Tutorial Session 1a: Using Past Models to Bridge to Open Models and Open Science Using nlmixr2

Chairs: Matthew L. Fidler and William S. Denney

Matthew Fidler	Basics of nlmixr2/rxode2 Model Syntax
William Denney	How to import NONMEM and Monolix Models into the nlmixr2/rxode2 Model Function using nonmem2rx and monolix2rx
Mirjam Trame	Simulate New Dosing Scenarios using RxODE2
Theodoros Papathanasiou	Re-estimate New Data Using nlmixr2
Justin Wilkins	Individualize Dosing Using posologyr

Please ensure that you have all packages installed and course materials downloaded

https://github.com/nlmixr2/courses/tree/main/ACoP2024

nlmixr

Basics of nlmixr2/rxode2 Model Syntax

Matthew L. Fidler, M.Stat., Ph.D.
Director, Pharmacometrics
Novartis

Phoenix, AZ

November 10 - 13, 2024

nlmixr^2: an open-source package for pharmacometric modeling in R

ACoP Conference 2024 tutorial

Matthew Fidler

On behalf of the nlmixr2 development team:

Matt Fidler, Bill Denney, Richard Hooijmaijers, Rik Schoemaker, Mirjam Trame, Theodoros Papathanasiou, Justin Wilkins, John Harrold

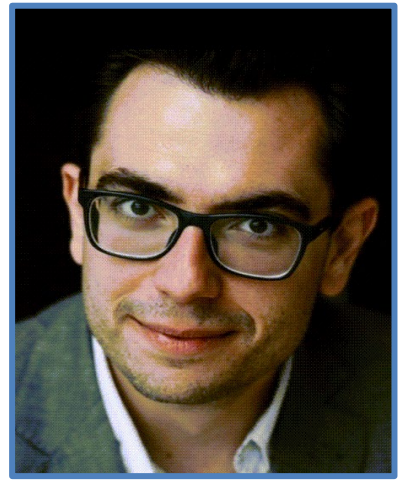
Active nlmixr2 team

Matthew Fidler, PhD

Bill Denney, PhD

John Harrold, PhD

Richard Hooijmaijers, BSc



Theo Papathanasiou, PhD

Rik Schoemaker, PhD

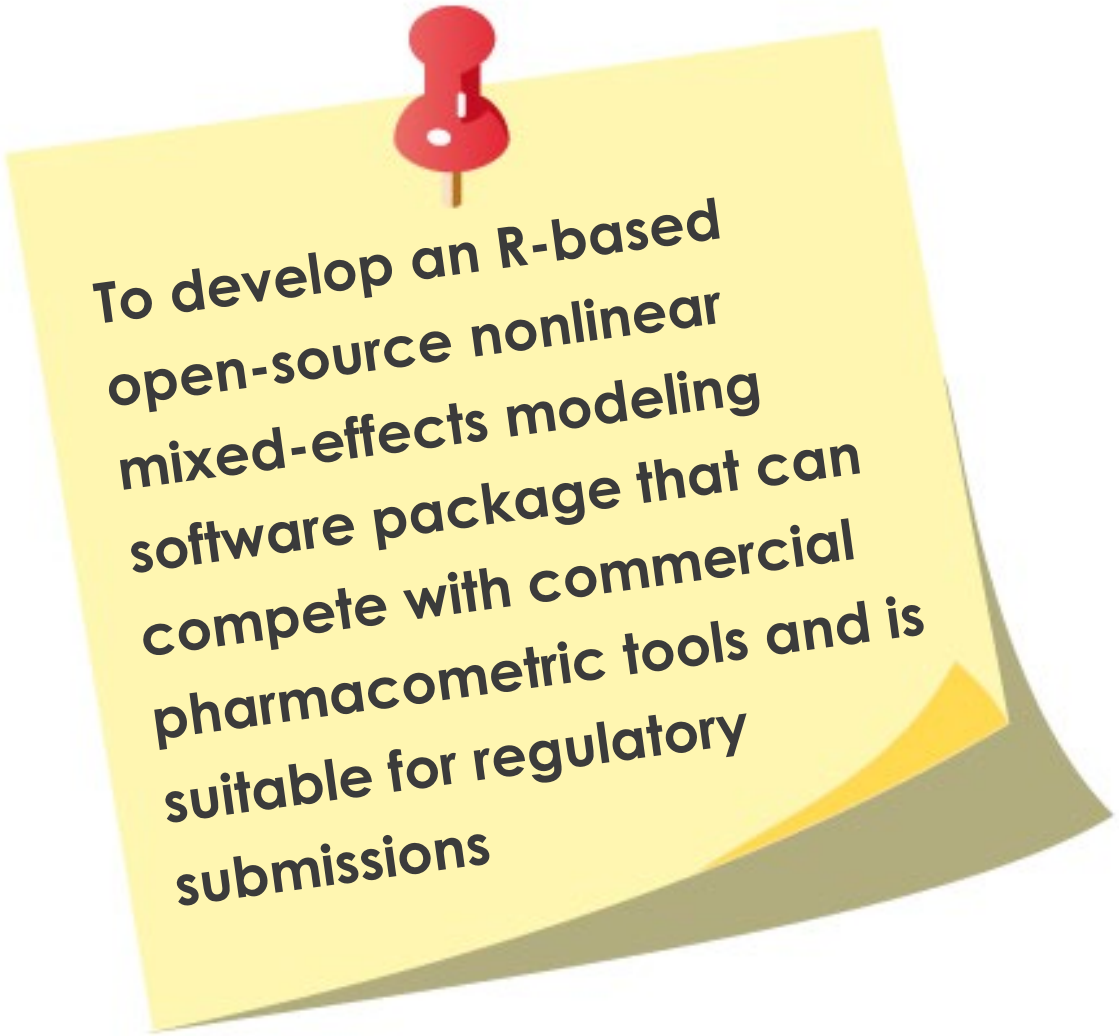
Mirjam Trame, PhD

Justin Wilkins, PhD

Max Taubert, PhD

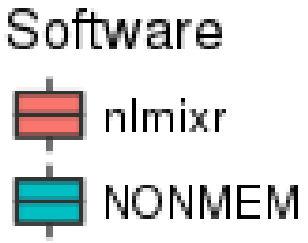
nlmixr²

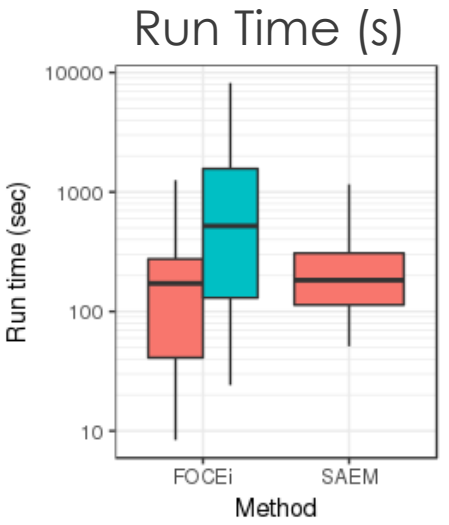
Vision of nlmixr²

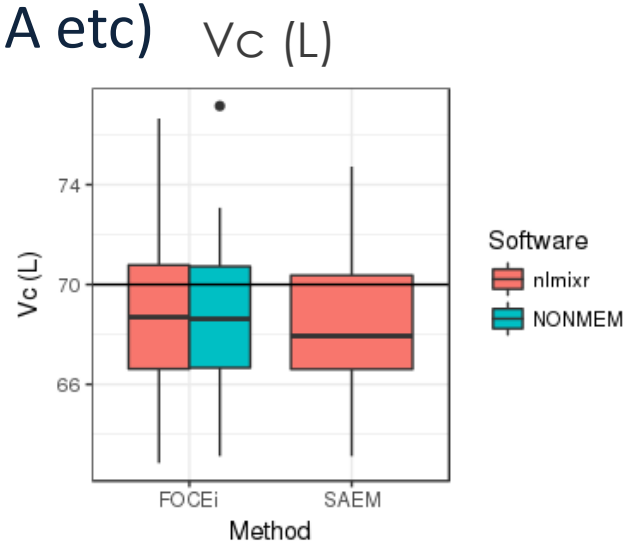


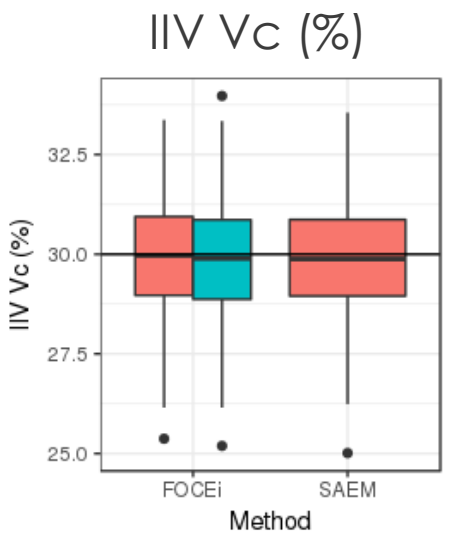
nlmixr² is a nonlinear mixed effects modeling R package with comparable performance to commercial software

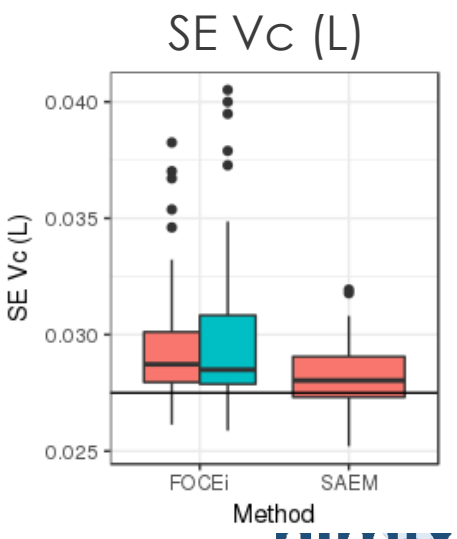
- Run time for ODE model:
  - FOCEi: nlmixr runs faster than NONMEM
  - SAEM: nlmixr runs as fast as Monolix and both are faster than NONMEM
- Parameter Estimates were similar for all three NLME tools.
- Multiple known submission/approval to regulators (FDA, EMA etc)



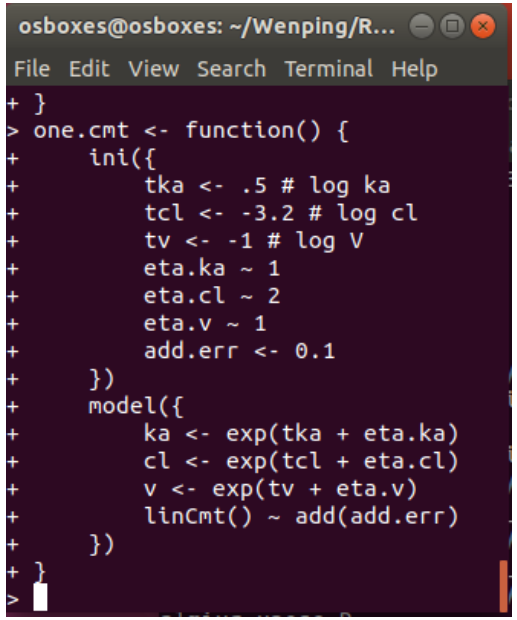








modeling syntax, running nlmixr^2 models and nlmixr^2 output



nlmixr^2

A nlmixr model has two main parts: initialization and model

Initialization ini({ })

```
ini({
    lCl  <- 1.6; label("log Cl (L/hr)")
    lVc   = log(90); label("log V (L)")
    lKa   = fix(1)  #log Ka (1/hr)
    add.sd = 0.2
    eta.Ka ~ 0.1 #IIV Ka
    eta.Cl ~ c(0.1)
    eta.Vc ~ c(0.005, 0.1)
})
```

label() or # (interactive only)

Lower triangular block matrix

- Population and Residual Estimates are defined using assign operators (=)
- Random Effects (ETAs) defined using a model formula (~; aka modelled by)

Model model({ })

```
model({ Relationship of Fixed/Random Pars First
    Cl = exp(lCl + eta.Cl)
    Vc = exp(lVc + eta.Vc)
    KA = exp(lKa + eta.Ka)

    linCmt() ~ add(add.sd)

})
```

- Parameters defined based on ini block
- Fixed/Random relationships defined first
- Model (Solved/RxODE) defined next
- Unexplained error defined by formula (~)

nlmixr uses defined parameters to select 1, 2 or 3 solved compartment model with linCmt() → closed-form solutions

Solved System Parameterization Support

1 Compartment	2 Compartment	3 Compartment
Cl, V	Cl, V, Q, Vp	Cl, Vc, Q1, Vp1, Q2 Vp2
Kel, V	Kel, k12, k21, V	Kel, k12, k21, k13, k31, V
A, alpha	A, alpha, B, beta	A, alpha, B, beta, C, gamma

nlmixr also uses parameter aliases; Examples:

- V =Vc = V1 and Q = Cld.
- Parameter case does not matter

Parameter aliases are context dependent.

- The first can be Volume = Vc, (Can start with V2)
- Second numbered Volume = Vp
- All NONMEM style parameters are supported.

CMT #1 = depot (w/Ka) / central (without Ka) compartment

Model model({ })

```
model({
      Cl  = exp(lCl + eta.Cl)
      Vc  = exp(lVc + eta.Vc)
      KA  = exp(lKa + eta.Ka)
      Vp  = exp(lVp)
      Cld = exp(lCld)
      linCmt() ~ prop(prop.sd)
})
```

- 1 compartment solved model is specified by linCmt()
- 2 and 3 compartment model is also specified by linCmt()
- Type of model depends on provided parameters

https://nlmixr2.github.io/rxode2/articles/rxode2-model-types.html#solved-compartment-models

A nlmixr model block in case of no closed-form solution or PD model and ODE model block is required → linCmt cannot be used

Initialisation ini({ })

```
ini({
    lCl    = 1.6; label("log Cl (L/hr)")
    lVc    = log(90); label("log V (L)")
    lKa    = 1        #log Ka (1/hr)
    prop.sd = 0.2
    eta.Ka ~ 0.1 #IIV Ka
    eta.Cl + eta.Vc ~ c(0.1,
                        0.005, 0.1)
})
```

label() or # (interactive only)

Lower triangular block matrix

- Population and Residual Estimates are defined using assign operators (=)
- Random Effects (ETAs) defined using a model formula (~; aka modelled by)

Model model({ })

```
model({ Relationship of Fixed/Random Pars First
    Cl  = exp(lCl + eta.Cl)
    Vc  = exp(lVc + eta.Vc)
    KA  = exp(lKa + eta.Ka)
    kel = Cl / Vc
    d/dt(depot) = -KA*depot
    d/dt(centr) =  KA*depot-kel*centr
    cp = centr / Vc
    cp ~ prop(prop.sd)
})
```

➢ instead of linCMT

- Parameters defined based on ini block
- Fixed/Random relationships defined first
- Model (Solved/RxODE) defined next
- Unexplained error defined by formula (~)

Add Bioavailability (F) and lag time (alag) to the model

Initialisation ini({ })

```
ini({
    lCl   = 1.6; label("log Cl (L/hr)")
lVc   = log(90);   label("log V (L)")
    lKa   = 1      #log Ka (1/hr)
    lf    = log(1)
    lalag = log(0.5)
    prop.sd = 0.2
    eta.Ka ~ 0.1 #IIV Ka
    eta.Cl + eta.Vc ~ c(0.1,
                        0.005, 0.1)
})
```

label() or # (interactive only)

Lower triangular block matrix

- Population and Residual Estimates are defined using assign operators (**=**)
- Random Effects (ETAs) defined using a model formula (**~**; aka modeled by)

Model model({ })

```
model({
    Cl    = exp(lCl + eta.Cl)
    Vc    = exp(lVc + eta.Vc)
    KA    = exp(lKa + eta.Ka)
    fD    = exp(lf)
    lagD  = exp(lalag)
    kel = Cl / Vc
    d/dt(depot) = -KA*depot
    alag(depot) = lagD
    f(depot)    = fD
    d/dt(centr) = KA*depot-kel*centr
    cp = centr / Vc
    cp ~ prop(prop.sd)
})
```

Relationship of Fixed/Random Pars First

Can also add rate/dur for modeled duration and rate

Residual Error models and Multiple Endpoints

Error Model	Coding	Supported By
Additive/Normal	Y ~ add(add.sd)	nlme, fo, foi, foce, focei, saem
Proportional	Y ~ prop(prop.sd)	nlme, fo, foi, foce, focei, saem
Additive + Proportional	Y ~ add(add.sd) + prop(prop.sd)	nlme, fo, foi, foce, focei, saem
Lognormal/Exponential ***Note*: normal scale OBJF**	Y ~ lnorm(lnorm.sd)	fo, foi, foce, focei, saem
Power Model	Y ~ pow(pow.sd, pow)	fo, foi, foce, focei, saem
Additive + Power	Y ~ add(add.sd) + pow(pow.sd, d)	fo, foi, foce, focei, saem
Box-Cox transform both sides	Y ~ add(add.sd) +boxCox(lambda)	fo, foi, foce, focei, saem
Yeo-Johnson transform both sides	Y ~ add(add.sd) + yeoJohnson(lambda)	fo, foi, foce, focei, saem

Multiple Endpoint:

```
PK ~ add(add.sd) + prop(prop.sd)   | depot
PD ~ add(pd.sd)                    | err
```

Now generalized llik for focei

Finalizing and checking a nlmixr model verifies nlmixr detects the correct solved model (or RxODE model), as well as showing the parsed initial estimates

Finalising models

```
osboxes@osboxes: ~/Wenping/R...
File  Edit  View  Search  Terminal  Help
+ }
> one.cmt <- function() {
+     ini({
+         tka <- .5 # log ka
+         tcl <- -3.2 # log cl
+         tv <- -1 # log V
+         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)
+     })
+ }
>
```

To finalize a model, put the **`ini`** and **`model`** in a named function

Checking how the model is parsed

```
osboxes@osboxes: ~/Wenping/RxODE
File  Edit  View  Search  Terminal  Help
> nlmixr(one.cmt)
── 1-compartment model with first-order absorption in terms of Cl ──
── Initialization: ──
Fixed Effects ($theta):
 tka  tcl   tv
 0.5 -3.2 -1.0

Omega ($omega):
       eta.ka eta.cl eta.v
eta.ka      1      0     0
eta.cl      0      2     0
eta.v       0      0     1
── Model: ──
        ka <- exp(tka + eta.ka)
        cl <- exp(tcl + eta.cl)
        v <- exp(tv + eta.v)

>
```

By calling **`nlmixr`** on the named R function, it will tell you how **nlmixr** parsed the model; This is especially useful in checking what solved system **nlmixr** detected before running the entire model

Fitting nlmixr models takes the estimation method (with its options) and produces a nlmixr combined dataset/fit object

```
fit <- nlmixr(one.cmt, data, est = "saem", table=tableControl(cwres=TRUE, npde=TRUE))
```

Assigned R object

Function Name is run name

Estimation methods = ("nlme", "saem", "focei", "foce", "foi", "fo")

Optional if cwres and or npde calculations wanted

```
> fit
── nlmixr SAEM(Solved); OBJF calculated from FOCEi approximation f
       OBJF     AIC      BIC Log-likelihood Condition Number
FOCEi 116.102 130.102 150.2816        -58.051         20.20522

── Time (sec; fit$time): ──────────────────────────────────────────
   saem    setup optimize covariance table
124.263 243.7769 0.048766      5e-06 0.413
```

```
── Population Parameters (fit$parFixed): ──────────────────────────
     Parameter    Est.      SE  %RSE Back-transformed(95%CI) BSV(CV%)
tka    log Ka  0.4503  0.1935 42.97     1.569 (1.074, 2.292)   72.12%
tcl    log Cl -3.216 0.08164 2.539 0.04013 (0.0342, 0.0471)   26.85%
tv      log V -0.7836 0.04353 5.556  0.4568 (0.4194, 0.4974)   13.55%
add.err       0.6919                                 0.6919

Shrink(SD)%
tka     -1.050%
tcl      4.763%
tv       9.939%
add.err
```

Labels

Model Based Est.

Transformed to be on "natural" scale. CIs are on the same scale

η in %CV/SD

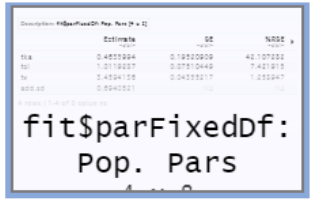
$$1 - \frac{SD(\eta)}{\omega}$$

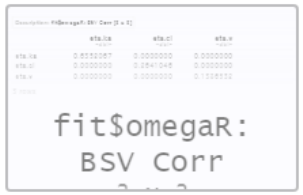
```
Covariance Type (fit$covMethod): fim
No correlations in between subject variability (BSV) matrix
Full BSV covariance (fit$omega) or correlation (fit$omegaR; diagonals=SDs)
Distribution stats (mean/skewness/kurtosis/p-value) available in fit$shrink
```

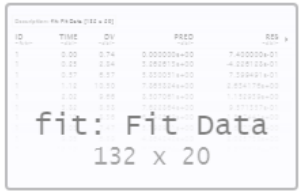
```
── Fit Data (object fit is a modified tibble): ────────────────────
# A tibble: 132 x 25
  ID     TIME    DV  PRED    RES   WRES IPRED   IRES
* <fct> <dbl> <dbl> <dbl>  <dbl>  <dbl> <dbl>  <dbl>
1 1     0      0.740  0    0.740  1.07     0   0.740
2 1     0.250  2.84   2.82 0.0178 0.0105   3.85 -1.01
```

In Rstudio's Rmarkdown or notebook, the output is similar but in tabular form that is easier to click through

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







Description: fit$parFixedDf: Pop. Pars [4 x 8]

	Estimate <dbl>	SE <dbl>	%RSE <dbl>	Back-transformed <dbl>	CI Lower <dbl>	CI Upper <dbl>	BSV(CV%) <dbl>
tka	0.4635994	0.19520909	42.107282	1.5897859	1.084367	2.330778	70.50083

Inclusion of Covariates into a SAEM nlmixr model

Initialization ini({ })

```
ini({
    lCl      = 1.6      #log Cl (L/hr)
    lVc      = log(90)  #log V (L)
    lKa      = fix(1)   #log Ka (1/hr)
    beta.wt = 0.75      #estimate of covariate effect
    prop.sd = c(0,0.2,1)
    eta.Ka  ~ 0.1 #IIV Ka
    eta.Cl + eta.Vc ~ c(0.1,
                        0.005, 0.1)
})
```

```
Cl = exp(   tCl   +   eta.Cl   +   beta.wt * lnWt70   )
```

- `tCl`: Fixed or Population Parameter
- `eta.Cl`: Random or Individual Parameter
- `beta.wt * lnWt70`: Covariate Estimate times transformed covariate

$$\exp(t_{Cl} + e_{Cl})\left(\frac{\mathrm{WT}}{70}\right)^{\mathrm{WT}_{CL}}$$

$$\exp(t_{Cl} + e_{Cl} + \mathrm{WT}_{CL} \cdot \mathrm{logWt70})$$

Resources, documentation and further reading

- Home of nlmixr2, rxode2, xpose.nlmixr2, support packages (most recent versions)
  - https://github.com/nlmixr2 New version of nlmixr
- Documentation: continually evolving
  - https://nlmixr.org/
- Open course material and recent discussion:
  - https://blog.nlmixr2.org
- LinkedIn: https://www.linkedin.com/groups/8621368/

How to import NONMEM and Monolix Models into the nlmixr2/rxode2 Model using nonmem2rx and monolix2rx

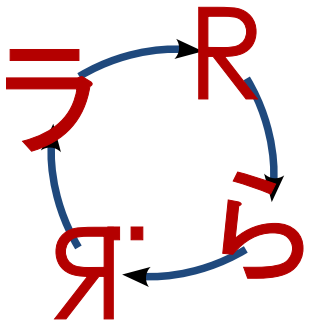
William S. Denney, Ph.D.
Chief Scientist
Human Predictions

Phoenix, AZ

November 10 - 13, 2024

Hands-on for importing NONMEM models into nlmixr2/rxode2 (nonmem2rx)

- Please load “nonmem2rx_example.R”
- nonmem2rx and monolix2rx
  - Loads existing models from NONMEM or Monolix
  - Validates model loading
    - Rerun the model using the population and individual parameters.
    - Does rxode2/nlmixr2 get the same answer as the original model?
- nonmem2rx and monolix2rx can
  - Enable easy simulation of that model
  - Enable easy reporting of that model
  - Enable easy reproduction of that model in different software



nlmixr

Simulate New Dosing Scenarios using RxODE^2

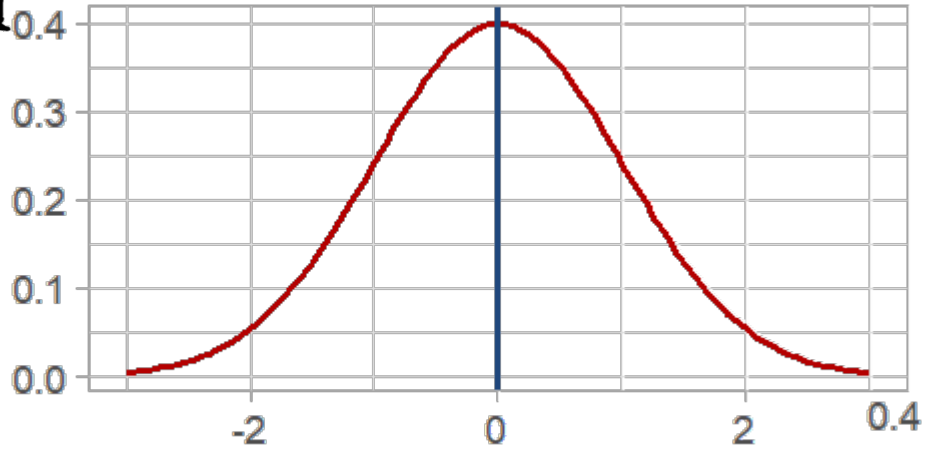
Mirjam N. Trame, Pharm.D., Ph.D.
VP, Pharmacometrics
Certara Drug Development Solutions

Phoenix, AZ

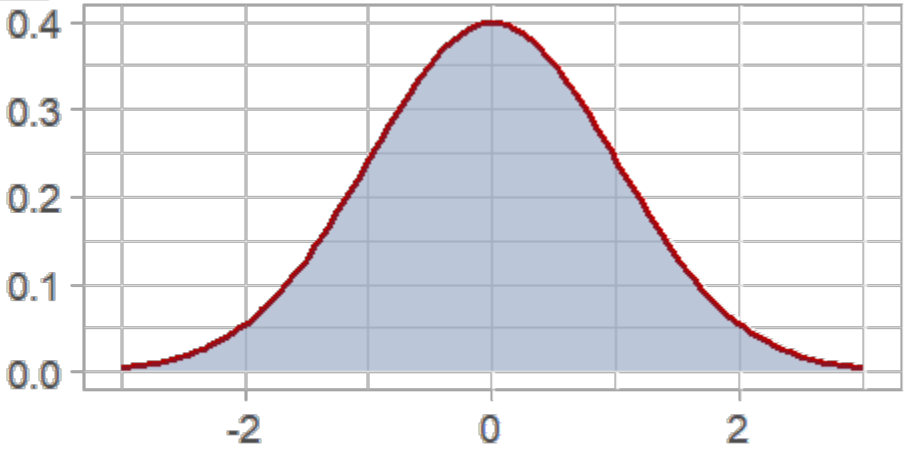
November 10 - 13, 2024

simulating from a nlmixr^2 model

multi-subject



Without uncertainty



With uncertainty

nlmixr^2

Excursion to RxODE^2

RxODE^2 is the ODE solving engine that powers the estimation methods of nlmixr^2

What do I need to know about RxODE^2?

What in RxODE^2 is used by nlmixr^2?

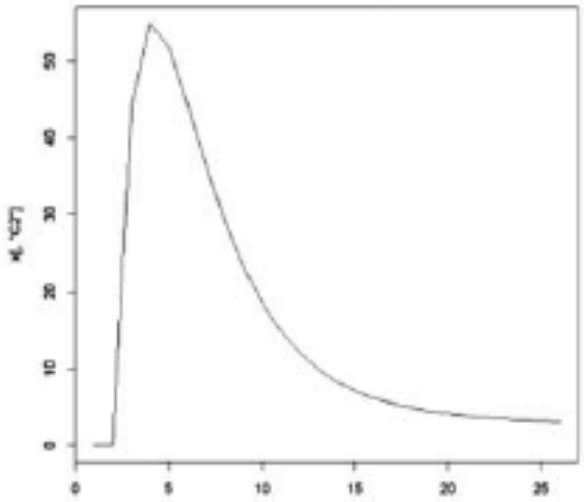
What in RxODE^2 is output by nlmixr^2?

Specify a variety of dosing schedules using the add.dosing() function

Single dose `ev$add.dosing(dose=10000, nbr.doses=1)`

Can also use NONMEM-style events by et

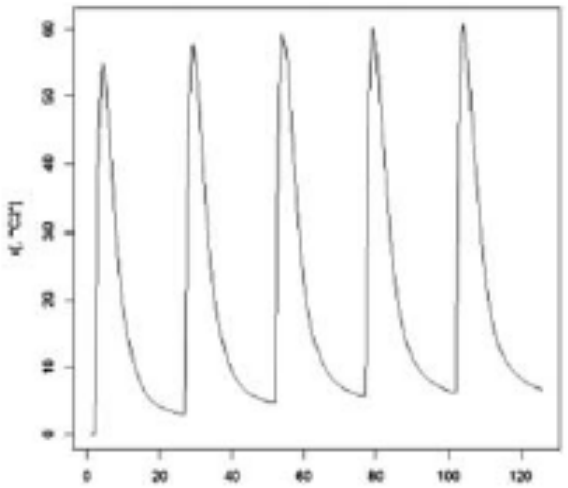
et(amt=10000)



Multiple doses `ev$add.dosing(dose=10000, nbr.doses=5, dosing.interval=24)`

Can also use NONMEM-style events by et

et(amt=10000, addl=4, ii=24)



Wang W. et al. A Tutorial on RxODE: Simulating Differential Equation Pharmacometric Models in R. CPT:PSP (2016)5,3-10

Creating an eventTable with et(), add.sampling(), add.dosing()


```
add.sampling = function(      Sampling Times   )
add.dosing = function(
    dose,                     # dose amount
    nbr.doses,                # number of doses

    dosing.interval=24,       # dosing interval
    dosing.to=1,              # where to dose
    rate=NULL,                # infusion rate
    start.time=0              # dosing start time
)
```

```
et = function(  Sampling times)
et = function(
amt,              # dose amount
addl,             # additional doses
                  nbr.doses - 1
ii,               # dosing interval
cmt,              # where to dose
rate              # infusion rate
time              # dosing start time
)
```

add.dosing(), add.sampling() or et() can be called incrementally multiple times

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

Back to Simulating with nlmixr^2

Simulating using nlmixr^2
Simulate a new regimen (BID) – No Parameter Uncertainty

Load nlmixr2	`library(nlmixr2)`

Specify ODE models	`model({`
	`  C2 = centr/V2;`
	`  d/dt(depot) =-KA*depot;`
	`  d/dt(centr) = KA*depot - CL*C2;`
	`})`

Specify Dosing & Sampling	`ev <- eventTable()# For nlmixr & RxODE simulations`
	`ev$add.dosing(dose=4.7,nbr.doses=2,dosing.interval=12) #assume BID`
	`ev$add.sampling(seq(0,24,length.out=51))`

Simulate Events	`n <- 300`
	`bid <- simulate(fit, events=ev, nSub=n*n)`
	Solved RxODE2 Object
	nlmixr2 Object
	Simulation Output: sim.id, time, ipred, sim

Simulation Output

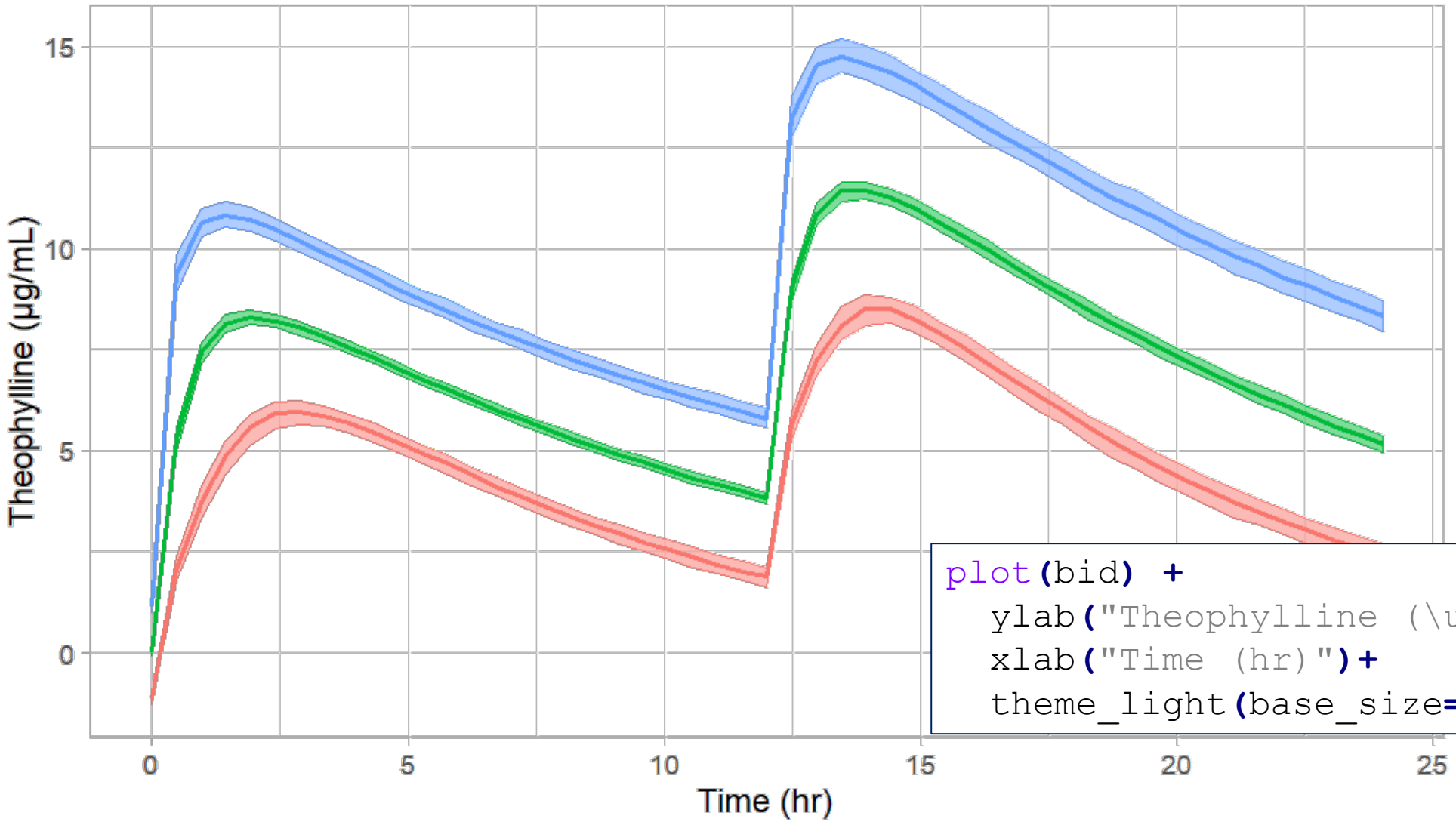
The simulation has information about the parameters that were used in the simulation from the model fit (= Solved RxODE2 object)

```
> bid
── Solved RxODE object ──
-- Parameters (bid$params): ------------------------------------------
# A tibble: 90,000 x 7
  sim.id eta.ka   tka   eta.cl   tcl   eta.v     tv
   <int>  <dbl> <dbl>    <dbl> <dbl>   <dbl>  <dbl>
1      1  0.146 0.455  0.311   -3.22  0.121  -0.785
2      2 -0.529 0.455  0.243   -3.22 -0.393  -0.785
3      3 -0.600 0.455 -0.0181  -3.22  0.0209 -0.785
4      4 -0.823 0.455  0.00427 -3.22 -0.0266 -0.785
5      5 -0.316 0.455 -0.00435 -3.22
6      6  0.620 0.455  0.423   -3.22
# ... with 89,994 more rows
-- Initial Conditions (bid$inits): --
 depot center
     0      0
-- First part of data (object): -----
# A tibble: 4,590,000 x 4
  sim.id  time ipred     sim
   <int> <dbl> <dbl>   <dbl>
1      1  0     0    -0.0737
2      1  0.48  5.17  5.22
3      1  0.96  7.07  6.23
4      1  1.44  7.62  9.40
5      1  1.92  7.61  7.43
6      1  2.4   7.39  7.00
# ... with 4,589,994 more rows
```

Constant Population Parameters

```
> summary(bid$params)
     sim.id           eta.ka              tka              eta.cl
 Min.   :    1   Min.   :-2.9624044   Min.   :0.4546   Min.   :-1.2343951
 1st Qu.:22501   1st Qu.:-0.4482224   1st Qu.:0.4546   1st Qu.:-0.1794427
 Median :45001   Median :-0.0005998   Median :0.4546   Median : 0.0010281
 Mean   :45001   Mean   :-0.0018905   Mean   :0.4546   Mean   : 0.0005723
 3rd Qu.:67500   3rd Qu.: 0.4434868   3rd Qu.:0.4546   3rd Qu.: 0.1824282
 Max.   :90000   Max.   : 2.7591093   Max.   :0.4546   Max.   : 1.1282721
      tcl               eta.v               tv
 Min.   :-3.215   Min.   :-0.5714017   Min.   :-0.7848
 1st Qu.:-3.215   1st Qu.:-0.0895164   1st Qu.:-0.7848
 Median :-3.215   Median : 0.0009726   Median :-0.7848
 Mean   :-3.215   Mean   : 0.0008345   Mean   :-0.7848
 3rd Qu.:-3.215   3rd Qu.: 0.0919098   3rd Qu.:-0.7848
 Max.   :-3.215   Max.   : 0.5585632   Max.   :-0.7848
```

Simulate a new regimen (BID) – No Parameter Uncertainty
→ Simulation Plot




```
plot(bid) +
  ylab("Theophylline (\u03BCg/mL)")+
  xlab("Time (hr)")+
  theme_light(base_size=18)
```

Simulating using nlmixr^2
Simulate a new regimen (BID) – With Parameter Uncertainty
→ add one additional parameter "nStud"

Load nlmixr2	`library(nlmixr2)`

Specify ODE models	`model({`
	`  C2 = centr/V2;`
	`  d/dt(depot) =-KA*depot;`
	`  d/dt(centr) = KA*depot - CL*C2;`
	`})`

Specify Dosing & Sampling	`ev <- eventTable()# For nlmixr & RxODE simulations`
	`ev$add.dosing(dose=4.7,nbr.doses=2,dosing.interval=12) #assume BID`
	`ev$add.sampling(seq(0,24,length.out=51))`

Simulate Events	`n <- 300` `bid <- simulate(fit, events=ev, nSub=n, nStud=n)`

Solved RxODE2 Object

nlmixr2 Object

nStud = # of "Studies" sampled
nSub = # subjects per study

Simulation Output: sim.id, time, ipred, sim

https://nlmixrdevelopment.github.io/RxODE/articles/RxODE-event-table.html

Simulation Output

The simulation has information about the parameters that were used in the simulation from the model fit (= Solved RxODE2 object)

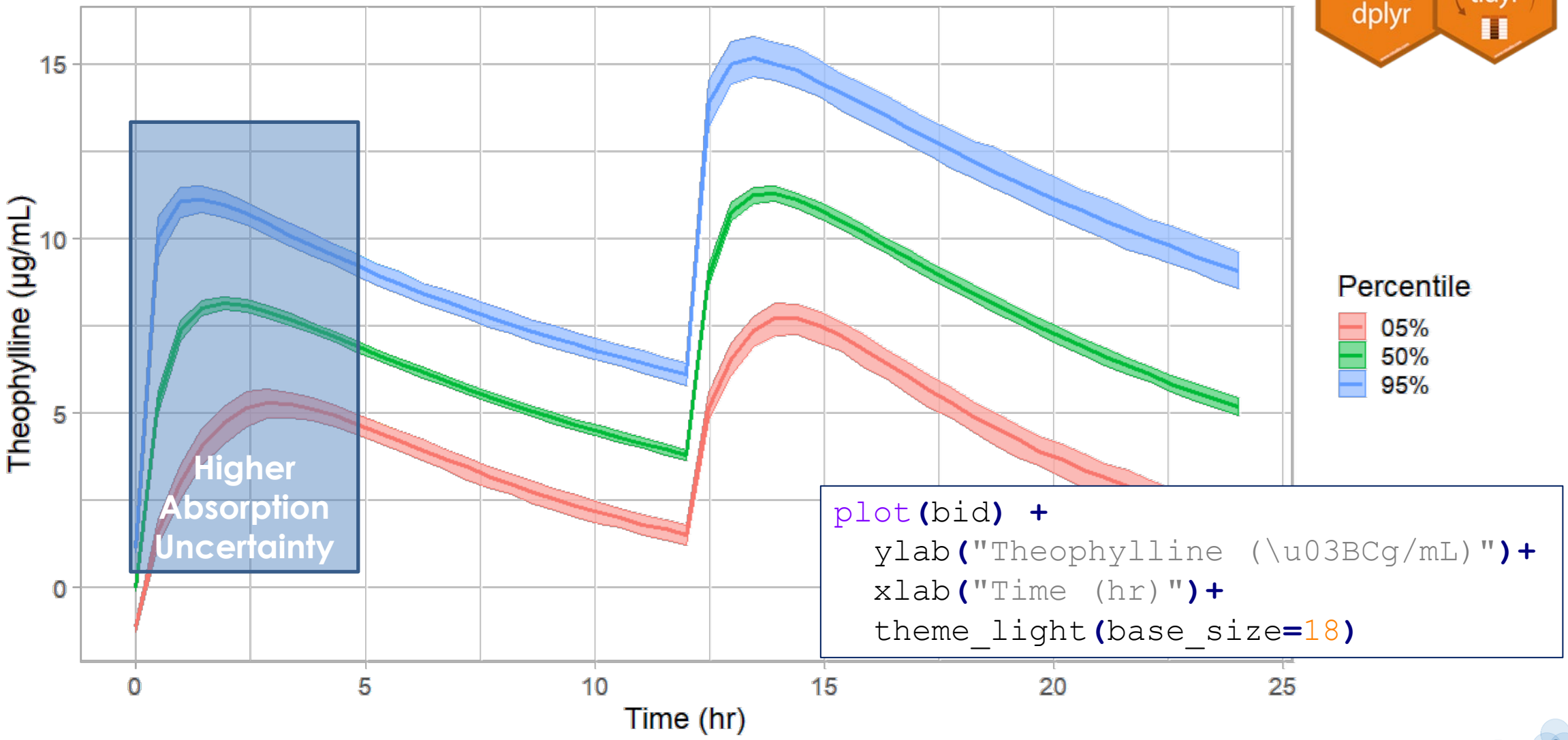
```
> bid
── Solved RxODE object ──
-- Parameters (bid$params): ------------------------------------------
# A tibble: 90,000 x 7
  sim.id eta.ka   tka  eta.cl   tcl    eta.v     tv
   <int>  <dbl> <dbl>   <dbl> <dbl>    <dbl>  <dbl>
1      1  1.11  0.199  0.193  -3.32  0.0108  -0.774
2      2 -1.80  0.199 -0.263  -3.32  0.00394 -0.774
3      3 -0.113 0.199  0.106  -3.32  0.0518  -0.774
4      4  1.05  0.199  0.0292 -3.32 -0.00367 -0.774
5      5 -2.57  0.199 -0.426  -3.32
6      6  1.31  0.199  0.311  -3.32
# ... with 89,994 more rows
-- Initial Conditions (bid$inits): -
 depot center
     0      0
-- First part of data (object): ----
# A tibble: 4,590,000 x 4
  sim.id  time ipred    sim
   <int> <dbl> <dbl>  <dbl>
1      1  0     0    0.0207
2      1  0.48  8.13 8.07
3      1  0.96  9.15 8.78
4      1  1.44  8.98 8.90
5      1  1.92  8.62 9.44
6      1  2.4   8.25 8.25
# ... with 4,589,994 more rows
```

Changing Population Parameters

```
> summary(bid$params)
     sim.id          eta.ka                tka                eta.cl
 Min.   :    1   Min.   :-5.910739   Min.   :-0.09778   Min.   :-1.3511807
 1st Qu.:22501   1st Qu.:-0.363984   1st Qu.: 0.30723   1st Qu.:-0.0734019
 Median :45001   Median : 0.001011   Median : 0.45434   Median :-0.0004093
 Mean   :45001   Mean   : 0.001559   Mean   : 0.43990   Mean   :-0.0007771
 3rd Qu.:67500   3rd Qu.: 0.364199   3rd Qu.: 0.56614   3rd Qu.: 0.0724550
 Max.   :90000   Max.   : 6.003083   Max.   : 0.94869   Max.   : 1.7316982
      tcl                eta.v                 tv
 Min.   :-3.435   Min.   :-1.0236210   Min.   :-0.9172
 1st Qu.:-3.263   1st Qu.:-0.0250715   1st Qu.:-0.8122
 Median :-3.213   Median :-0.0000393   Median :-0.7860
 Mean   :-3.212   Mean   : 0.0000356   Mean   :-0.7848
 3rd Qu.:-3.160   3rd Qu.: 0.0252582   3rd Qu.:-0.7530
 Max.   :-2.954   Max.   : 0.8727193   Max.   :-0.6597
```


nlmixr²

Simulate a new regimen (BID) – With Parameter Uncertainty
→ Simulation Plot




```
plot(bid) +
  ylab("Theophylline (\u03BCg/mL)")+
  xlab("Time (hr)")+
  theme_light(base_size=18)
```

Design evaluation and optimization with PopED, babelmixr2 and nlmixr2

Theodoros Papathanasiou, MPharm, PhD
Associate Director, Clinical Pharmacology Modeling & Simulation
GSK

Phoenix, AZ

November 10 - 13, 2024

nlmixr²

Design evaluation and optimization with PopED, babelmixr2 and nlmixr2

Theo Papathanasiou, MPharm, PhD
GSK, Zug, Switzerland

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

and the nlmixr2 Development Team

Course 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 pharmacometric tool that aims to convert nlmixr2 syntax to other commonly used tools.
- Babelmixr2 can help you by:
  - Running your nlmixr2 model in 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] RxODE packagedown: https://nlmixr2.github.io/rxode2

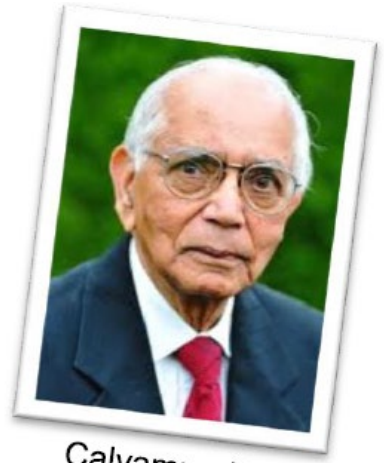
Background
Optimal design

- Clinical trial simulation limitations
  - Computationally expensive
  - Often intuition-based
- Optimal designs (OD)
  - Experimental designs that are optimal with respect to some statistical criterion
  - Allow parameters to be estimated **without bias** and with **minimum variance**

Background
Cramer-Rao inequality

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

Harald Cramér
1893 - 1985



Calyampudi
Radhakrishna Rao
1920 -

- 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 relative standard errors (RSEs) can be calculated by the diagonals of the inversed of the FIM!

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

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

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

Also $DESIGN in NONMEM 7.5

- 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 optimal design software language
- **Repetitive**: Translation needs to happen for every new modeling exercise
- NONMEM $DESIGN helps with brining optimal design and estimation/simulation in one software
- **babelmixr2** is now extending its US to seamlessly interact with PopED

Nyberg, Bazzoli, Ogungbenro, et al. (2014)

PopED syntax

- PopED requires 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 )
}
```

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 ))
}
```

PopED syntax

Design and design space definition

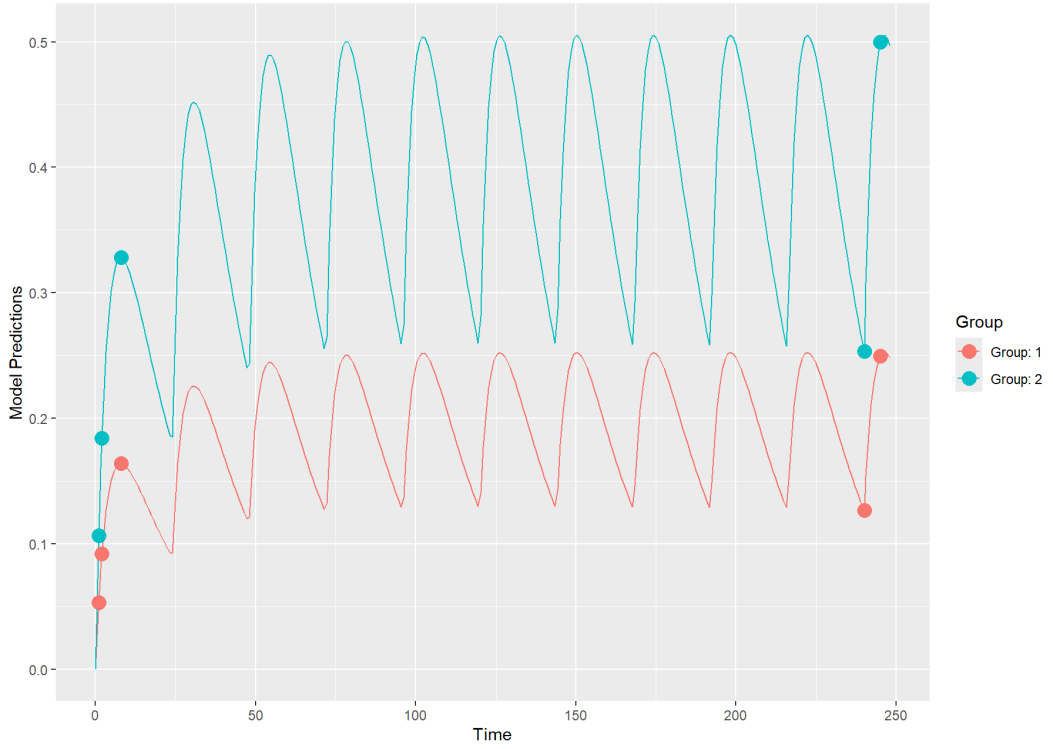
Hide

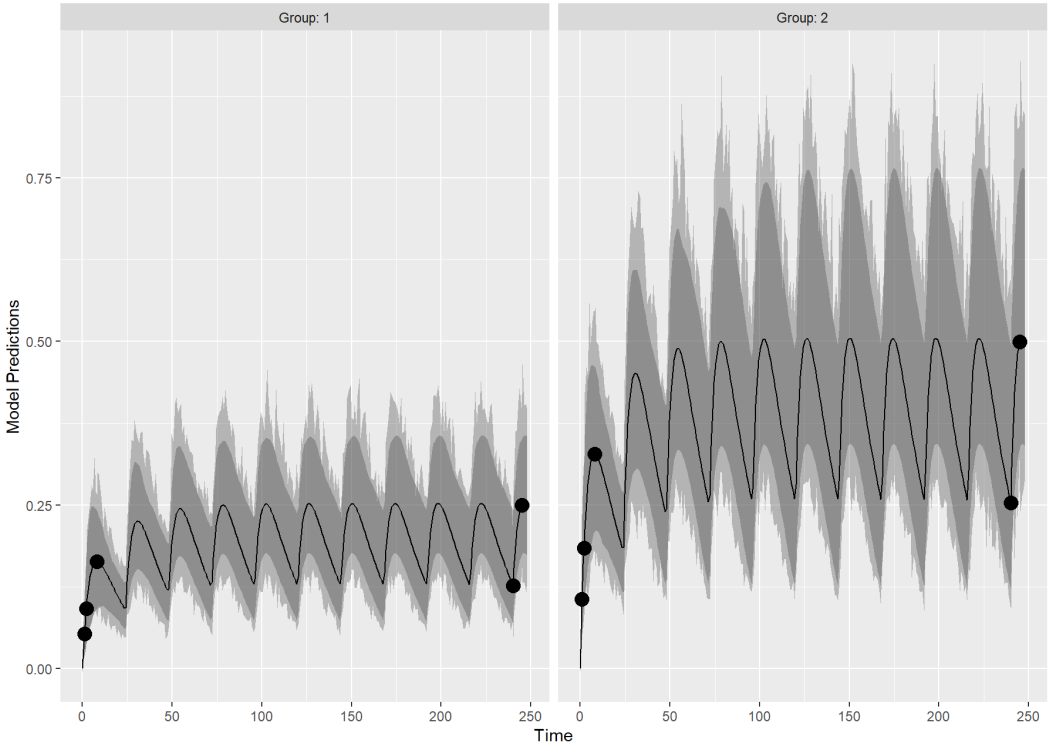
```
# Define design and design space
poped.db <- create.poped.database(
      # Definition of the user defined functions
      ff_fun="ff",
      fg_fun="sfg",
      fError_fun="feps",

      # 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))
```

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 in nlmxir2 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!

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

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

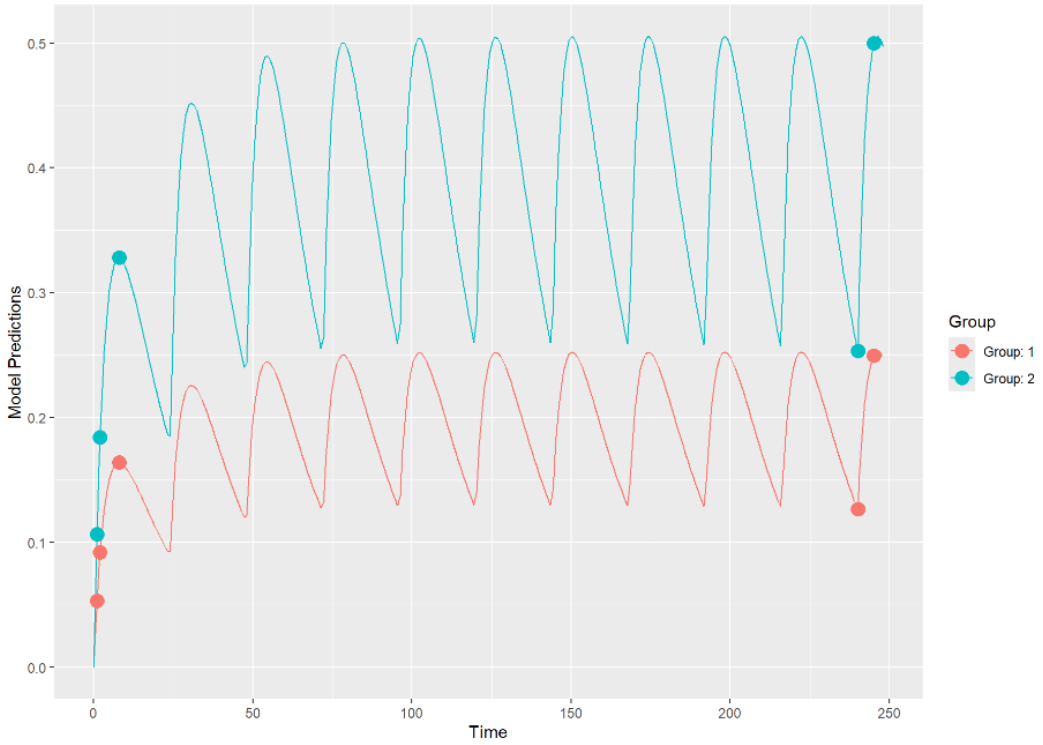
Downstream PopED functions compatibility

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
```

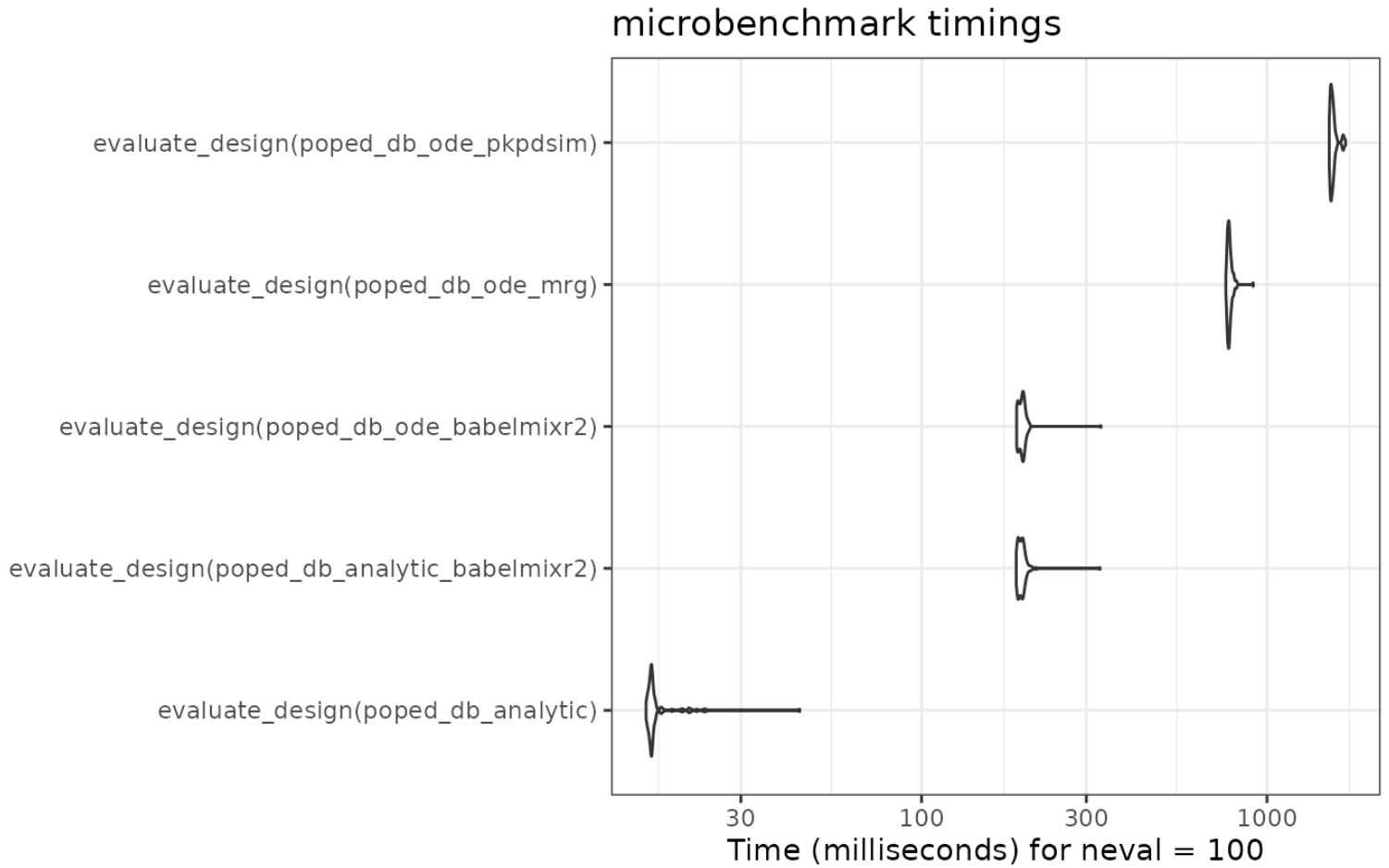

nlmixr²

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

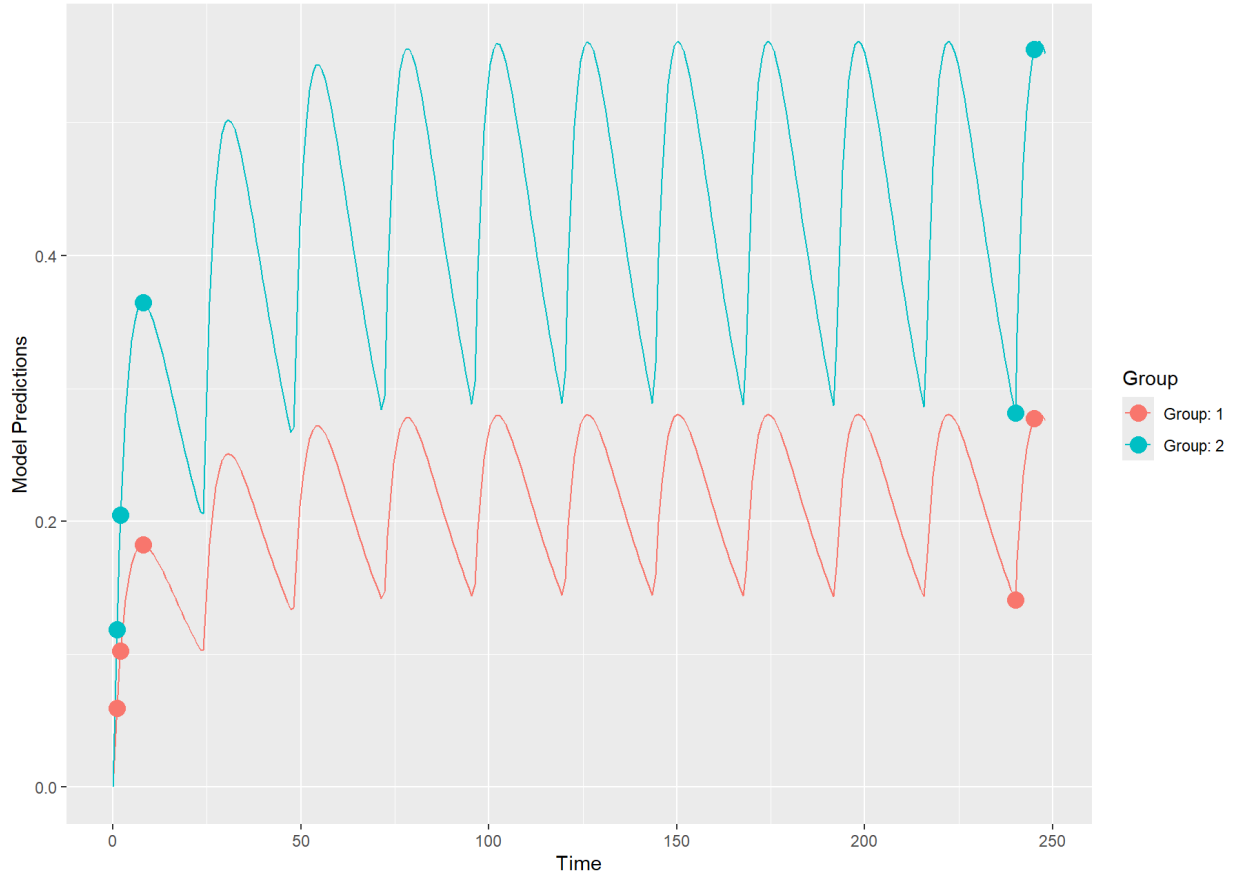
- babelmixr2 ode solver: the fastest ode solver in this comparison.
- Among other things, this is because the model is loaded into memory and does not need to be setup each time.
- Results may be problem depended
- As benchmarks: the mrgsolve, and PKPDsim implementations on the PopED website



nlmixr²

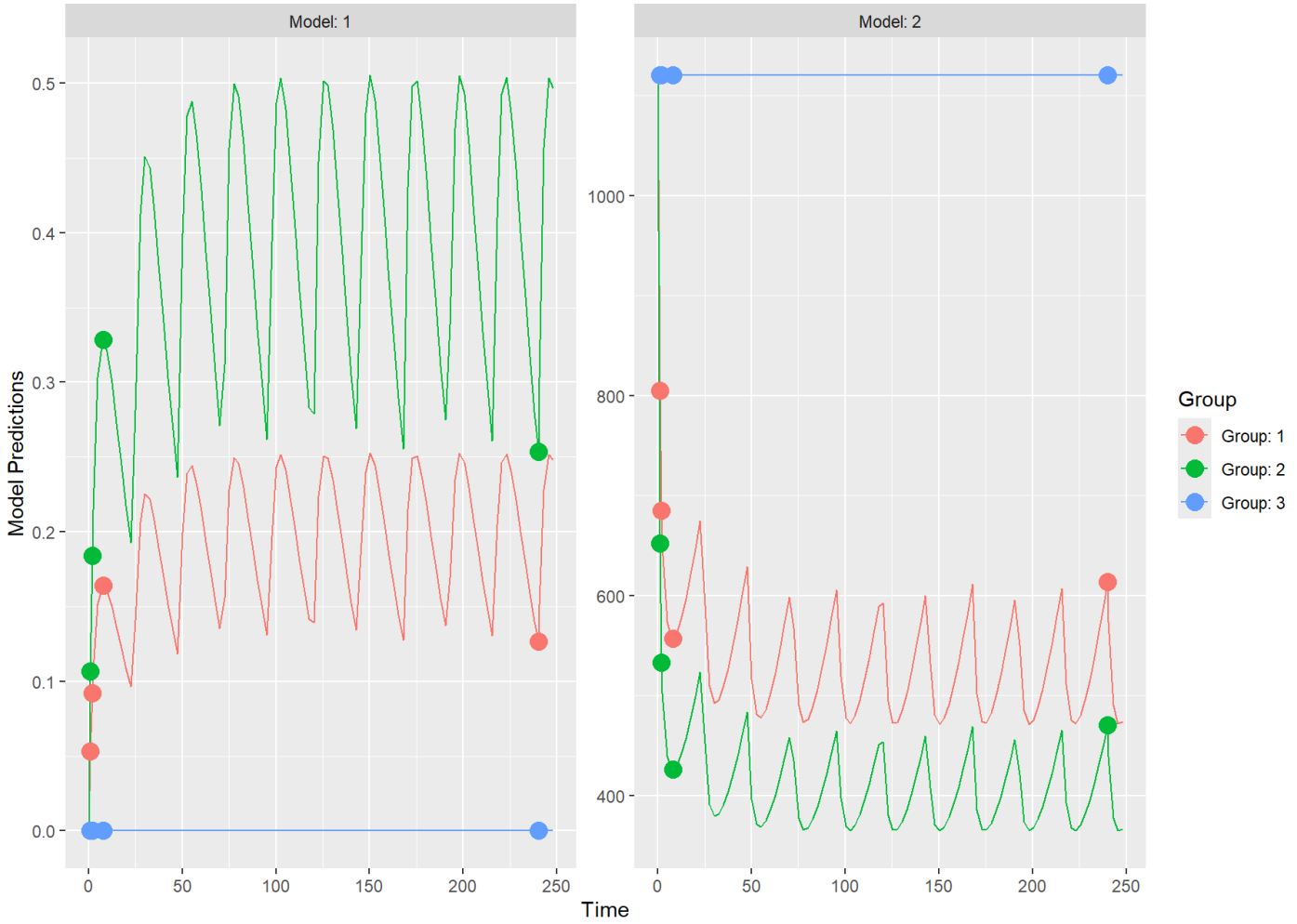
Hand-on 1: Closed form and ODE PK model

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



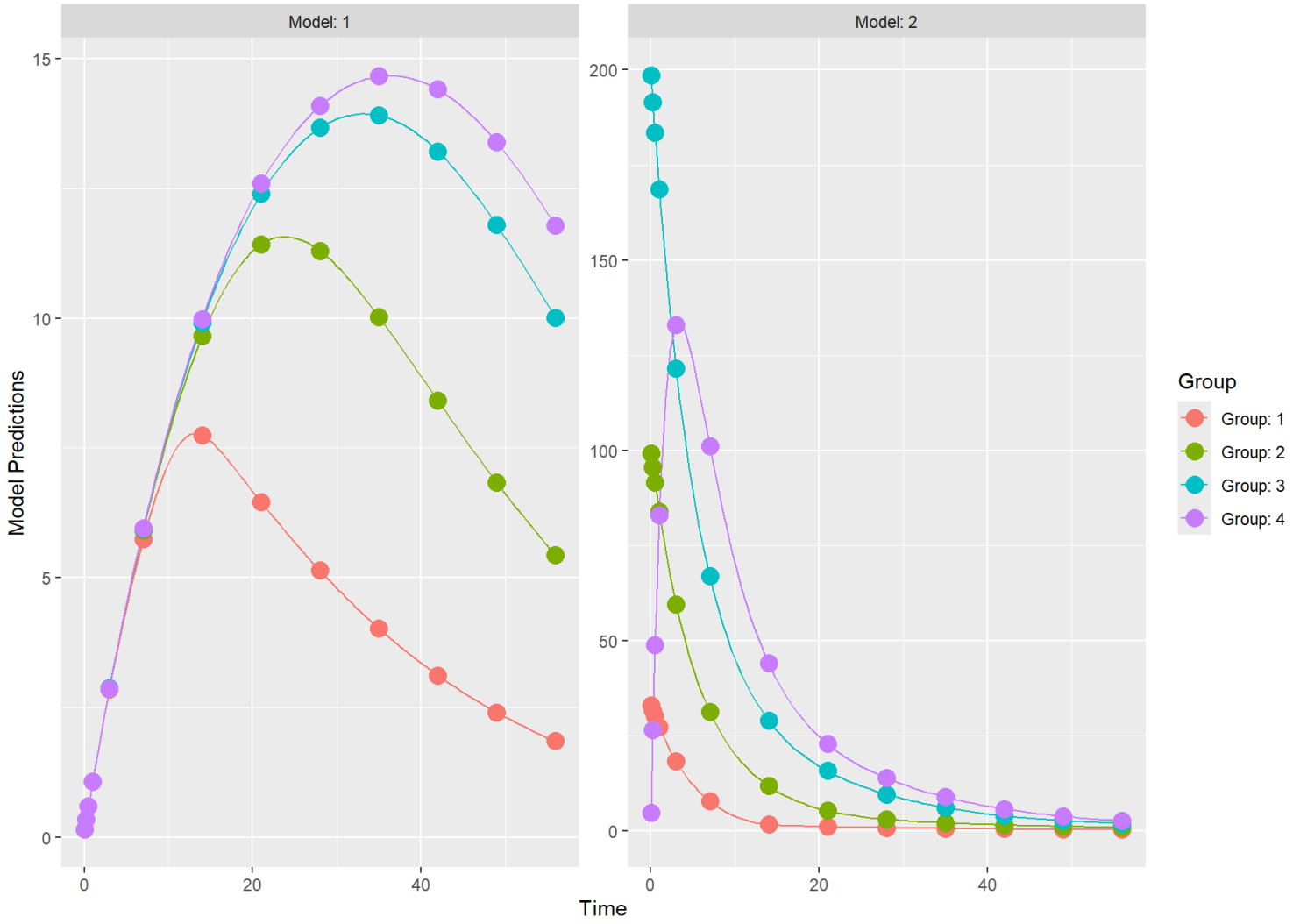
Hand-on 2: Closed form PKPD model

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



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 model conversions
- The ease of translating from NONMEM or Monolix to nlmixr2 and then PopED through babemixr2
- Our hope is that the new UI will alleviate part of the entry burden to optimal design applications

Individualize Dosing Using posologyr

Justin Wilkins, Ph.D.
Owner and Senior Consultant
Occams

Phoenix, AZ

November 10 - 13, 2024

nlmixr²

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

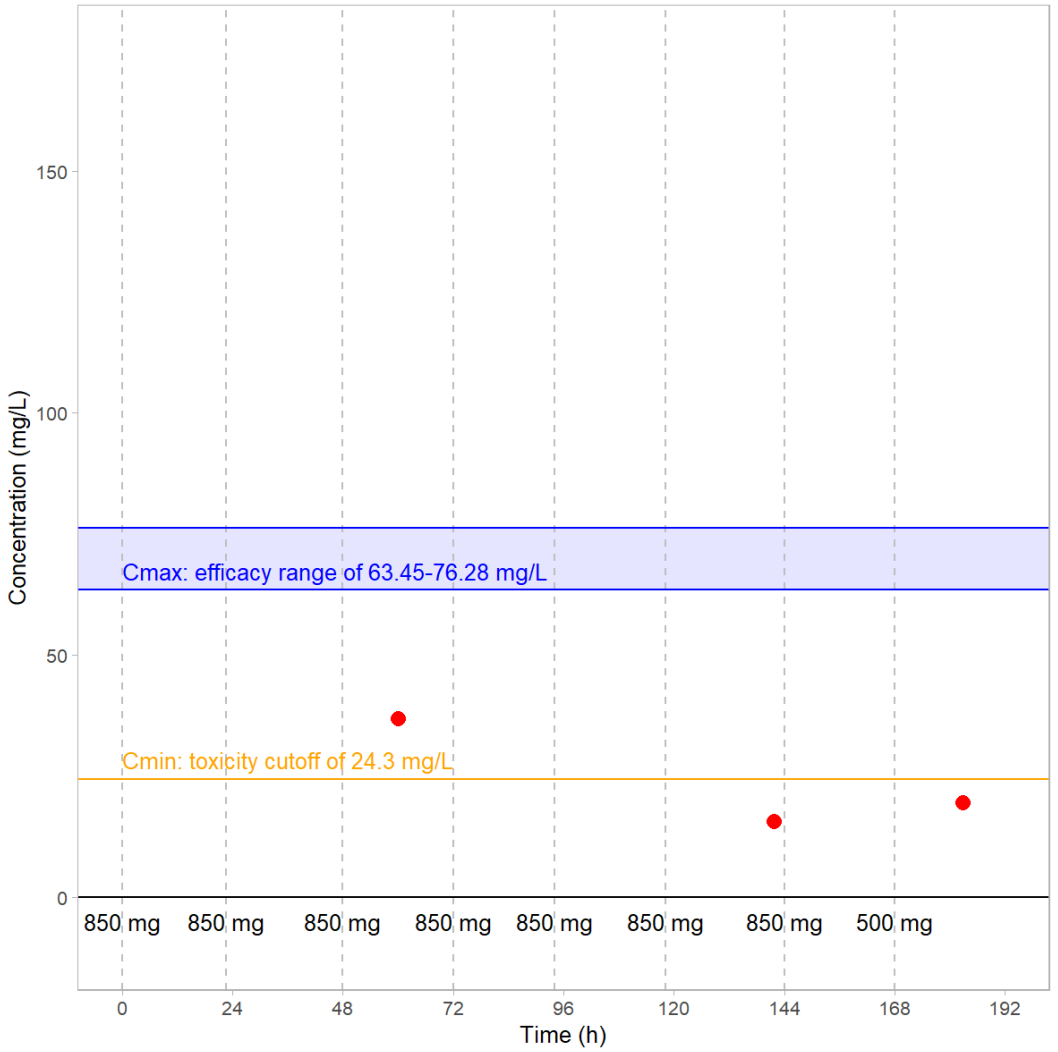
Dvorchik B, Arbeit RD, Chung J, Liu S, Knebel W, Kastrissios H. Population Pharmacokinetics of Daptomycin. Antimicrobial Agents and Chemotherapy. 2004 Aug;48(8):2799–807

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?

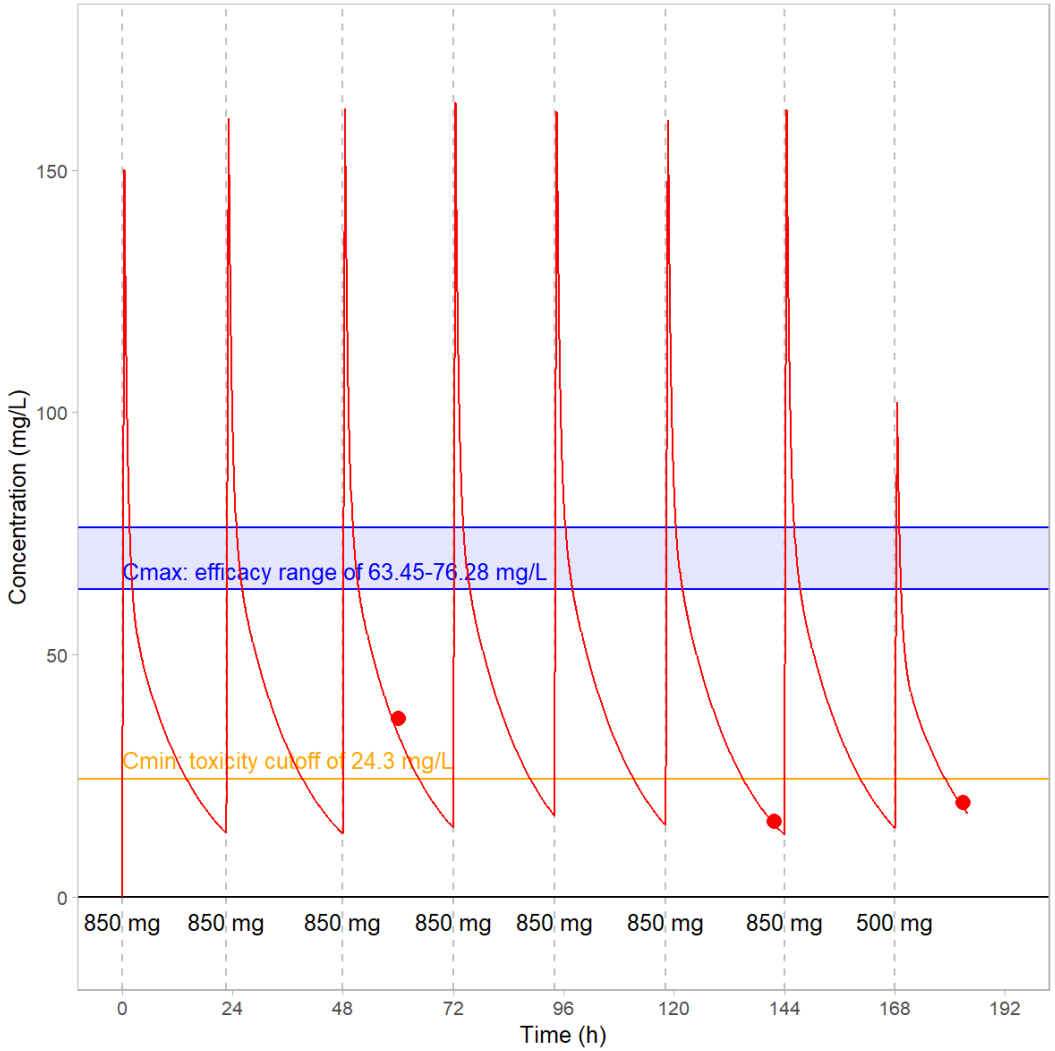
- Let's find out…

This patient's data

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

The rxode2 model

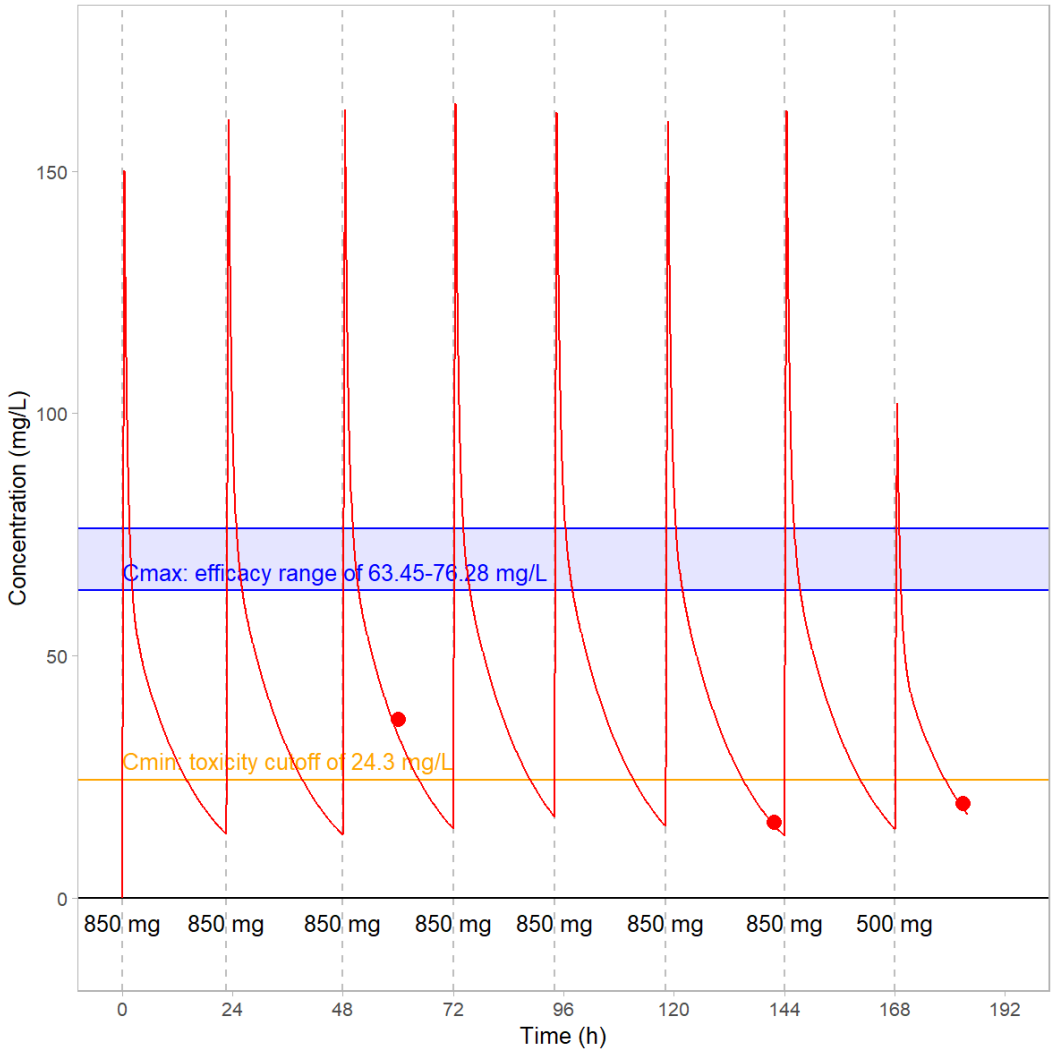
- poso_estim_map() provides:
  - Maximum *a posteriori* (MAP) individual parameters for individual patients based upon their data and the model…
  - Predicted concentration-time curve based on their dosing history
  - Rolling AUC



The dose seems to be OK, but…

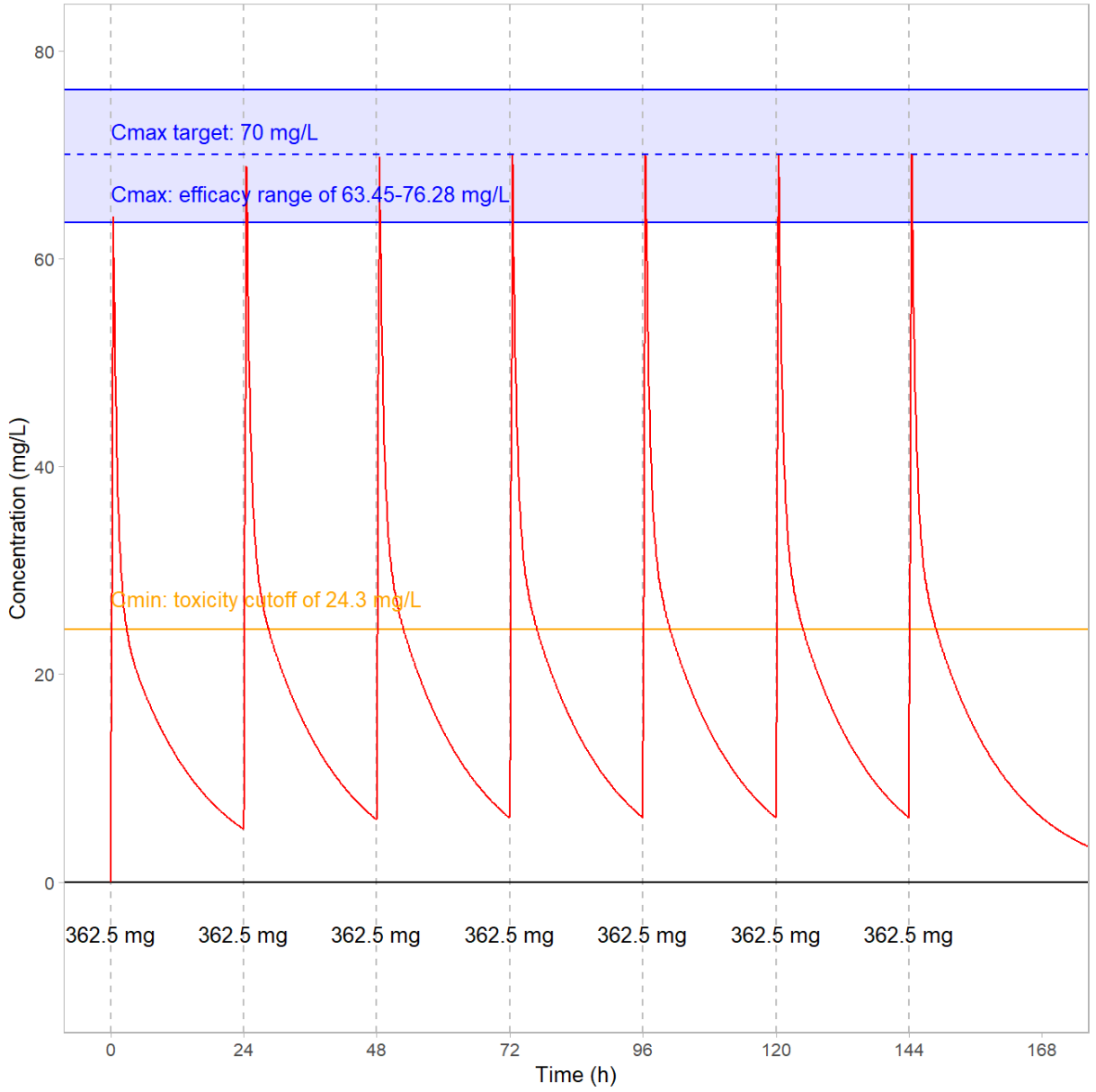
- C_{min} appears to be on target, but C_{max} might be higher than it needs to be…
- AUC between the 48 h and 72 h doses is 1003.7 mg.h/L
- Since MIC is 1 mg/L, AUC/MIC=1003.7, which is >> 666

- 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)?

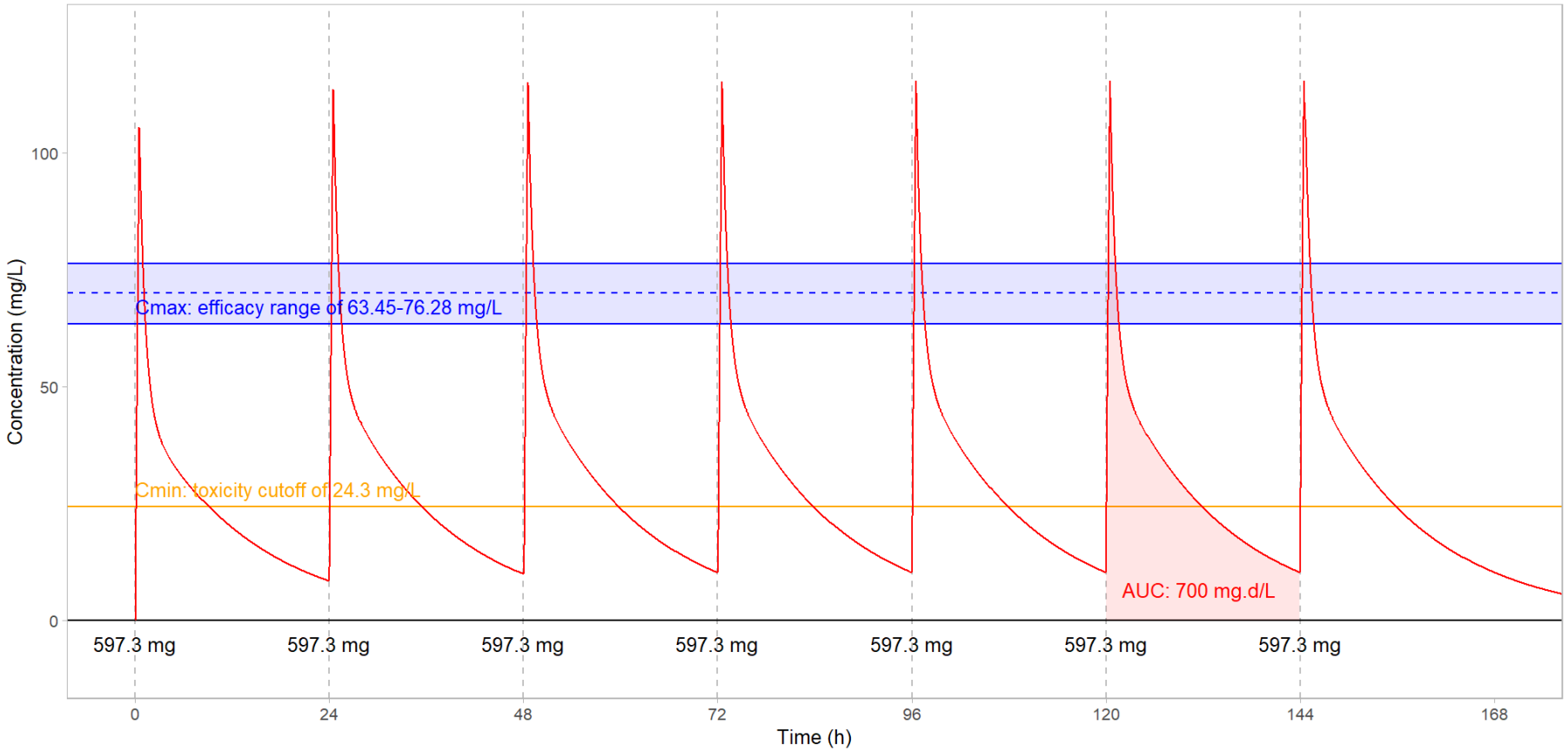
```
poso_dose_conc(dat = subset(dat, ID==17), prior_model = rx, time_c=(6*24)+0.5,
time_dose=0, target_conc = 70, duration=0.5, indiv_param=pat17, add_dose=7,
interdose_interval=24)

$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?

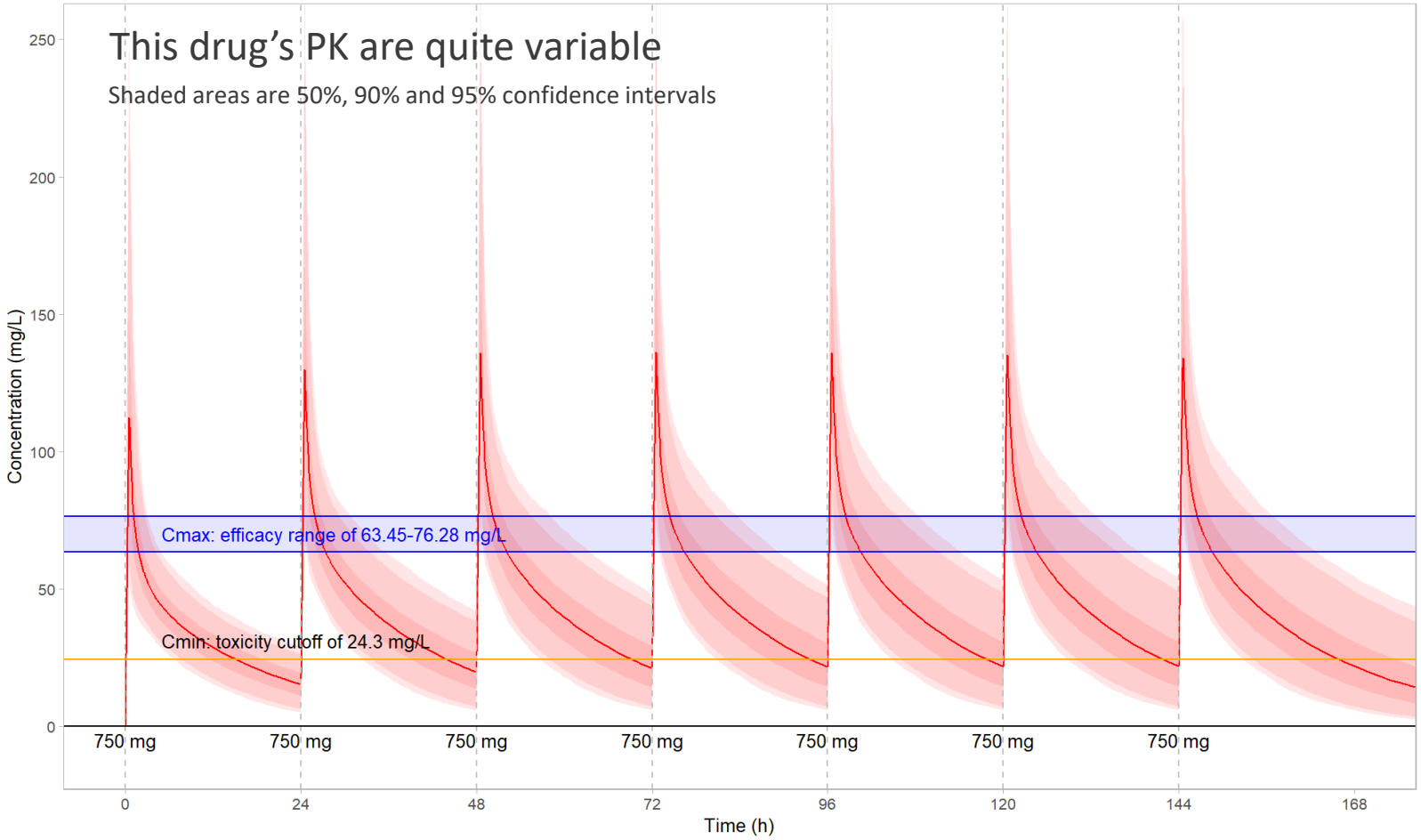
```
poso_dose_auc(dat = subset(dat, ID==17), prior_model = rx, starting_time = 144, time_auc=24, target_auc = 700, duration=0.5, indiv_param=pat17, interdose_interval = 24, add_dose = 7)
```

```
$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

nlmixr2 further direction

Matt Fidler
Novartis

nlmixr^2: a short history

Nlmixr2 a brief history (1/2)

- 2005
  - Inspiring Monolix Course
- 2005
  - First C saem
- 2011
  - Nmutil simulator
- 2015
  - First RxODE
- 2016
  - First nlmixr nlme
- 2017
  - First CRAN
- 2017
  - Saem & nlmixr2 fn
- 2018
  - fo/focei(python)
- 2018
  - ShinyMixR
- 2018
  - Multiple Endpoint
- 2019
  - First tutorial
- 2019
  - Comparison paper
- 2019
  - Lag/rate/dur/f
- 2019
  - NONMEM EVIDs
- 2019
  - Legible Parsing err
- 2021
  - Drop python

Nlmixr2 a brief history (2/2)

- 2021
  - Parallel random
- 2021
  - Unbranched
- 2021
  - Censoring
- 2022
  - Nlmixr2 &rxode2
- 2022
  - Nlmixr2=rxode2 fn
- 2022
  - ETA Std. Errors
- 2022
  - babelmixr2
- 2023
  - +9 Pop Est Methods
- 2023
  - User Functons
- 2023
  - ODE EVID expansion
- 2023
  - Merge $fitMerge
- 2024
  - monolix2rx
- 2024
  - Poped

nlmixr2: where we are and looking forward

Features Already Implemented in nlmix2/rxode2

- Time varying covariates
- Parallel ODE solving in rxode2 and saem (still needs to be worked out for other methods)
- Generalized likelihood for certain population/mixed effects model
- Censored data (M3/M4) via LIMIT, CENS columns
- User Defined functions interfacing R and possibly converting to C (with derivatives)

Estimation methods – Naive Pooled

Method	Bounded	Gradient Free	Gradient	Hessian	Likelihood
bobyqa	☑	☑	☐	☐	☑
Uobyqa	☐	☑	☐	☐	☑
optim	☐	☑	☐	☐	☑
	method="Nelder-Mead“, "SANN“ or method="Brent"				
nls	☐	☐	☑	☑	☐
nlminb	☑	☐	☑	☑	☑
nlm	☐	☐	☑	☑	☑
lbfgsb3c	☑	☐	☑	☐	☑
n1qn1	☐	☐	☑	☐	☑
optim	☑(L-BFGS-B) ☐	☐	☑	☐	☑
	method="L-BFGS-B“, "BFGS“ or "CG"				
focei	☑	☑ or ☐ (outerOpt)	☑	☐	☑

Estimation Methods – mixed effects

Method	Bounded	Gradient Free	Gradient Inner	Mu-ref linear	Parallelized	Likelihood
fo	☑	☑	☐	☐	☐	☑
foi	☐	☑	☐	☐	☐	☑
foce	☐	☑	☐	☐	☐	☑
focei	☑	☑ or ☐ (outerOpt)	☑	☐	☐	☑
Saem	☐	☑	☐	☑	☑	☐

Future estimation methods?

- adfocei – instead of using forward sensitivity for inner gradients, use automatic-differentiation
- mufocei – replace etas with phi and determine population/covariate effects by linear models
- ffocei – calculate outer gradient of problem (can possibly be mixed with mufocei)
- fsaem – Implement f-SAEM (2020)
- Gc – Gausian Quadrature

Interaction with other tools with babelmixr2

Method	Import	Notes
nonmem	nonmem2rx	Creates NONMEM control stream, runs and imports into nlmixr2 • Can help diagnose NONMEM focei issues • Validates translation (also in the import)
monolix	monolix2rx	Creates Monolix project, runs and imports into nlmixr2 • Can add CWRES to monolix model • Validates translation (also in the import, work in progress)
poped	--	
pknca	--	

Other integration future features?

- Other ODE integrators, DEsolve, PKPDsim, mrgsolve
- Other modeling frameworks (dMod, torstan, stanette)

Features on the roadmap

- Linear-compartment model update
- Between Occasion variability (and other levels of variability; can be worked around)
- Easy way to code/simulate survival models (and perhaps survival specific regression techniques)
- Neural Network-based ODEs (Implemented in PAGE poser)
- Mixture models
- Adding prior parameters to do Bayesian analysis with adjusted likelihoods (likely needed before torstan/stanette integration)
- Proper different distributions of between subject variabilities (?)
- Delay Differential Equations (?)
- Matrix Exponential / Inductive Linearization
- Other ODE methods (we have lsoda and dop853)
- Different methods of covariance (eg SIR) and likelihood calculation (for SAEM)

Missing features not currently on the roadmap

- Autocorrelation
- Non-parametric estimation

Fit integration into other tools

- Goodness of fit plots
  - xpose (via xpose.nlmixr2)
  - ggPMX
  - pmPlot (through accessing merged dataset)
- VPC
  - Vpc package (regular, pred-corrected, and censoring VPCs)
  - Other VPC packages by simulating the vpc data
- Reporting
  - Nlmixr2rpt – word reports
  - Nlmixr2 models to LaTeX equations
- Tools
  - Some covariate selection methods (like SCM, Lasso, Horseshoe-prior)
  - Automatic model selection by nlmixr2auto (UCL)
  - Bootstrapping, preconditioning
- Shiny
  - shinyMixR for run management
- Pharmpy
  - Import/export nlmixr2 models

nlmixr^2 acknowledgements

- Nlmixr team past & present
- Advisory Committee
- Novartis support & internal advocates
  - Lisa Hendricks
  - Mick Looby
  - Kai Grosh
  - Farkad Ezzet
  - Andy Stein
  - Maja Skataric
- External Advocates