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

R in Pharma 2019
Harvard University, Cambridge
August 22nd 2019

Mirjam N. Trame

On behalf of the nlmixr development team:

Matt Fidler, Richard Hooijmaijers, Teun Post, Rik Schoemaker, Mirjam Trame, Justin Wilkins, Yuan Xiong and Wenping Wang

Who we are

Wenping Wang, PhD

Matthew Fidler, PhD

Yuan Xiong , PhD

Mirjam Trame, PharmD, PhD

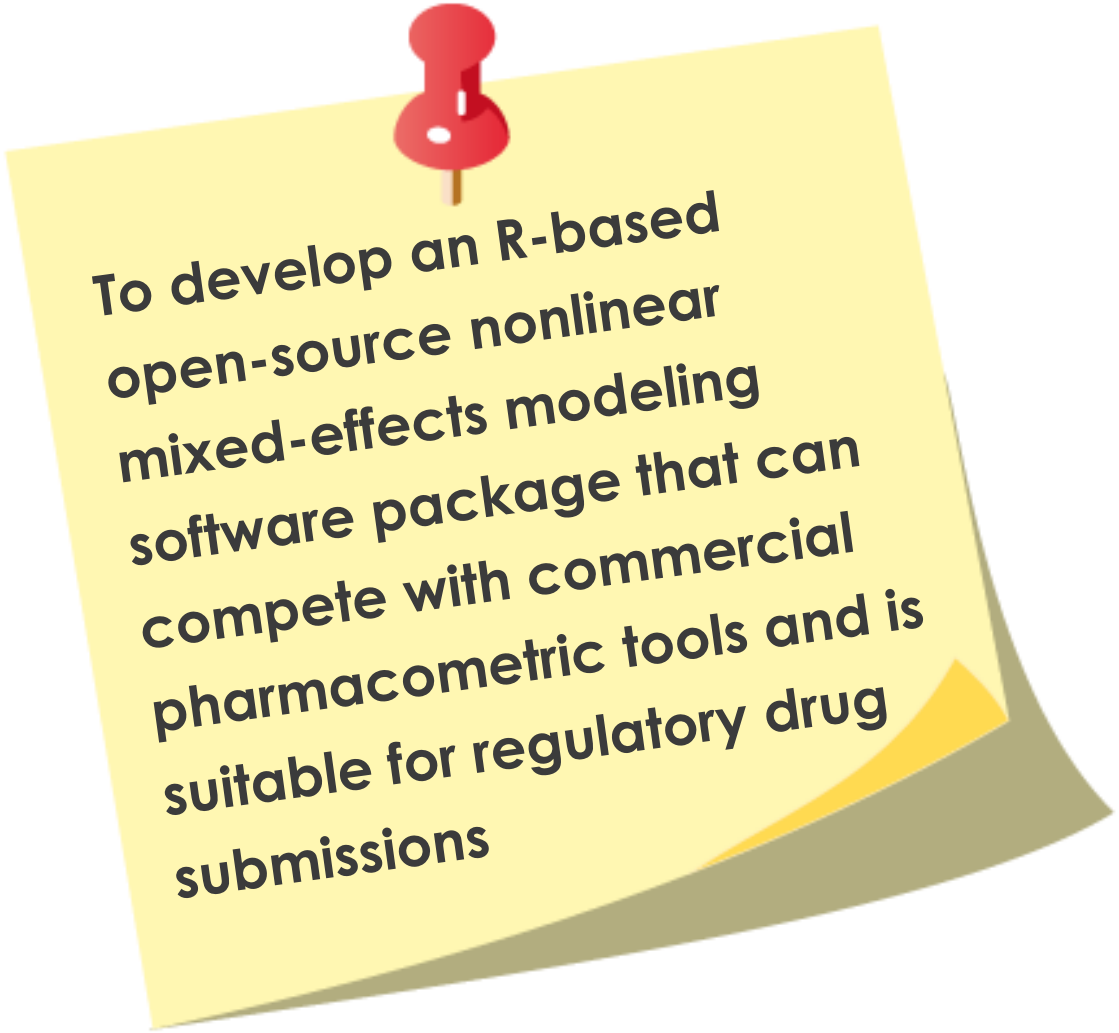
Justin Wilkins, PhD

Rik Schoemaker, PhD

Richard Hooijmaijers, BSc

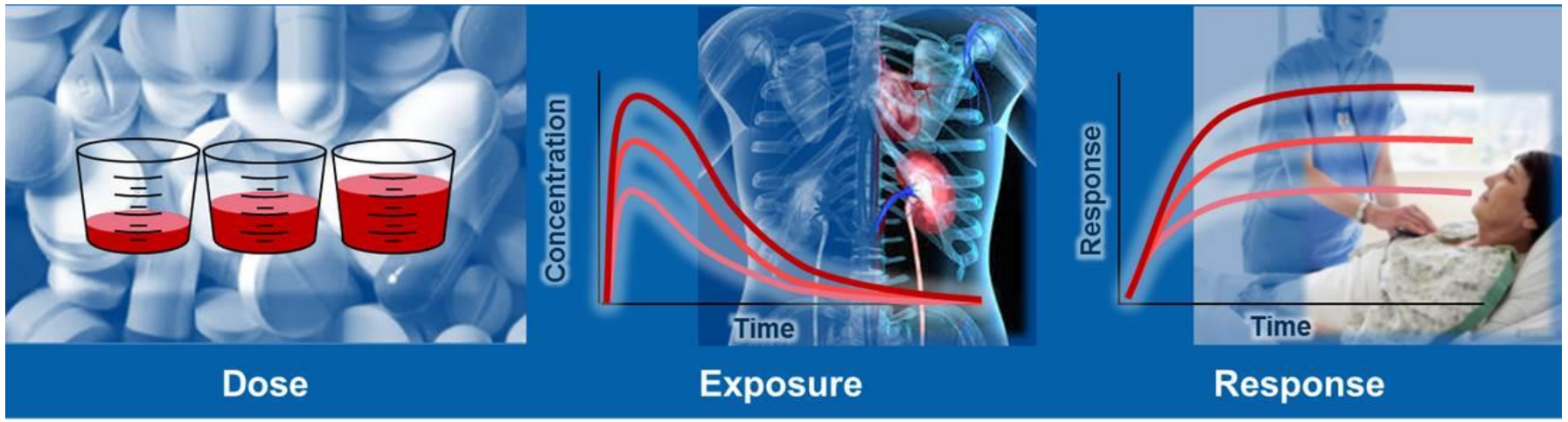
Teun Post, PharmD, PhD

Vision of nlmixr



What is Pharmacometrics?

- Pharmacometrics is the integrative science using in silico models to understand the drug-patient-disease relationships and to Optimize Drug Therapy through Dose-Exposure-Response Analysis



- Pharmacometrics helps with:
  - Understanding the potential that compounds offer patients, by optimizing dose, dosing regimen, time course of treatment, individualization, while accounting for disease progression
  - Supporting clinical trial designs, program decisions, and regulatory submissions

Why is Finding the Right Dose of a Drug Important?

- Tylenol helps
  and is safe a
- >5,000 mg/c
- >10,000 mg/
- You can lear
  from clinical
- Making best
  models to in

Swiss physician **Paracelsus** (1493-1541) credited with being

"the father of modern toxicology."

"All substances are poisons: there is none which is not a poison. The right dose differentiates a poison from a remedy."

nol) Johnson&Johnson

Why is Dose Finding Hard?

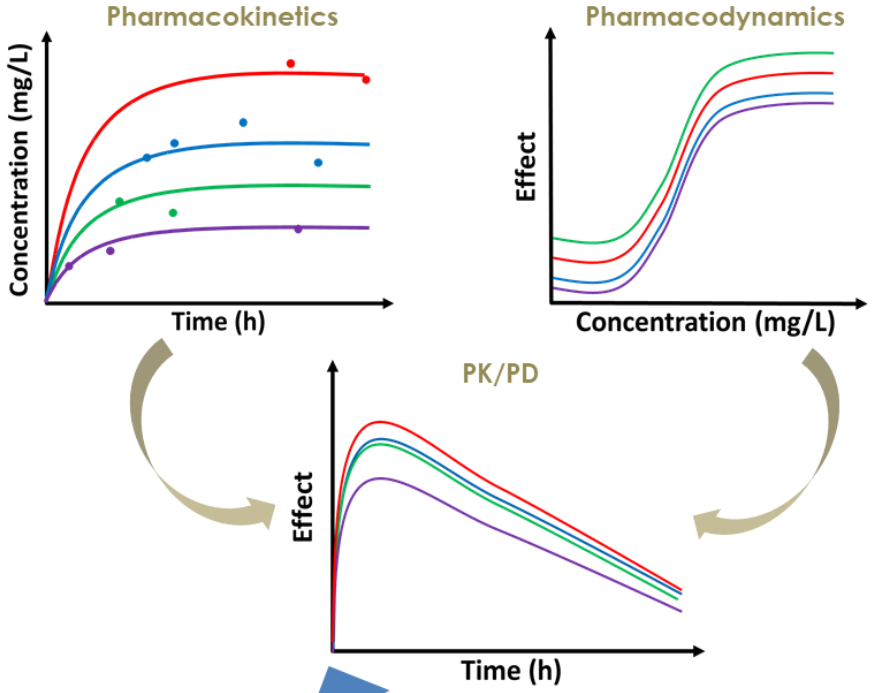
- ✓ Variability
- ✓ Non-linearity
- ✓ Change over time
- ✓ Ability to capture and measure disease markers

+

Population PK/PD Analysis using

non-linear mixed effects modeling

nlmixr



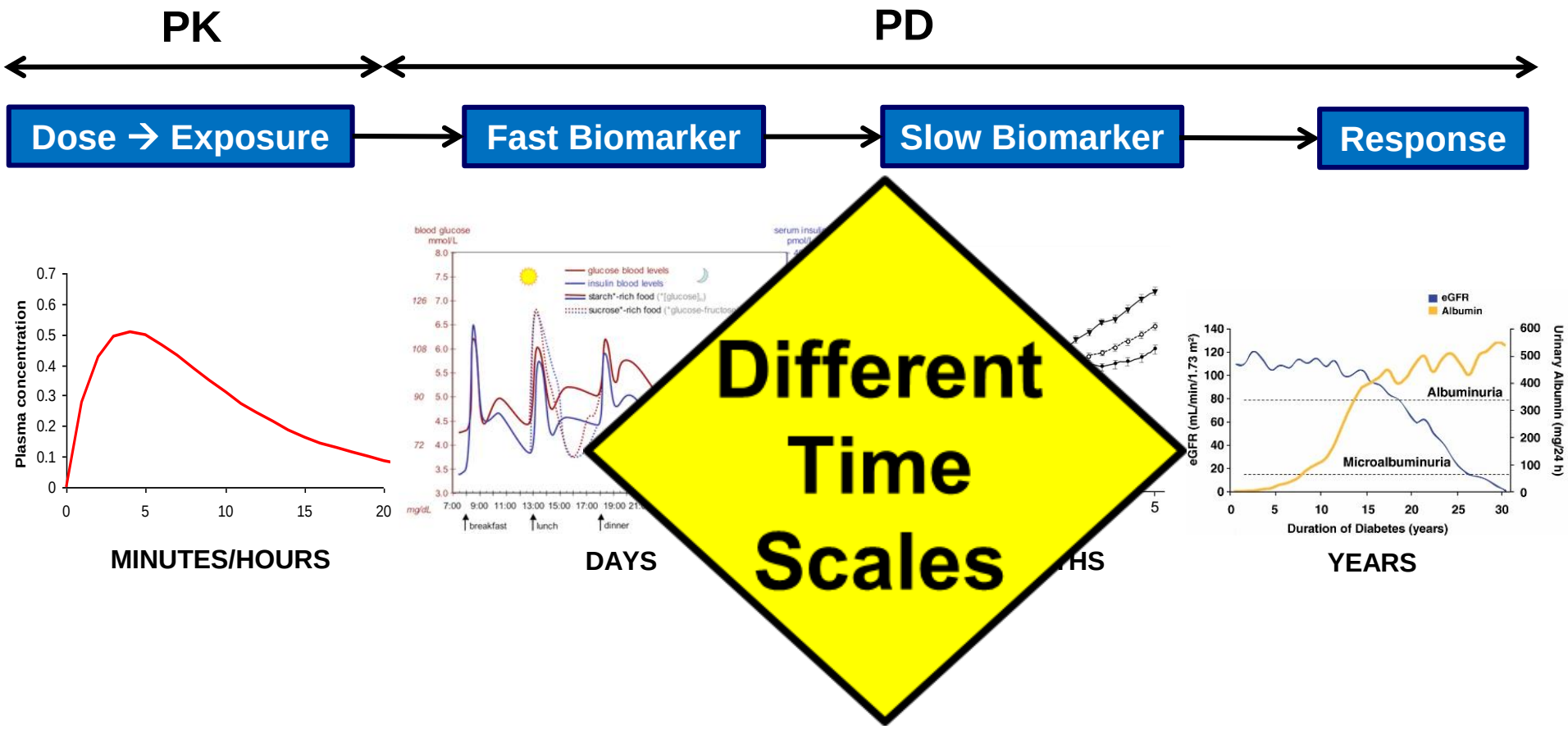
New population with **different genetic pool and rare diseases (i.e. Malaria, TB)**

?

Inclusion of **population specific covariates** to predict Dose-Exposure-Response

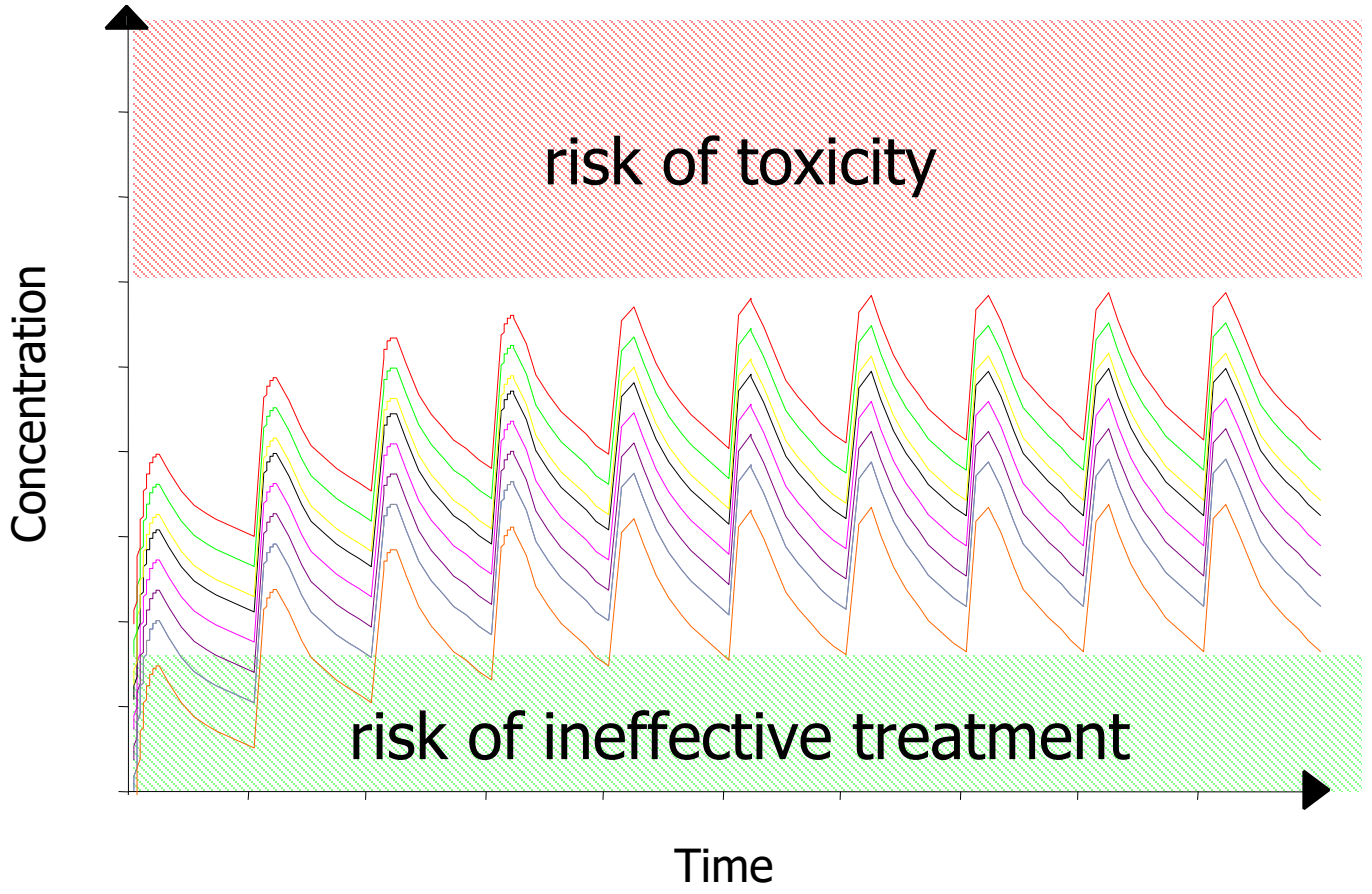
- ➢ **Dose Individualization for Special Population**

Drug Concentrations and Response Wax and Wane Over Time



nlmixr

Pharmacometrics Used to Explain Inter-Individual Variability in Drug Concentrations



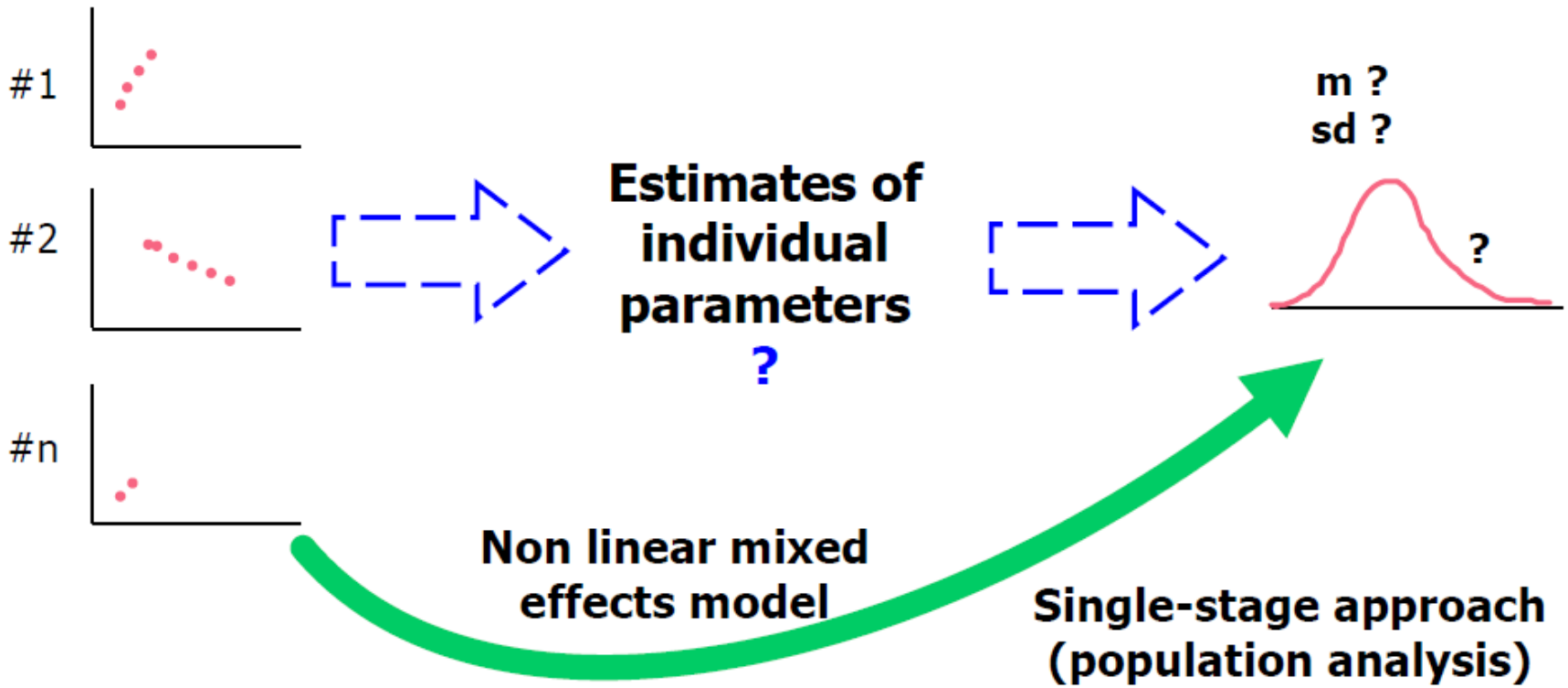
Therapeutic Failure:
Drug is too toxic

Therapeutic Range

Therapeutic Failure:
Drug doesn't work

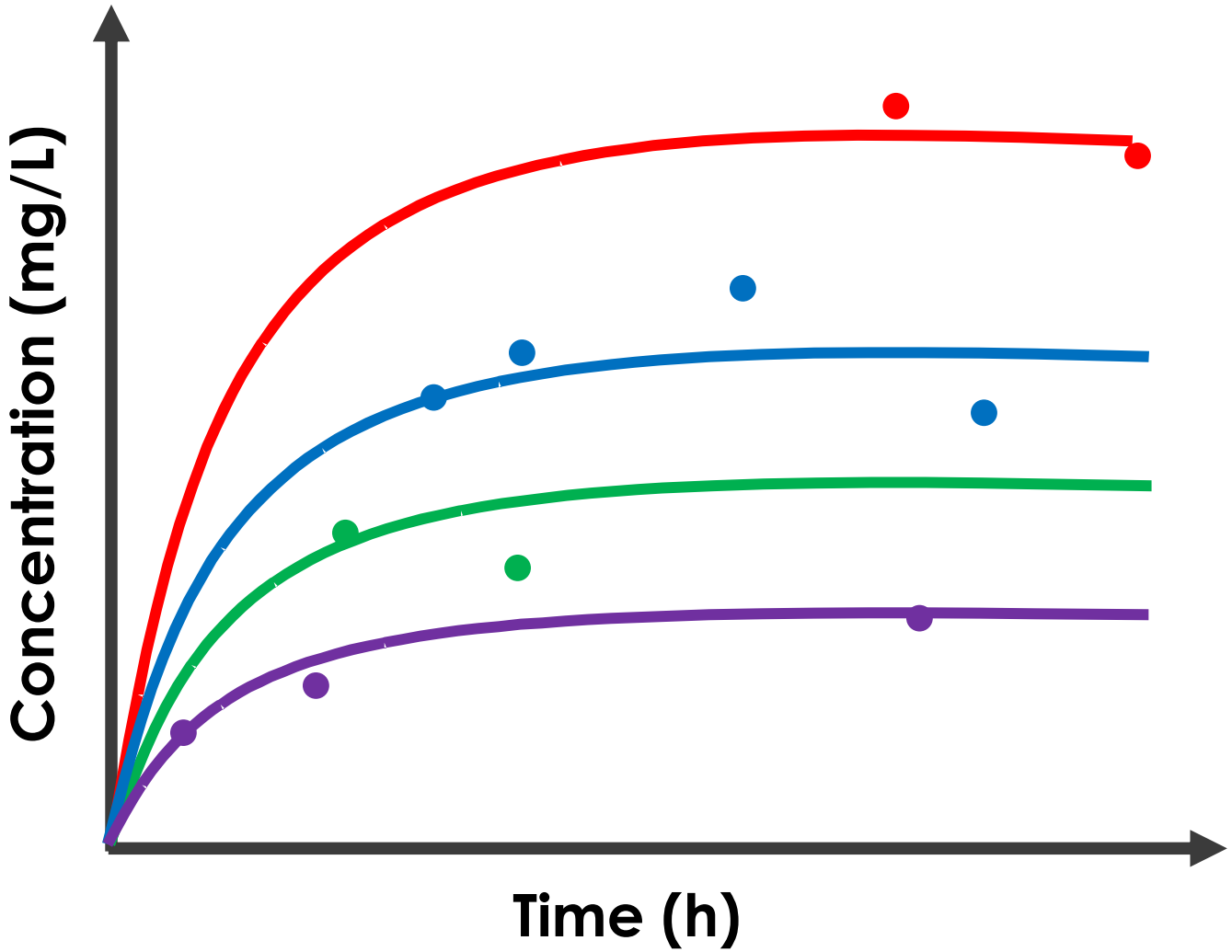
average blood levels for a particular dose

NLME = Population Approach



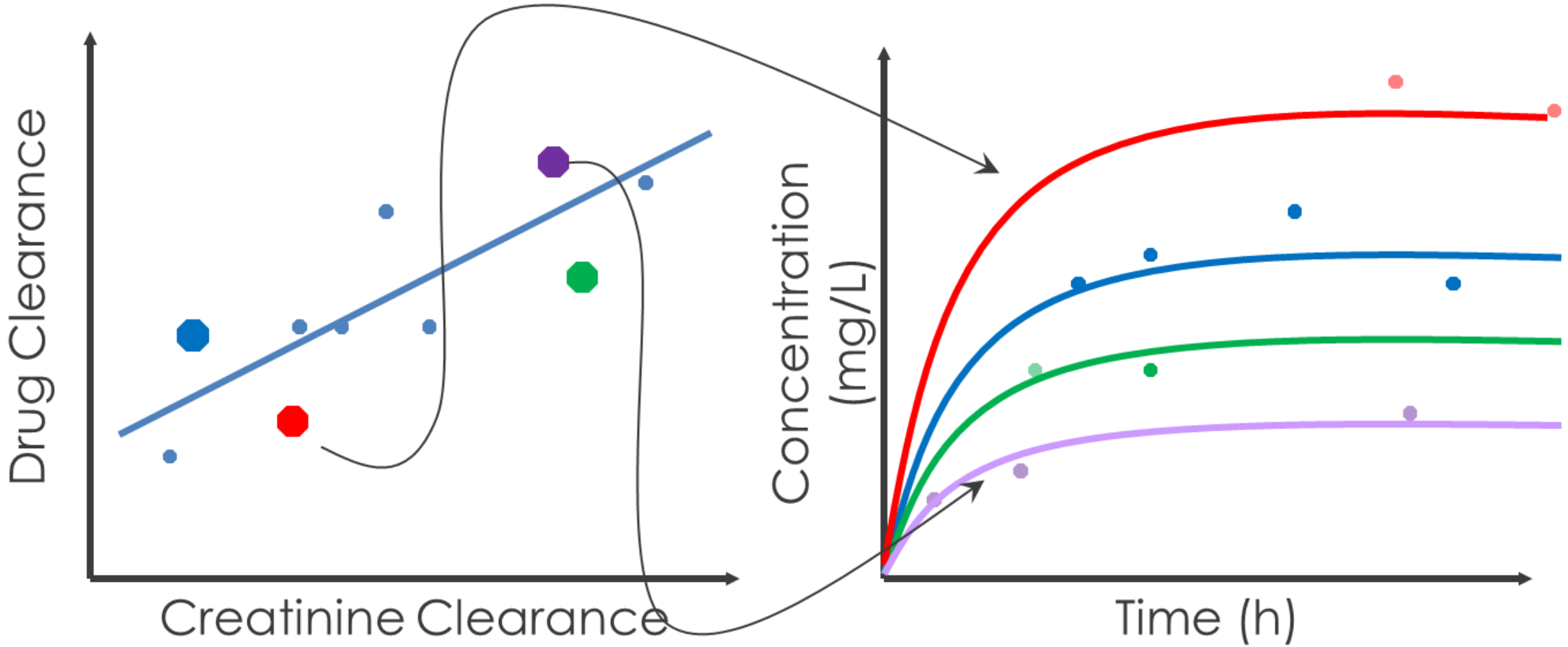
- **NLME modeling:** Build PK model to characterize fixed effects, between-subject variability, and residual variabilities

Pop-PK & Sparse Sampling

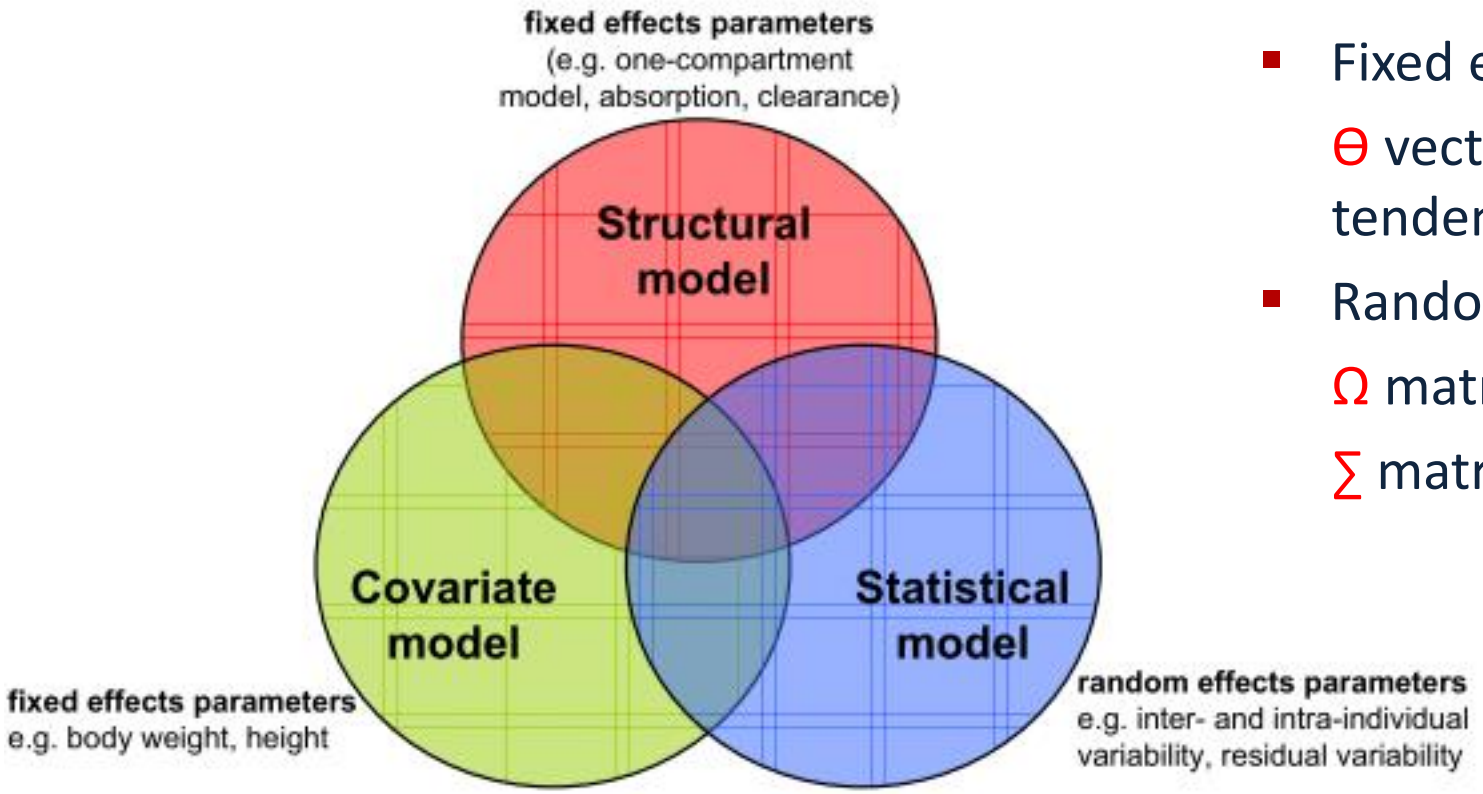


nlmixr

Impact of Interindividual Differences in Creatinine Clearance on Drug Concentration



Population Mixed-Effect Modeling



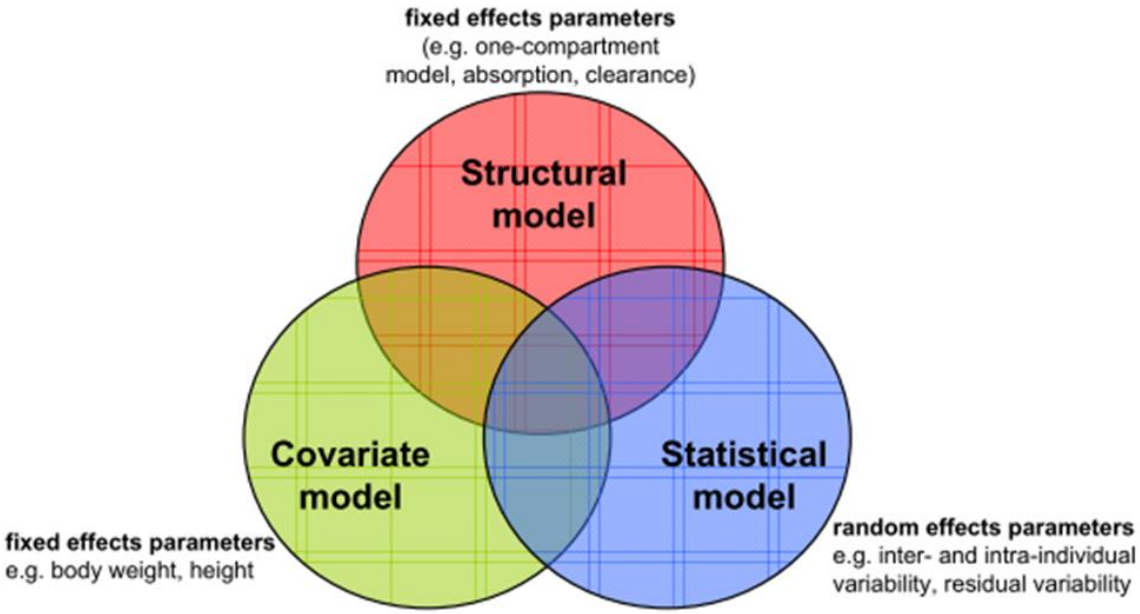
- Simultaneous estimation of parameters relating fixed effects and nested random effects to observed data:
  - Fixed effects (give point estimates):

    Θ vector of typical (mean) values or central tendencies of parameters
  - Random effects (at least 2 levels):

    Ω matrix of interindividual variance estimates

    ∑ matrix of residual variance estimates

Anatomy of a nlmixr control stream for a popPK model compared to NONMEM




```
library(nlmixr)
library(xpose.nlmixr)

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


uif <- 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)
  })
}
fit <- nlmixr(uif, data, est="saem")
```

Initial values for fixed effects

Initial values for random effects

Initial values for error model

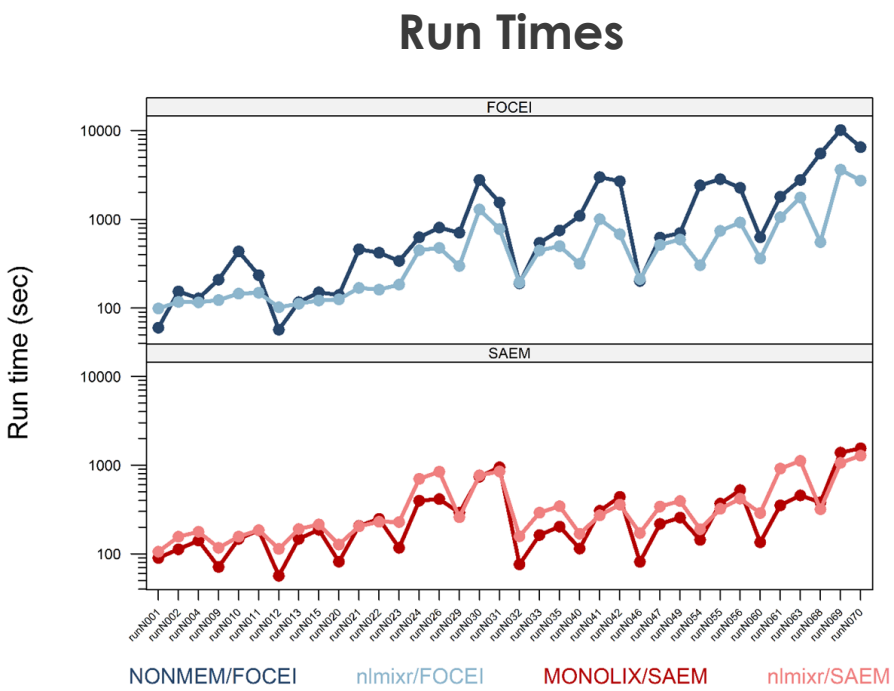
Determines compartment model

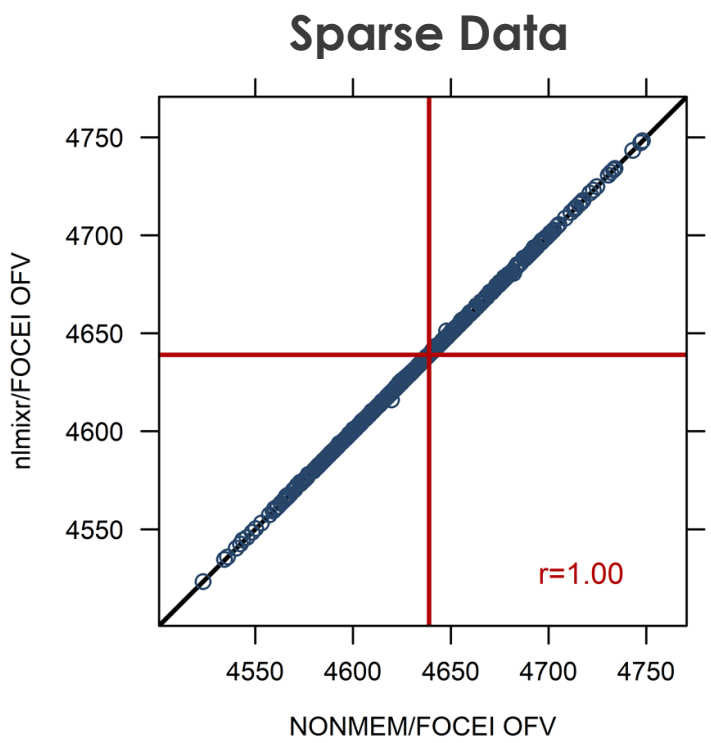
nlmixr

- nlmixr is a freely available open-source package for R [1,2] that does not depend on any commercial software, and is available on CRAN [3] and GitHub [4]
- The package allows structural models to be implemented using a system of ordinary differential equations (ODEs), and allows fully flexible dosing definitions in terms of the type (e.g. bolus doses or infusions), the timing, the number of doses, and their amount, which can vary between individuals
- nlmixr builds on RxODE [5], a fast and efficient R package for simulating nonlinear mixed effect models using ODEs, with rapid execution due to compilation in C [6]
- The package comes with its dedicated project manager shinyMixR [7] that runs in a web-browser, and is linked to xpose [8,9] for graphical exploration and goodness of fit plots for model evaluation
- nlmixr implements a number of parameter estimation algorithms that can be accessed through a common model definition language [10,11]
  - nlme
  - Stochastic approximation expectation maximization (SAEM)
  - First-order conditional estimation with interaction (FOCEI)
  - Further algorithms and parallel computation are in active development
- For a new tool to be accepted by the pharmacometric modeling and simulation community, it is essential that its estimation algorithms can be demonstrated to perform well and provide results comparable to widely used standards such as NONMEM and Monolix [12-14]

➢ **Does nlmixr show similar results compared to commercial software?**

Performance Comparison: nlmixr to Commercial Software





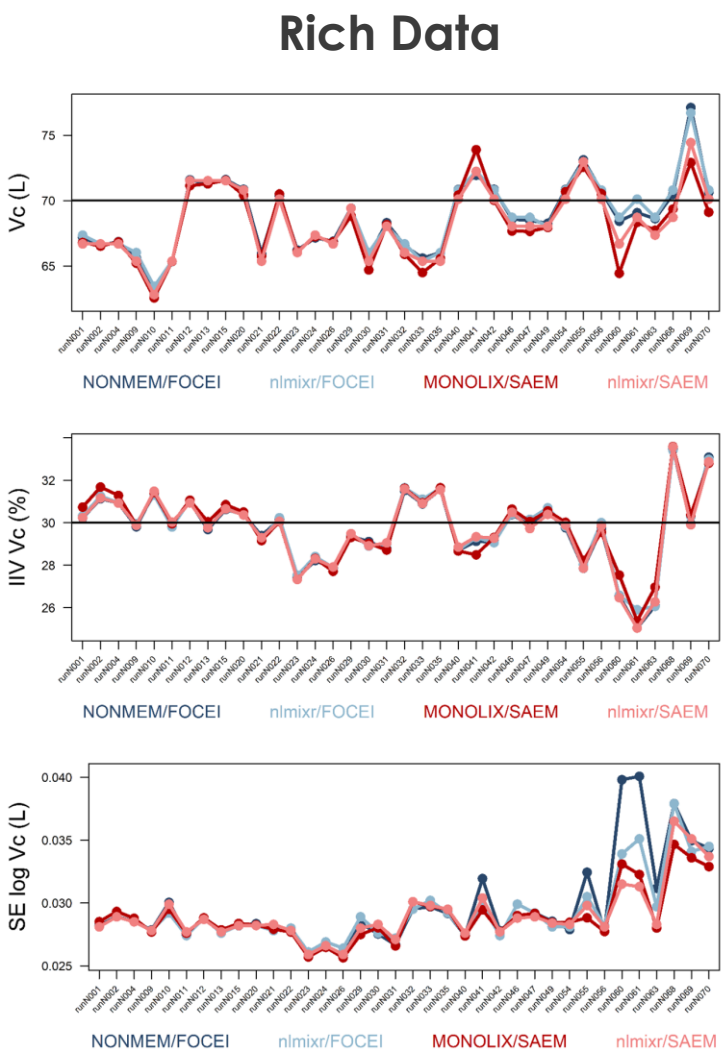
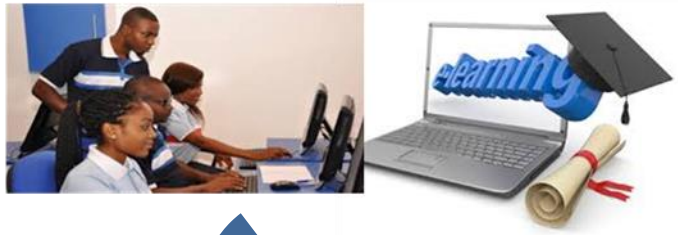


Figure Left: Run times on logarithmic scale using single-thread implementation for the four different estimation algorithms (top: FOCEI, bottom: SAEM). **Middle**: Sparse-data single-model objective function value (OFV) comparison results for nlmixr/FOCEI *vs.* NONMEM/FOCEI. **Right**: Results for central volume (Vc) for 36 models with richly-sampled data sets and multiple models and inputs. Model complexity increases from left to right. Population typical parameters (top), their inter individual variability (IIV, middle), and the standard error (SE) of their log estimate (bottom). Horizontal black line: values used in simulation.

nlmixr Helps Patients in Africa by Optimizing a Drug for this Population using an Open-source Free Tool

- $$$$ commercial Pharmacometrics software
- no costs for nlmixr due to open-source software tool

Training next Generation of Clinical Pharmacometricians in LMIC "Train the Trainer"



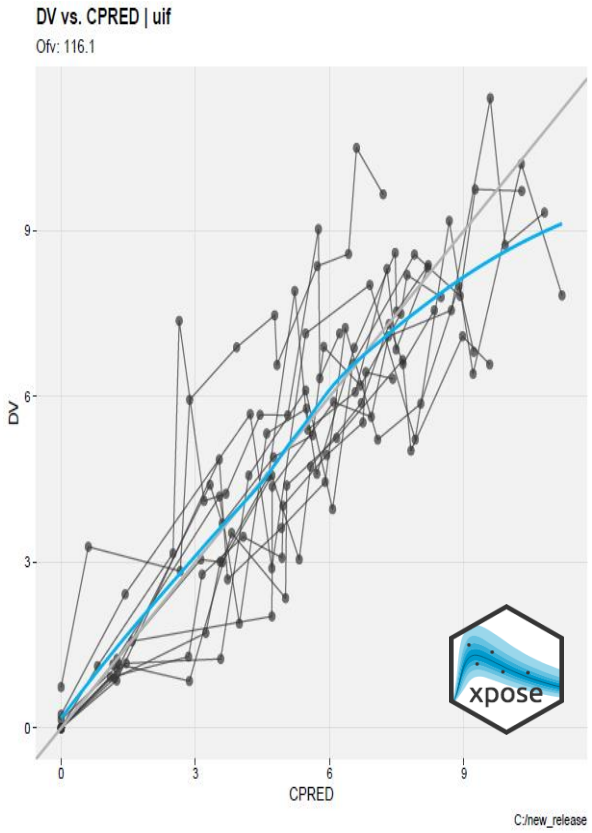
Access untapped data to predict the right dose for a different genetic pool

Decrease disease burden of rare diseases specific for LMIC

Better Health

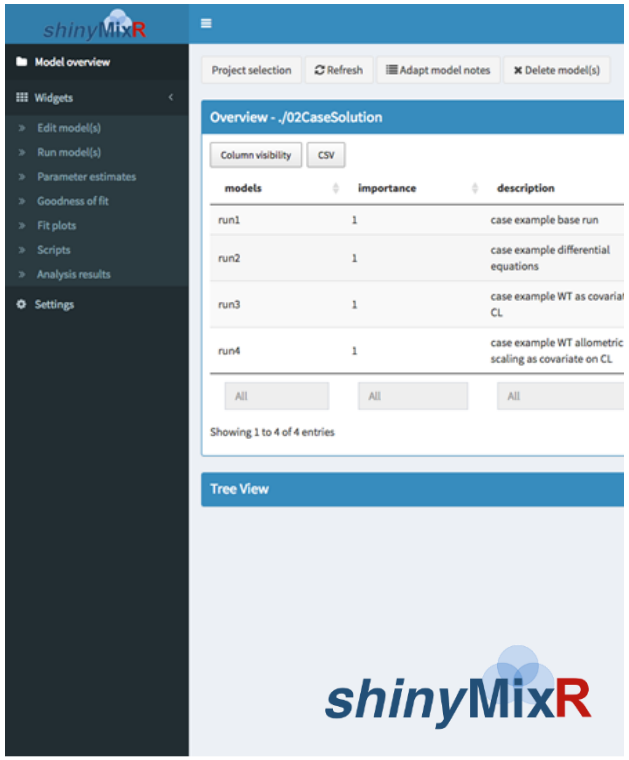
By the people: nlmixr support and add-ons

xpose interface
Justin Wilkins



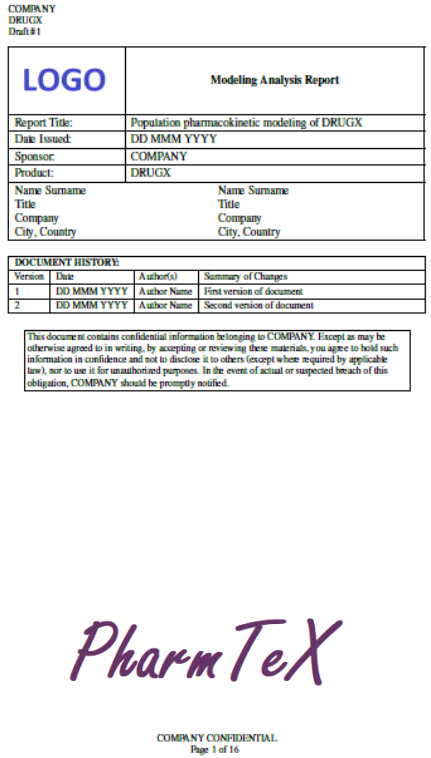
https://github.com/UUPharmacometrics/xpose
https://github.com/nlmixrdevelopment/xpose.nlmixr

GUI project management
Richard Hooijmaijers



https://github.com/RichardHooijmaijers/shinyMixR

Automatic reporting
Christian Rasmussen


COMPANY
DRUGX
Draft #1

LOGO	Modeling Analysis Report
Report Title:	Population pharmacokinetic modeling of DRUGX
Date Issued:	DD MMM YYYY
Sponsor:	COMPANY
Product:	DRUGX
Name Surname Title Company City, Country	Name Surname Title Company City, Country

DOCUMENT HISTORY:			
Version	Date	Author(s)	Summary of Changes
1	DD MMM YYYY	Author Name	First version of document
2	DD MMM YYYY	Author Name	Second version of document

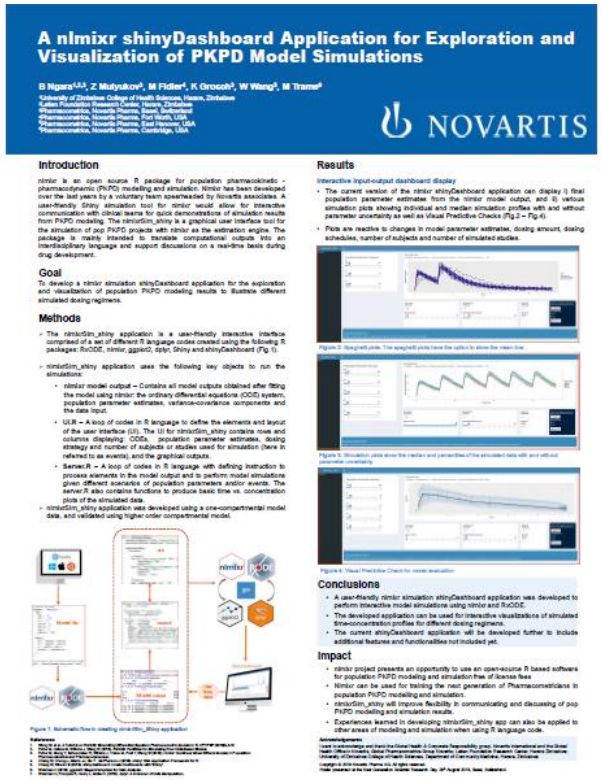
This document contains confidential information belonging to COMPANY. Except as may be otherwise agreed to in writing, by accepting or reviewing these materials, you agree to hold such information in confidence and not to disclose it to others (except where required by applicable law), nor to use it for unauthorized purposes. In the event of actual or suspected breach of this obligation, COMPANY should be promptly notified.

PharmTeX

COMPANY CONFIDENTIAL
Page 1 of 16

http://pharmtex.org/

GUI auto-simulator
Bernard Ngara



nlmixr: a popPKPD tool by pharmacometricians, for pharmacometricians

Contact us to participate @ nlmixr.info@gmail.com

Or just download the source code and get stuck in!

Conclusion

- Nlmixr is an open-source R-based free tool for pharmacometric analysis

- The estimation algorithms (FOCEI and SAEM) in nlmixr provide near-identical results to those obtained from commercial software (NONMEM and Monolix) for the same models and data

- **These results indicate that nlmixr may provide a viable open-source and free alternative to existing commercial tools for pharmacometric parameter estimation**

Thank you for your attention!

www.nlmixr.org

nlmixr Tutorial. CPT Pharmacometrics Syst. Pharmacol. (2019) XX, 1–13

Resources, documentation and further reading

- Home of nlmixr, RxODE, xpose.nlmixr, support packages (most recent versions)
  - www.nlmixr.org
- Documentation: continually evolving
  - https://nlmixrdevelopment.github.io/nlmixr_bookdown/
- Open course material:
  - PAGE (Montreux, 2018): https://github.com/nlmixrdevelopment/PAGE-2018
  - ACoP9 (San Diego, 2018): https://github.com/nlmixrdevelopment/ACoP9-2018
- Cheat sheets and posters:
  - https://github.com/nlmixrdevelopment/Publications
- Twitter: @nlmixr
- LinkedIn: https://www.linkedin.com/groups/8621368/

References

- [1]R Development Core Team (2018). R version 3.5.1. https://www.R-project.org
- [2]Fidler M et al. nlmixr Tutorial. CPT Pharmacometrics Syst. Pharmacol. (2019) XX, 1–13
- [3]https://CRAN.R-project.org/package=nlmixr
- [4]https://github.com/nlmixrdevelopment/nlmixr
- [5]Wang W, Hallow KM, James DA. A Tutorial on RxODE. CPT:PSP (2016) 5:3–10.
- [6]https://nlmixrdevelopment.github.io/nlmixr
- [7]https://github.com/RichardHooijmaijers/shinyMixR
- [8]https://github.com/nlmixrdevelopment/xpose.nlmixr
- [9]https://github.com/UUPharmacometrics/xpose
- [10]https://CRAN.R-project.org/package=nlme
- [11]Kuhn E and Lavielle MM. Maximum likelihood estimation in nonlinear mixed effects models. Comput Stat Data An (2005) 49:1020–1038.
- [12]Beal S, Sheiner LB, Boeckmann A, and Bauer RJ. 1989-2013. NONMEM Users Guides. Icon Development Solutions, USA.
- [13]Monolix. Antony, France: Lixoft SAS, 2018/2019. http://lixoft.com/products/monolix/
- [14]Laveille C, *et al*. Evaluation of the PK and PK-PD libraries of MONOLIX: A comparison with NONMEM. PAGE 17 (2008) Abstr 1356 [www.page-meeting.org/?abstract=1356]

Back-up on how to use nlmixr

A nlmixr model has two main parts: initialisation and model

Initialisation ini({ })

```
ini({
    lCl    = 1.6
    lVc    = log(90)
    lKa    = fix(1)
    add.sd = 0.2
    eta.Ka ~ 0.1 #IIV Ka
    eta.Cl + eta.Vc ~ c(0.1,
                        0.005, 0.1)

})
```

Labeled with #

```
#log Cl (L/hr)
#log V (L)
#log Ka (1/hr)
```

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://nlmixrdevelopment.github.io/RxODE/articles/RxODE-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
    lVc    = log(90)
    lKa    = 1
    prop.sd = 0.2
    eta.Ka ~ 0.1 #IIV Ka
    eta.Cl + eta.Vc ~ c(0.1,
                        0.005, 0.1)
})
```

Labeled with #
#log Cl (L/hr)
#log V (L)
#log Ka (1/hr)

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 lag time (alag) to the model

Initialisation ini({ })

```
ini({
    lCl    = 1.6
    lVc    = log(90)
    lKa    = 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)
})
```

Labeled with #
#log Cl (L/hr)
#log V (L)
#log Ka (1/hr)

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

Relationship of Fixed/Random Pars First

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

Initialisation ini({ })

```
ini({
    lCl    = 1.6
    lVc    = log(90)
    lKa    = 1
    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)
})
```

Labeled with #
#log Cl (L/hr)
#log V (L)
#log Ka (1/hr)

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({
    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

Finalising 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))
```

- `fit <-` : Assigned R object
- `nlmixr(one.cmt, data,` : Function Name is run name
- `est = "saem",` : Estimation methods = ("nlme", "saem", "focei", "foce", "foi", "fo")
- `table=tableControl(cwres=TRUE, npde=TRUE))` : 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
```


nlmixr

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
Power Model	Y ~ pow(pow.sd, pow)	fo, foi, foce, focei
Additive + Power	Y ~ add(add.sd) + pow(pow.sd, d)	fo, foi, foce, focei
Cox-Box transform both sides	Y ~ add(add.sd) + coxBox(lambda)	fo, foi, foce, focei
Yeo-Johnson transform both sides	Y ~ add(add.sd) + yeoJohnson(lambda)	fo, foi, foce, focei
Bernoulli	p ~ bern()	saem
Poisson	lambda ~pois()	saem

Multiple Endpoint:

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


nlmixr

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

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

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

Overview of non-linear mixed effect model algorithms in nlmixr:

- First Order with FOCEi posthoc (foi) or FOCE posthoc (fo)

```
## eg nlmixr(modelfn, data, "foi");
```

```
## eg nlmixr(modelfn, data, "fo");
```

- First Order Conditional Estimate (FOCE)

```
## eg nlmixr(modelfn, data, "foce");
```

- First Order Conditional Estimate with interaction (FOCEi)

```
## eg nlmixr(modelfn, data, "focei");
```

- Stochastic Approximation Estimation-Maximization (SAEM)

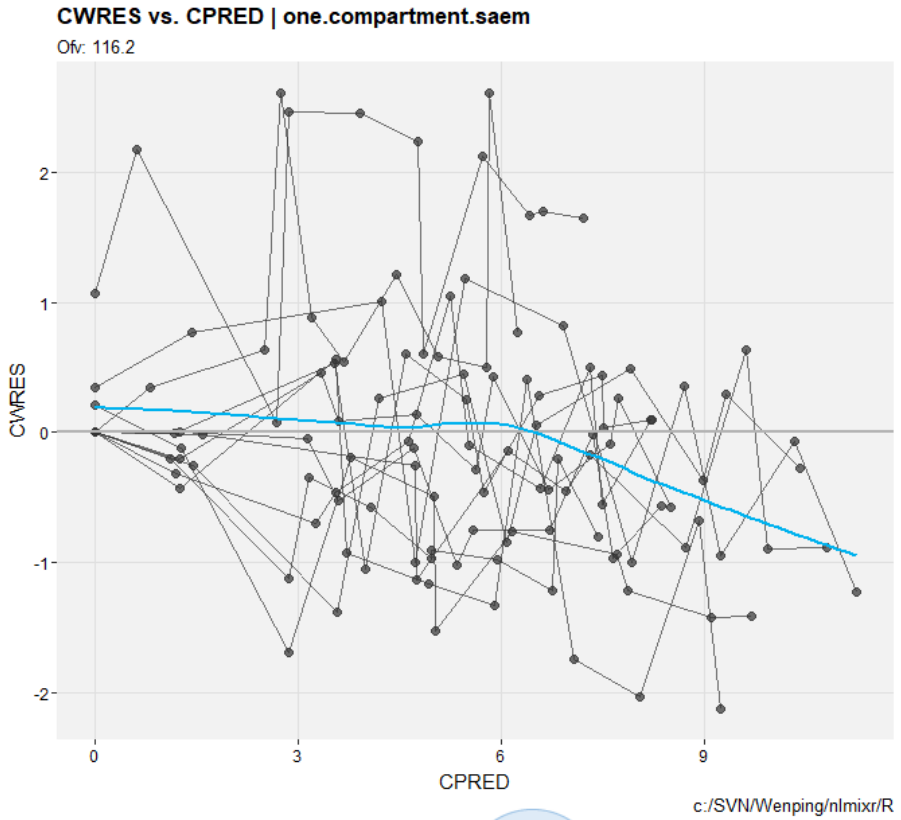
```
## eg nlmixr(modelfn, data, "saem");
```

- Adaptive Gaussian quadrature (with Laplacian approximation as a special case see glmn and glmm2)

- Traditional R nlme

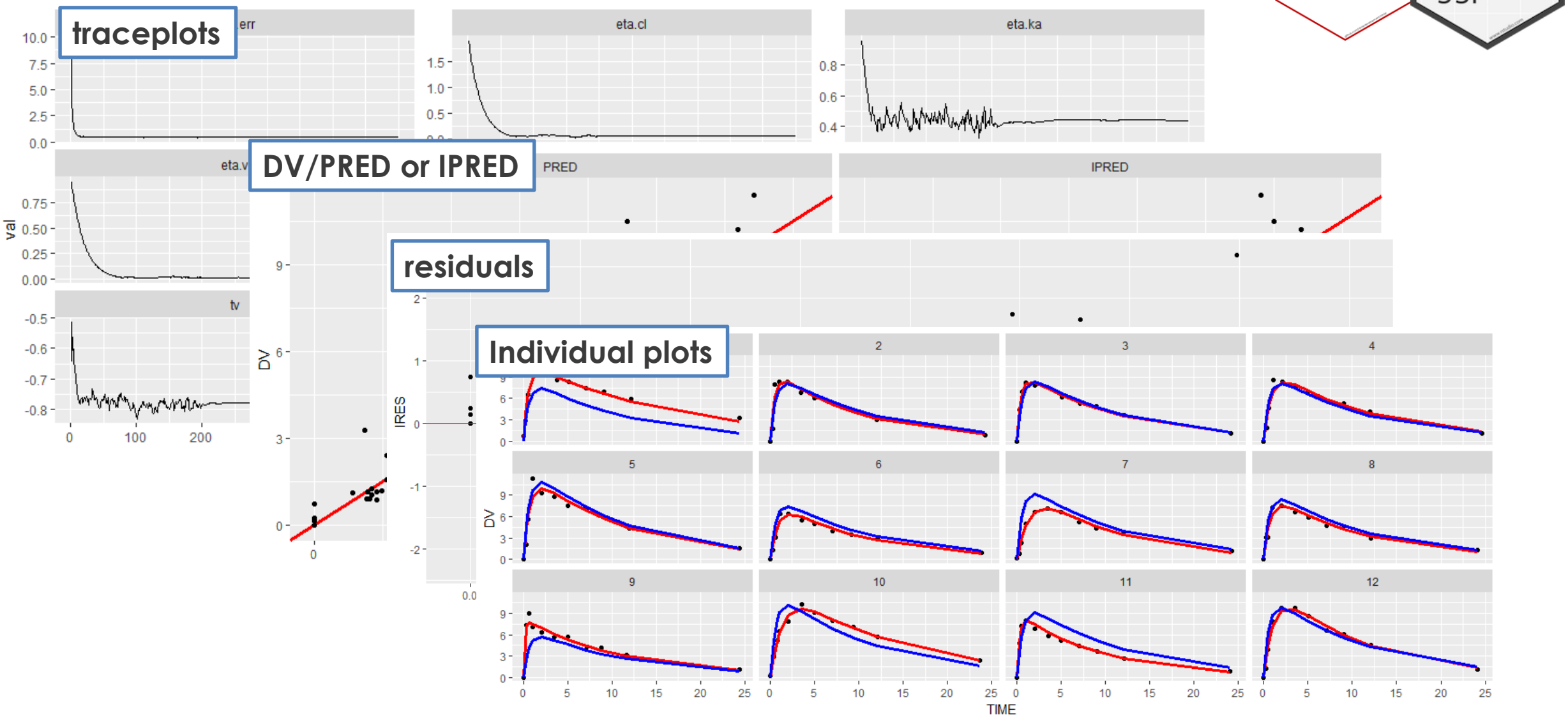
```
## eg nlmixr(modelfn, data, "nlme");
```

diagnostic plots from a nlmixr model



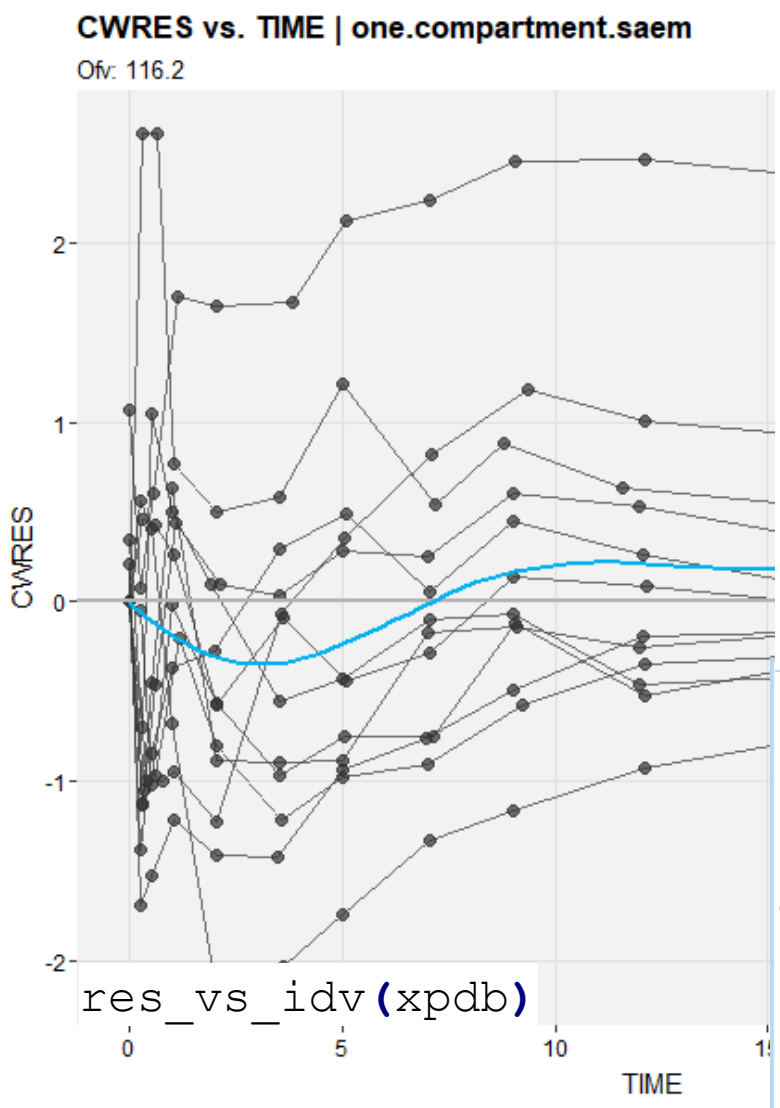
nlmixr

Simple goodness of fit plots can be produced by a simple plot(fit)

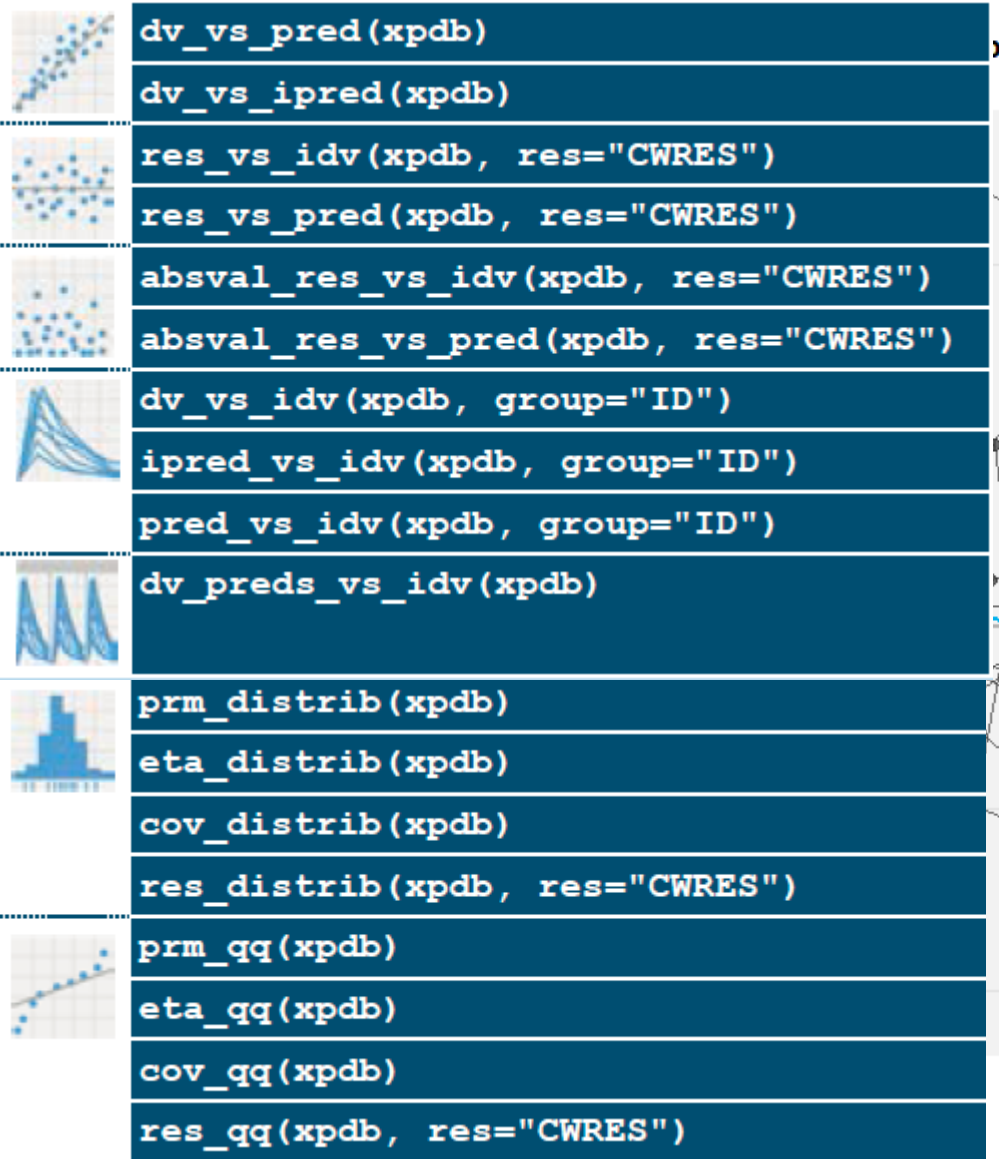


Xpose Plots can be created by converting fit to xpose data

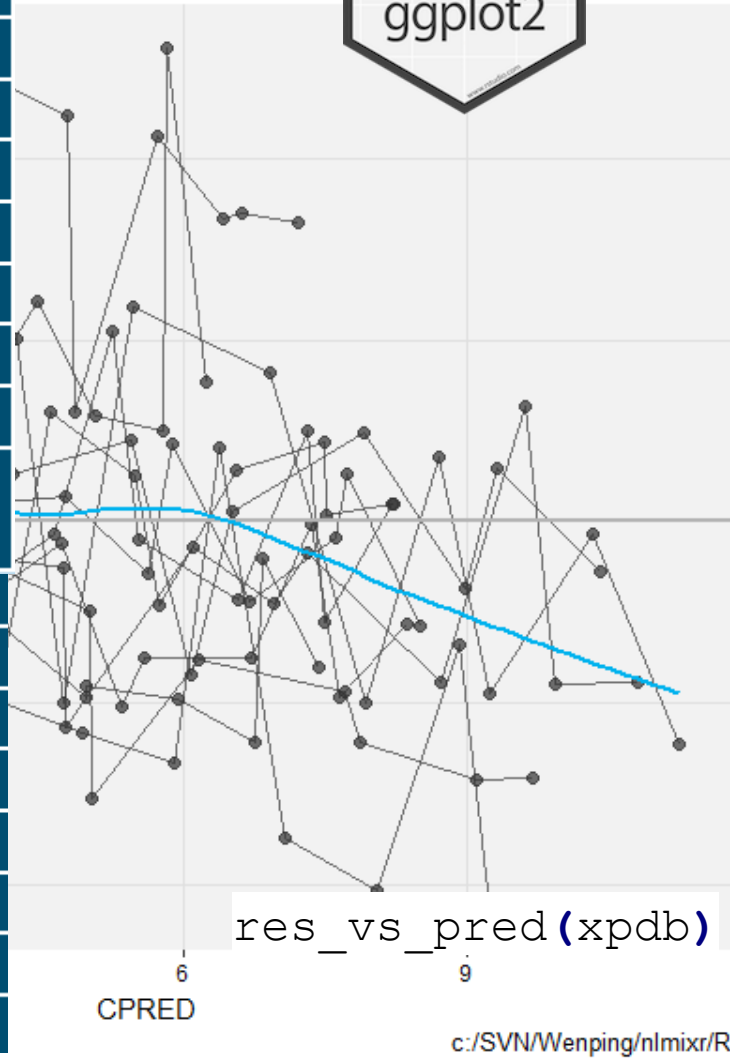
```
xpdb<-xpose_data_nlmixr(fit,xp_theme= theme_xp_nlmixr())
```



res_vs_idv(xpdb)



```
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)
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")
```

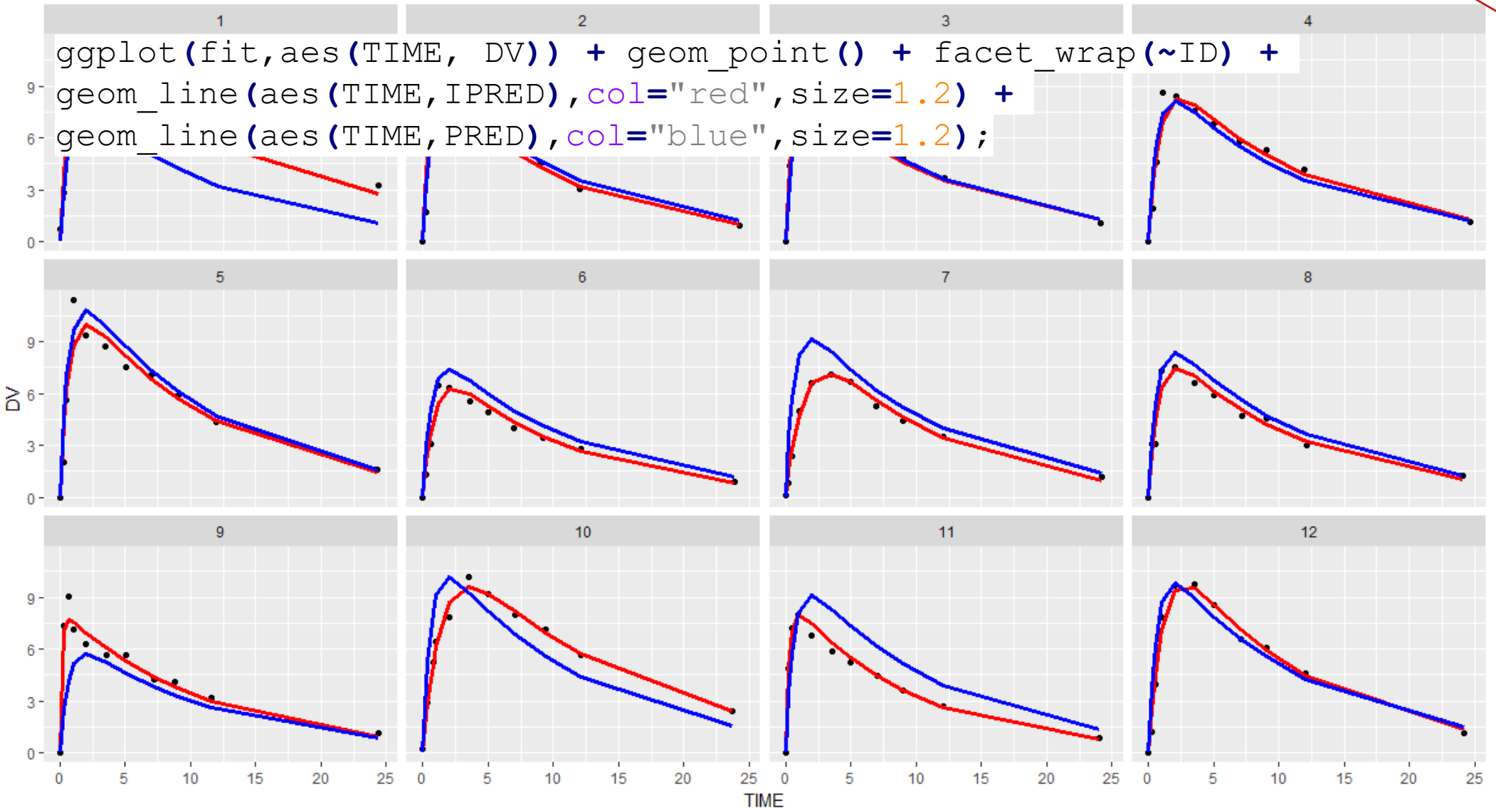


res_vs_pred(xpdb)

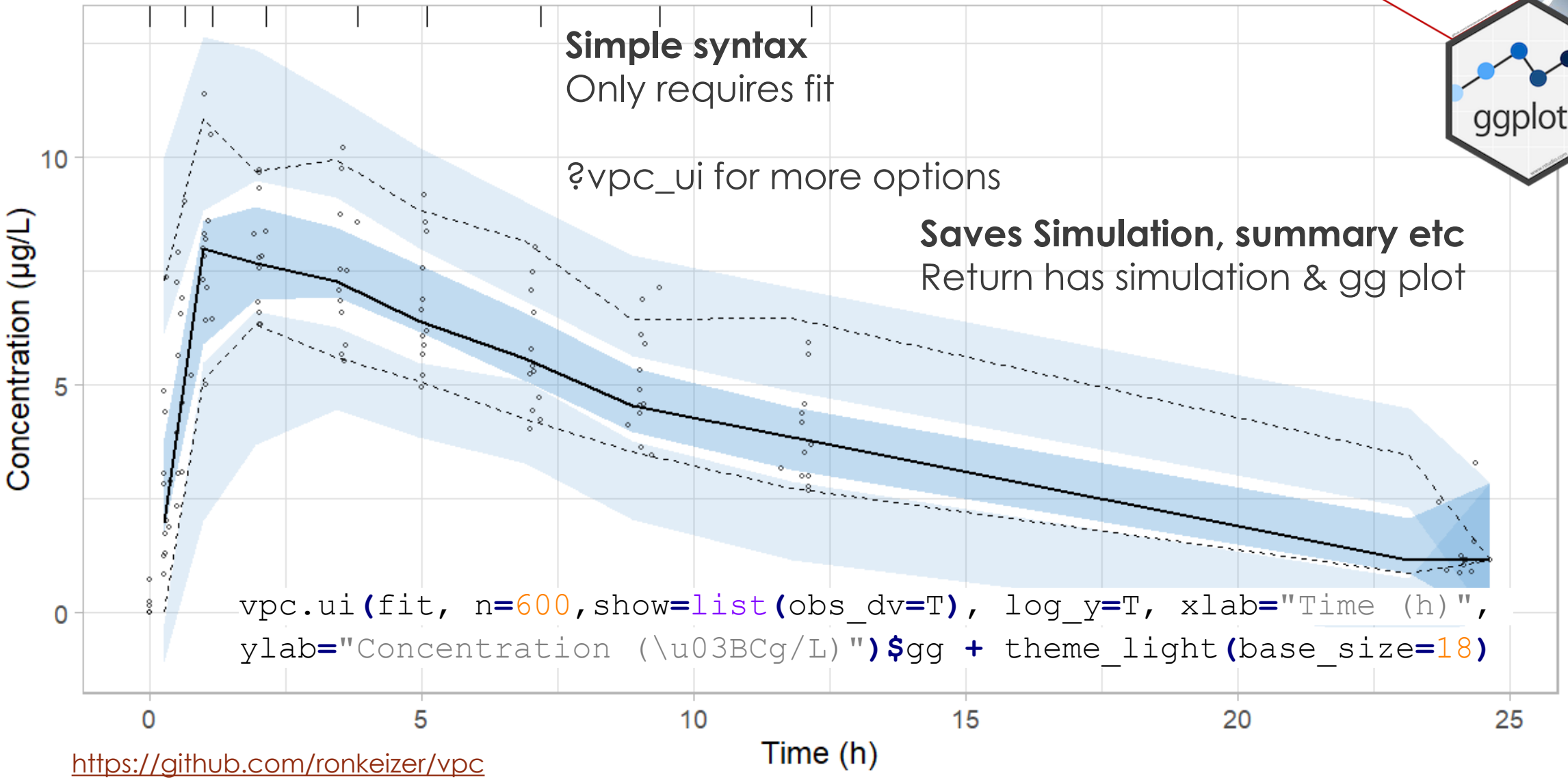
c:/SVN/Wenping/nlmixr/R

Individual Plots Using Fit Data
Shows observations & connects the dots for individual points

```
ggplot(fit,aes(TIME, DV)) + geom_point() + facet_wrap(~ID) +
geom_line(aes(TIME,IPRED),col="red",size=1.2) +
geom_line(aes(TIME,PRED),col="blue",size=1.2);
```



Model Evaluation with Visual Predictive Check



Simple syntax
Only requires fit

?vpc_ui for more options

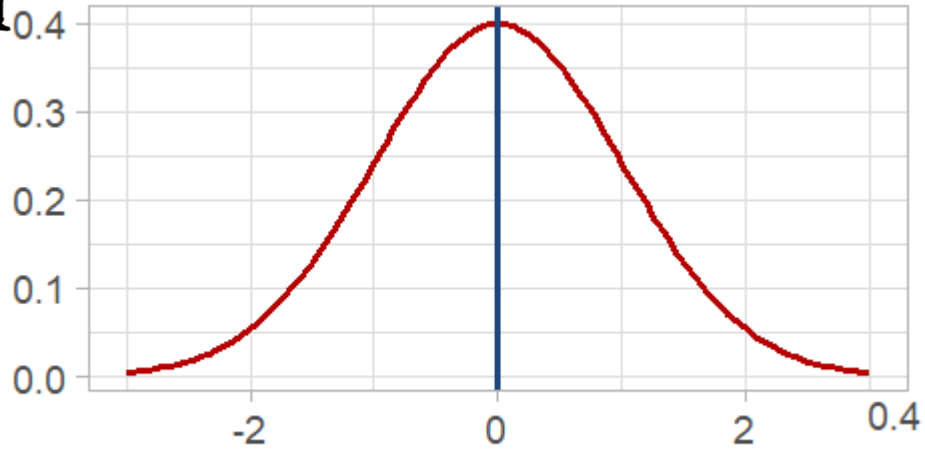
Saves Simulation, summary etc
Return has simulation & gg plot

```
vpc.ui(fit, n=600,show=list(obs_dv=T), log_y=T, xlab="Time (h)",
ylab="Concentration (\u03BCg/L)")$gg + theme_light(base_size=18)
```

https://github.com/ronkeizer/vpc

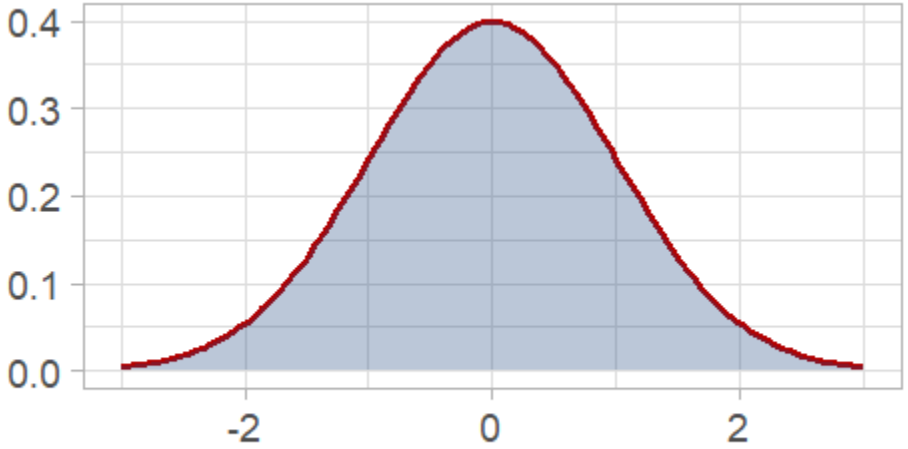
simulating from a nlmixr model

multi-subject



Without uncertainty

nlmixr



With uncertainty

Excursion to RxODE

https://nlmixrdevelopment.github.io/RxODE/

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

What do I need to know about RxODE?

What in RxODE is used by nlmixr?

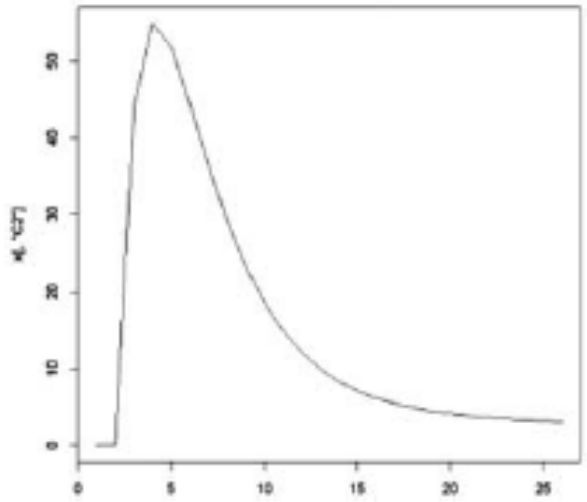
What in RxODE is output by nlmixr?

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

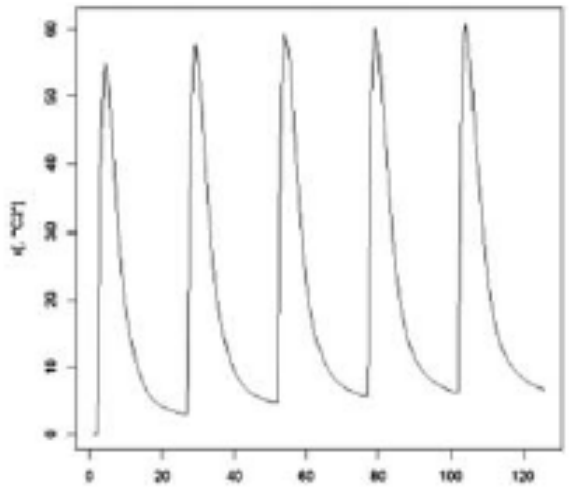
et(amt=10000)



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);
```

Links to RxODE Manual (https://nlmixrdevelopment.github.io/RxODE/index.html)

- RxODE Introduction: https://nlmixrdevelopment.github.io/RxODE/articles/RxODE-intro.html
- RxODE Covariates: https://nlmixrdevelopment.github.io/RxODE/articles/RxODE-covariates.html
- RxODE Data Frames: https://nlmixrdevelopment.github.io/RxODE/articles/RxODE-data-frame.html
- RxODE Events: https://nlmixrdevelopment.github.io/RxODE/articles/RxODE-events.html
- RxODE Solved Systems and ODEs: https://nlmixrdevelopment.github.io/RxODE/articles/RxODE-mix-lin-ode.html
- RxODE and Shiny: https://nlmixrdevelopment.github.io/RxODE/articles/RxODE-shiny.html
- RxODE Simulation: https://nlmixrdevelopment.github.io/RxODE/articles/RxODE-sim-var.html
- RxODE Benchmarking: https://nlmixrdevelopment.github.io/RxODE/articles/RxODE-speed.html
- RxODE Jacobian specification and Stiff Systems: https://nlmixrdevelopment.github.io/RxODE/articles/RxODE-stiff.html
- RxODE ODE solving syntax: https://nlmixrdevelopment.github.io/RxODE/articles/RxODE-syntax.html
- RxODE Transit Compartment Models: https://nlmixrdevelopment.github.io/RxODE/articles/RxODE-transit-compartments.html

Back to Simulating with nlmixr

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

Load nlmixr	`library(nlmixr)`

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 RxODE Object

nlmixr Object

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

Simulation Output

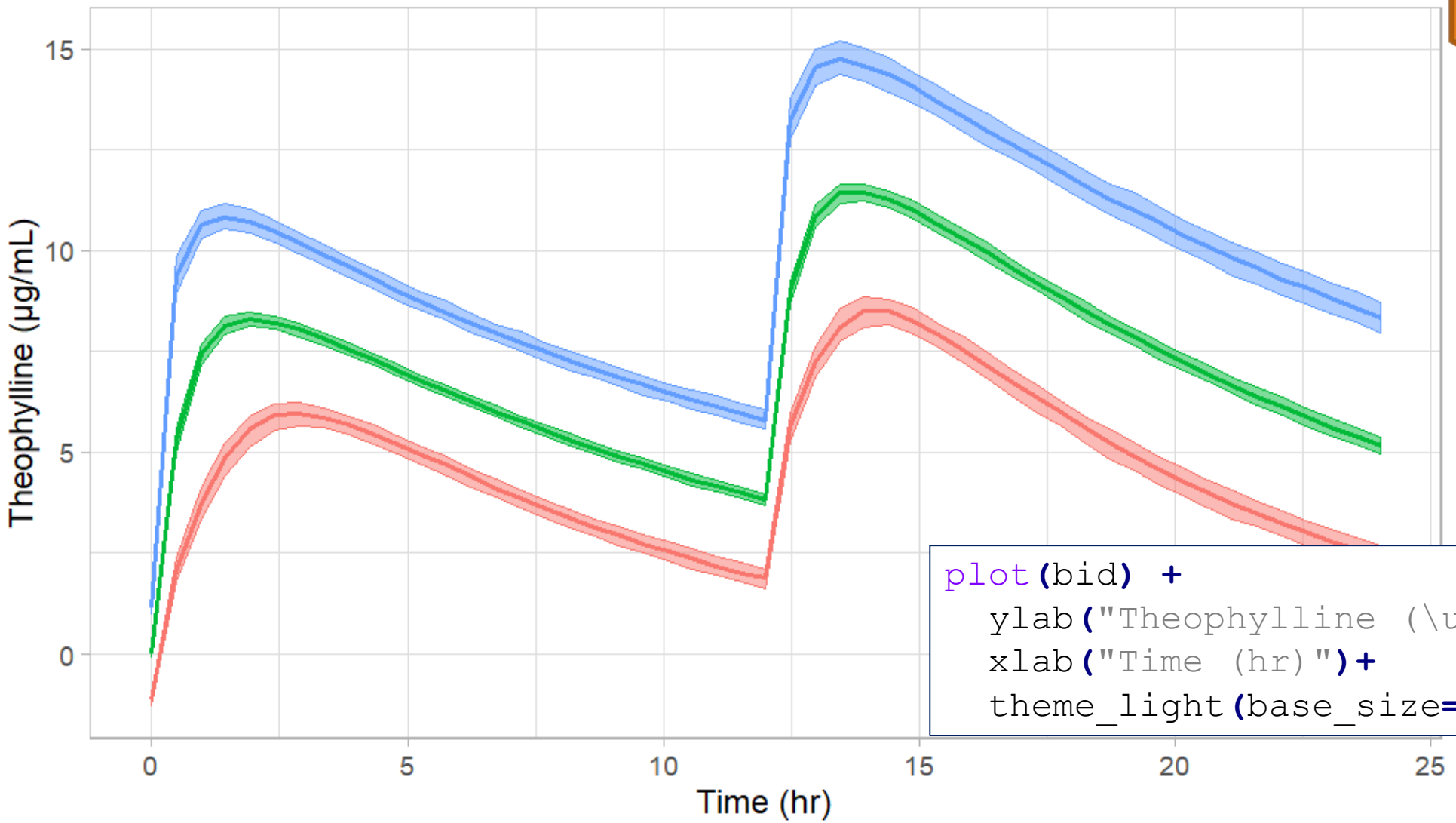
The simulation has information about the parameters that were used in the simulation from the model fit (= Solved RxODE object)

```
> 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
Simulate a new regimen (BID) – With Parameter Uncertainty
→ add one additional parameter "nStud"

Load nlmixr	`library(nlmixr)`

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 RxODE Object

nlmixr Object

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

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 RxODE 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
```

The “Study” realization of each “omega” matrix and “sigma” matrix can also be accessed

```
> head(bid$omega.list,3)
[[1]]
             [,1]        [,2]         [,3]
[1,] 1.241368413 0.20987892 0.008933452
[2,] 0.209878919 0.11947895 0.012484270
[3,] 0.008933452 0.01248427 0.018430285

[[2]]
            [,1]         [,2]         [,3]
[1,] 1.37419522  0.09899824  0.01474735
[2,] 0.09899824  0.07532544 -0.01726544
[3,] 0.01474735 -0.01726544  0.01979003

[[3]]
            [,1]         [,2]         [,3]
[1,] 0.55100595  0.04605451  0.01535430
[2,] 0.04605451  0.11197438 -0.01655395
[3,] 0.01535430 -0.01655395  0.01563234
```

Sampled from Inverse Wishart (Scaled), df=#Sub

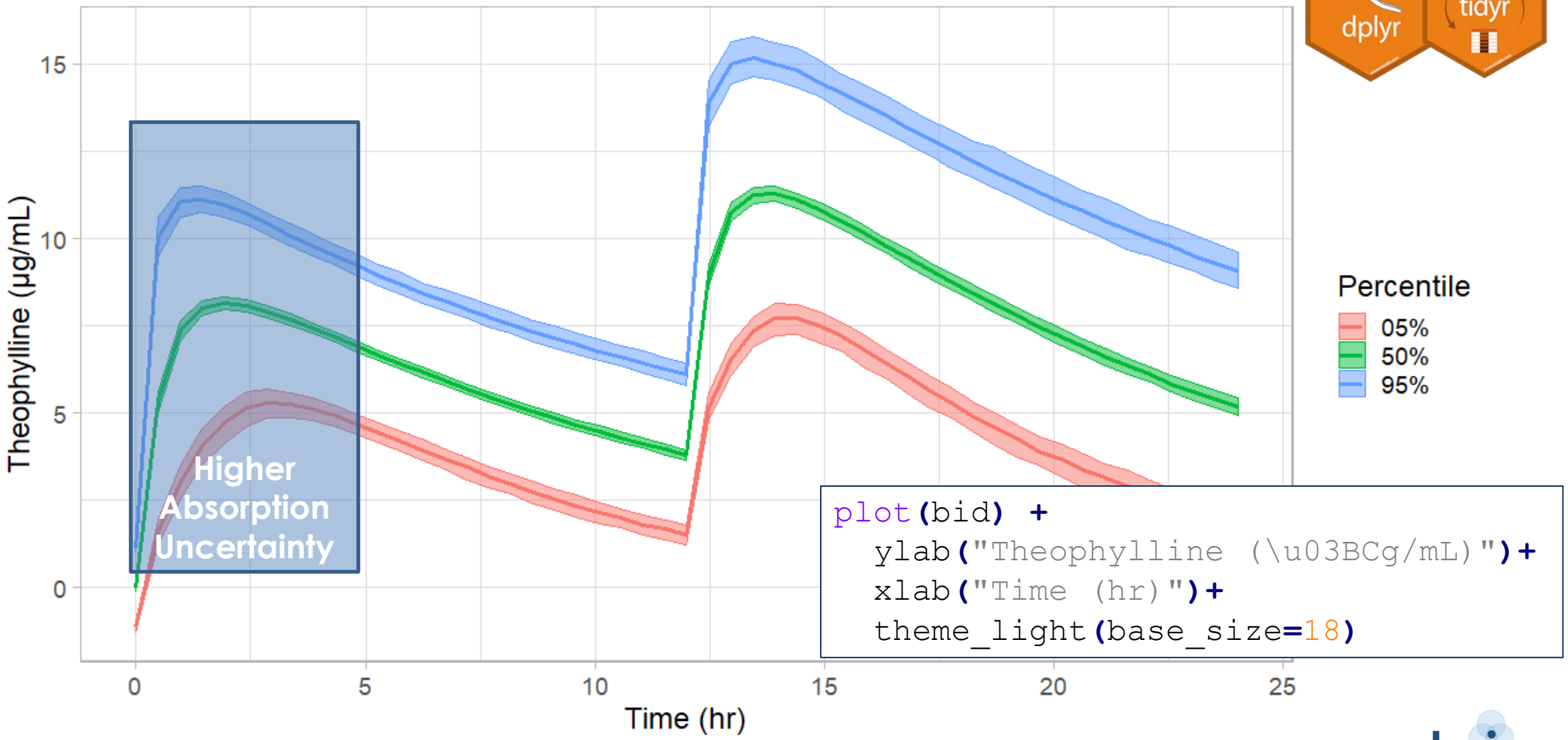
```
> head(bid$sigma.list,3)
[[1]]
             [,1]
[1,] 0.5943641

[[2]]
             [,1]
[1,] 0.5408789

[[3]]
             [,1]
[1,] 0.4281149
```

Sampled from Inverse Wishart (Scaled), df=#Obs

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




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

Hands-on

Case Theophylline

1) Open Run1.R and execute the model in R
   - Output the model parameter estimates
   - Plot the GOF plots and VPC
2) Run2: Code the model provided in Run1.R using ODE's
3) Run3: Use model Run1 and add covariate WT on CL
4) Run4: Use model Run1 and add covariate WT allometrically scaled and estimate exponent
5) Run5: Use model Run4 and fix allometric exponent to estimate value from Run4
6) Run6: Use model Run2 and add estimation of lag time

Additional Hands-on Examples

1) https://github.com/nlmixrdevelopment/nlmixr-examples

Update/change models incrementally on the fly in nlmixr

Modeling Piping

Update/change models incrementally on the fly

- **Changing and fixing parameter values in models**
  1) Change parameter estimates before or after running a model (ie. update(tka=0.5))
  2) Fix parameters to arbitrary values, or estimated values (ie. update(tka=fix(0.5)) or update(tka=fix))

- **Changing model features**
  - Adding or removing between subject variability
  - Adding covariate effects
  - Changing residual errors
  - Adding diagnostic plots

  - **Exercise: modelPiping_Page.R**

Estimation of Multiple Endpoints in nlmixr

Multiple Endpoints (e.g. joint PKPD model, multiple PD endpoints)

Example: joint PKPD model

```
example <- function(){
  ini({
     .........
})
  model({
     ........
     d/dt(depot)  = -ktr * depot
     d/dt(gut)      =  ktr * depot -ka * gut
     d/dt(center) =  ka * gut - cl / v * center
     d/dt(effect)  = kin*PD -kout*effect

     cp = center / v
     cp ~ prop(prop.err) + add(pkadd.err) | center
     effect ~ add(pdadd.err) | effect
   })
}
```

 # PK
 # PD

One error model per endpoint/compartment

- Error parameters to be estimated in ini block
- **|compartment** → needed to specify which error is for which compartment

#Data conversion to nlmixr data standard: DVID to CMT

```
# Read data
homedir="."
data.pkpd <- read.csv(file = paste(homedir,"warfarin_dat.csv",sep="/"))
## Convert the dvid data item to compartment names.
## The easiest way to do this is to convert it to a factor.
## Alternatively you could count the compartment number and
## assign it directly. by data.pkpd$dvid + 2
##
data.pkpd$cmt  <- factor(data.pkpd$dvid,c(1,2),c("center","effect"))
```

Dataset example in NONMEM format

id	time	amt	dv	dvid	evid	wt	age	sex
1	0	100	0	1	1	66.7	50	1
1	0.5	0	0	1	0	66.7	50	1
1	1	0	1.9	1	0	66.7	50	1
1	2	0	3.3	1	0	66.7	50	1
1	3	0	6.6	1	0	66.7	50	1
1	6	0	9.1	1	0	66.7	50	1
1	9	0	10.8	1	0	66.7	50	1
1	12	0	8.6	1	0	66.7	50	1
1	24	0	5.6	1		66.7	50	1
1	24	0	44	2			50	1
1	36	0	4	1			50	1
1	36	0	27	2			50	1
1	48	0	2.7	1	0	66.7	50	1
1	48	0	28	2	0	66.7	50	1
1	72	0	0.8	1	0	66.7	50	1
1	72	0	31	2	0	66.7	50	1
1	96	0	60	2	0	66.7	50	1
1	120	0	65	2	0	66.7	50	1

1 = PK
2 = PD

Example Warfarin PKPD model:
https://github.com/nlmixrdevelopment/nlmixr-examples/tree/master/case-study-warfarin

Multiple Endpoints (e.g. joint PKPD model, multiple PD endpoints)

Example: multiple PD endpoints → PK model parameters were fixed in this model

```
example <- function(){
  ini({
     .........
})
  model({
     ........
     d/dt(CENTRAL) = -C1*CL + Q*(C2-C1);
     d/dt(PERIF)       =              - Q*(C2-C1);
     d/dt(RET)         = KTR*PROG2*(1+EDRUG)-KRET*RET;
     d/dt(RBC)         =                                KRET*RET-KRBC*RBC;
     d/dt(HGB)         =                                KRET*RET+KRBC*RBC-KHGB*HGB;
     d/dt(PROG1)    = KPROL*PROG1*(2*RBC0/HGB)-KTR*PROG1;
     d/dt(PROG2)    =                                    KTR*PROG1-KTR*PROG2*(1+EDRUG);

     log(HGB) ~ add(logHgb) | HGB
     log(RBC) ~ add(logRbc) | RBC
     log(RET)  ~ add(logRet) | RET
   })
}
```

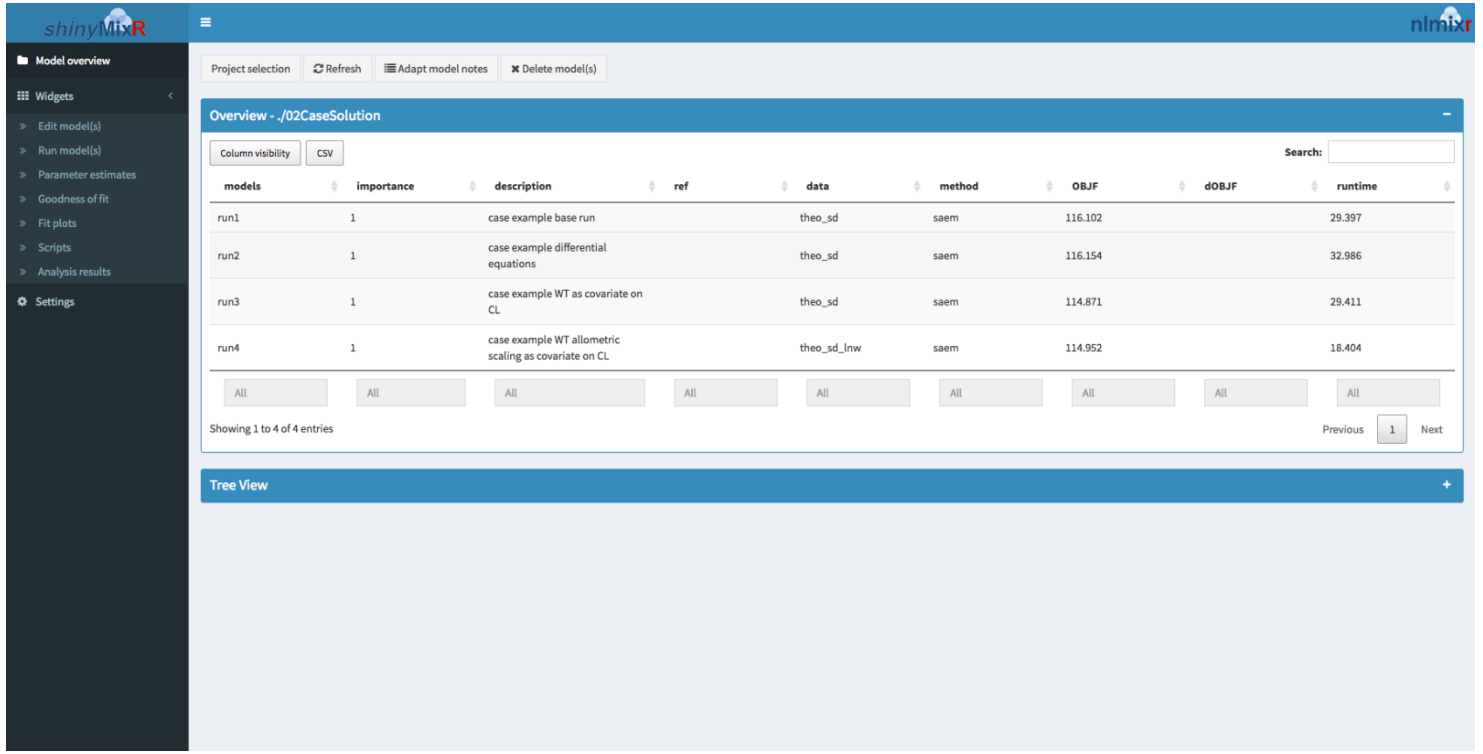
One error model per endpoint/compartment

- Error parameters to be estimated in ini block
- **|compartment** → needed to specify which error is for which compartment

Reynaldo-Fernández G. et al. Semi-mechanistic Pharmacokinetic/Pharmacodynamic model of three pegylated rHuEPO and ior®EPOCIM in New Zealand rabbits. European Journal of Pharmaceutical Sciences 120 (2018) 123–132

A shiny GUI for nlmixr: shinyMixR

shinyMixR



nlmixr

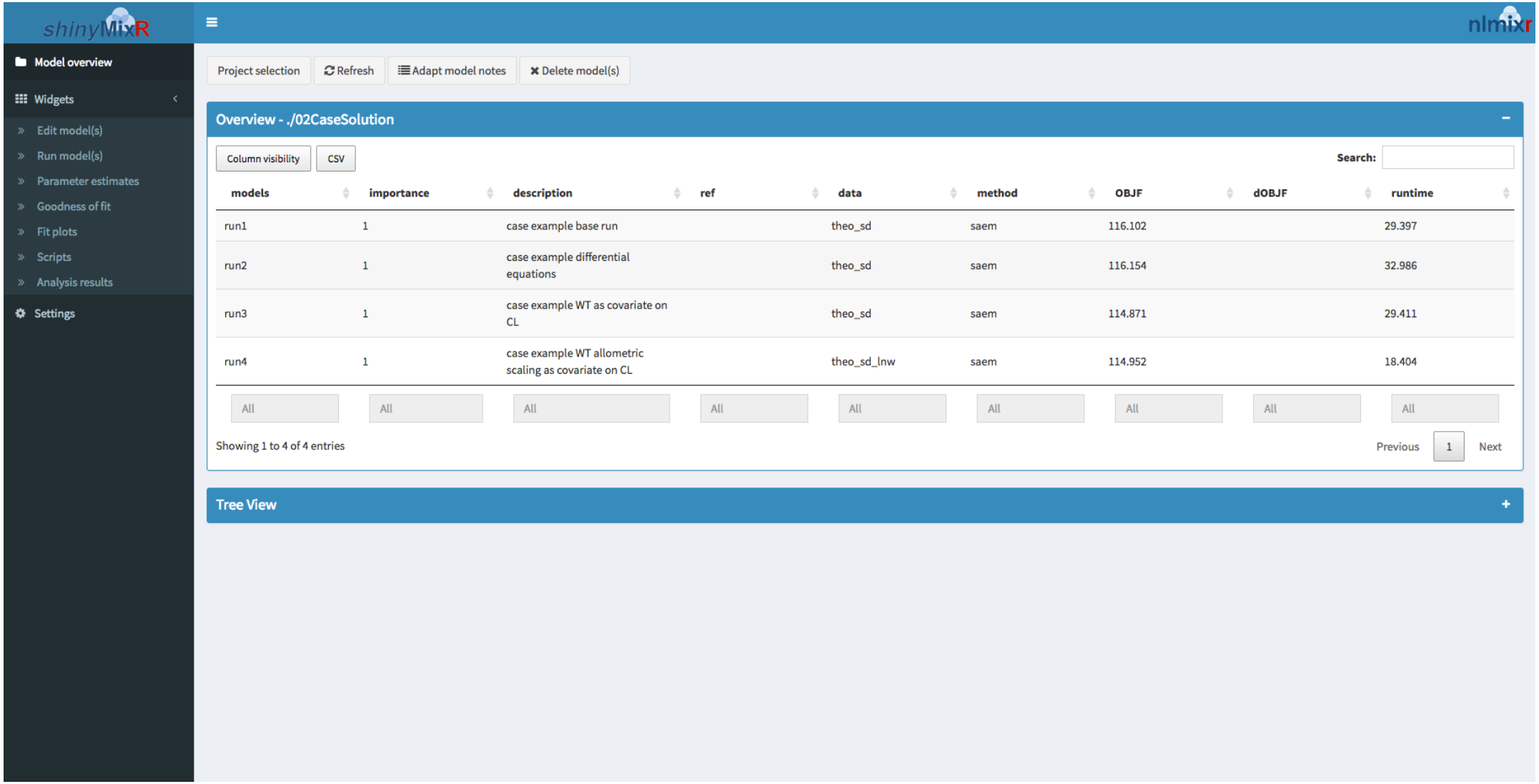
nlmixr and shinyMixR - background

There are two main ways of working with nlmixr models:

- Via the `nlmixr` package
  - `nlmixr` is the engine for running models
  - Provides output in a model fit object, which can be read out and approached
  - R cannot be used while running models – new R session can be opened
  - Model fit object is in the global environment

- Via the `shinyMixR` package
  - Provides a **graphical user interface** around `nlmixr`
  - Structures a project
  - Models are submitted in separate sessions – R can be used while running models
  - Model fit object is automatically saved to disk
  - Modeling output and models run via `nlmixr` cannot be imported
  - *Note*: several `shinyMixR` package functions can also be used interactively on command line

shinyMixR


shinyMixR
nlmixr
Model overview
Widgets
Edit model(s)
Run model(s)
Parameter estimates
Goodness of fit
Fit plots
Scripts
Analysis results
Settings
Project selection
Refresh
Adapt model notes
Delete model(s)
Overview - ./02CaseSolution
Column visibility
CSV
Search:
models
importance
description
ref
data
method
OBJF
dOBJF
runtime
run1 1 case example base run theo_sd saem 116.102 29.397
run2 1 case example differential equations theo_sd saem 116.154 32.986
run3 1 case example WT as covariate on CL theo_sd saem 114.871 29.411
run4 1 case example WT allometric scaling as covariate on CL theo_sd_lnw saem 114.952 18.404
All
Showing 1 to 4 of 4 entries
Previous
1
Next
Tree View

Introduction

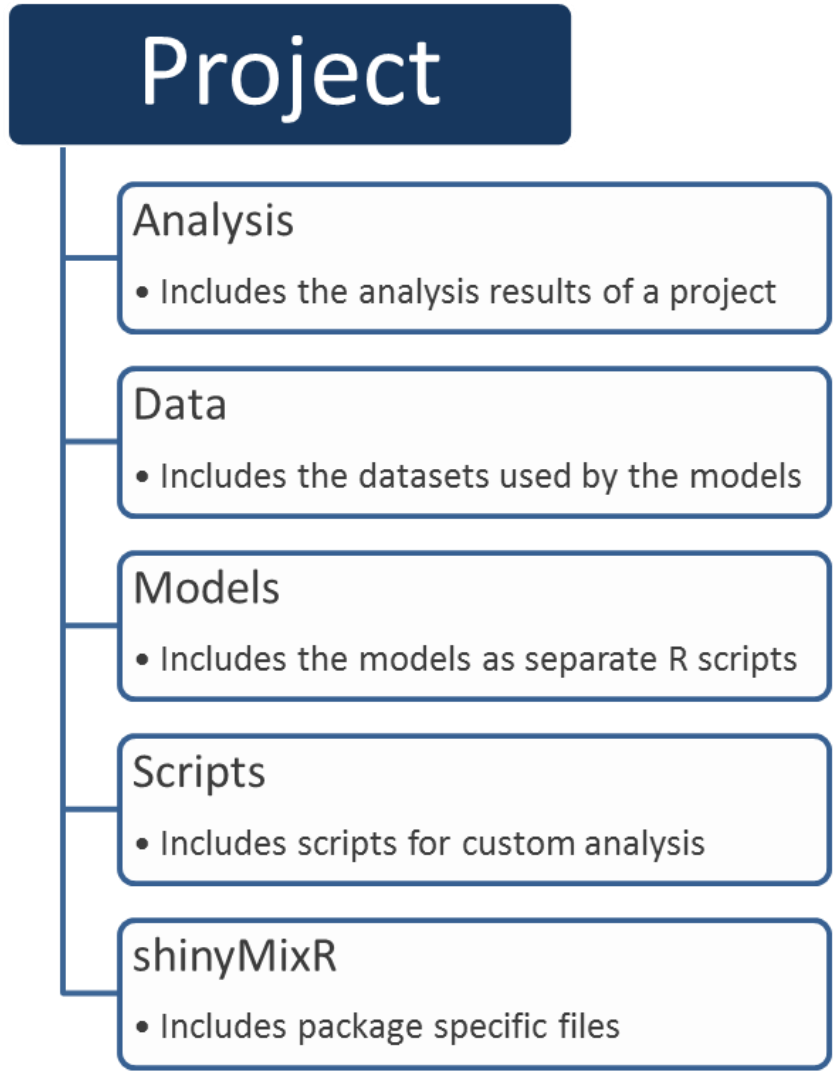
- The `shinyMixR` package is a **graphical user interface (GUI)** tool for managing pop PKPD projects with `nlmixr` as the estimation engine
- The package is intended to **view, edit, run, compare, analyze and report** nlmixr models
- It organizes your project – model, data, metadata, settings and results are kept together
- The interface enables browsing between specific project folders
- The application can be started via a shortcut or via the R command line in the project folder
- The interface is created using the `shiny` and `shinydashboard` packages
- *Note*: 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

Project Structure

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

The folder structure is important because:

1. The package monitors changes in specific folders and keeps track of this in a project object
2. Files are read and saved from specific locations to disk
3. When working with many models an organized folder structure is key – model, data, metadata, setting and results are kept together



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

Project Object

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

- **The object has information regarding the available models, meta data, high level results, etc.**
- All changes within a project are monitored/saved to disk in this object:
  - When model is changed
  - When new results are generated
  - When 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)
- *Note*: for an interactive session, in some cases updating of this object can be done using the `get_proj()` function

nlmixr and shinyMixR - nuances

nlmixr – model and nlmixr fit function

```
m1 <- function() {
  ini({
    tka <- .5
    tcl <- -3.2
…
fit1 <- nlmixr(m1, theo_sd, est="saem", … )
```

run `nlmixr()` via R or use the **Run model(s)** widget via `shinyMixR`

`nlmixr` arguments moved to metadata within the model code (run1.R model file) – analogous to NONMEM $PROBLEM, $DATA, $EST, etc.

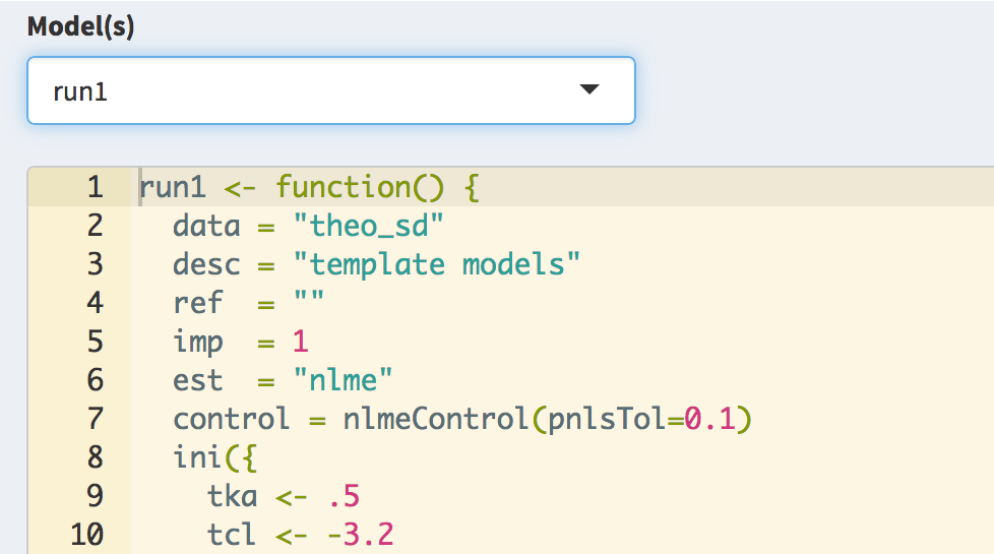
`shinyMixR` stores results in R data objects (e.g. run1.res.rds)

shinyMixR – metadata and run button or run_nmx function

```
run1 <- function() {
  data = "theo_sd"                    # csv or rds file
  desc = "case example base run"      # model description
  ref  = ""                           # model reference
  imp  = 1                            # model importance
  est  = "saem"                       # estimation method
  control = list()                    # est control options
  ini({
    tka <- .5
    tcl <- -3.2
…
```

Overview of shinyMixR functionality

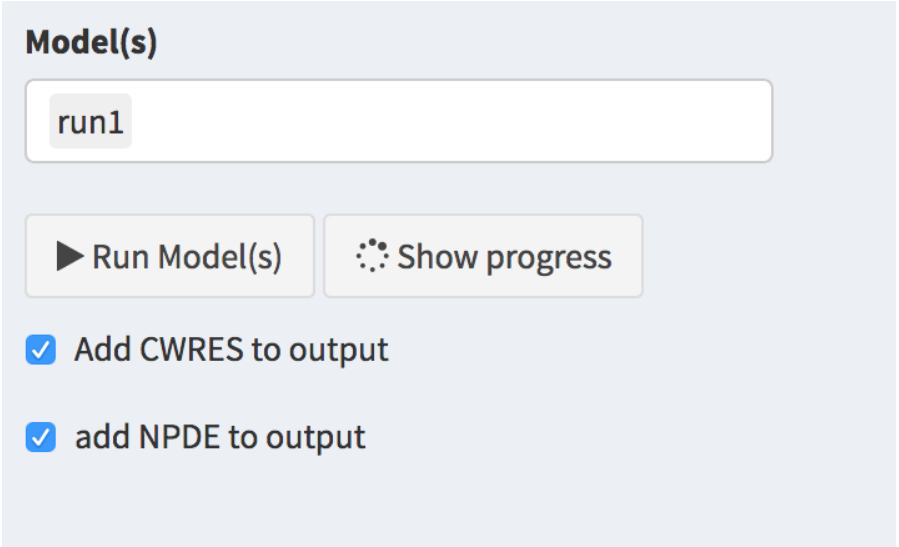
Edit Models



The **edit model(s)** widget is used to edit models within an editor including syntax coloring (using **shinyAce**).

It is also possible to create new models using various templates or to duplicate existing models.

Run Models



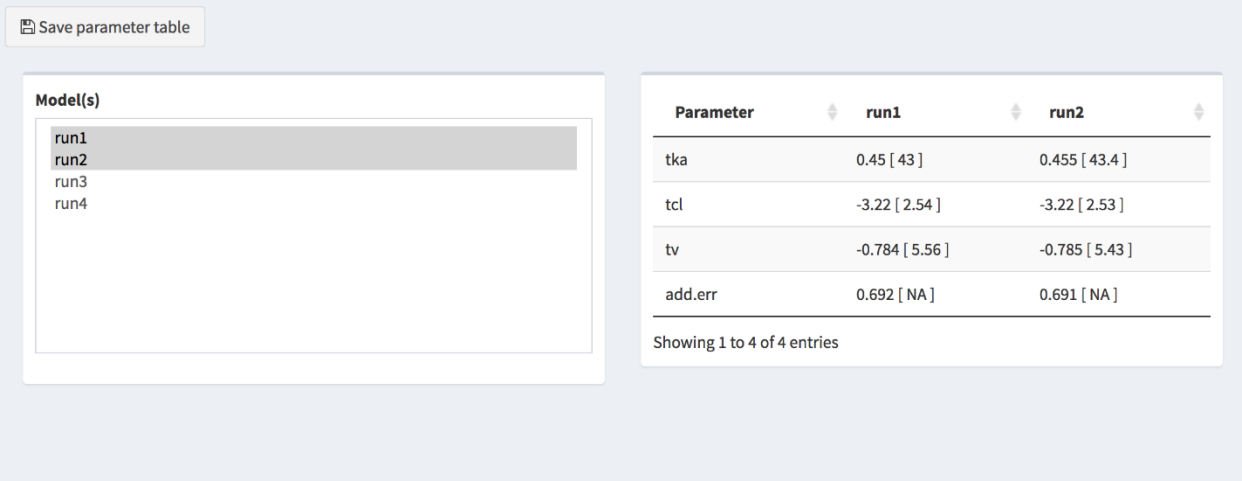
The **run model(s)** widget is used to run models. It is possible to run one or multiple models at once.

It is also possible to assess the intermediate output or progress for an nlmixr run.

data in the **data** folder; models in the **model** folder

Overview of shinyMixR functionality

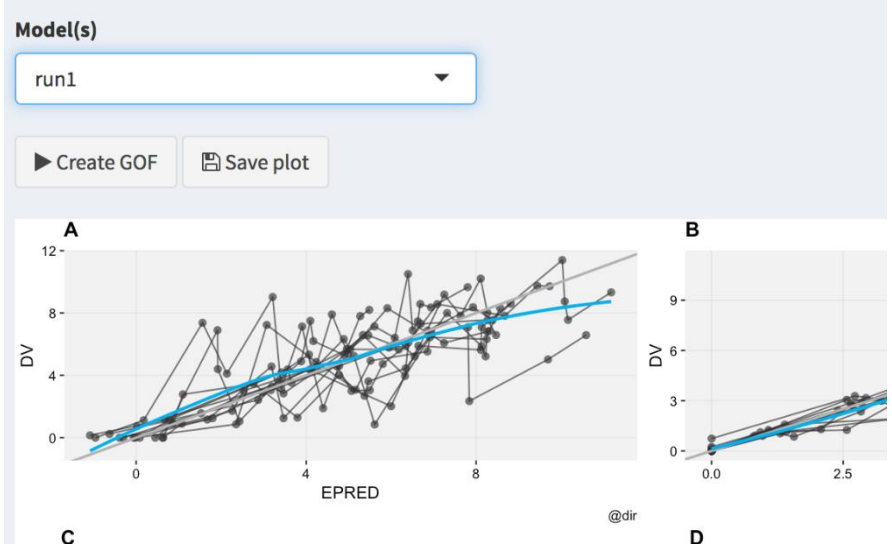
Parameter Estimates



The **parameter estimates** widget is used to generate a table with parameter estimates. In case multiple models are selected the table will show the results of each run in a separate column.

Output stored in the **analysis** folder; fit results and specific files in the **shinyMixR** folder

GOF Plots

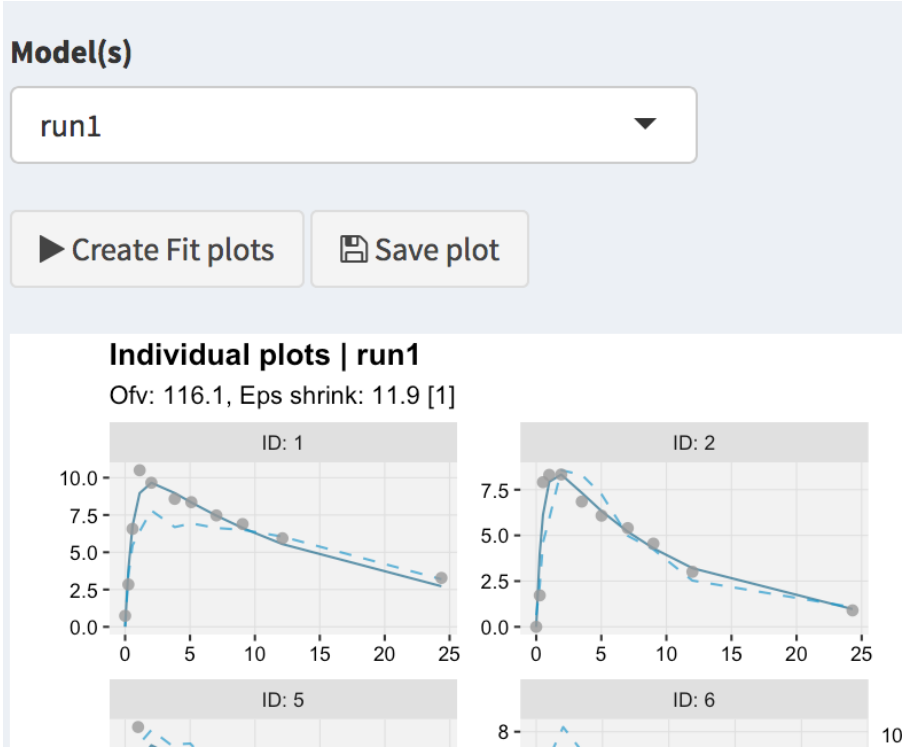


The **goodness of fit** widget is used to generate a combination of 4 goodness of fit plots combined.

By default **nlmixr.xpose** is used but direct **ggplot2** can also be used directly by specifying this in the **settings** widget.

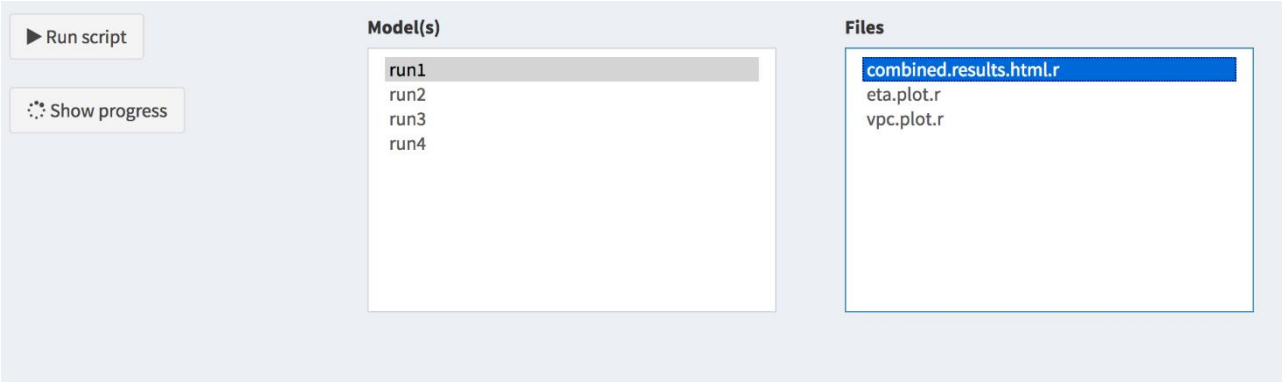
Overview of shinyMixR functionality

Fit Plots



The **fit plots** widget is used to generate individual fit plots. The same plotting options are present here as for the goodness of fit plots.

Scripts



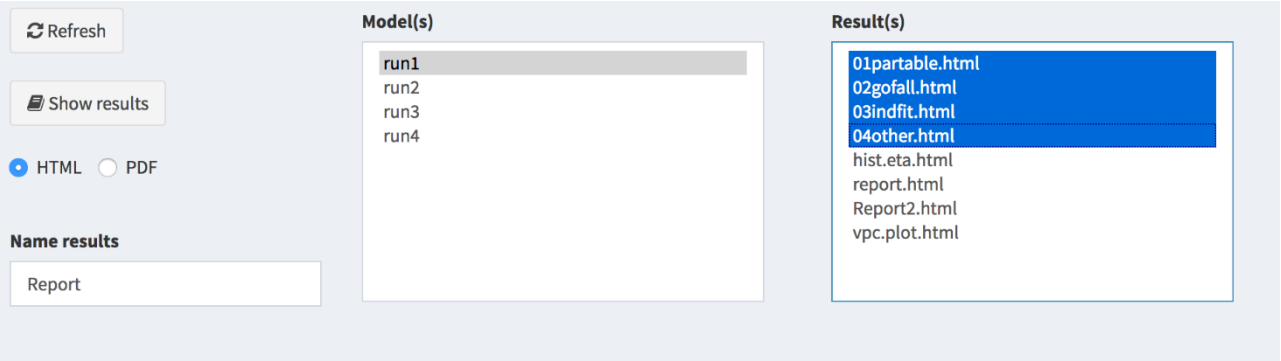
It is possible to write your own scripts (See **scripts** folder and **scripts** widget) that can be used to analyze model results.

The scripts can be used to process the result for one or multiple models at once (the interface will include the name of the selected models in the script).

Output stored in the **analysis** folder; scripts in the **scripts** folder; fit results and specific files in the **shinyMixR** folder

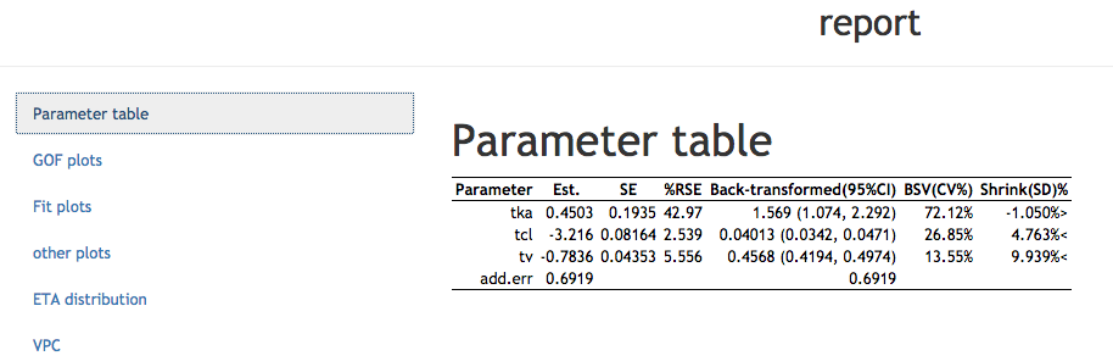
Overview of shinyMixR functionality

Analysis results



Output from Scripts and Widgets (plots and tables) are available in the **Analysis results** widget.

It is possible to save, view and combine the results from the models within a project into an HTML or PDF (if LaTeX is present) document.



Output stored in the **analysis** folder

Summary of shinyMixR

- The `shinyMixR` package is a **graphical user interface (GUI)** tool for managing popPKPD projects with `nlmixr` as the estimation engine
- It structures your project – model, data, metadata, settings and results are kept together and saves the results to disk
- The interface enables browsing between specific project folders
- The package has functionalities to view, edit, run, compare, analyze and report nlmixr models
- Models are submitted in separate sessions – R can be used while running models