nlmixr² Cheat Sheet

June 2022

https://nlmixr2.org/

Getting nlmixr

What you need:

- R-4+; R-4.2.0 preferred
- Rtools (if you use Windows)
- For installation, run:

```
install.packages(
"nlmixr2",
dependencies=TRUE)
```

See our homepage (nlmixr2.org) and blog (blog.nlmixr2.org) for more information.

Optional extras

xpose.nlmixr2 and ggPMX: Graphical diagnostics

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

```
model <- 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 =)

Random effects (~)

Residual error (<-)

Initial estimates

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, 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 improces stability. For SAEM, calculate logWT70<-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(
  model,
  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 sigidig 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

Solved system

```
model <- function() {
  ini({
    lcl  <- log(0.135) #log Cl (L/h)
    lv   <- log(8)     #log V (L)
    prop.err <- 0.15   #RUV (SD/mean)
    eta.cl ~ 0.1
    eta.v  ~ 0.1 })
  model({
    cl <- exp(lcl + eta.cl)
    v  <- exp(lv + eta.v)
    linCmt() ~ prop(prop.err)})}
```

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 D1 (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)})}
```

Turnover simultaneous PKPD model

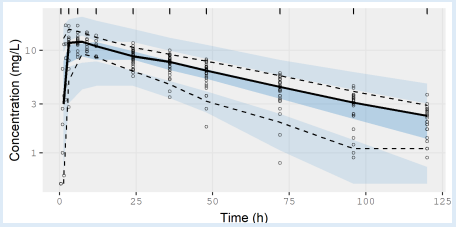
```
model <- function() {
  ini({
    tcl  <- log(0.1) # log CL (L/hr)
    tv   <- log(8)   # log Vc (L)
    eta.cl ~ 0.1
    eps.pkprop <- 0.1
    tc50  <- log(1)    #log ec50 (mg/L)
    tkout <- log(0.05) #log tkout (/h)
    e0    <- 100       #e0
    eta.c50  ~ .5
    eps.pdadd <- 100})
  model({
    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 rxode2 and the vpc package to generate VPCs directly from the fitted model object:

nlmixr

```
vpcPlot(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

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

Graphical diagnostics: xpose

Loading a model into xpose

In order to use the functionality of xpose, we first need to convert our nlmixr model object into an xpose database using the xpose.nlmixr2 package.

xpose.nlmixr

```
xpdb <- 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	
point	Options for geom_point
line	Options for geom_line
guide	Options for geom_abline
smooth	Options for geom_smooth
text	Options for geom_text
xscale	Options for scale_x_continuous or scale_x_log10
yscale	Options for scale_y_continuous or scale_y_log10

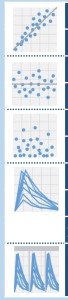
Layers for distributions	
histogram	Options for geom_histogram
density	Options for geom_density
rug	Options for geom_rug
xscale	Options for scale_x_continuous or scale_x_log10
yscale	Options for scale_y_continuous or scale_y_log10

Access functions

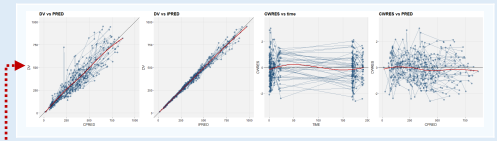
get_code(xpdb)	Display model
get_data(xpdb)	Extract data
print(xpdb)	Display summary of xpose data object

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

Basic goodness-of-fit

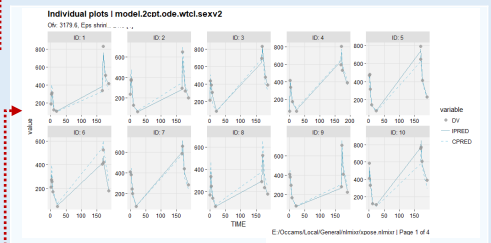


```
dv_vs_pred(xpdb)
dv_vs_ipred(xpdb)
res_vs_idv(xpdb, res="CWRES")
res_vs_pred(xpdb, res="CWRES")
absval_res_vs_idv(xpdb, res="CWRES")
absval_res_vs_pred(xpdb, res="CWRES")
dv_vs_idv(xpdb, group="ID")
ipred_vs_idv(xpdb, group="ID")
pred_vs_idv(xpdb, group="ID")
dv_preds_vs_idv(xpdb)
```



Individual plots

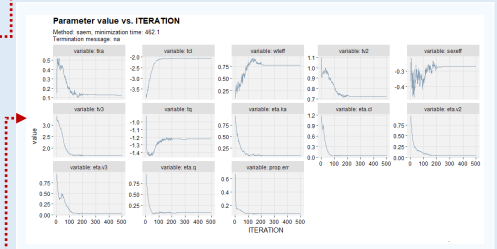
```
ind_plots(xpdb)        #xpose version
plot(augPred(myfit)) #nlmixr version
```



Distributions

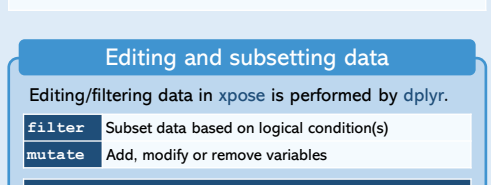
```
prm_distrib(xpdb)
eta_distrib(xpdb)
cov_distrib(xpdb)
res_distrib(xpdb, res="CWRES")
```

```
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	
p	Point
l	Line
s	Smooth
t	Text

Distributions	
h	Histogram
d	Density line
r	Rug

Editing and subsetting data

Editing/filtering data in xpose is performed by dplyr.

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

xpose

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

Editing data types

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

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

xpose

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