



## Cheat Sheet

November 2019

<https://nlmixr.github.io/>

### Getting nlmixr

What you need:

- R 3.5.0 or later
    - RxODE
    - SnakeCharmR
    - nlmixr
  - Rtools (if you use Windows)
  - Python with SymPy
- See our GitHub homepage ([nlmixr.github.io](https://nlmixr.github.io/)) for a detailed installation guide and links to nlmixr installers.

### Optional extras

xpose.nlmixr: Graphical diagnostics using xpose

shinyMixR: A GUI for building nlmixr models in shiny

### Solved systems

Linear compartmental PK models with either oral or IV dosing all have closed-form solutions similar to NONMEM ADVANS.

#### model

```
linCmt() ~ add(add.err)
```

The `linCmt()` term replaces the ODEs. nlmixr will guess the model form from the parameters specified. Currently only for nlme and SAEM.

### Residual error

Additive, proportional and combined additive and proportional error models are available.

#### model

```
cp ~ add(add.err)
cp ~ prop(prop.err)
cp ~ add(add.err) +
  prop(prop.err)
```

## Writing models

```
nlmixr <- function() {
  ini({
    tka <- log(1.5)
    tcl <- log(4)
    tv <- log(20)
    eta.ka ~ 0.5
    eta.cl ~ 0.5
    eta.v ~ 0.2
    prop.err <- 0.1
  })

  model({
    ka <- exp(tka + eta.ka)
    cl <- exp(tcl + eta.cl)
    v <- exp(tv + eta.v)

    d/dt(depot) = -ka * depot
    d/dt(cent) = ka * depot -
      cl / v * cent

    cp = cent / v
    cp ~ prop(prop.err)
  })
}
```

Fixed effects (<- or =)      Initial estimates

Random effects (~)      Residual error (<-)

Model parameters

ODEs

Concentration

Residual error

Model

Models are defined as functions, with `ini` (initial estimates) and `model` (model) blocks. Parameters are best defined on the log scale. Assignments can use <- or =. Random effects are expressed as variances using the tilde (~).

Bounds are supported for FOCEi (but not currently for nlme or SAEM), parameters can be fixed, and parameters can be labelled with #:

#### ini

```
tcl <- c(-3, 0.1, 5) # log CL (FOCEi only)
allCL <- fix(0.75) # allometric exponent
```

### Off-diagonal random effects

Parameter correlations are expressed as triangular blocks (zeroes should not be used):

#### ini

```
eta.cl + eta.v ~ c(0.1,
  0.005, 0.1)
```

### Mu-referencing

SAEM random effects and covariates must be added to the population parameters (mu-referencing). This is implemented for exponential random effects as additive on log-scale. While not strictly required for FOCEi, it improves stability. For SAEM, calculate  $\log(WT70) < -\log(WT/70)$  in the data set, and not in the model block.

#### model

```
cl <- exp(tcl + allCL*logWT70 + eta.cl)
v <- exp(tv + CovSex*SEX + eta.v)
```

## Running models

```
nlmixr(
  object,
  data,
  est = "saem",
  saemControl(print=50,
    nBurn=200, nEm=300),
  tableControl(cwres=TRUE))
```

Model

NONMEM/RxODE data

Estimation method

Control parameters

Calculate conditional weighted residuals

### Estimation method options

est = "focei", "foce", "foi", "fo"	
These methods are based on our interpretation of the NONMEM routines.	
foceiControl()	
outerOpt	Outer optimization routine c("nlminb", "bobyqa", and many others)
sigdig	Controls tolerances of estimation and ODE solving routines. Not the same as NONMEM sigdig parameter but with similar meaning (3)
maxOuterIterations	Maximum number of outer iterations; 0 provides Bayesian feedback estimates
print	Iterations printed to console (1)
...	Additional arguments (too many to mention!)
est = "saem"	
An implementation of the stochastic approximation expectation-maximization algorithm. No termination criteria, can be slow when using ODEs.	
saemControl()	
seed	Random seed (99)
nBurn	Number of iterations in the SA (burn-in) step (200)
nEm	Number of iterations in the EM step (300)
nmc	Number of Markov chains (3)
atol	Absolute convergence tolerance (1e-8)
print	Iterations to complete before printing to console (1)
...	Additional arguments

#### est = "posthoc"

Uses posthoc step of FOCEi algorithm for Bayesian feedback. Similar to using `foceiControl(maxOuterIterations=0)`.

#### tableControl()

Controls additional table outputs included in the final nlmixr model.

cwres	Boolean indicating if you need to calculate conditional weighted residuals (CWRES). On by default for FOCE(i) routines. This will also generate WRES, CPRED and CRES. Additionally this will add the FOCEi objective function value
npde	Calculate npde residuals (NPDE). This will also generate EPRED and ERES
nsim	Number of simulations used for NPDE (default 300)
ties	Boolean indicating if noise will be added to avoid ties in NPDE calculation (TRUE)
Seed	Random seed to use for npde calculation (1009)

#### Adding table items after fit

```
fit <- fit %>% addCwres()
fit <- fit %>% addNpde()
```

## Example code

### Zero-order absorption

```
model <- function() {
  ini({
    ka <- 1.2 #ka (/h)
    lcl <- -2.0 #log CL (L/hr)
    v <- -8.0 #v (L)
    ltk0 <- 0.5 #log zero-order
    #infusion duration (h)

    prop.err <- 0.15
    eta.cl ~ 0.1}) #IIV CL

  model({
    cl <- exp(lcl + eta.cl)
    D1 <- exp(ltk0)
    d/dt(depot) = -ka*depot
    d/dt(C2) = ka*depot - (cl/v)*C2
    dur(depot) = D1
    cp = C2 / v
    cp ~ prop(prop.err) }) }
```

### Lag-time

```
model <- function() {
  ini({
    ltlag <- log(0.5)
    ...
  })
  model({
    Tlag <- exp(ltlag + eta.tlag)
    d/dt(depot) = -ka * depot
    lag(depot) = Tlag
    ...
  }) }
```

### Turnover simultaneous PKPD model

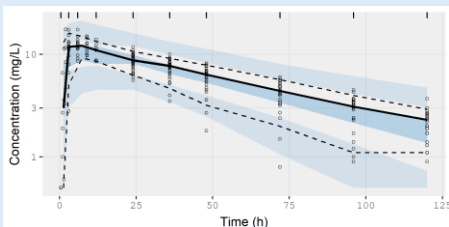
```
model <- function() {
  ini({
    tcl <- log(0.1) # log CL (L/hr)
    tv <- log(8) # log Vc (L)
    eta.cl ~ 0.1
    eps.pkprop <- 0.1
    tc50 <- log(1) #log ec50 (mg/L)
    tkout <- log(0.05) #log tkout (/h)
    e0 <- 100 #e0
    eta.c50 ~ .5
    eps.pdadd <- 100})

  model({
    cl <- exp(tcl + eta.cl)
    v <- exp(tv)
    c50 = exp(tc50 + eta.c50)
    kout = exp(tkout)
    cp = center / v
    d/dt(center) = - cl * cp
    effect(0) = e0
    kin = e0*kout
    PD = 1-cp/(c50+cp)
    d/dt(effect) = kin*PD -kout*effect
    #specify CMT or DVID (1/2) in data
    cp ~ prop(eps.pkprop) | center
    effect ~ add(eps.pdadd) | effect }) }
```

## VPCs: vpc

nlmixr uses the simulation capabilities of **RxODE** and the **vpc** package to generate VPCs directly from the fitted model object:

```
nlmixr
vpc_ui(myfit, n=500, show=list(obs_dv=TRUE),
  log_y=TRUE, log_y_min=0.5,
  xlab="Time (h)",
  ylab="Concentration (mg/L)")
```



## Most useful VPC options

<b>fit</b>	nlmixr fit object
<b>n</b>	Number of simulation iterations
<b>bins</b>	Either "density", "time", or "data", "none", or one of the approaches available in <code>classInterval()</code> such as "jenks" (default) or "pretty", or a numeric vector specifying the bin separators
<b>n_bins</b>	When using the "auto" binning method, what number of bins to use
<b>bin_mid</b>	Either "mean" for the mean of all timepoints (default) or "middle" to use the average of the bin boundaries
<b>show</b>	A list of what to show in VPC (obs_dv, obs_ci, pi, pi_as_area, pi_ci, obs_median, sim_median, sim_median_ci); see example
<b>stratify</b>	Character vector of stratification variables (max 2)
<b>smooth</b>	"Smooth" the VPC (connect bin midpoints) or show as rectangular boxes (default T)
<b>pred_corr</b>	Perform prediction-correction (default F)
<b>pi</b>	Simulated prediction interval to plot. Default is c(0.05, 0.95)
<b>ci</b>	Confidence interval to plot. Default is (0.05, 0.95)
<b>facet</b>	"wrap", "columns", or "rows"
<b>log_y</b>	Logarithmic y-axis? (default F)
<b>xlab</b>	Label for x-axis
<b>ylab</b>	Label for y-axis
<b>title</b>	Title
<b>uloq</b>	Upper limit of quantification (default NULL)
<b>lloq</b>	Lower limit of quantification (default NULL)
<b>vpc_theme</b>	Theme. Expects list of class <code>vpc_theme</code> created with function <code>vpc_theme()</code>

## Loading a model into xpose

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

```
xpose.nlmixr
xpdbs <- xpose_data_nlmixr(myfit,
  xp_theme = theme_xp_nlmixr())
```

The **xp\_theme** option allows a **ggplot2** theme object (defining how plots will be drawn) to be specified.

## Plot layers and aesthetics

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

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

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

Layers for distributions	
<b>histogram</b>	Options for <code>geom_histogram</code>
<b>density</b>	Options for <code>geom_density</code>
<b>rug</b>	Options for <code>geom_rug</code>
<b>xscale</b>	Options for <code>scale_x_continuous</code> or <code>scale_x_log10</code>
<b>yscale</b>	Options for <code>scale_y_continuous</code> or <code>scale_y_log10</code>

## Access functions

<b>get_code(xpdb)</b>	Display model
<b>get_data(xpdb)</b>	Extract data
<b>print(xpdb)</b>	Display summary of xpose data object

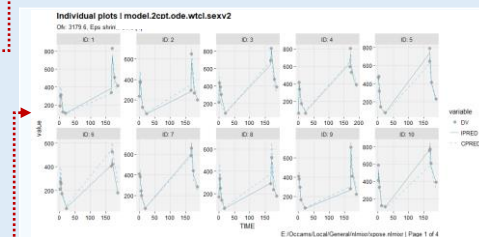
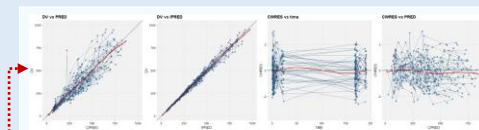
Icons and content for **xpose** courtesy of Ben Guistaennec and the **xpose** team! **Xpose** can do much more than this – get the official cheat sheet at [uopharmacometrics.github.io/xpose/reference/figures/cheatsheet.pdf](https://uopharmacometrics.github.io/xpose/reference/figures/cheatsheet.pdf)

## Graphical diagnostics: xpose



## Basic goodness-of-fit

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



## Individual plots

```
ind_plots(xpdb) #xpose version
plot(augPred(myfit)) #nlmixr version
```



## Distributions

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



## SAEM iteration trace plots

```
prm_vs_iteration(xpdb) #xpose version
traceplot(myfit) #nlmixr version
```



## Plot types

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

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

Scatterplots		Distributions	
<b>p</b>	Point	<b>h</b>	Histogram
<b>l</b>	Line	<b>d</b>	Density line
<b>s</b>	Smooth	<b>r</b>	Rug
<b>t</b>	Text		

## Editing and subsetting data

Editing/filtering data in **xpose** is performed by **dplyr**.

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

```
xpose
xpdbs %>%
  filter(WT>70) %>%
  dv_vs_pred()
```

## Editing data types

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

<b>list_vars(xpdb)</b>	Display variable assignments
<b>set_var_types(xpdb, ...)</b>	Modify variable assignments

```
xpose
list_vars(xpdb1)
xpdbs2 <- set_var_types(xpdb1, .problem = 1,
  catcov='sex')
```