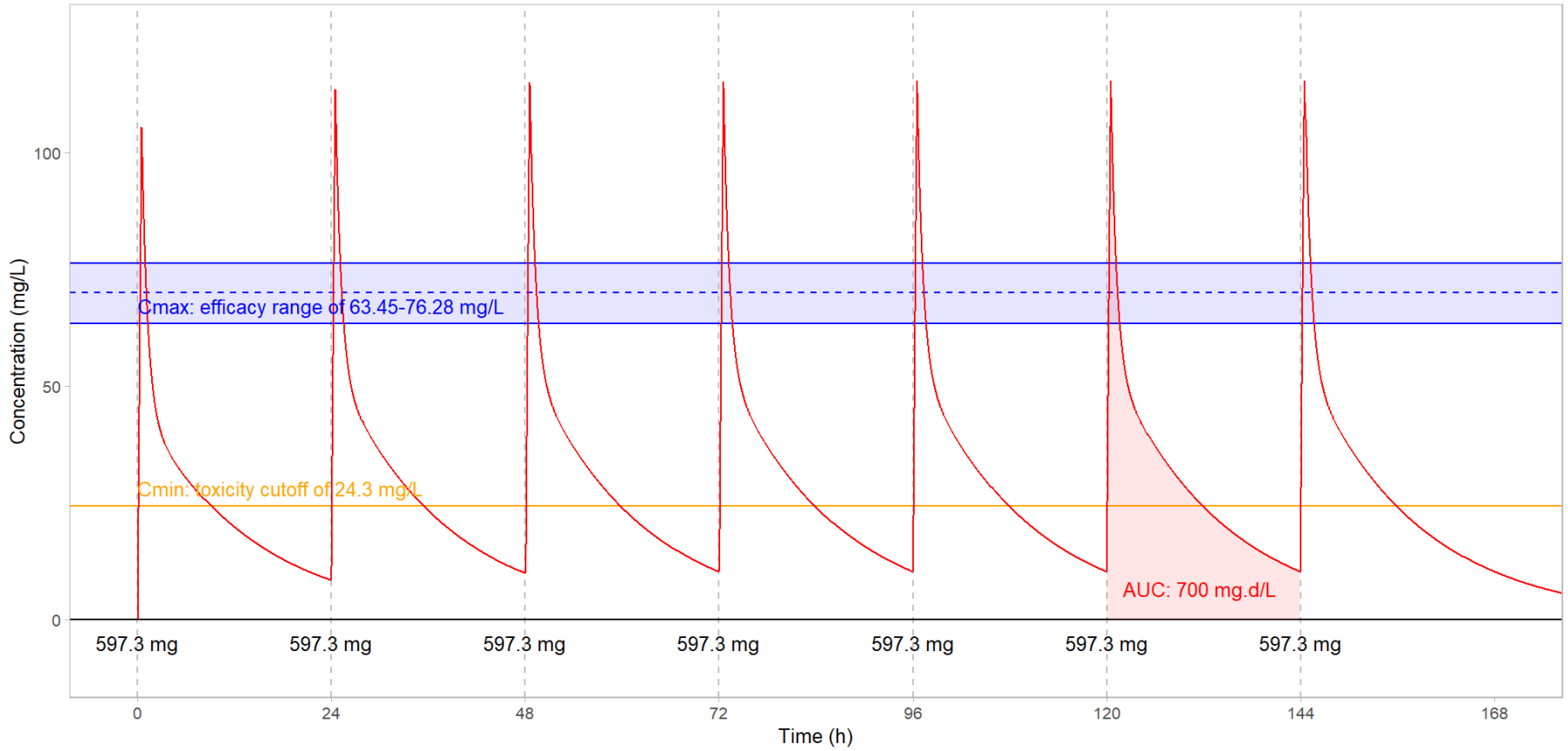
```
poso_dose_conc(dat = subset(dat, ID==17), prior_model = rx,
time_c=(6*24)+0.5, time_dose=0, target_conc = 70,
duration=0.5, indiv_param=pat17, add_dose=7,
interdose_interval=24)

$dose
[1] 362.5345

$type_of_estimate
[1] "point estimate"

$conc_estimate
[1] 70

$indiv_param
# A tibble: 1 × 16
ID Cl_pop Q_pop V1_pop V2_pop beta_Cl_SEX_1
<dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 17 -0.439 1.29 1.62 1.63 -0.282
# ℹ 10 more variables: beta_Cl_logtClCr <dbl>,
# beta_V2_logtWT <dbl>, a <dbl>, omega_Cl <dbl>,
# omega_Q <dbl>, omega_V1 <dbl>, omega_V2 <dbl>, SEX <dbl>,
# WT <dbl>, ClCr <dbl>
```



What if we wanted an AUC of 700 mg.d/L?

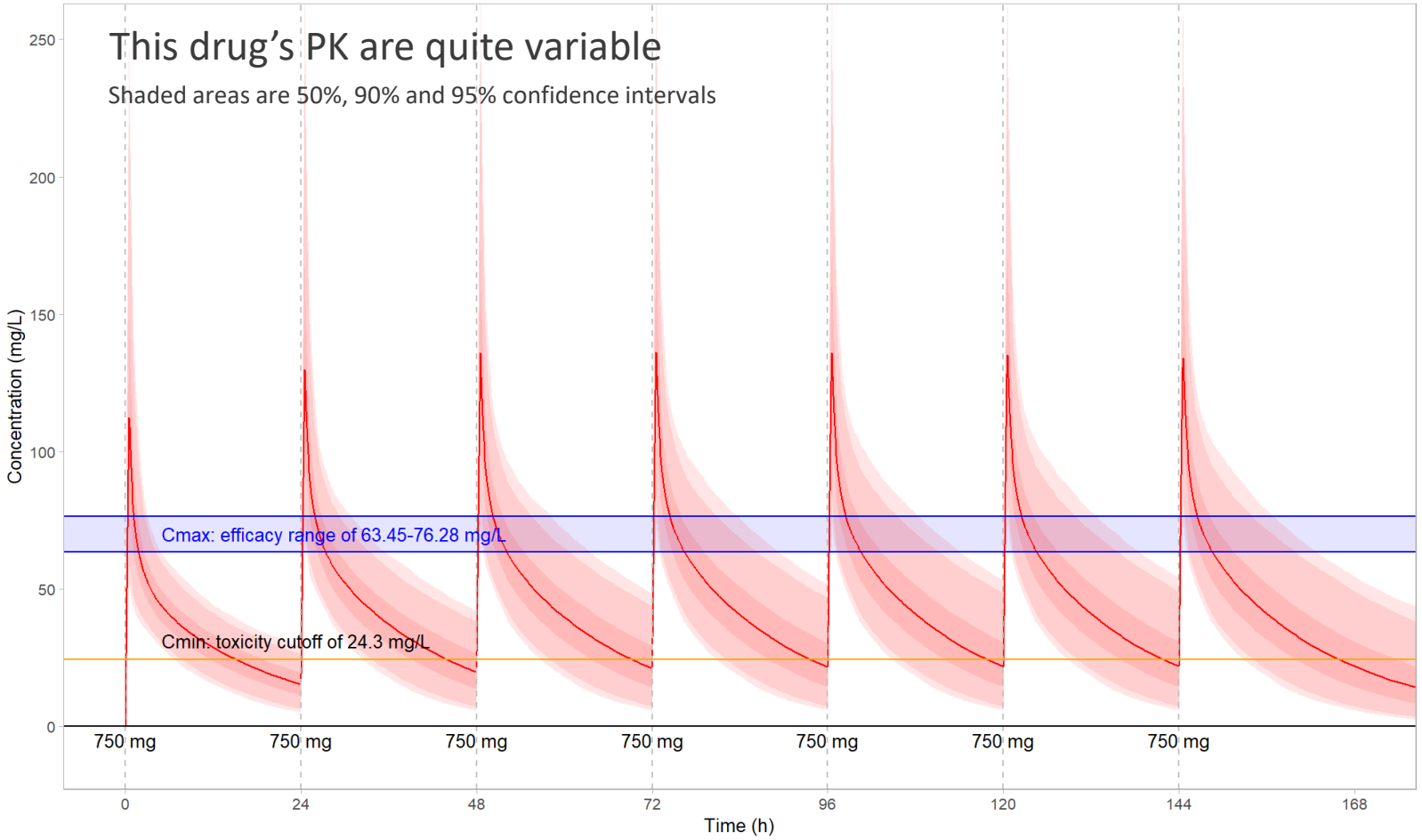
```
poso_dose_auc(dat = subset(dat, ID==17), prior_model = rx, starting_time = 144, time_auc=24, target_auc = 700,
duration=0.5, indiv_param=pat17, interdose_interval = 24, add_dose = 7)
```

```
$dose
[1] 597.3235

$type_of_estimate
[1] "point estimate"

$auc_estimate
[1] 700
```



And what if we had no samples for this subject, but only their weight, CrCL and sex…?

```
poso_simu_pop(dat=simskeleton17, prior_model=rx, n_simul=2000, return_model=T)
```



- We can simulate profiles using our patient's covariate profile
- Clearly, a dose of 750 mg has a chance of just under 50% of exceeding the C_{min} safety limit
- We would probably want to start this patient on a lower dose, collect TDM samples, and then use `poso_estim_map()` and `poso_dose_conc()` and/or `poso_dose_auc()` to find the best possible dose

And there's more!

- **posologyr** provides several more utility functions:
  - `poso_estim_mcmc()`: Estimate the posterior distribution of individual parameters by MCMC
  - `poso_estim_sir()`: Estimate the posterior distribution of individual parameters by SIR
  - `poso_inter_cmin()`: Estimates the optimal dosing interval to consistently achieve a target Cmin, given a dose, a population pharmacokinetic model, a set of individual parameters, and a target concentration
  - `poso_time_cmin()`: Estimates the time required to reach a target trough concentration (Cmin) given a population pharmacokinetic model, a set of individual parameters, a dose, and a target Cmin
  - `poso_replace_et()`: Update a model with events from a new **rxode2** event table, while accounting for and interpolating any covariates or inter-occasion variability

In conclusion

- **posologyr** provides a toolkit to leverage pharmacometrics models with very limited PK/PD data in order to optimize individual dosing in a clinical setting
- It's capable of a lot more than we've shown you, and is in ongoing development
- This presentation is based on version 1.27, currently on CRAN

posologyr is developed by **Cyril Leven** with contributions from **Matthew Fidler**, **Emmanuelle Comets**, **Audrey Lavenu** and **Marc Lavielle**

https://levenc.github.io/posologyr/

https://github.com/levenc/posologyr/

https://cran.r-project.org/web/packages/posologyr/index.html

More tools of note

Justin J. Wilkins
Occams, Amstelveen, The Netherlands

and the nlmixr Development Team

nlmixr2lib

Richard Hooijmaaijers, Matt Fidler, Bill Denney

- A model library for nlmixr2
- 50+ models already included, more added regularly

name	description
PK_1cmt	One compartment PK model with linear clearance
PK_1cmt_des	One compartment PK model with linear clearance using differential equations
PK_2cmt	Two compartment PK model with linear clearance
PK_2cmt_des	Two compartment PK model with linear clearance using differential equations
PK_2cmt_no_depot	Two compartment PK model with linear clearance using differential equations
PK_2cmt_tdcl_des	Two compartment PK model with time-dependent clearance using differential equations (structured like nivolumab PK model)
PK_3cmt	Three compartment PK model with linear clearance
PK_3cmt_des	Three compartment PK model with linear clearance using differential equations
phenylalanine_charbonneau_2021	Phenylalanine model for absorption and metabolism in healthy subjects and patients with PKU
indirect_0cpt_transitEx	Two compartment PK model with Michealis-Menten clearance using differential equations
indirect_1cpt_inhi_kin	One compartment indirect response model with inhibition of kin.
indirect_1cpt_inhi_kin_CLV	One compartment indirect response model with inhibition of kin.

https://nlmixr2.github.io/nlmixr2lib/

nlmixr2rpt

John Harrold

- Templated Word and PowerPoint reporting for nlmixr2
- Designed to automate the reporting of analyses performed in both PowerPoint and Word
- Accomplished via a YAML file that contains specifications for report elements (figures and tables) as well as the contents of Word and PowerPoint documents
- Internal templates for both the documents and the report YAML file are included
- Can be customized:
  - The format of the document templates can be customized for your organization by using the **onbrand** package
  - The report contents can be customized by creating a copy of the included YAML file and modifying it to suit your needs

https://nlmixr2.github.io/nlmixr2rpt/

ruminate

John Harrold

- Facilitates the exploration of PMx data
- Shiny interface to different tools for data transformation (**dplyr** and **tidyr**), plotting (**ggplot2**), noncompartmental analysis (**PKNCA**), and ODE-based adaptive trial simulations (**rxode2**)
- These results can be reported in Excel, Word or PowerPoint
- The state of the app can be saved and loaded at a later date
- When saved, a script is generated to reproduce the different actions in the Shiny interface

https://ruminate.ubiquity.tools/

nlmixr2autoinit, nlmixr2auto

Zhonghui Huang

- **nlmixr2autoinit** generates initial estimates for nlmixr2 models automatically
- **nlmixr2auto** automates model exploration

```
library(nlmixr2autoinit)
library(nlmixr2auto)

outs<-sf.operator(dat=pheno_sd,
            search.space = "ivbase",
            filename =  "pheno_sd",
            foldername =   "pheno_sd" )
print(outs)

# Infometrics                              Value
# ---------------------------------------  --------------
#   Dose Route                               bolus
# Dose Type                                combined_doses
# Total Number of Subjects                 59
# Total Number of Observations             155
# Subjects with First-Dose Interval Data   35
# Observations in the First-Dose Interval  35
# Subjects with Multiple-Dose Data         56
# Observations after Multiple Doses        120
# ---------------------------------------  ------
#   Estimating half-life.....................
# Half-life estimation complete: Estimated t½ = 16.44 h
# Evaluating the predictive performance of calculated one-compartment model parameters.................
# (hybrid mode: parameters combined across sources).....................
# Base PK parameter analysis finished. Estimated ka: NA, estimated CL: 0.0087, estimated Vd: 1.25
# Run parameter sweeping on nonlinear eliminiation kinetics PK parameters.....................
# Run parameter sweeping on multi-compartmental PK parameters.....................
# Running Stepwise 1. Structural Model----------------------------------------------------
#   Test number of compartments----------------------------------------------------
#   [Success] Model file created in current working directory:
#   /home/zhonghuihuang/Step_2025-08-11-pheno_sd_359e255ca82e284a7b41aefd444b39a1_temp/mod1.txt
# [Success] Model file created in current working directory:
#   /home/zhonghuihuang/Step_2025-08-11-pheno_sd_359e255ca82e284a7b41aefd444b39a1_temp/mod2.txt
# [Success] Model file created in current working directory:
#   /home/zhonghuihuang/Step_2025-08-11-pheno_sd_359e255ca82e284a7b41aefd444b39a1_temp/mod3.txt
# Analyse elimination type----------------------------------------------------
#   [Success] Model file created in current working directory:
#   /home/zhonghuihuang/Step_2025-08-11-pheno_sd_359e255ca82e284a7b41aefd444b39a1_temp/mod4.txt
# [Success] Model file created in current working directory:
#   /home/zhonghuihuang/Step_2025-08-11-pheno_sd_359e255ca82e284a7b41aefd444b39a1_temp/mod5.txt
# Test IIV on parameters----------------------------------------------------
#   [Success] Model file created in current working directory:
#   /home/zhonghuihuang/Step_2025-08-11-pheno_sd_359e255ca82e284a7b41aefd444b39a1_temp/mod6.txt
# [Success] Model file created in current working directory:
#   /home/zhonghuihuang/Step_2025-08-11-pheno_sd_359e255ca82e284a7b41aefd444b39a1_temp/mod7.txt
```

https://github.com/ucl-pharmacometrics/nlmixr2auto
https://github.com/ucl-pharmacometrics/nlmixr2autoinit