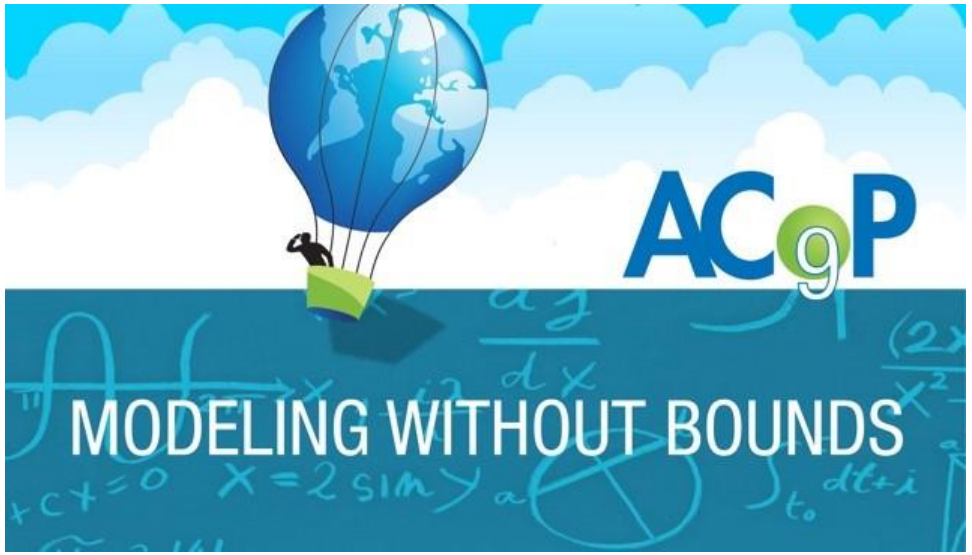


Tutorial 1:
nlmixr: Population Modeling in R

Chairs:

Matthew Fidler and Wenping Wang

Hands On Tutors:

Teun M. Post, Yuan Xiong, Mirjam N. Trame, Justin Wilkins, and Rik Schoemaker

Introduction to nlmixr: an open-source package for pharmacometric modelling in R

Matt Fidler

On behalf of the nlmixr development team:

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

nlmixr

San Diego, California

October 7 – 11, 2018

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

Why open-source in R?

Default tool for many

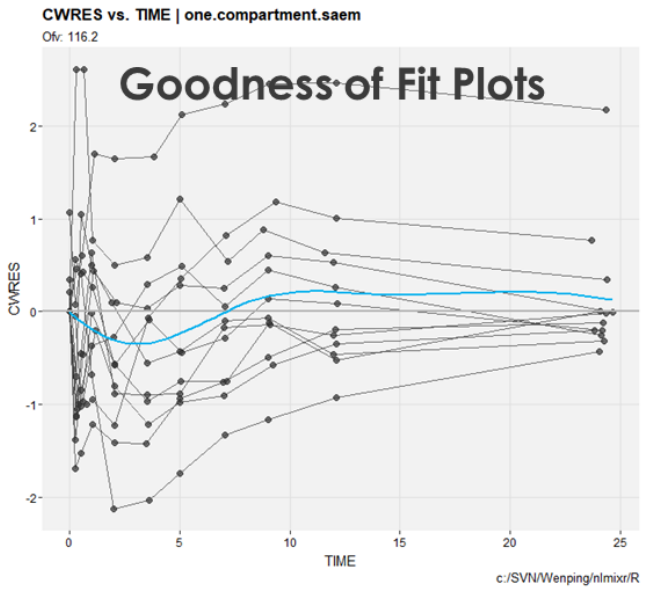
- Default tool for most pharmacometricians for exploratory analyses, model assessment and prediction
- Increasing use for data management/data preparation/report building
- Increasing use in statistics over SAS these days

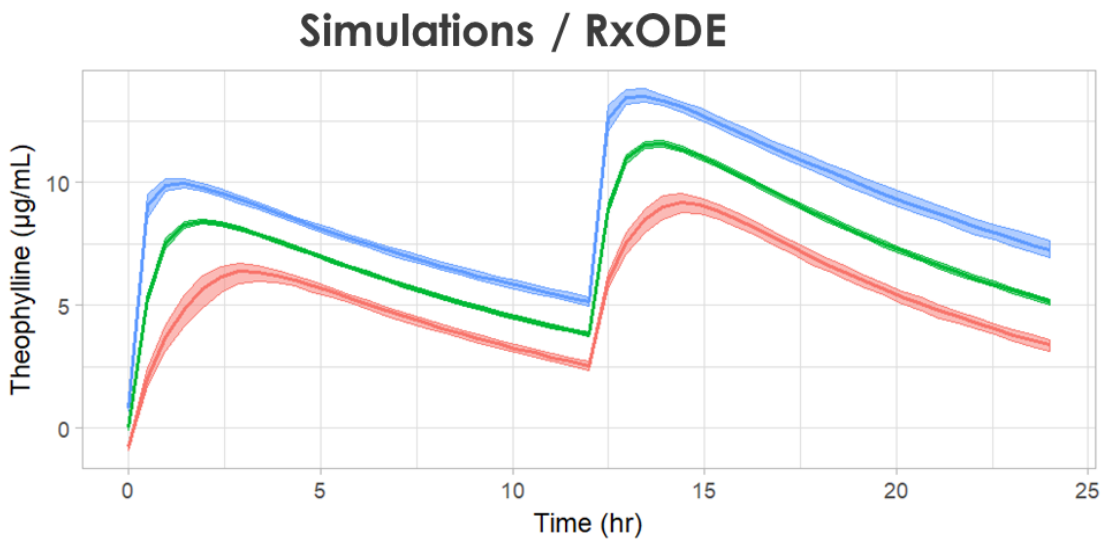
R is a self-sufficient ecosystem

- Rich collection of numerical tools (BLAS, LAPACK, RcppEigen. RcppArmadillo, optimization routines)
- Powerful graphical tools (ggplot2, lattice, grid)
- Powerful report generation tools (knitr, bookdown, pkgdown)
- Convenient interface to general programming languages (.C, .Fortran, .Call, Rcpp)

Model Syntax

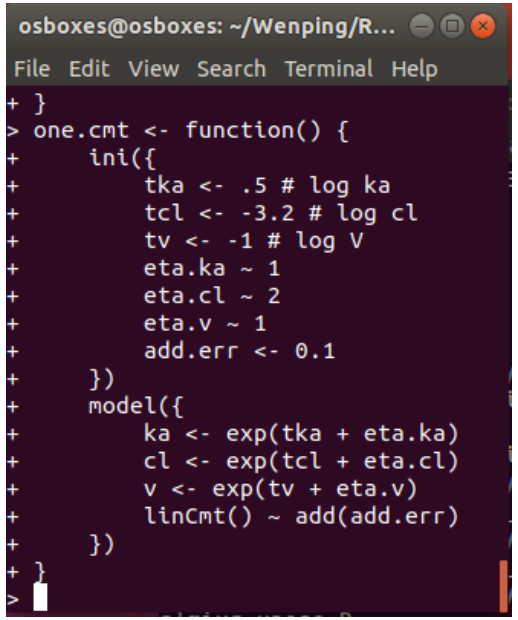
```
osboxes@osboxes: ~/Wenping/R...
File  Edit  View  Search  Terminal  Help
+ }
> one.cmt <- function() {
+     ini({
+         tka <- .5 # log ka
+         tcl <- -3.2 # log cl
+         tv <- -1 # log V
+         eta.ka ~ 1
+         eta.cl ~ 2
+         eta.v ~ 1
+         add.err <- 0.1
+     })
+     model({
+         ka <- exp(tka + eta.ka)
+         cl <- exp(tcl + eta.cl)
+         v <- exp(tv + eta.v)
+         linCmt() ~ add(add.err)
+     })
+ }
>
```





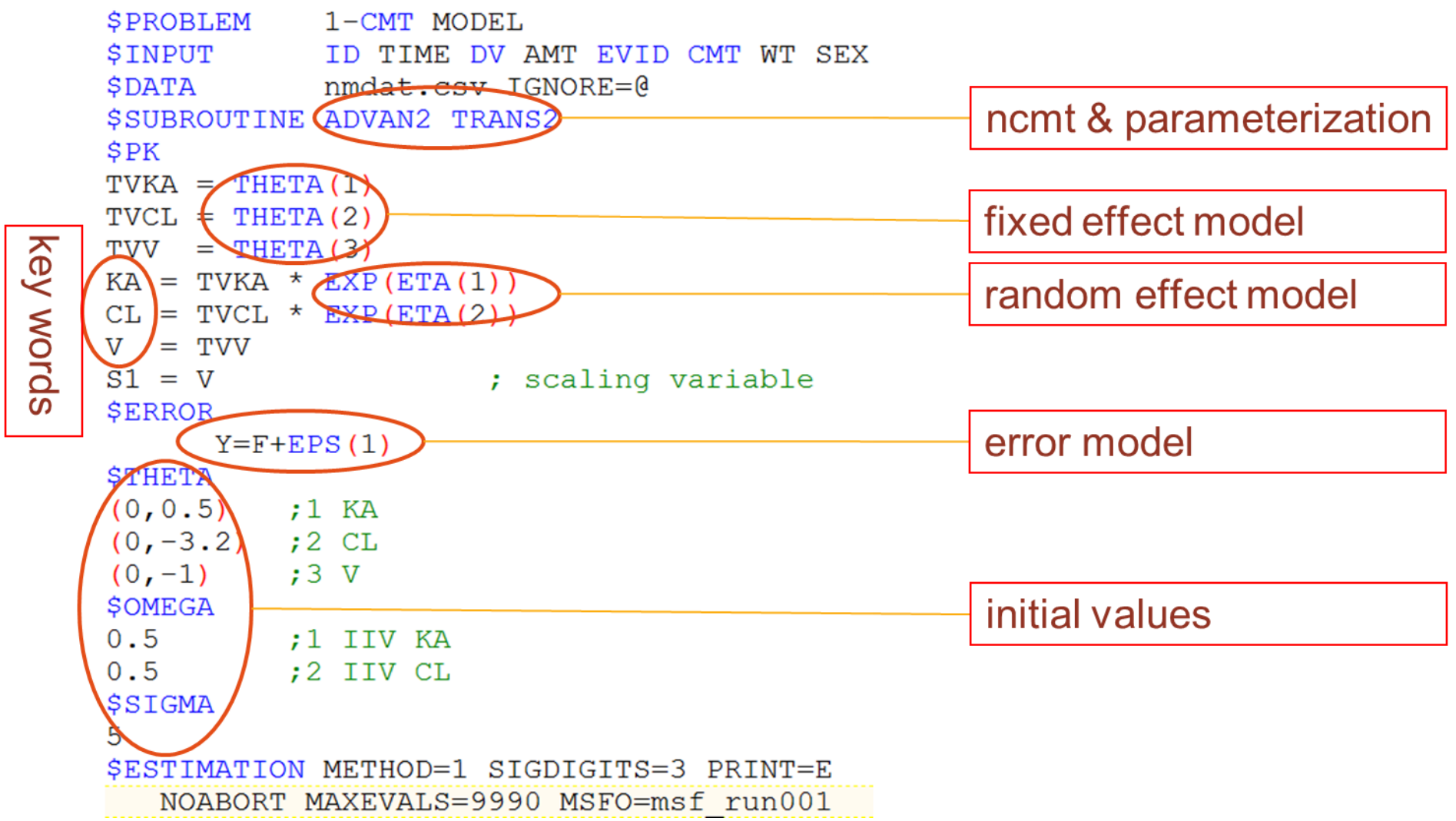
nlmixr

modeling syntax, running nlmixr models and nlmixr output



nlmixr

Anatomy of a NONMEM control stream for a popPK model



nlmixr

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

NONMEM Dataset

NONMEM Dataset converter to convert EVID column to nlmixr syntax

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

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

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")
```

NONMEM $PK like

Initial values for fixed effects

Initial values for random effects

Initial values for error model

ADVAN & TRANS like

A nlmixr model has two main parts: initialization and model

Initialization ini({ })

```
ini({
    lCl    = 1.6
    lVc    = log(90)
    lKa    = fix(1)
    add.err = c(0,0.2,1)
    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.err)
})
```

- 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, Vc, Q, Vp	Cl, Vc, Q, Vp1, 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
- 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)
      Q = exp(lCld)
      linCmt() ~ prop(prop.err)
})
```

- 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

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

Initialisation ini({ })

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

➢ instead of linCmt

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

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( uif, data, est = "saem", table=tableControl(cwres=TRUE, npde=TRUE))
```

- `fit <-` : Assigned R object
- `uif, data,` : xpose 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.err) + prop(prop.err)  | depot
PD ~ add(pd.err)                    | 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
    p.err    = 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

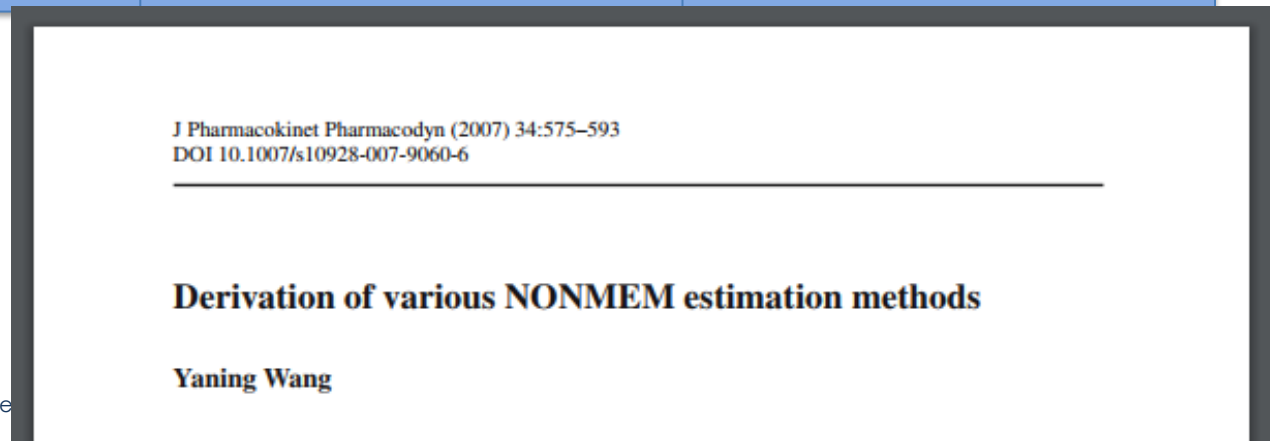
```
## eg nlmixr(modelfn, data, "nlme");
```

nlmixr's FOCEi-related objective functions reproduces NONMEM's FOCEi-related objective functions

Model Type	Error Model	NONMEM	OBJF nlmixr
FOCEi	Additive	-2.059	-2.059
FOCE		-2.059	-2.059
FO		0.026	0.026
FOCEi	Proportional	39.458	39.458
FOCE		39.207	39.207
FO		39.213	39.213
FOCEi	Additive + Proportional	39.735	39.735
FOCE		39.499	39.499
FO		39.505	39.505

Wang 2007 provided an example dataset and NONMEM OBJF for FOCEi

Used same examples as Wang's papers and add+prop with add=0.1 and prop=0.1

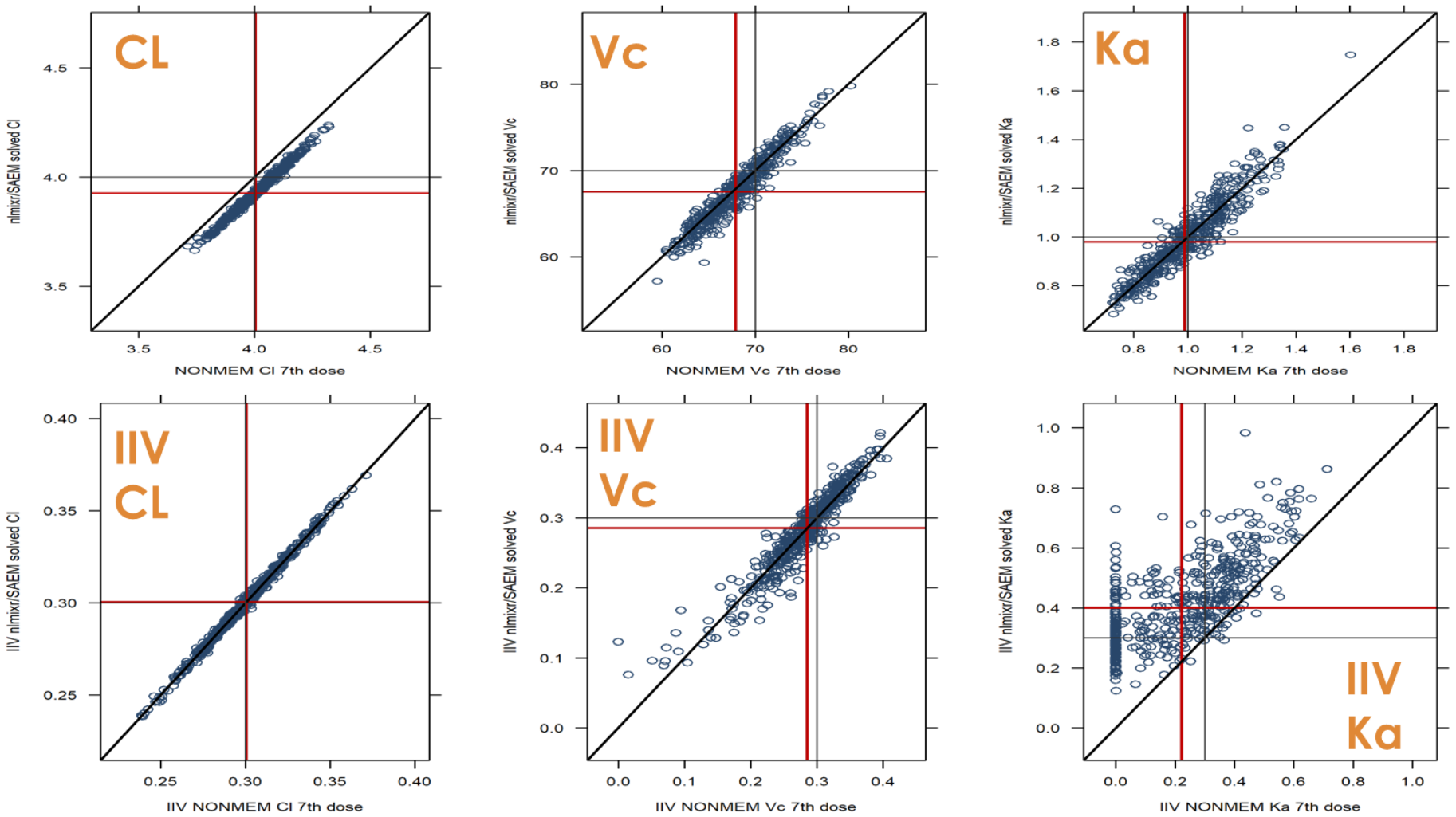

J Pharmacokinet Pharmacodyn (2007) 34:575–593
DOI 10.1007/s10928-007-9060-6

Derivation of various NONMEM estimation methods

Yaning Wang

nlmixr/SAEM sparse day 7 estimates vs NONMEM/FOCEi estimates

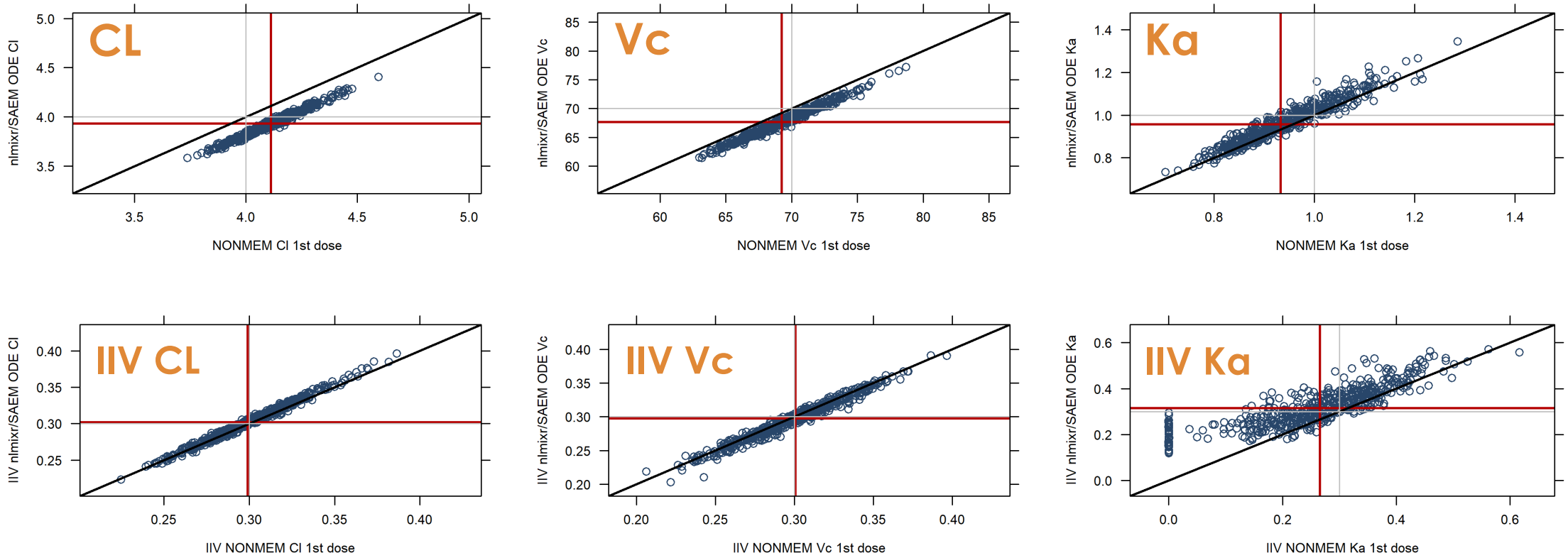
Nlmixr SAEM Estimates



NONMEM FOCEi Estimates

Sparse data analysis results for 1st dose: NONMEM FOCE-I solved vs. nlmixr/SAEM ODE

NImixr SAEM Estimates



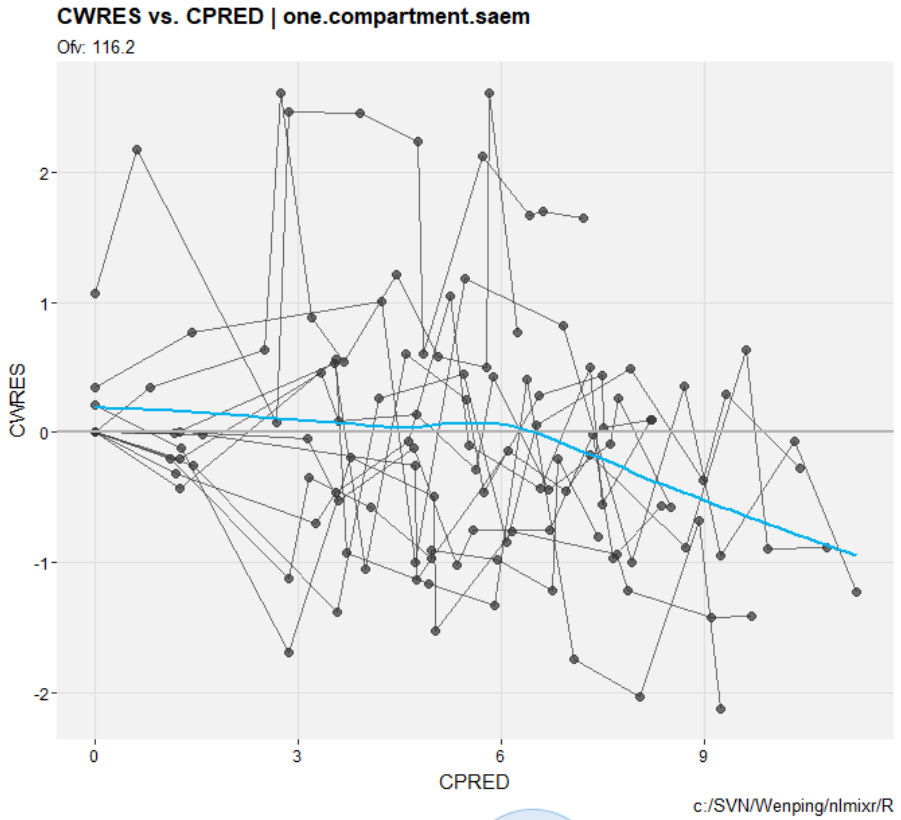
NONMEM FOCEi Estimates

Gray solid lines are "truth"; red solid lines are averages of estimates from NONMEM and nlmixr.

Covariance types in nlmixr:

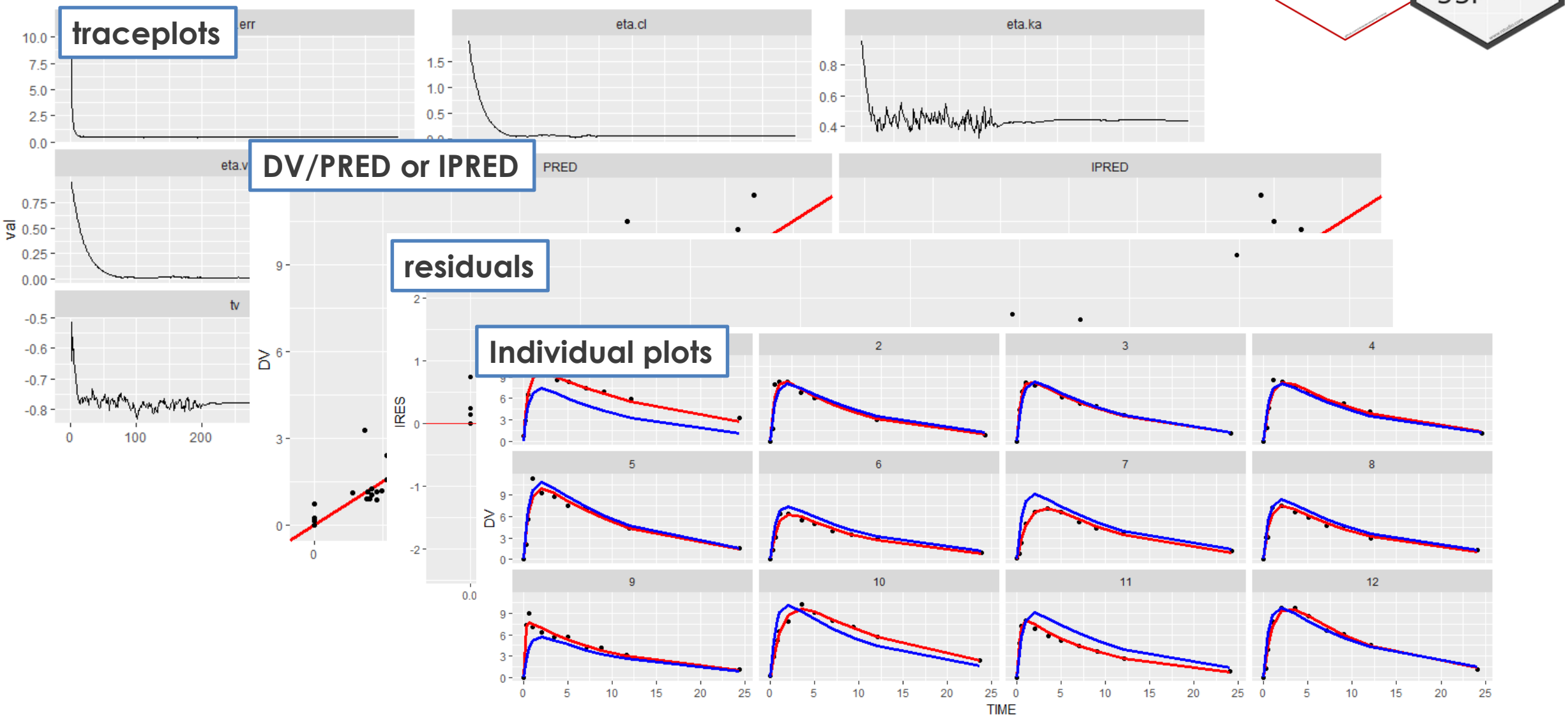
- FIM (fim)
  - SAEM estimate of the Fisher Information Matrix; cov = $(\mathbf{fim})^{-1}$
- R-matrix (r)
  - Same as NONMEM **R**= ½ Hessian Matrix; covR= $(2\mathbf{R})^{-1}$
- S-matrix (s)
  - Same as NONMEM **S** = ¼ Gradient Cross-product Matrix; covS= $4(\mathbf{S})^{-1}$
- R,S; sandwich matrix (r, s)
  - Same as NONMEM, covRS=$\mathbf{R}^{-1} \cdot \mathbf{S} \cdot \mathbf{R}^{-1}$
- Corrected Matrices
  - Regard covariance with caution
  - Positive definite matrices can be corrected by:
    - Adding a bit to the diagonal (r+ or s+; Uses generalized Cholesky decomposition)
    - By using the following equation $\mathrm{sqrtm}(\mathbf{M} \cdot \mathbf{M}^T)$. Can be |fim|, |r|, or |s|
  - Uses pseudo-inverse for non-invertable matrices

diagnostic plots from a nlmixr model



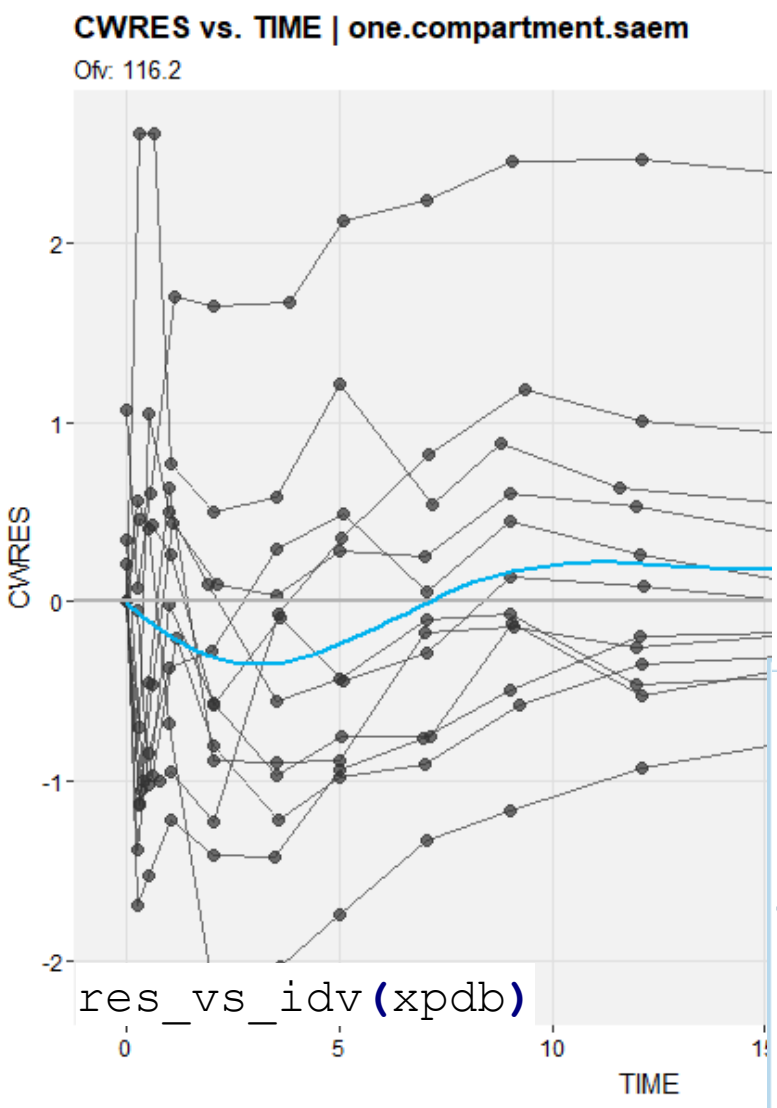
nlmixr

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

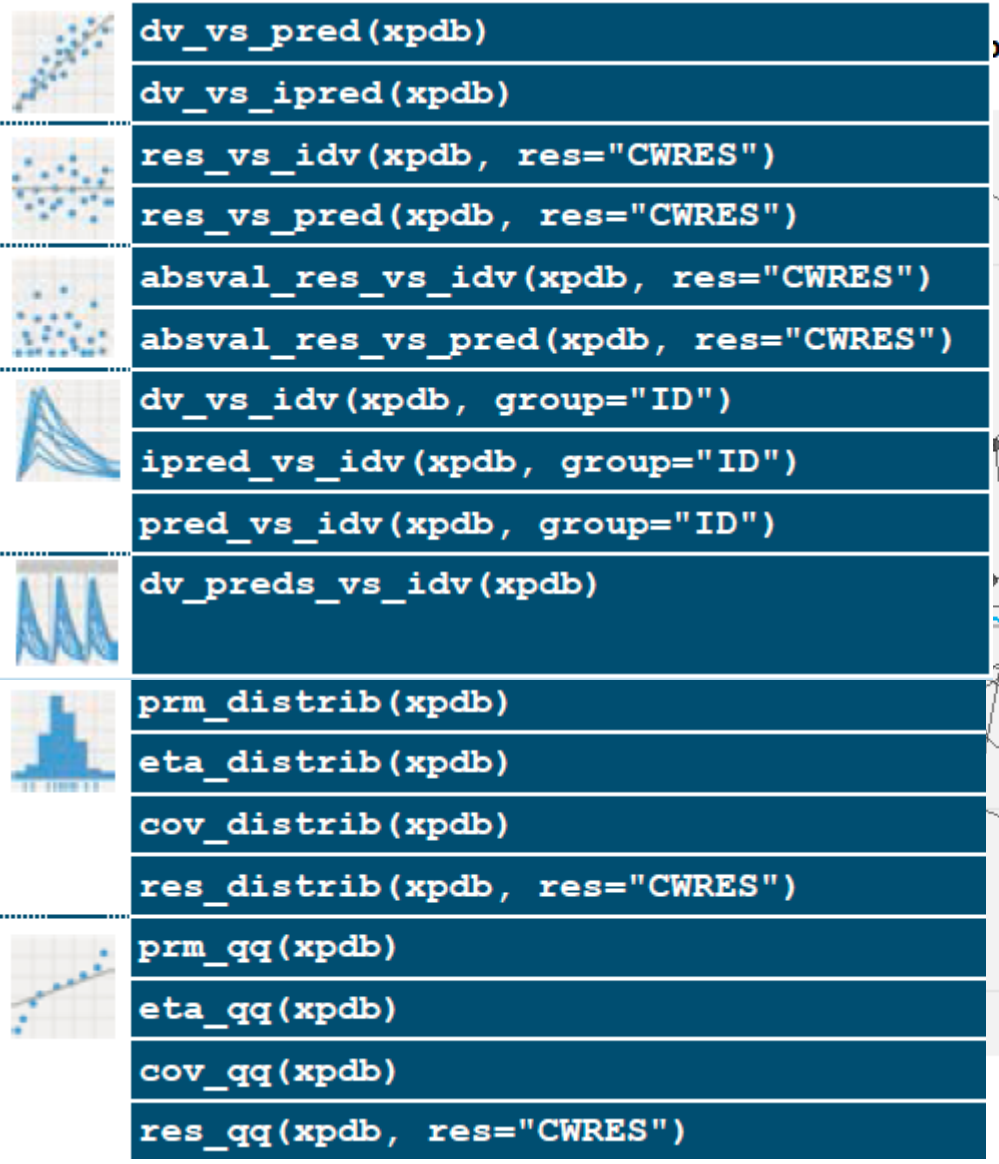


Xpose Plots can be created by converting fit to xpose object

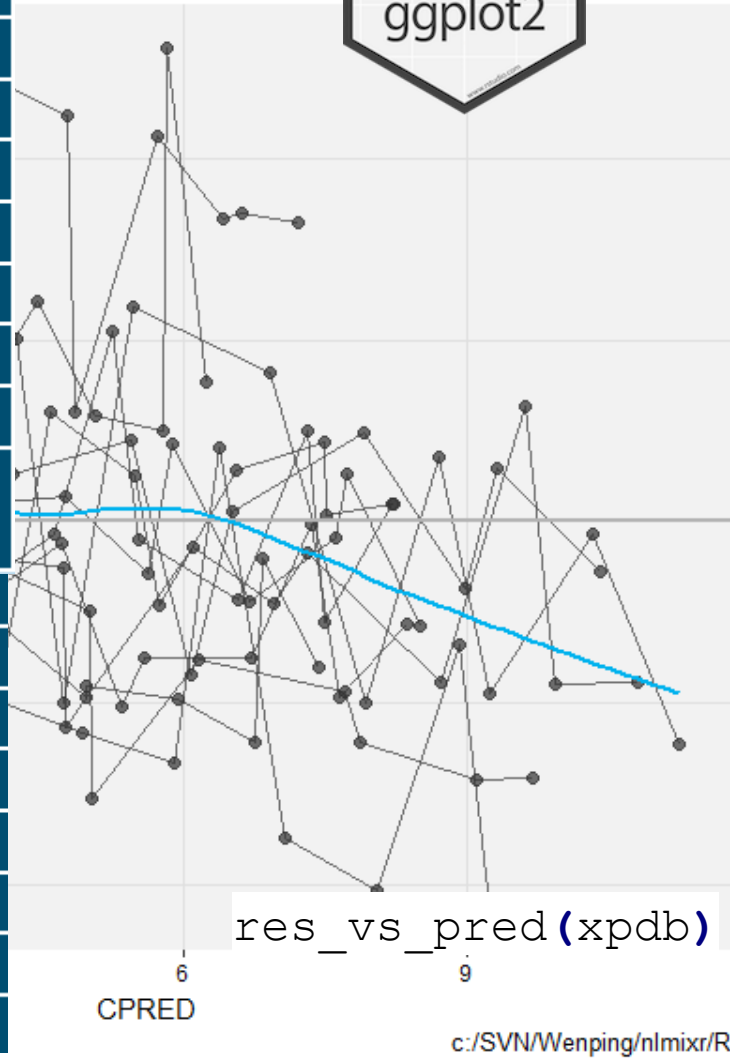
```
xpdb<-xpose_data_nlmixr(fit,xp_theme= theme_xp_nlmixr())
```



res_vs_idv(xpdb)



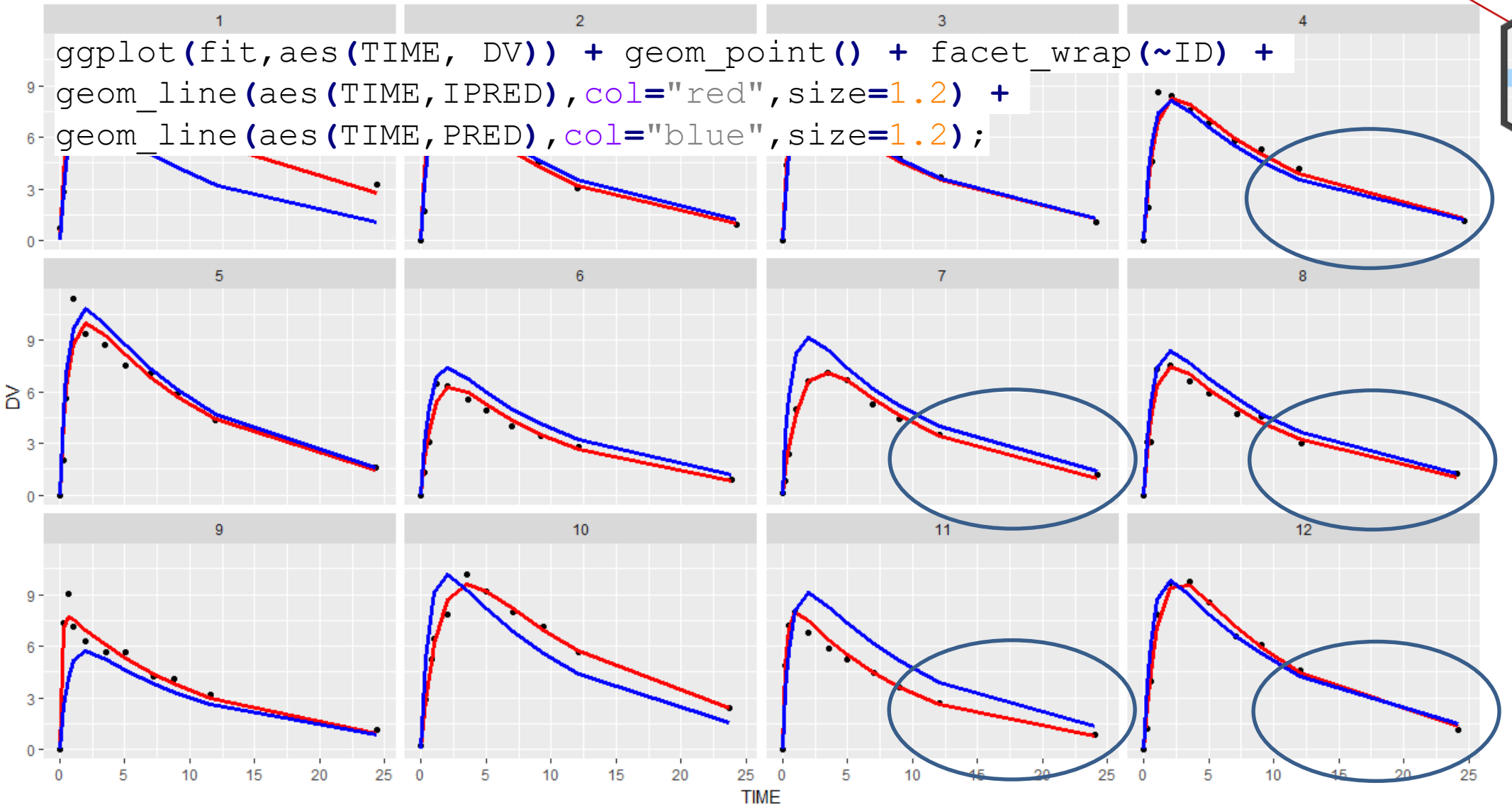
```
dv_vs_pred(xpdb)
dv_vs_ipred(xpdb)
res_vs_idv(xpdb, res="CWRES")
res_vs_pred(xpdb, res="CWRES")
absval_res_vs_idv(xpdb, res="CWRES")
absval_res_vs_pred(xpdb, res="CWRES")
dv_vs_idv(xpdb, group="ID")
ipred_vs_idv(xpdb, group="ID")
pred_vs_idv(xpdb, group="ID")
dv_preds_vs_idv(xpdb)
prm_distrib(xpdb)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

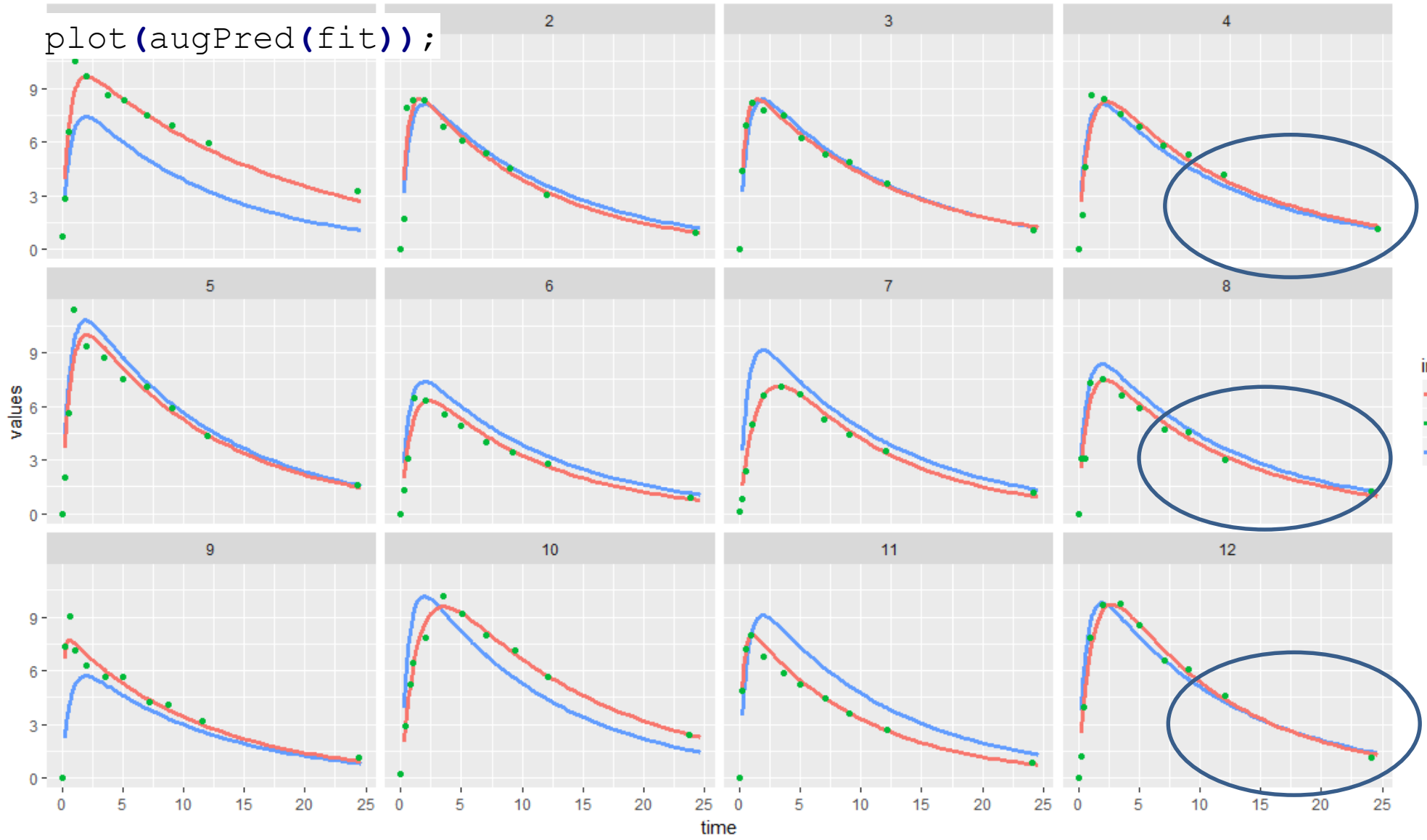
Individual Plots Using Fit Data
Shows observations & connects the dots for individual points

```
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);
```



augPred predicts model points outside of observed this smooths the model predictions

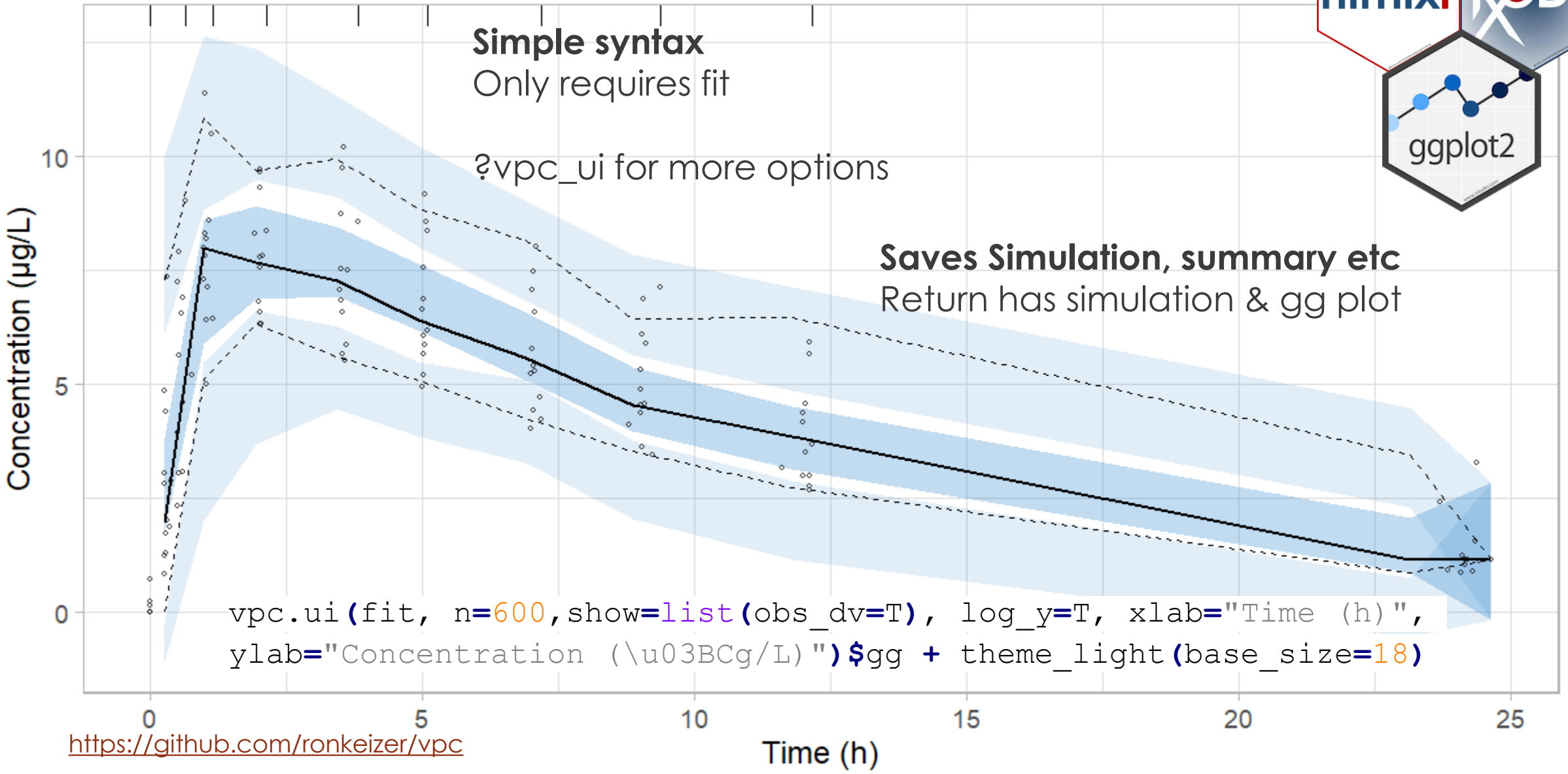
```
plot(augPred(fit));
```



With Sparse data shows model with few data points

augPred alone returns a data.frame that can be manipulated as needed

Model Evaluation with Visual Predictive Check



Simple syntax
Only requires fit

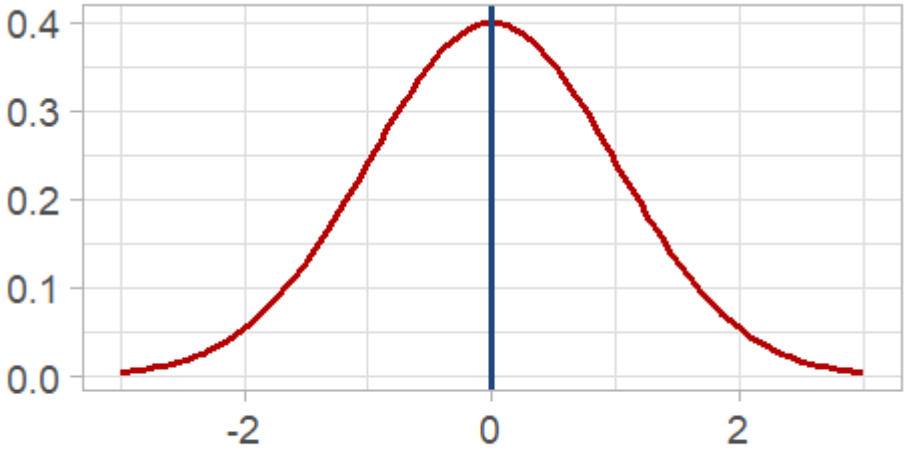
?vpc_ui for more options

Saves Simulation, summary etc
Return has simulation & gg plot

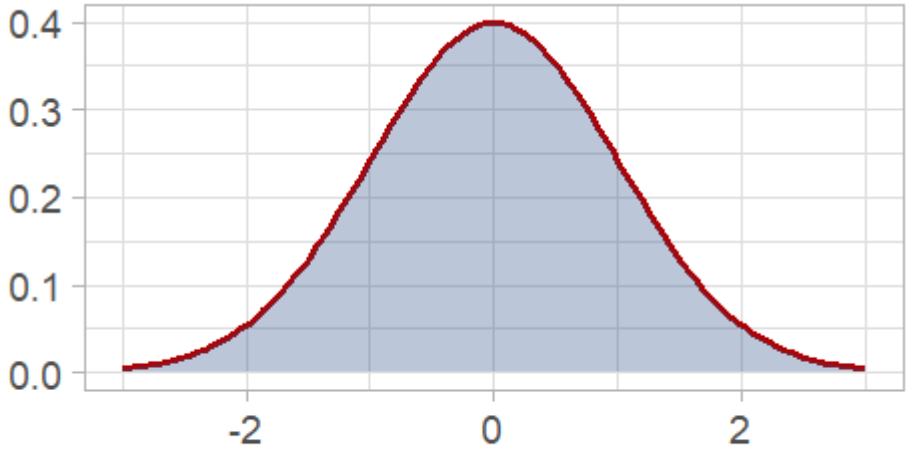
```
vpc.ui(fit, n=600,show=list(obs_dv=T), log_y=T, xlab="Time (h)",
ylab="Concentration (\u03BCg/L)")$gg + theme_light(base_size=18)
```

https://github.com/ronkeizer/vpc

simulating from a nlmixr model



Without uncertainty



With uncertainty

nlmixr

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?

Creating an eventTable with 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
)
```

add.dosing() 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);
```

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

Load nlmixr	`library(nlmixr)`
Fit model	`fit <- nlmixr(one.cmt, theo_sd, "saem")`

ODEs from nlmixr model block	`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

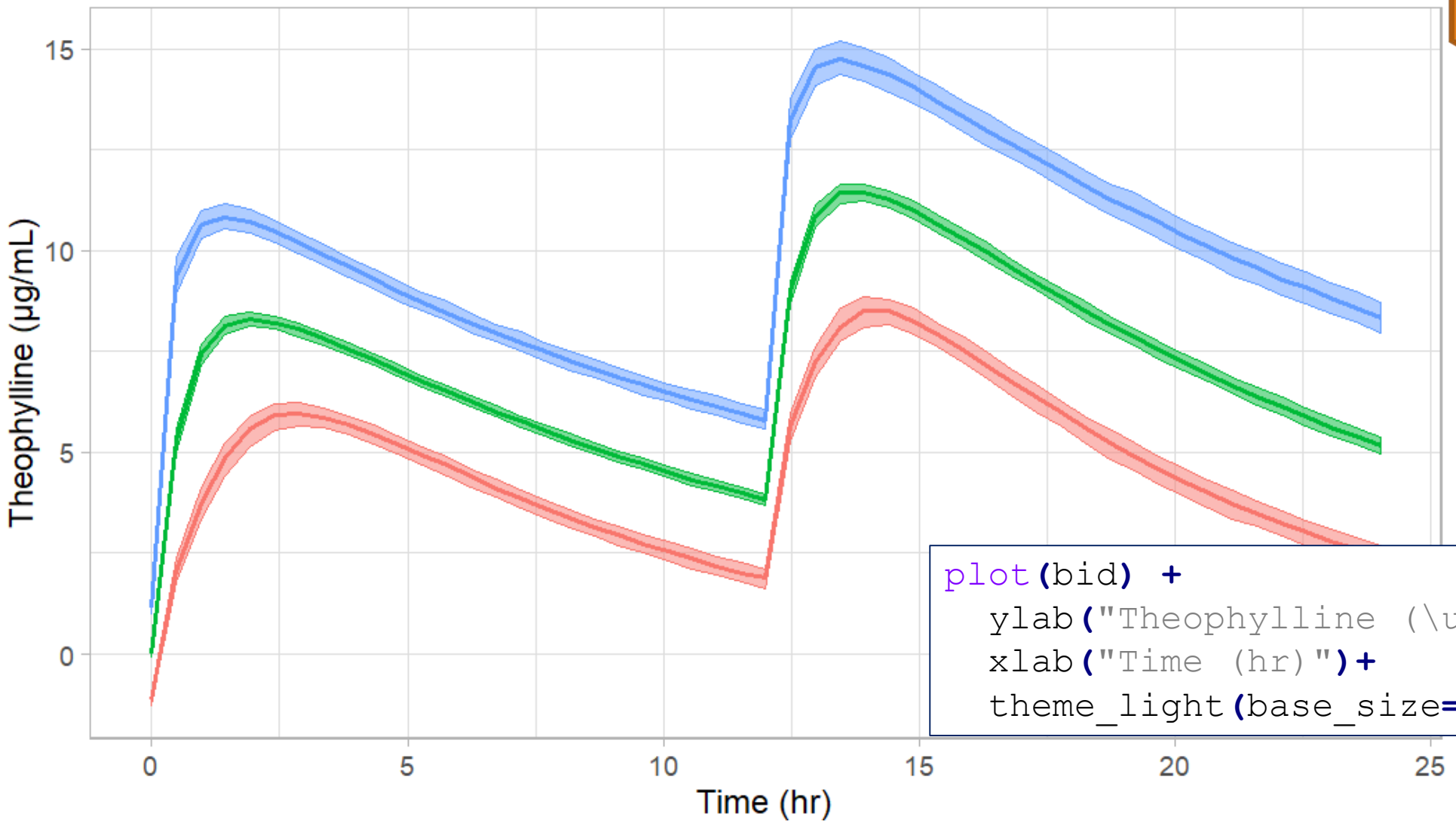
The simulation has information about the parameters that were used in the simulation from the model fit (= Solved RxODE object)

```
> 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)`
Fit model	`fit <- nlmixr(one.cmt, theo_sd, "saem")`

ODEs specified by nlmixr model	`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
```

Simulating with parameter uncertainty assumes that model estimates are distributed in certain ways

Population Parameter Estimates have a Multivariate Normal Distribution

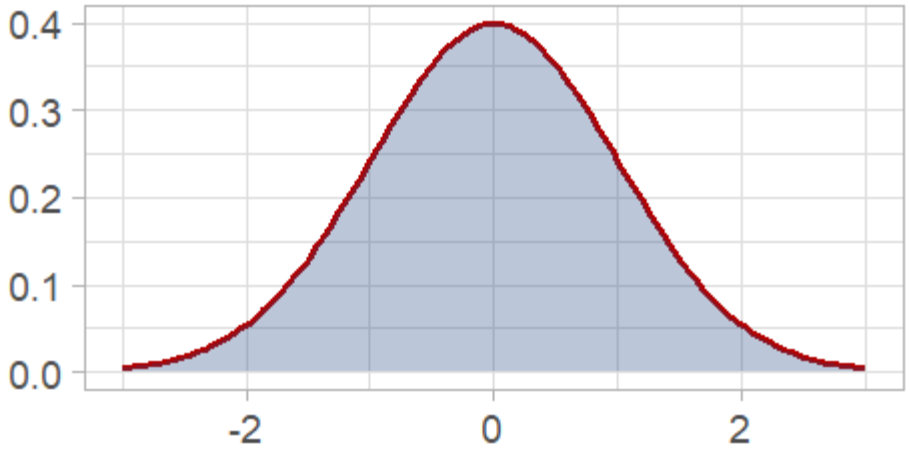
N(θ, COV(θ))

Can be simulated by mvnfast

Covariance Parameter Estimates have a Inverse Wishart (scaled) Distribution

Inv Wish(Cov, Degrees of Freedom=#Obs, #Sub)

Degrees of freedom=#Items used for Cov matrix



With uncertainty

nlmixr simulations

Used for:

- Between subject variability (df=#sub)
- Unexplained variability (df=#obs)

Simulated by RxODE cvPost

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
```

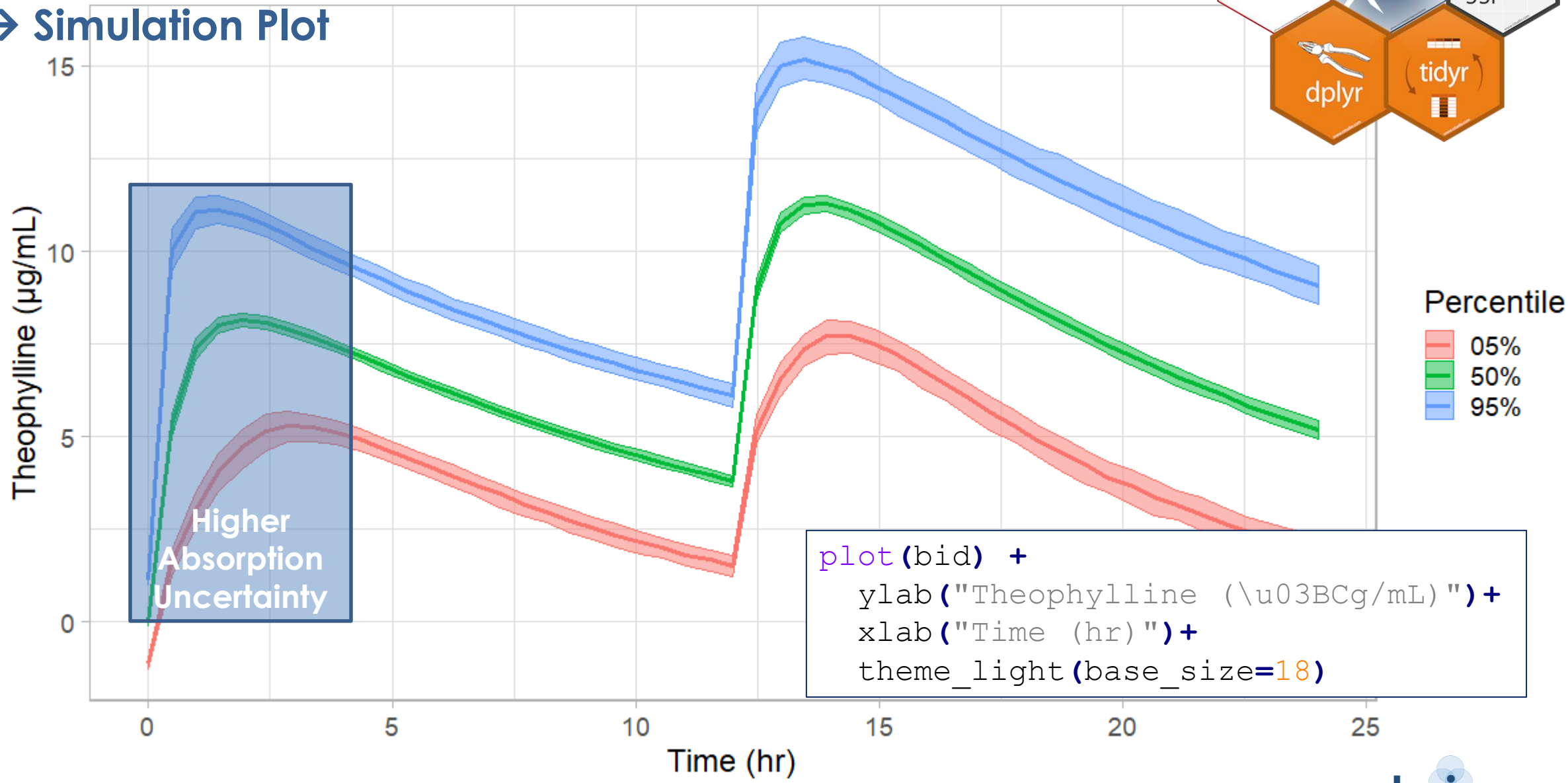
Sampled from Inverse Wishart (Scaled), df=#Sub

```
> 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)
```

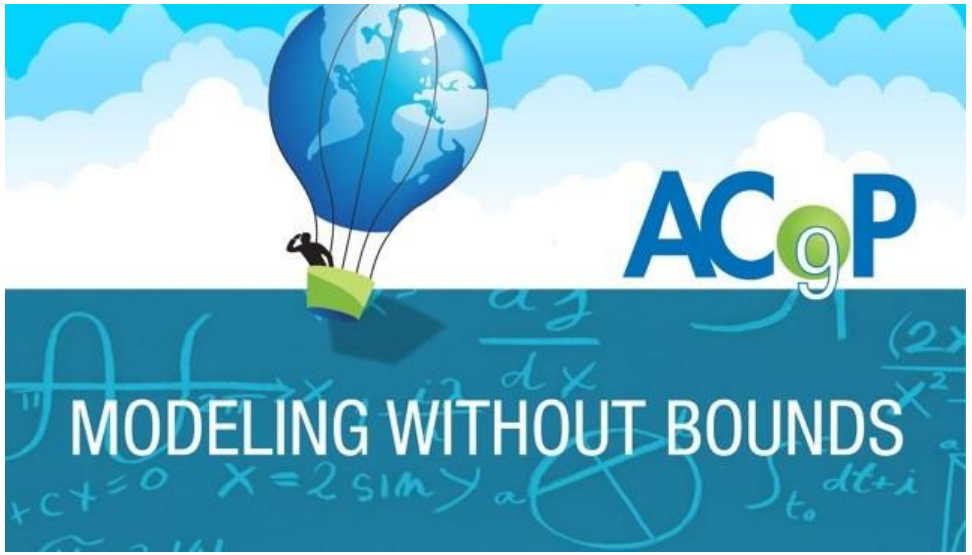
Hands-on

Case Theophylline

1) Open Run1.R and execute the model in R
   - Output the model parameter estimates
   - Plot the GOF plots and VPC
2) Code the model provided in Run1.R using ODE's
3) Use model Run1 and add covariate WT on CL
4) Use model Run1 and add covariate WT allometrically scaled

Case Nimotuzumab

1) Follow the introduction slides to Case Nimotuzumab (https://github.com/nlmixrdevelopment/PAGE-2018/blob/master/Case%20Study_1_PAGE2018.zip)
2) Additional examples: https://github.com/nlmixrdevelopment/PAGE-2018



Tutorial 1:
nlmixr: Population Modeling in R

Chairs:
Matthew Fidler and Wenping Wang
Hands On Tutors:
Teun M. Post, Yuan Xiong, Mirjam N. Trame, Justin Wilkins, and Rik Schoemaker

Introduction to the shinyMixR Package

Teun Post

On behalf of the nlmixr development team:
Matt Fidler, Richard Hooijmaijers, Teun Post, Rik Schoemaker, Mirjam Trame, Justin Wilkins, Yuan Xiong and Wenping Wang

nlmixr

San Diego, California **October 7 – 11, 2018**

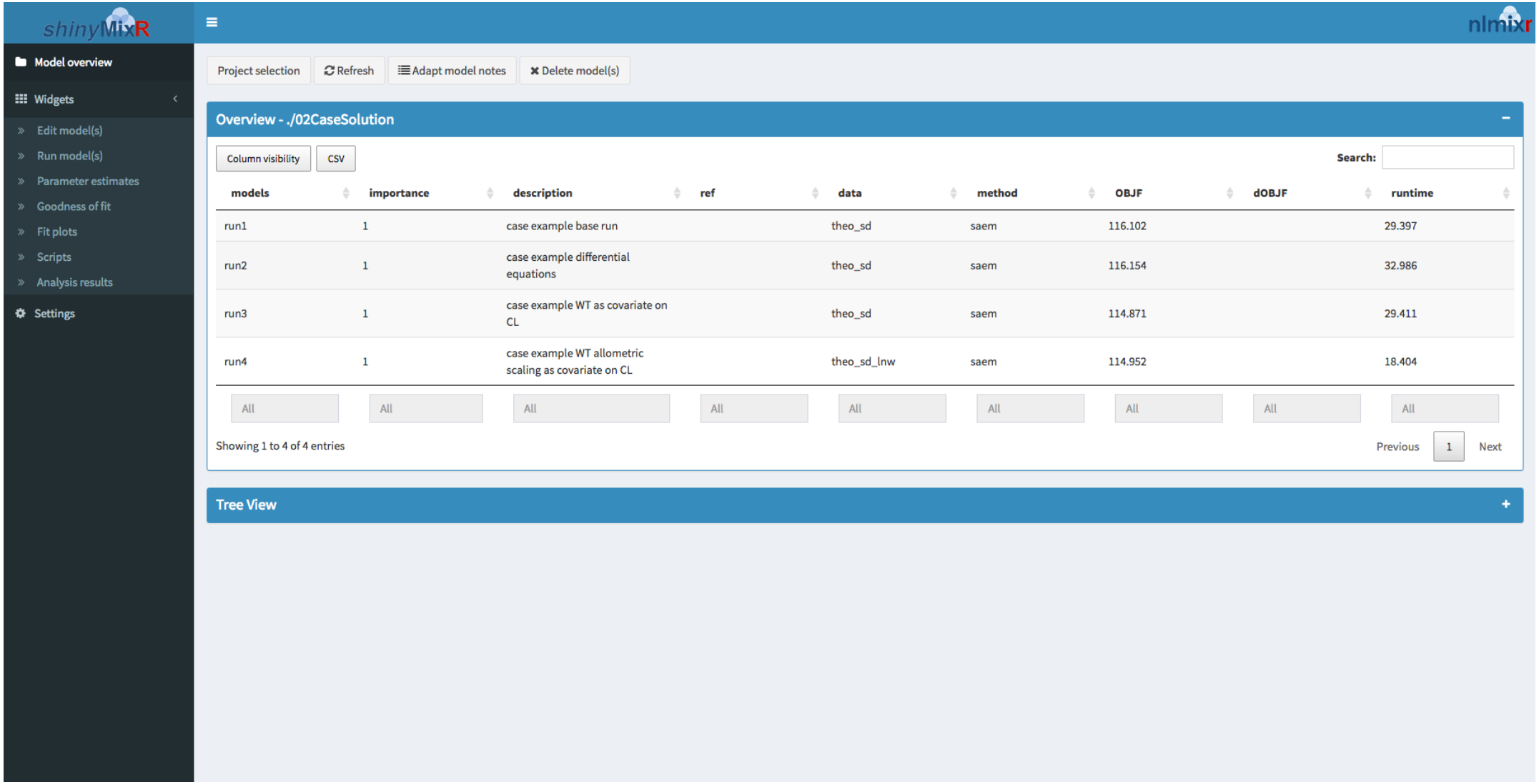
nlmixr and shinyMixR - background

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 – what does it look like



Introduction

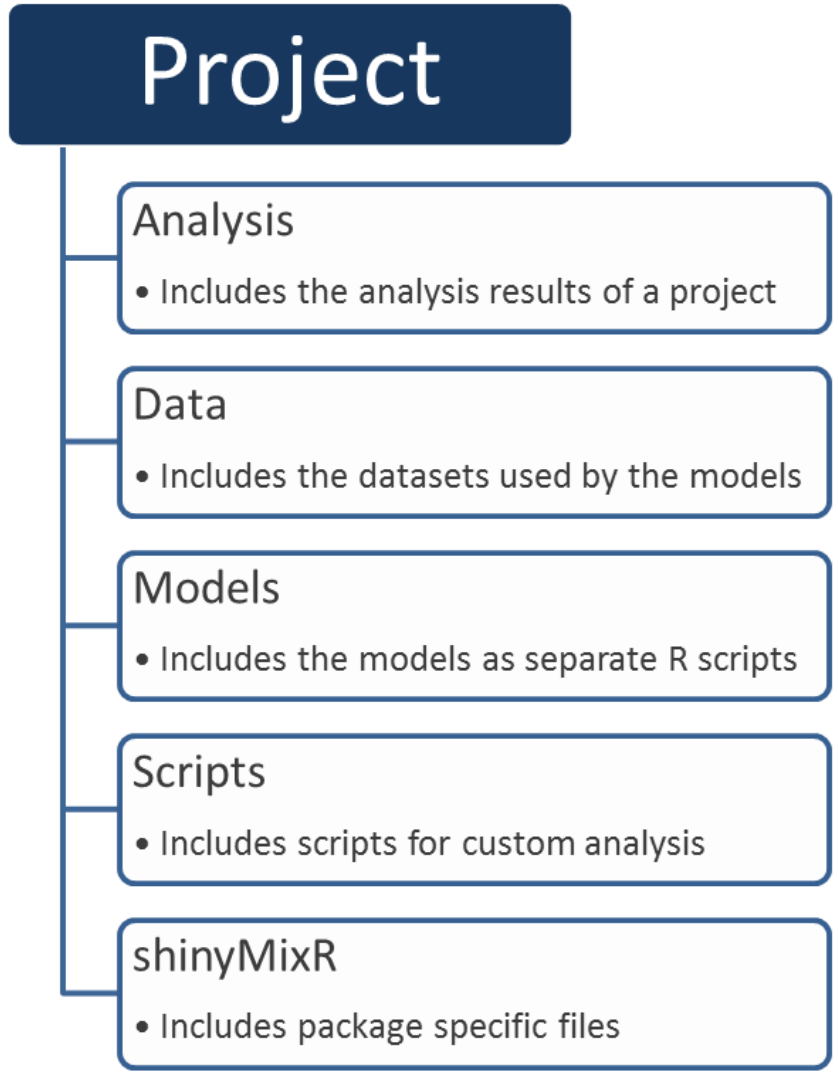
- The `shinyMixR` package is a **graphical user interface (GUI)** tool for managing pop PKPD projects with `nlmixr` as the estimation engine
- The package is intended to view, edit, run, compare, analyze and report nlmixr models
- It organizes your project – model, data, metadata, settings and results 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`

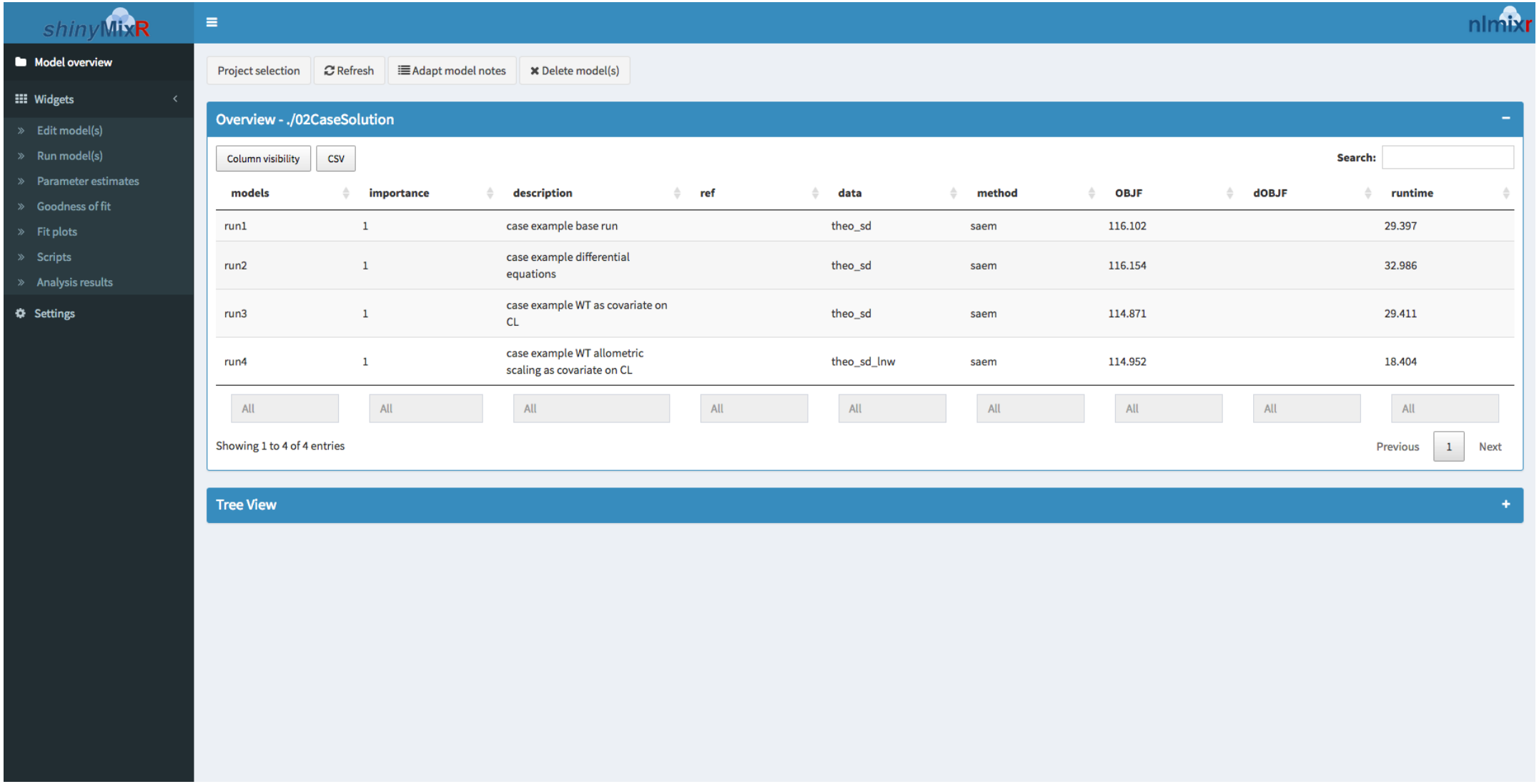
`nlmixr` arguments moved to metadata within the model code (run1.r model file) – analogous to NONMEM $PROBLEM, $DATA, $EST, etc.

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

shinyMixR – metadata and run button or run_nmx function

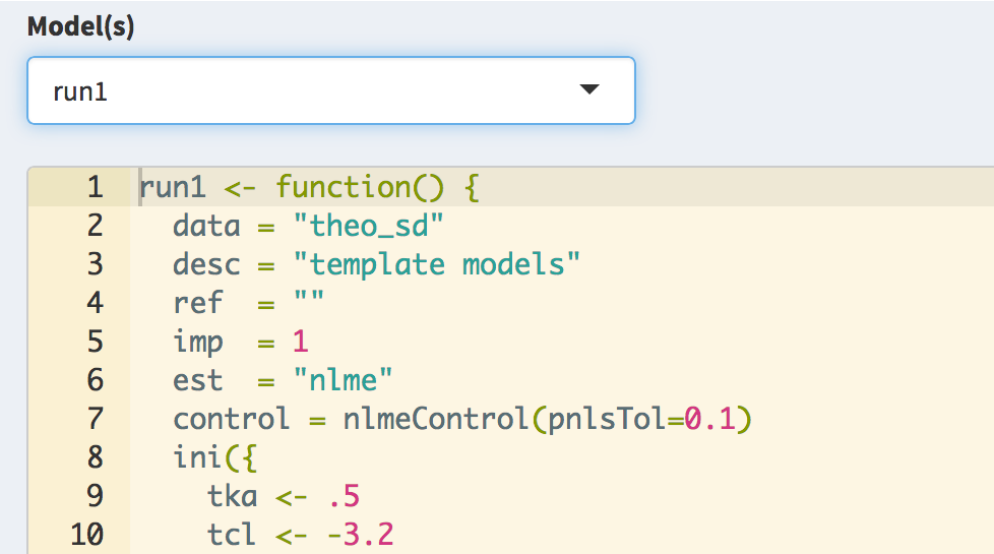
```
run1 <- function() {
  data = "theo_sd"                    # csv or rds file
  desc = "case example base run"      # model description
  ref  = ""                           # model reference
  imp  = 1                            # model importance
  est  = "saem"                       # estimation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

shinyMixR - functionalities



Overview of shinyMixR functionalities

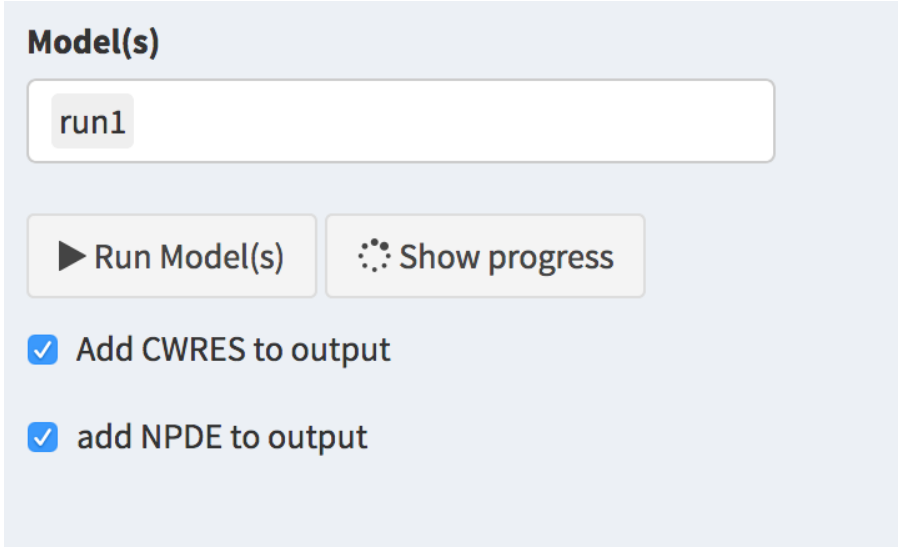
Edit Models



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



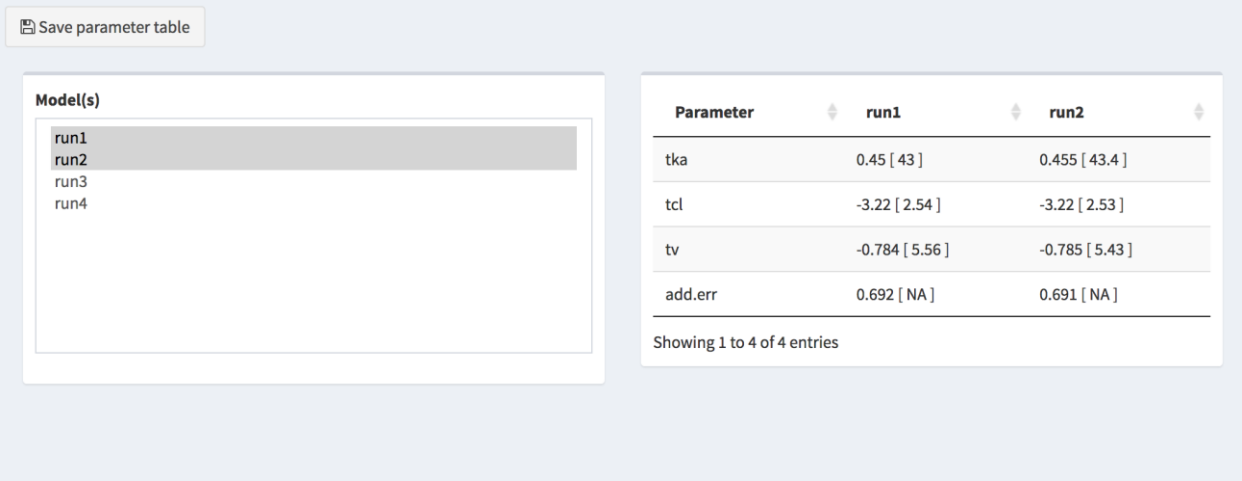
The **run model(s)** widget is used to run models. It is possible to run one or multiple models at once.

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 functionalities

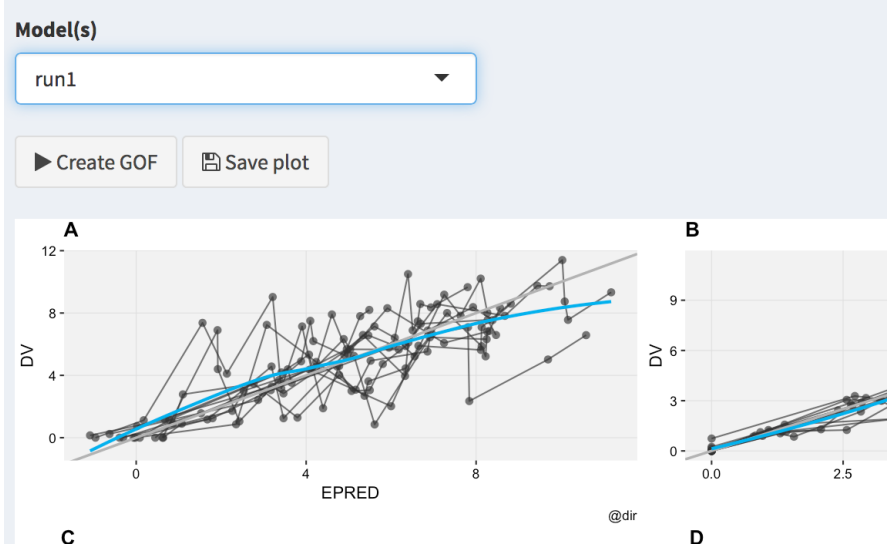
Parameter Estimates



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

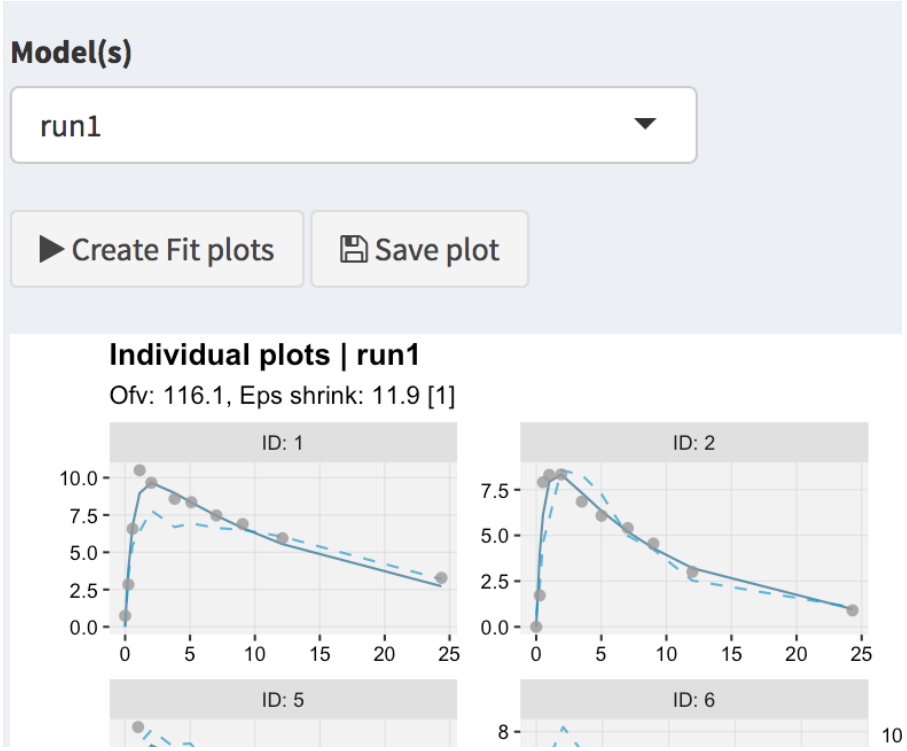


The **goodness of fit** widget is used to generate a combination of 4 goodness of fit plots combined.

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

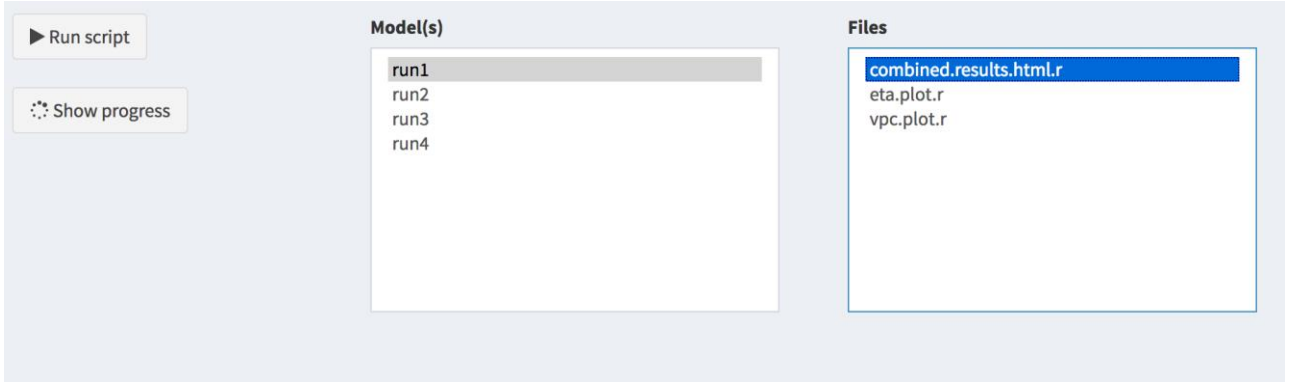
Overview of shinyMixR functionalities

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



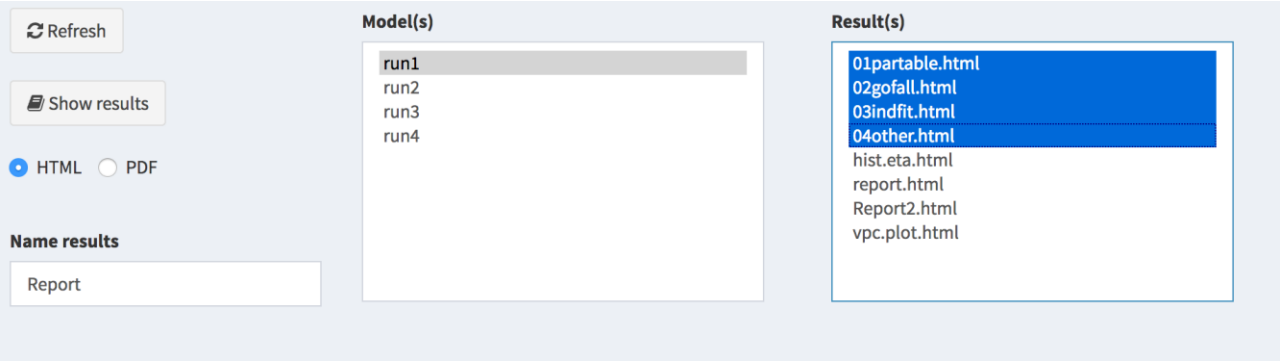
It is possible to write your own scripts (See **scripts** folder and **scripts** widget) that can be used to analyze model results.

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

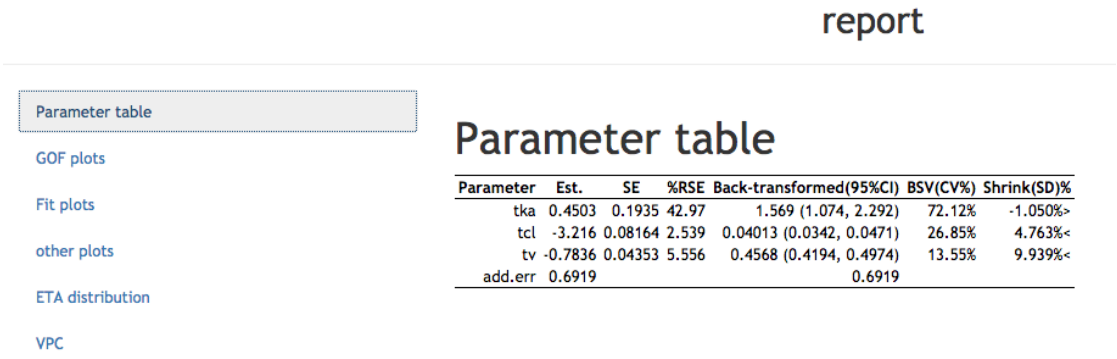
Overview of shinyMixR functionalities

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 shinyMixR

- The `shinyMixR` package is a **graphical user interface (GUI)** tool for managing pop PKPD 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

Assignment - Theophylline

The assignment and solutions were provided, and can be found in **google drive**

The assignment and related files to start the assignment are available at:
"**via_nlmixr**" or "**via_shinyMixR**"
(Open **background.pdf** and **assignment.pdf** to go trough the background and questions, respectively)

Assignment - Theophylline

Additional guidance on using shinyMixR (next slides)

1. Create New Project (general)
   - via `shinyMixR` shortcut (preferred), or
   - via `shinyMixR` command line
2. Return To Existing Project (general)

… and guidance in **background.pdf** and **cheatsheets**

Back-up ShinyMixR

Create New Project (general) – via `shinyMixR` shortcut

1. Create a folder for your project (e.g., <ProjectFolder>)
2. Click the shortcut (Windows, Mac or Linux specific)
   - ➢ *The shortcut should only be copied the first time you start using shinyMixR*
3. The shinyMixR application opens
   - ➢ *If the shortcut does not work – follow "via `shinyMixR` command line" on the next slide to start the interface*
4. Browse to your project folder (e.g., <ProjectFolder>)

- By default, the folder structure is created within the current directory, if not present. The following folders are created:
  - **analysis**: in this folder all plots and tables are saved in a structured way to make them accessible to the interface
  - **data**: data files used by the models in R data format (.rds)
  - **models**: models, available as separate R scripts according the unified user interface in nlmixr
  - **scripts**: generic analysis scripts made available in the interface
  - **shinyMixR**: folder used by the interface to store temporarily files and results files

- Start creating and running your models
  - If needed, supplied Model(s) can be copied to the models folder
  - If needed, supplied Data can be copied to the data folder

Create New Project (general) – via `shinyMixR` command line

1. Create a folder for your project (e.g., <ProjectFolder>)
2. Open R or RStudio and set the working directory to your project folder (e.g., <ProjectFolder>)
   - Use `setwd()`, or
   - via Rstudio: *Session > Set working directory > Choose directory*
3. On the command line or in a script run
   - `library(shinyMixR, quietly=TRUE)`
   - `create_proj()` (only needed once per project)

- By default, the folder structure is created within the current directory, if not present. The following folders are created:
  - **analysis, data, models**, **scripts**, **shinyMixR**

- Once there is a folder structure present, the interface can be started:

4. On the command line or in a script run
   - `library(shinyMixR, quietly=TRUE); run_shinymixr(launch.browser=TRUE)`
   - *Note*: other functions like `run_nmx()` can also be used directly on the command line or in a script to run models

- Start creating and running your models
  - If needed, supplied Model(s) can be copied to the models folder
  - If needed, supplied Data can be copied to the data folder

Return To Existing Project (general)

1. Click the **shortcut** (Windows, Mac or Linux specific)
   - *The shortcut should only be copied the first time you start using shinyMixR*
2. The shinyMixR application opens
3. Browse to your project folder (e.g., <ProjectFolder>)

OR

1. Open R or Rstudio and set the working directory to your project folder (e.g., <ProjectFolder>)
   - Use `setwd()`, or
   - via RStudio: *Session > Set working directory > Choose directory*
2. On the command line or in a script run
   - `library(shinyMixR, quietly=TRUE)`
   - `run_shinymixr(launch.browser=TRUE)` → if interface run in web browser

- The interface will open up showing all models previously stored in this directory.

shinyMixR

Cheat Sheet

github.com/richardhooijmaijers/shinyMixR

Installation

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

Installation of the package should be done using devtools:

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

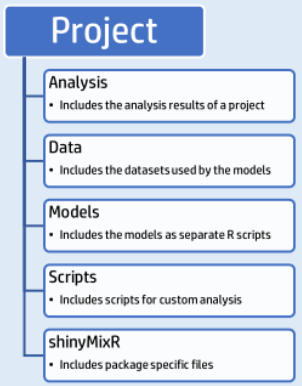
The package is based on the `shiny` and `shinydashboard` packages. Additional packages are off course `nlmixr`, and for `xpose` type plotting `xpose.nlmixr`.

Introduction

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

Folder Structure

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



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

```
create_proj()
```

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

Project Structure

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

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

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

Interactive usage

For interactive usage, the most important functions are:

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

Interface usage

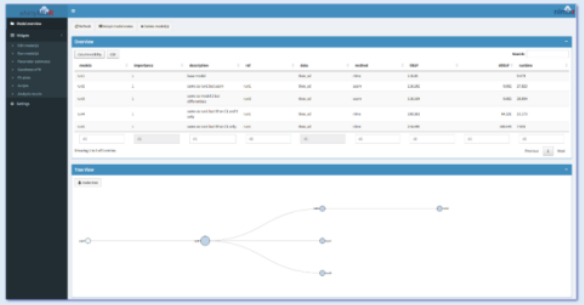
The interface can be started from the projects root folder:

```
run_shinymixr()
```

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

Usage

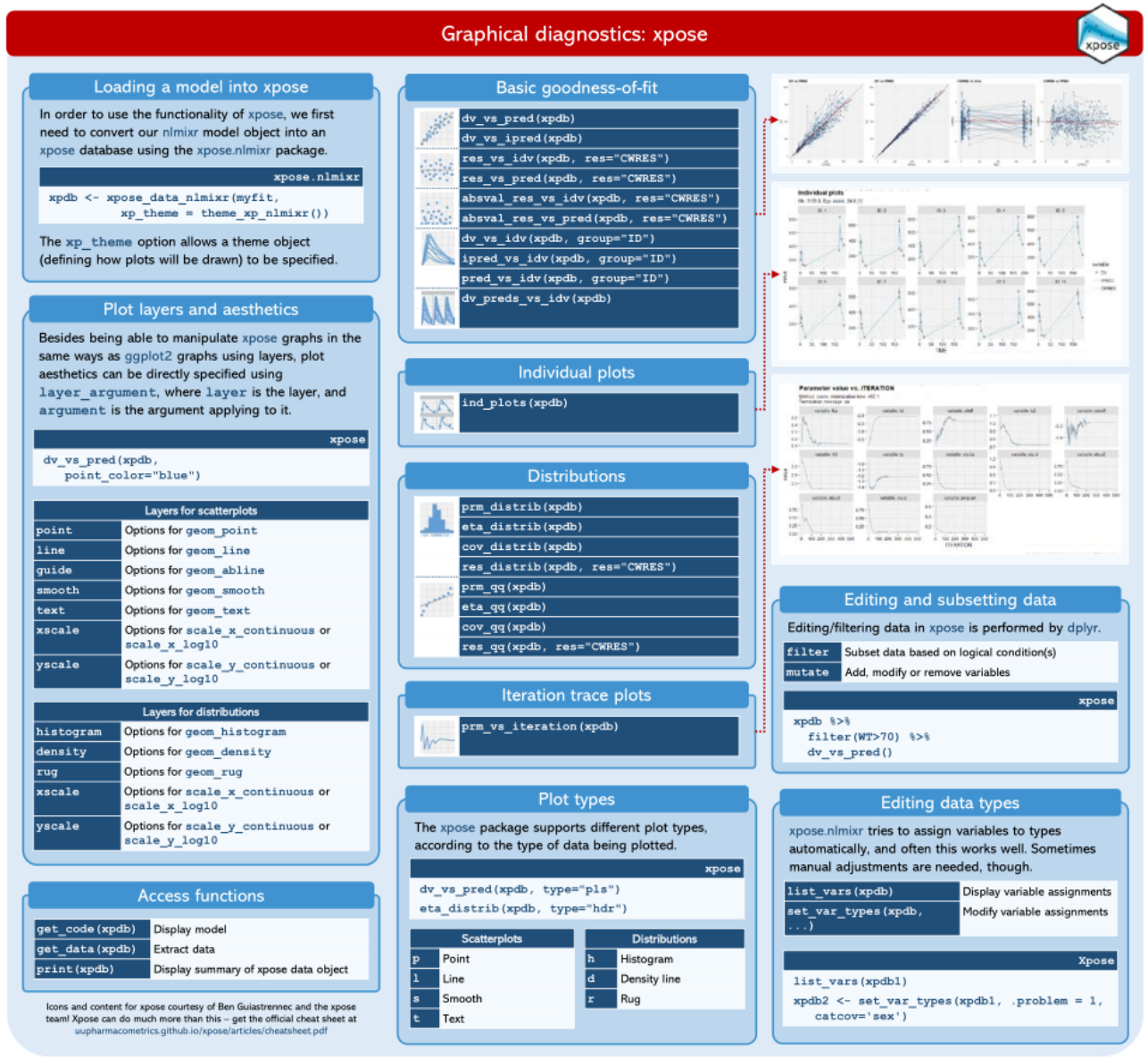
There are various widgets available on the left side of the screen to run and analyse your models.



Model overview	Starting point, contains overview of models including most important information. Possibility to add/adapt meta information. Change views, export overview. Visualize relationship between models.
Edit model	Create, save, duplicate and edit models. Possibility to use template models. Syntax highlighted editor using `shinyAce` editor.
Run model	Run one or multiple nlmixr models side-by-side. Perform model runs in separate R session(s). Keep track of progress of runs.
Parameter estimates	Create dense parameter table. Quickly observe parameter estimates and compare estimates between models. . Export to HTML or PDF output*.
Goodness of fit	Create a combination of most important goodness of fit plots. Export plots to HTML or PDF output*.
Fit plots	Create individual fit plots. Export to HTML or PDF output*.
Scripts	Run user generated scripts on selected model outputs. Possibility to run scripts included in the package or write your own scripts to create project specific output (scripts can also be used in interactive session).
Analysis results	Manage and view results created with `shinyMixR` either using the widgets, app scripts or ad-hoc (if named/saved according app conventions).
Settings	Adapt the settings of the app, like type of output and look of editor.

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

Workflow – model diagnostics



Elaborate information on how to use the `xpose.nlmixr` package is available in the cheatsheet

Installation of shinyMixR

- To get started, first install the package using:
  - devtools::install_github("richardhooijmaijers/shinyMixR")
- Be aware that the `nlmixr`, `nlmixr.xpose` and `R3port` package should be installed before installing `shinyMixR`, e.g.:
  - devtools::install_github("richardhooijmaijers/R3port")
  - devtools::install_github("nlmixrdevelopment/nlmixr")
  - devtools::install_github("nlmixrdevelopment/xpose.nlmixr")

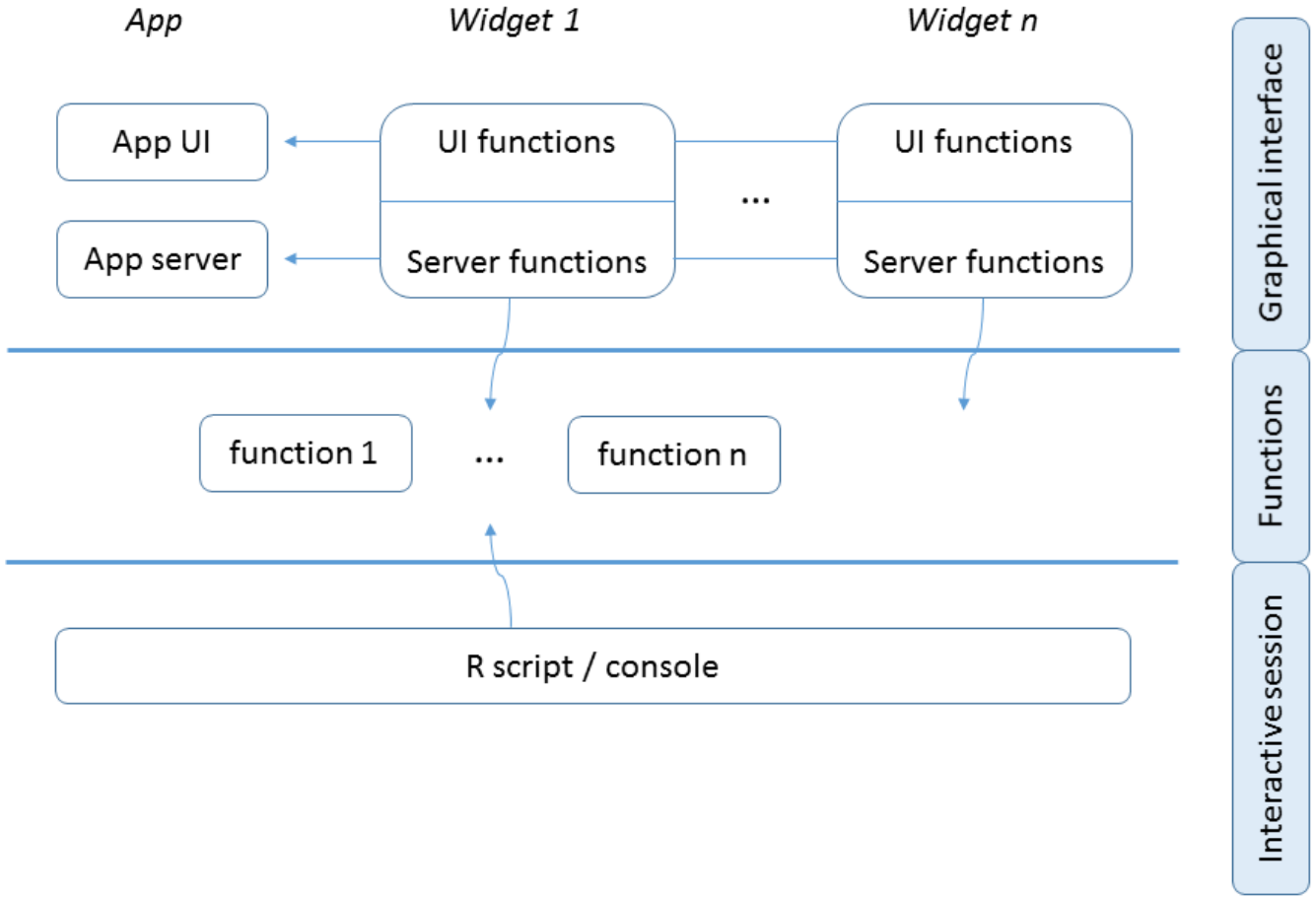
Package Information for shinyMixR

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

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

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

Package structure



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

Differences between Interactive Session and Interface

Functionality available in <u>both</u>

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

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

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

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

Workflow – via nlmixr

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

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

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

A model can be directly run in nlmixr:

1. Define model using the unified user interface.
2. Import, create and/or adapt the required data.
3. Use the nlmixr function to run the model.

Workflow – model diagnostics

```
print(fit)
plot(fit)



xpdb <- xpose_data_nlmixr(fit)

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

High level results can be printed in console and default plots can be created using `nlmixr`.

More elaborate plots can be generated using the `xpose.nlmixr` package

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

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

Workflow – via shinyMixR command line

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

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

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


# interface
run_shinymixr()
```

The model is defined in a separate file (run1.r) and includes metadata used by the package

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

The interface can be started using a single function