RxODE and nlmixr: open-source packages for pharmacometric modelling in R

Pharmacometrics Network Benelux Presentation

Rik Schoemaker, PhD

Groningen, 29 March 2018

The nlmixr development team:

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

nlmixr

RxODE is pharmacometric simulation software as an open-source R package

- Written by Wenping Wang and Matt Fidler, available on CRAN[1] and GitHub[2], and described in a tutorial in CPT:PSP[3]
- Simulation of ODEs was already possible in R (using deSolve), but was slow and virtually impossible to code with flexible dosing history
- RxODE has rapid execution due to compilation in C
- RxODE allows fully flexible dosing history
- Stable and mature software for Windows, OS X, Linux
- Requires external compilers (provided by Rtools on Windows)

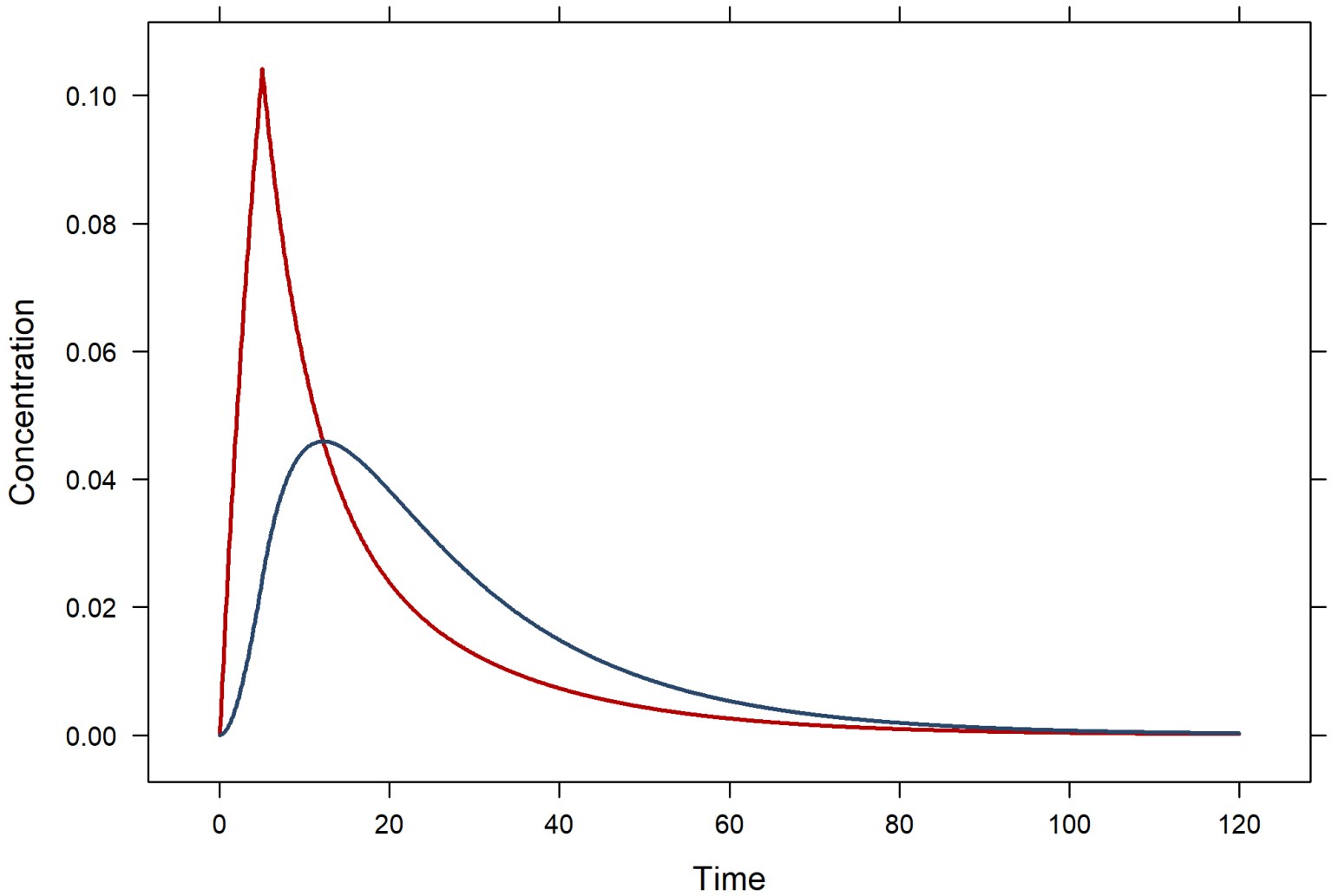
- New developments (alpha stage): parallelisation to increase speed even further

[1] CRAN: https://cran.r-project.org/web/packages/RxODE/index.html
[2] GitHub: https://github.com/nlmixrdevelopment/RxODE
[3] Wang W et al. CPT:PSP (2016) 5, 3–10.

Basic example

```
library(RxODE)

ode1 <- "
 K12 = CL2/V;
 K21 = CL2/V2;
 d/dt(centr)  = K21*periph-K12*centr-(VMAX*centr/V)/(KM+centr/V)-CL*centr/V;
 d/dt(periph) =-K21*periph+K12*centr;
 C1=centr/V;
 C2=periph/V2;
"
mod1 <- RxODE(model = ode1, modName = 'mod1')

ev <- eventTable()
ev$add.dosing(
    dose = 10,
    nbr.doses = 1,
    dosing.to = 1,
    rate = 2,
    start.time = 0
  )
ev$add.sampling(seq(0,120,0.1))
Params <- c(VMAX=2000, KM=700, CL=4, CL2=3, V=70, V2=30)
Res <- as.data.frame(mod1$run(Params, ev))

xyplot(C1+C2~time,data=Res,type='l',ylab="Concentration",xlab="Time")
```

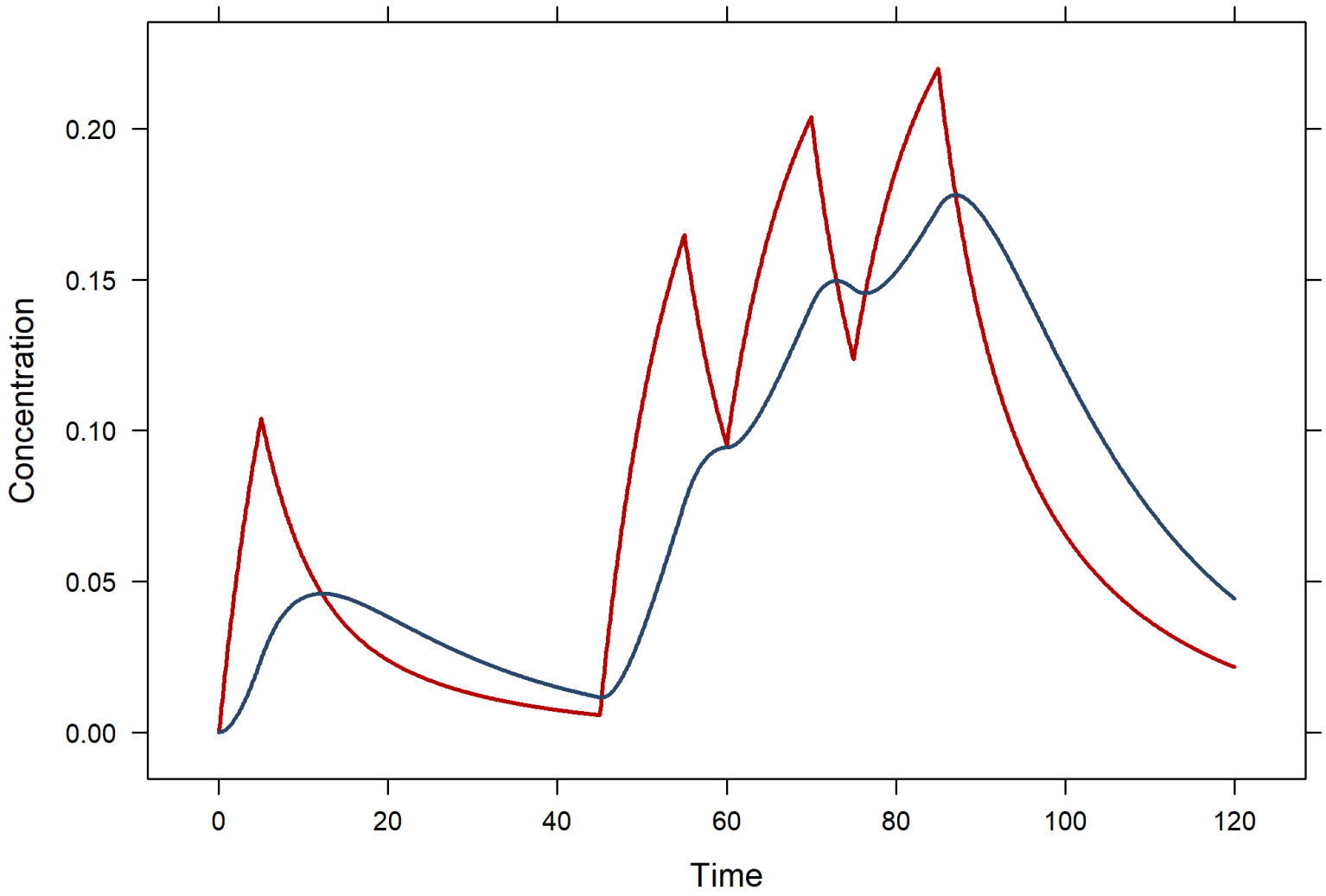
Single dose


0.10
0.08
0.06
0.04
0.02
0.00
Concentration
0
20
40
60
80
100
120
Time

Adding extra doses (expand the event table)

```
ev$add.dosing(
  dose = 20,
  nbr.doses = 3,
  dosing.to = 1,
  dosing.interval=15,
  rate = 2,
  start.time = 45
)
res<-as.data.frame(mod1$run(Params, ev))

xyplot(C1+C2~time,data=res,type='l',ylab="Concentration",xlab="Time")
```

Multiple dose



Generate a whole population of full individual IPRED curves starting from a NONMEM dataset (study this at home ☺)

```
library(data.table)
NMdat <- fread(file.path(datapath, "run100.csv"))
EBEs <- unique(NMdat[, .(ID, STD, VMAX, KM, CL, CL2, V, V2)])
subs <- unique(NMdat$ID)
N <- length(subs)
s = lapply(1:N, function(i) {
  params <- EBEs[ID == subs[i]]
  ev <- eventTable()
  DOSi <- NMdat[ID == subs[i] & AMT > 0]
  DOSi[, nTime := shift(TIME, 1L, type = 'lead')]
  timei <- NMdat$TIME[NMdat$ID == subs[i]]
  for (j in 1:length(DOSi$AMT)) {
    dos <- DOSi[j, ]
    ev$add.dosing(dose = dos$AMT,nbr.doses = 1,dosing.to = 1,
                  rate = dos$RATE,start.time = dos$TIME)
    #generate prediction time points (many points at dose and fewer at later times)
    if (is.na(dos$nTime)) {dos$nTime <- dos$TIME + 720}
    timei <-c(timei, dos$TIME + exp(seq(log(+0.01),log(dos$nTime - dos$TIME - 0.01),
        (log(dos$nTime - dos$TIME - 0.01) - log(+0.01)) / 100)))
  }
  times <- sort(unique(timei))
  ev$add.sampling(times)
  x <- as.data.table(mod1$run(params, ev))
  x[, ID := subs[i]]
  setnames(x, "C1", "IPRED")
})
df.sim = as.data.table(do.call("rbind", s))
```

You need to simulate before you can estimate

- With simulation covered, you can start to think about estimation
- Combine the simulation core with estimation routines and you get:

nlmixr!

nlmixr is an open-source R package

- Written by Wenping Wang and Matt Fidler, and available on GitHub and CRAN[1,2]:
  - builds on RxODE[3]
  - combined with nlme and SAEM estimation routines, provides an R package for parameter estimation in nonlinear mixed effect models
  - much, more to come (e.g. adaptive Gaussian quadrature for non-continuous data, and with FOCE-I under development)
- nlmixr is completely free and open, and does not depend on any other commercial tool such as NONMEM or Monolix
- nlmixr provides an efficient and versatile way to specify pharmacometric models (both closed-form and ODEs) and dosing scenarios, with rapid execution due to compilation in C

[1] https://github.com/nlmixrdevelopment/nlmixr
[2] https://cran.r-project.org/web/packages/RxODE/index.html
[3] Wang W et al. CPT:PSP (2016) 5, 3–10.

nlmixr is an open-source R package

- Models are defined using a unified user interface (UUI): common input and output structure for the various estimation algorithms
- xpose.nlmixr[1] written by Justin Wilkins provides linkage to the new Xpose package[2], written by Ben Guiastrennec, feeding the uniform output into a highly flexible diagnostics package
- The shinyMixR[3] project management tool written by Richard Hooijmaijers and Teun Post provides an interface to nlmixr from both the R command line and a user-friendly browser-based Shiny dashboard application
- nlmixr requires access to compilers (e.g. using Rtools) and Python: both a full-package windows installer is available, and instructions on managing your own installation
- Documentation is available in the form of a bookdown (nlmixr.github.io) written and curated by Teun Post
- Runs on Linux, Windows, and OS X

[1] https://github.com/nlmixrdevelopment/xpose.nlmixr
[2] https://CRAN.R-project.org/package=xpose
[3] https://github.com/RichardHooijmaijers/shinyMixR

The unified user interface

- Models are defined using a function containing an initialisation block (`ini`) and a model definition block (`model`)

```
mod1 <-  function() {
  ini({

  })
  model({

  })
}
```

The unified user interface

- The `ini` block defines the parameters
  - Thetas defined using assign operators (**<-** or **=**)
  - Residual error defined using assign operators (**<-** or **=**)
  - Etas defined using a model formula (**~**)
- Parameter names, starting values, labels (using #), bounds

```
mod1 <-  function() {
  ini({
    lCl <- 1.6      #log Cl (L/hr)
    lVc = log(90)   #log V (L)
    lKa <- 1        #log Ka (1/hr)
    prop.err <- c(0, 0.2, 1)
    eta.Ka ~ 0.1    #IIV Ka
    eta.Cl + eta.Vc ~ c(0.1,
                        0.005, 0.1)
  })
  model({

  })
}
```

The unified user interface

- The `model` block defines
  - the relationship between thetas and etas
  - the model structure using either ODEs or closed-form solutions
  - the residual error structure and where it is applied

```
mod1 <- function(){
  ini({

  })
  model({
    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.err)
  })
}
```

Parameterisation and mu-referencing

- For SAEM, parameters must be defined using 'mu-referencing' and this implies estimating log-parameters with the IIV added on the log-scale
- For nlme, mu-referencing is not strictly required, but is shown to provide superior estimation results

```
Data$logWT70 <- log(Data$WT/70)

mod1 <- function(){
  ini({
    ## For SAEM parameters must be defined using 'mu-referencing'
    ## For nlme mu-referencing is not strictly required but is shown to provide
    ##  superior estimation results
    lCl <- 1.6      #log Cl (L/hr)
    AllomCl <- 0.75 #log Cl (L/hr)
    ## .....
    eta.Cl ~ 0.1    # IIV Cl
    ## .....
  })
  model({
    ## Parameters are defined in terms of the initial estimates
    Cl <- exp(lCl + eta.Cl)
    ## or for implementing covariate effects:
    Cl <- exp(lCl + eta.Cl + logWT70*AllomCl)
    ## Data transformations should be done outside the model definition
    ## .....
})
}
```

Full example with proportional and additive error

```
mod1 <- function(){
  ini({
    ## Initial conditions for population parameters (sometimes
    ## called theta parameters) are defined by either `<-` or '='
    lCl <- 1.6      #log Cl (L/hr)
    ## Note that simple expressions that evaluate to a number are
    ## OK for defining initial conditions (like in R)
    lVc = log(90)   #log V (L)
    ## Also a comment on a parameter is captured as a parameter label
    lKA <- 0.1      #log Ka (1/hr)
    ## Bounds may be specified by c(lower, est, upper), like NONMEM:
    ## Residuals errors are assumed to be population parameters
    prop.err <- c(0, 0.2, 1)
    add.err  <- c(0,0.01)
    ## Initial estimate for ka IIV variance
    ## Labels work for single parameters
    eta.Cl ~ 0.1    # IIV Cl
    ## For correlated parameters, you specify the names of each
    ## correlated parameter separated by a addition operator `+`
    ## and the left handed side specifies the lower triangular
    ## matrix initial of the covariance matrix.
    eta.Vc + eta.KA ~ c(0.1,
                        0.005, 0.1)
    ## Note that labels are not defined for correlated parameters.
  })
  model({
    ## Parameters are defined in terms of the initial estimates
    Cl <- exp(lCl + eta.Cl)
    Vc <- exp(lVc + eta.Vc)
    KA <- exp(lKA + eta.KA)
    ## Next, the differential equations are defined
    kel <- Cl / Vc;
    d/dt(depot)  = -KA*depot;
    d/dt(centr)  =  KA*depot-kel*centr;
    ## And the concentration is then calculated
    cp = centr / Vc;
    ## Last, nlmixr is told that the plasma concentration follows
    ## a combined proportional/additive error
    cp ~ prop(prop.err)+add(add.err)
  })
}
```

And using a closed-form solution

```
mod1 <- function(){
  ini({
    lCl <- 1.6       #log Cl (L/hr)
    lVc <- 4.5       #log V (L)
    lKA <- 0.1       #log Ka (1/hr)
    prop.err <- c(0, 0.2, 1)
    eta.Cl ~ 0.1     # IIV Cl
    eta.Vc + eta.KA ~ c(0.1,
                        0.005, 0.1)
  })
  model({
    Cl <- exp(lCl + eta.Cl)
    Vc <- exp(lVc + eta.Vc)
    KA <- exp(lKA + eta.KA)
    ## Instead of specifying the ODEs, you can use
    ## the linCmt() function to use closed-form solutions.
    ## This function determines the type of PK solved system
    ## to use by the parameters that are defined.
    ## In this case it knows that this is a one-compartment model
    ## with first-order absorption.
    linCmt() ~ prop(prop.err)
  })
}
```

The nlmixr dataset

- Datasets need to comply with RxODE requirements
- EVID is more complex
  - 101 for bolus dose in compartment 1, 10101 for infusion in compartment 1
- No MDV item so no on-the-fly removal of unwanted records
- Infusions need two records: one to start infusion and one to stop infusion (at time of infusion stop with a negative rate)
- No SS dosing so steady state needs to be coded using multiple preceding doses
- NONMEM datasets can be converted using a special function based on code by Yuan Xiong:

```
dat <- nmDataConvert(dat);
```

Running nlmixr

- nlmixr is run using the following structure:

```
fit <- nlmixr(model.function,
              rxode.dataset,
              est = "est",
              control = estControl(options))
```

- Currently nlme and SAEM are implemented
  - Example for nlme:

```
fit <- nlmixr(mod1,
              dat,
              est = "nlme",
              control = nlmeControl(pnlsTol = .05))
```

  - Example for SAEM:

```
fit <- nlmixr(mod1,
              dat,
              est = "saem",
              control = saemControl(
                n.burn = 200,
                n.em = 300,
                print = 50
              ))
```

The shinyMixR interface can manage your runs

File Edit Code View Plots Session Build Debug Profile Tools Help

script_example.r

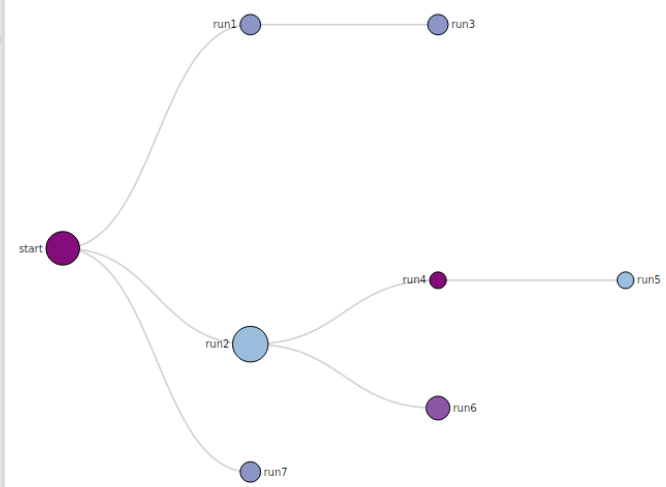
```r
# Example of workflow directly within R
library(shinyMixR)

# Create new project
create_proj()

# Obtain project information
proj <- get_proj()

# Run nmlmixr models (async in separate Rsession)
run_nmx("run1",proj)

# Use results created by the package
res <- readRDS("shinyMixR/run1.res.rds")
ggplot(res,aes(DV,PRED)) + geom_point(alpha=.6)  + geom_abline(intercept=0,slope=1,colour="darkblue",linetype=2)

# Several other functions written for the interface are available for use in R environment
oview()     # Create data frame with overview of the models within the project
makeTree()  # Create interactive tree view of the models within project
```

Environment | History

Global Environment

Data	
res	132 obs. of 15 variables
Functions	
AdaptModel	function (projlst, inp, session)
AdaptOverview	function (projlst, inp, session, type = "modal")
makeTree	function ()
oview	function ()
oviewUI	function (projlst)

Files | Plots | Packages | Help | Viewer



Console ~/Documents/nlmixr/171230/shinyMixR/V0.1.1/testproject/

The shinyMixR interface can be run from R…

```
# Example of workflow directly within R
library(shinyMixR)

# Create new project
create_proj()

# Obtain project information
proj <- get_proj()

# Run nmlmixr models (async in separate Rsession)
run_nmx("run1",proj)

# Use results created by the package
res <- readRDS("shinyMixR/run1.res.rds")
ggplot(res,aes(DV,PRED)) + geom_point(alpha=.6)  + geom_abline(intercept=0,slope=1,colour="darkblue",linetype=2)

# Several other functions written for the interface are available for use in R environment
oview()     # Create data frame with overview of the models within the project
makeTree()  # Create interactive tree view of the models within project
```

…or by launching a browser session

ShinyMixR

nlmixr

- Model overview
- Widgets
  - Edit model(s)
  - Run model(s)
  - Parameter estimates
  - Goodness of fit
  - Fit plots
  - Analysis results
- Settings

Refresh | Adapt model notes

Overview

Column visibility | CSV | Search:

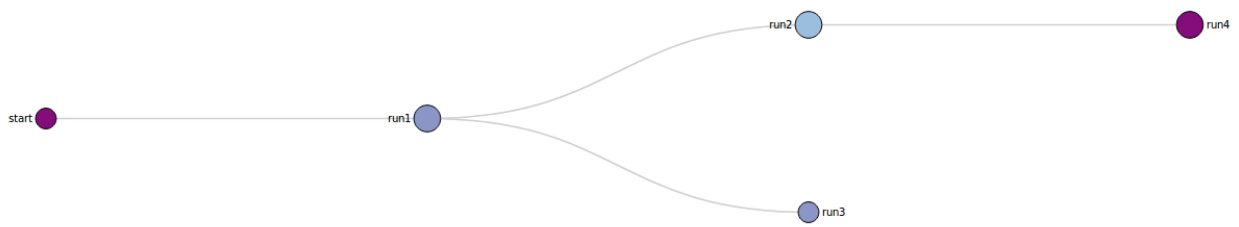
models	importance	description	ref	data	method	OBJF	dOBJF	runtime
run1	2	start model		theo_sd	saem	116.137		31.827
run2	3	test one	run1	theo_md	saem	349.627	233.49	79.909
run3	2	test two	run1	theo_sd	saem	116.137	0	24.843
run4	0	test three	run2		saem			
All	All	All	All	All	All	All	All	All

Showing 1 to 4 of 4 entries

Previous | 1 | Next

Tree View

make tree



Where models can be edited and run…

ShinyMixR

Model overview
Widgets
Edit model(s)
Run model(s)
Parameter estimates
Goodness of fit
Fit plots
Analysis results
Settings

Model(s)

run1

```
run1 <- function(){
ini({
    tka <- 0.5
    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)
    d/dt(depot) = -ka * depot
    d/dt(center) = ka * depot - cl/v * center
    cp = center/v
    cp ~ add(add.err)
})
meta <- list(data="theo_sd",desc="start model",ref="",est="saem",imp=2,control=list())
}
```

Save Model | Duplicate Model | New Model

New model

Name new model

run5.r

template

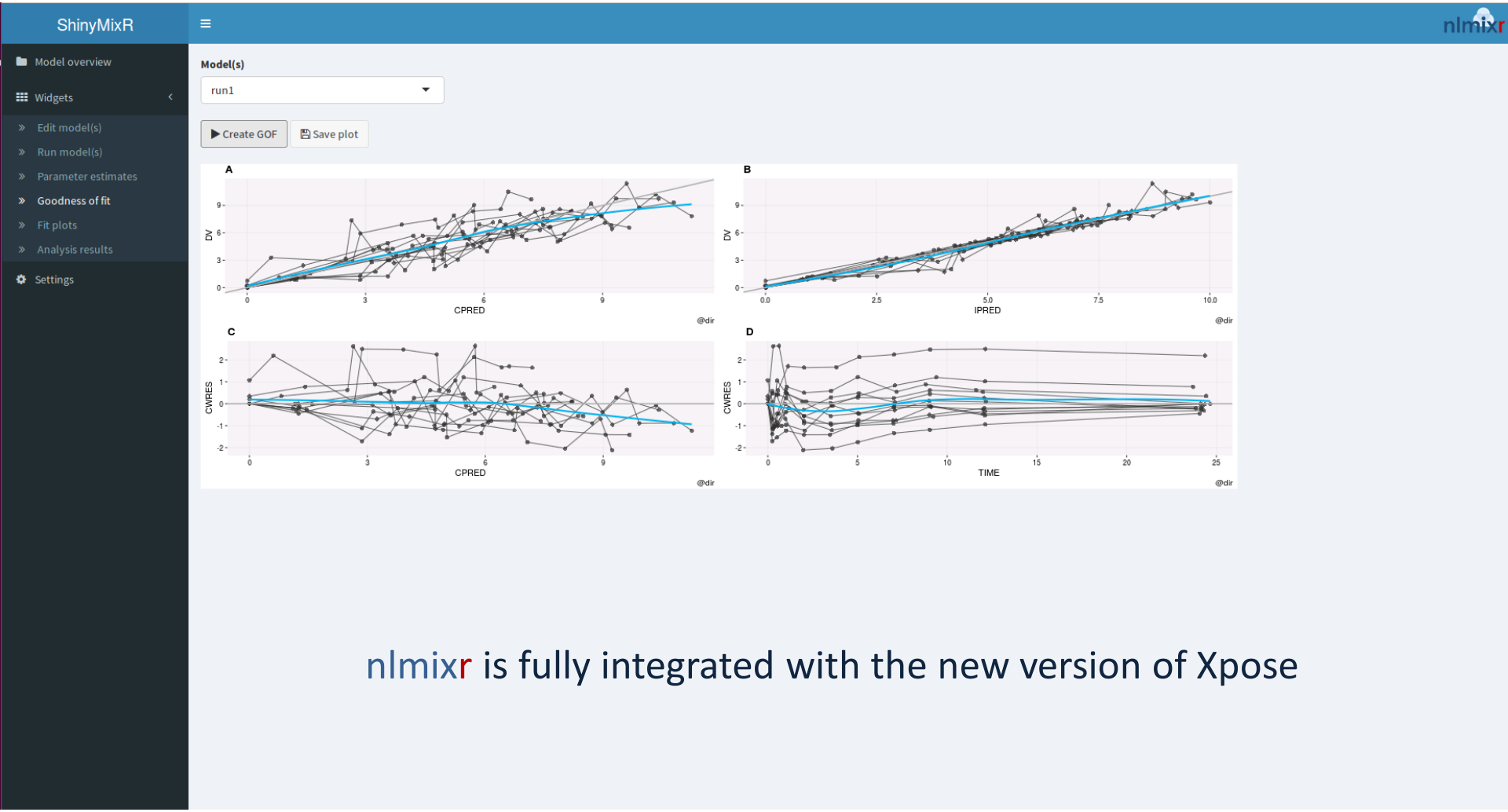
pk.1cmt.closed

▶ Go

Close

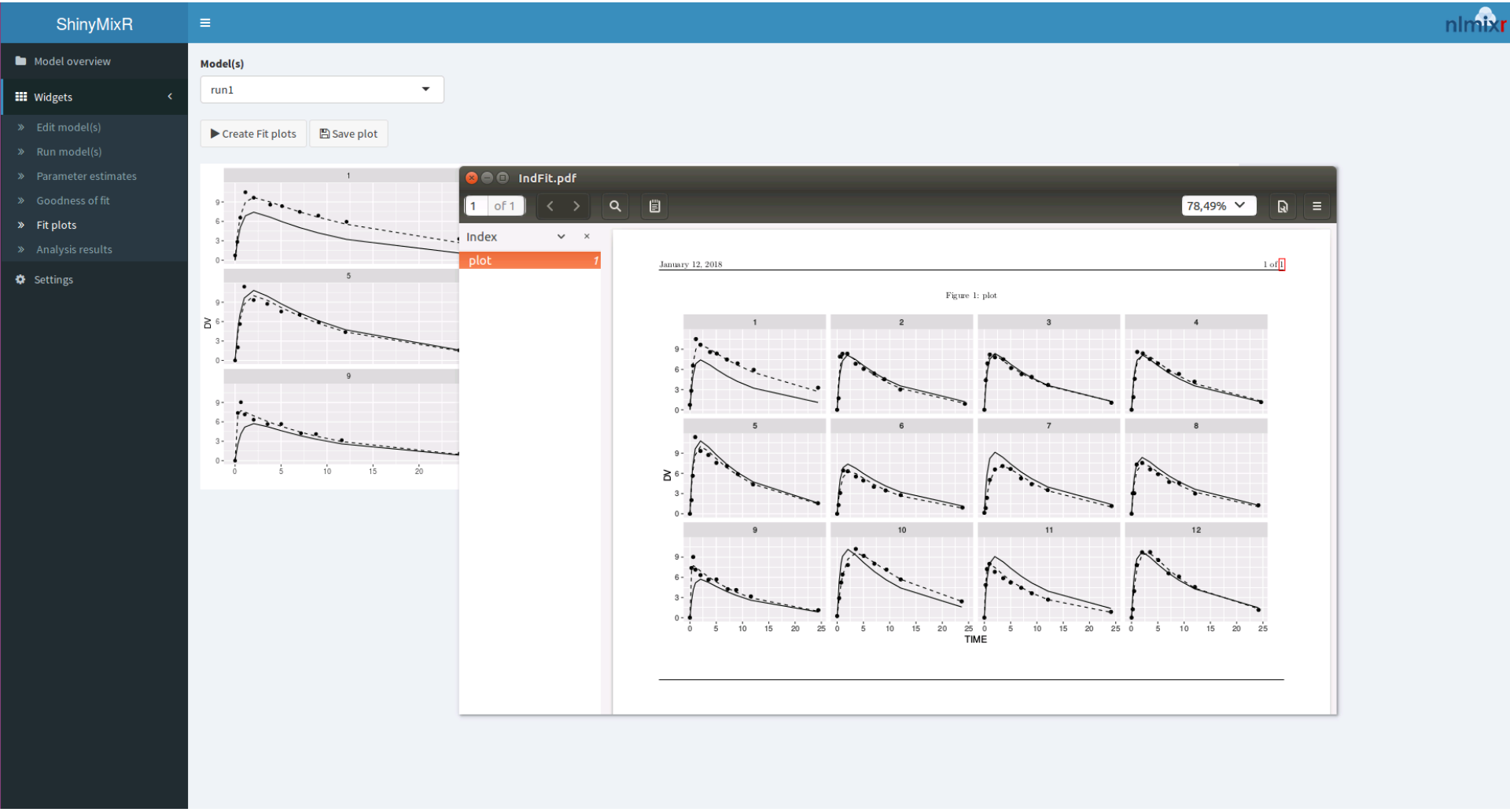
nlmixr

…and output like goodness of fit plots can be created using the new Xpose functionality, or using custom scripts…



nlmixr is fully integrated with the new version of Xpose

…and individual plots as well



Results can be exported to pdf or html

ShinyMixR

nlmixr

Model overview

Widgets

Edit model(s)

Run model(s)

Parameter estimates

Goodness of fit

Fit plots

Analysis results

Settings

Show results

PDF HTML

Model(s)

./analysis/run1
./analysis/run2
./analysis/run3

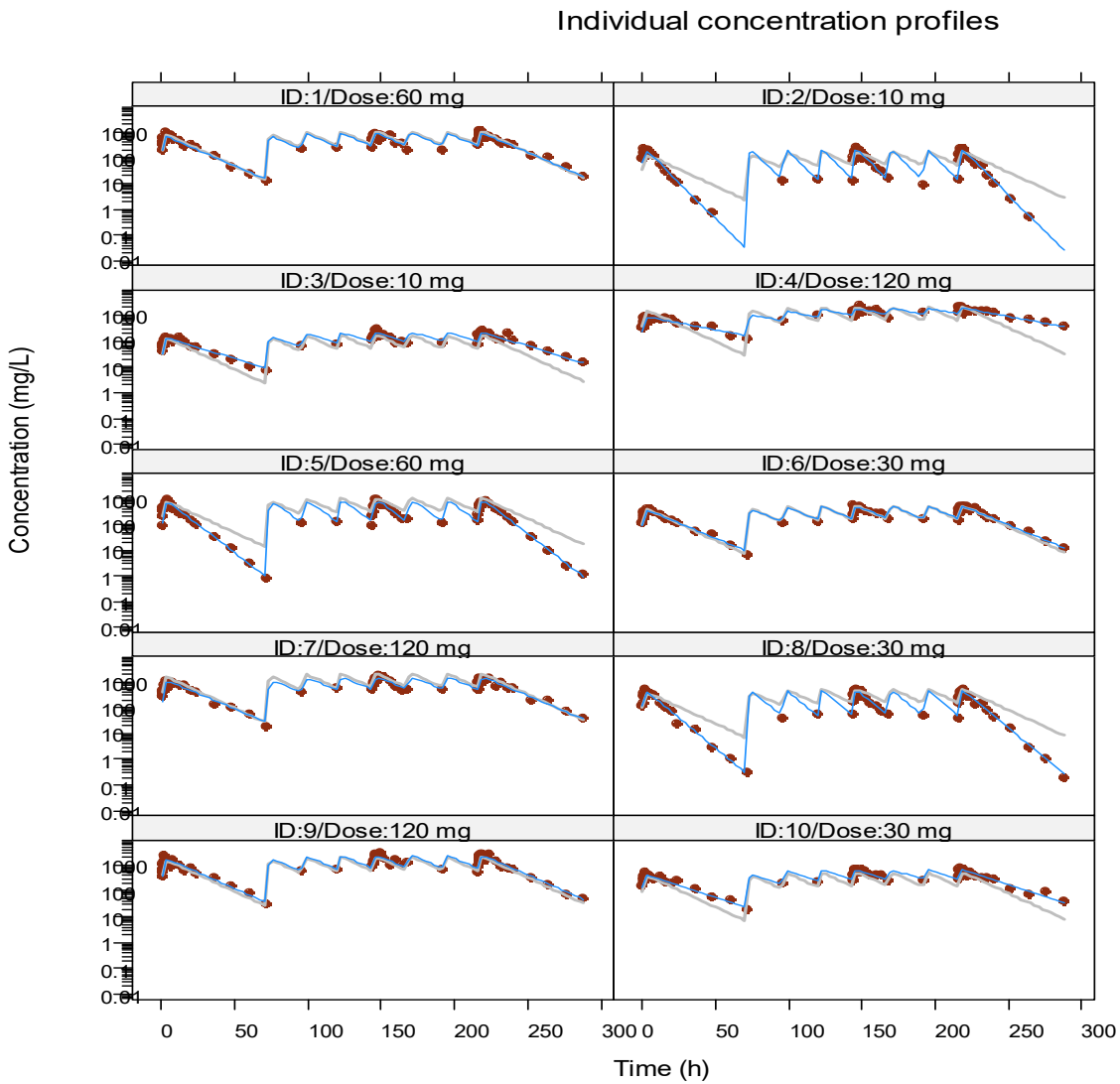
Result(s)

GOF.pdf
GOFtst.pdf
IndFit.pdf
ParTable.pdf

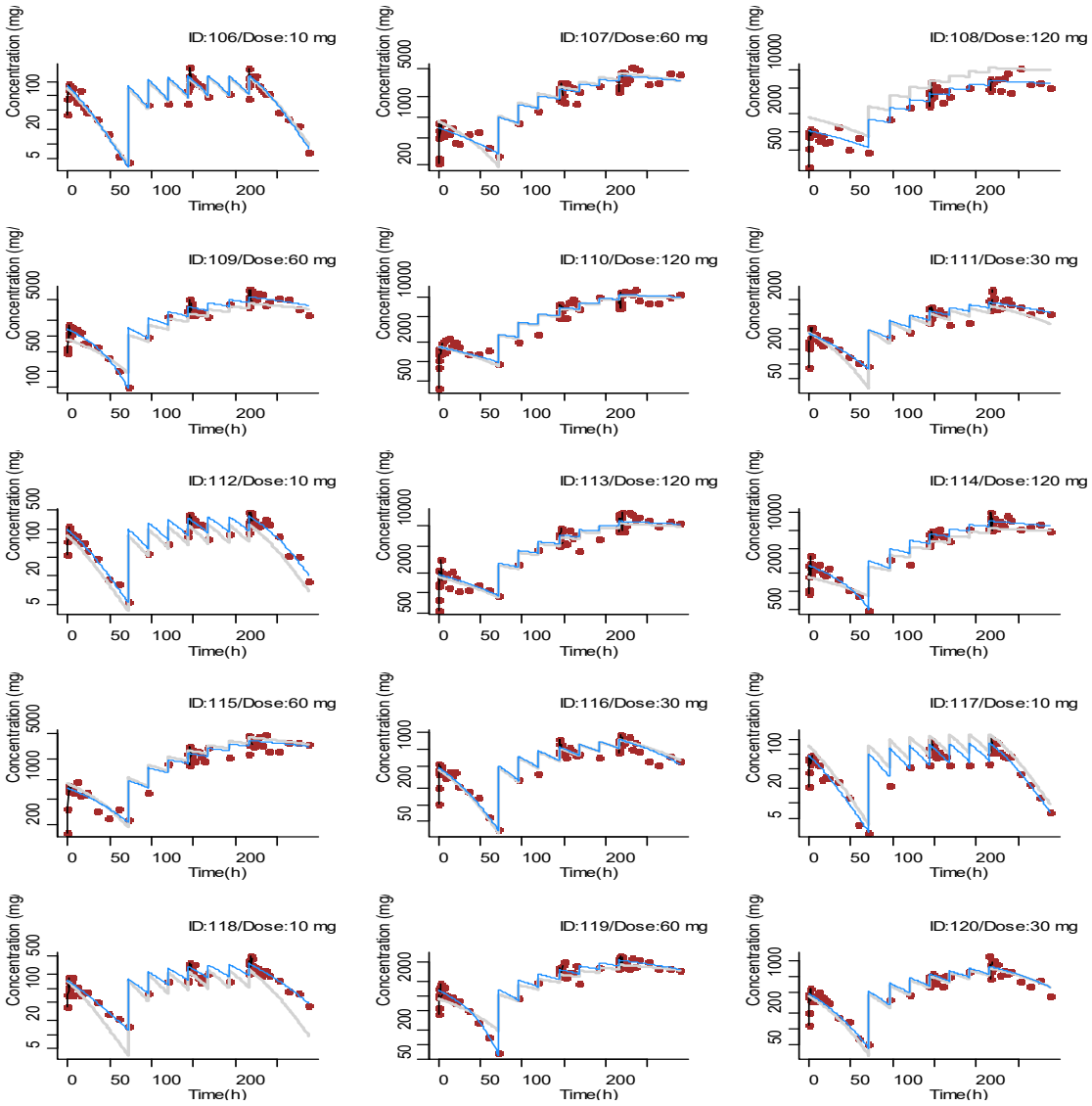
nlmixr performance

- 4 different dose levels (10, 30, 60 and 120 mg) of 30 subjects each as
  - single dose (over 72h)
  - multiple dose (4 daily doses)
  - single and multiple dose combined
  - and steady state dosing
- Range of test models:
  - 1- and 2-compartment disposition
  - with and without 1st order absorption
  - linear or Michaelis-Menten (MM) clearance
- A total of 42 test cases
  - all IIVs were set at 30%, residual error at 20%
  - overlapping PK parameters were the same for all models
- nlmixr estimation routines compared to NONMEM FOCE-I

Example full profiles (linear elimination)



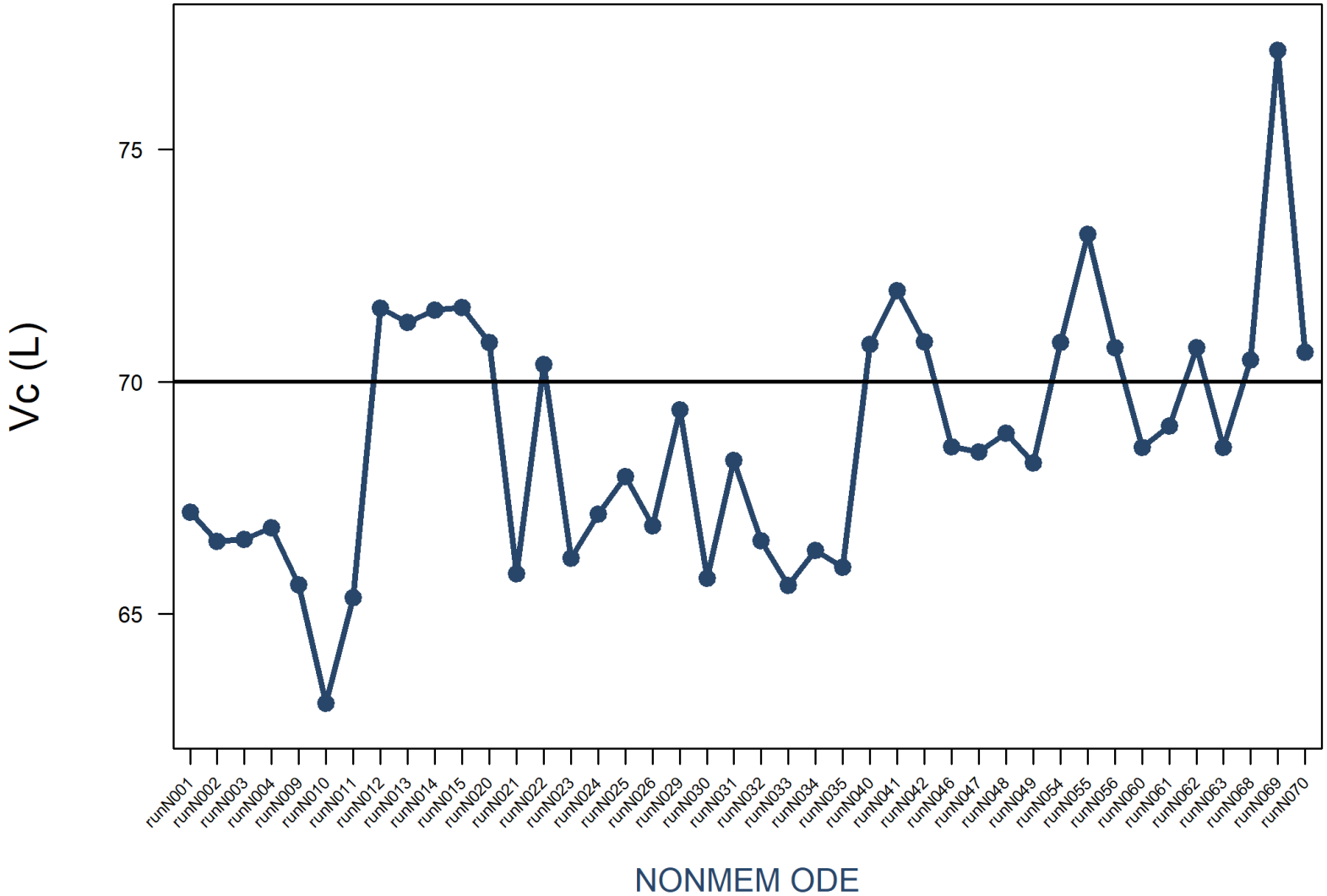
Example full profiles (MM elimination)



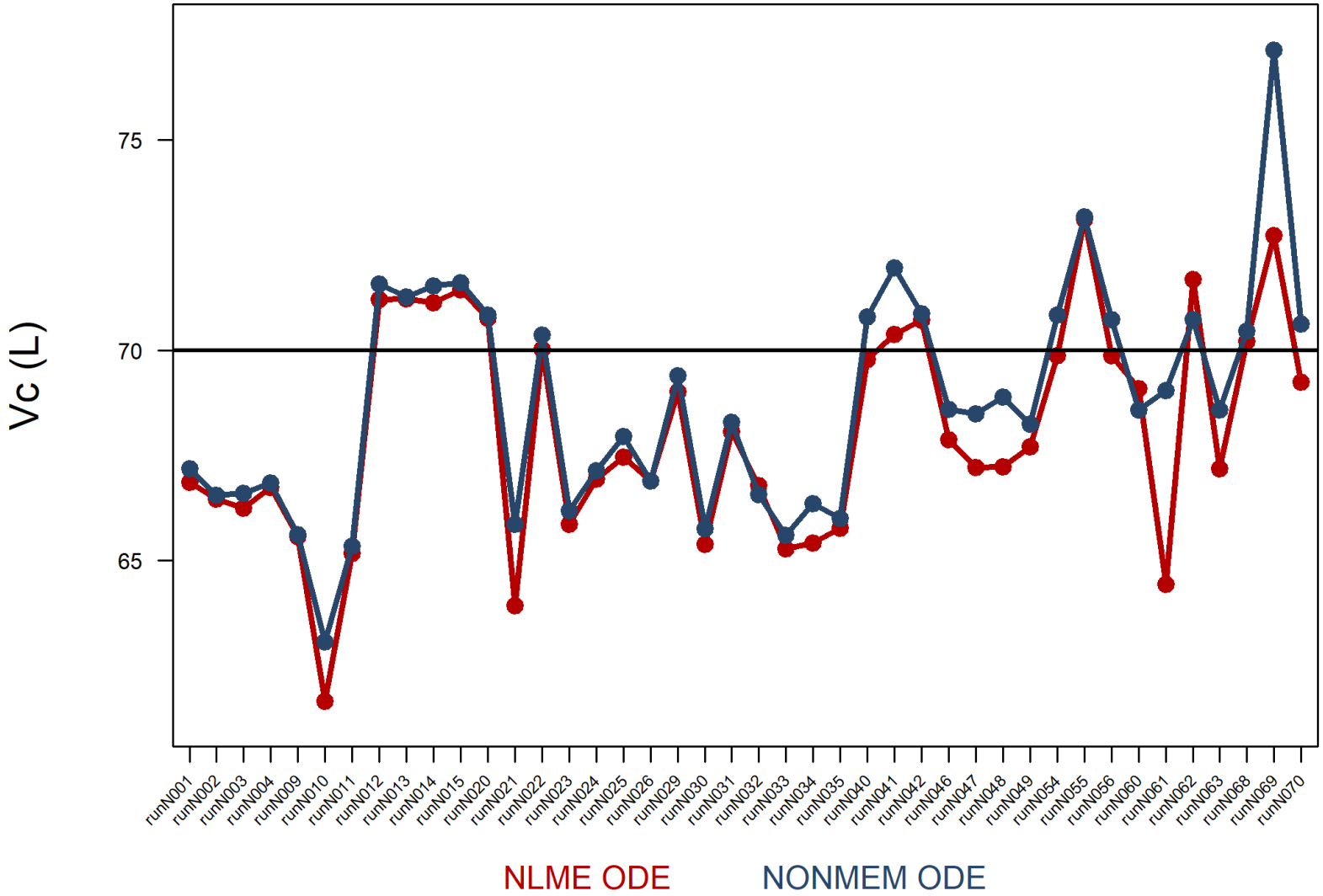
nlmixr

Vc is available in all models:
Theta estimates using NONMEM FOCE-I and ODE implementation

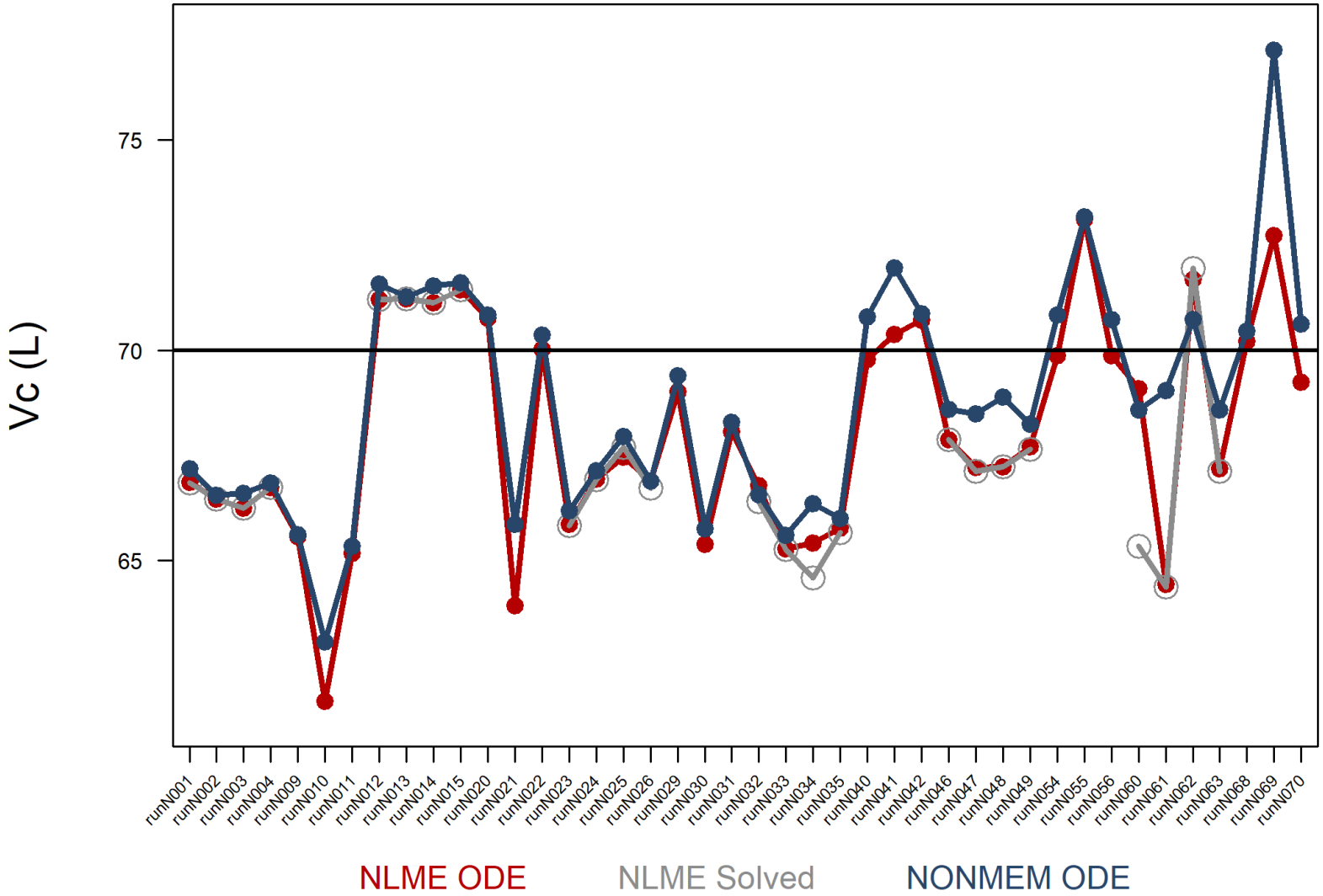
Horizontal black line: value used for simulation



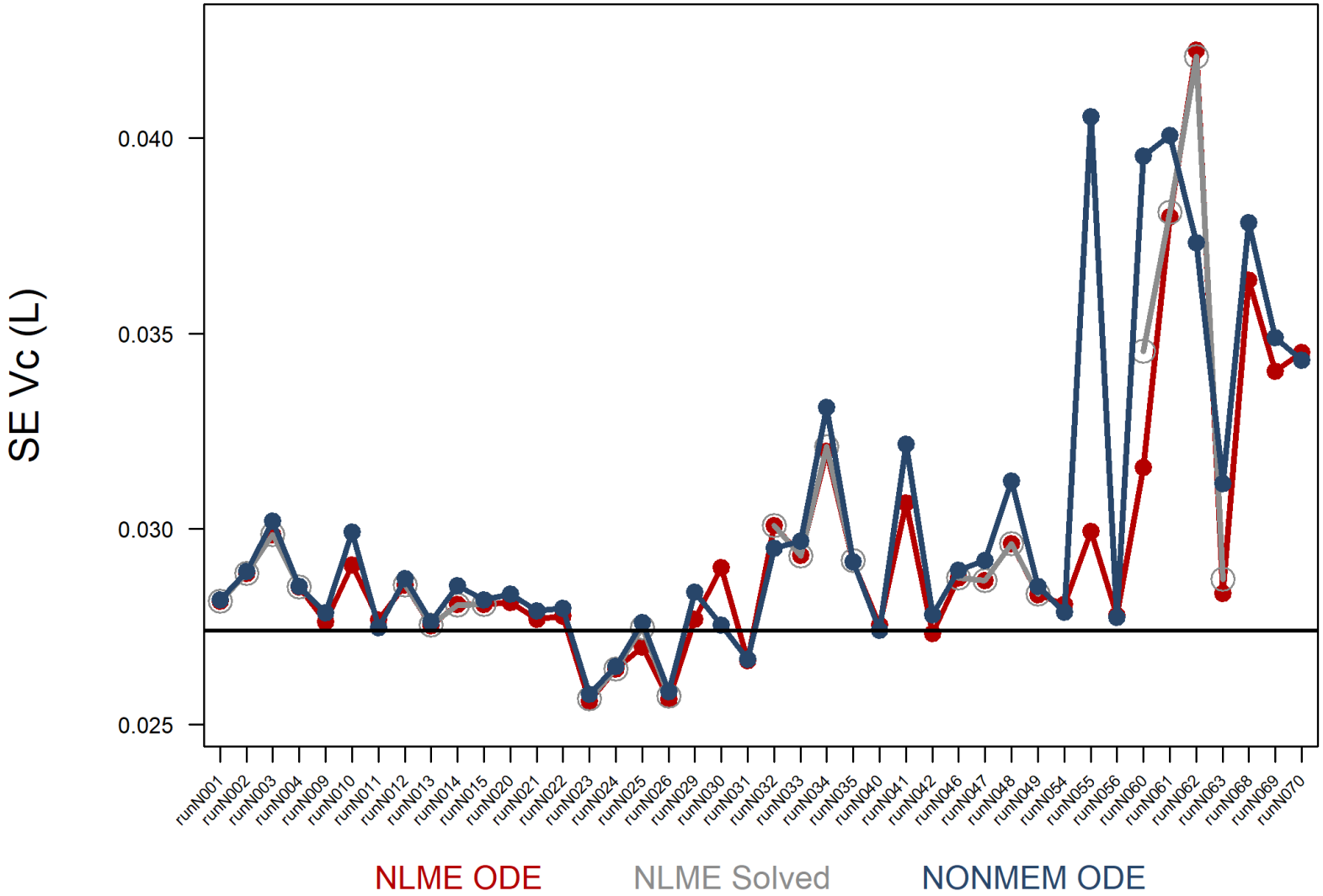
Red line: nlmixr/nlme estimates using ODEs



Non MM models also implemented using closed-form solutions: Grey line: nlmixr/nlme estimates using closed-form solutions

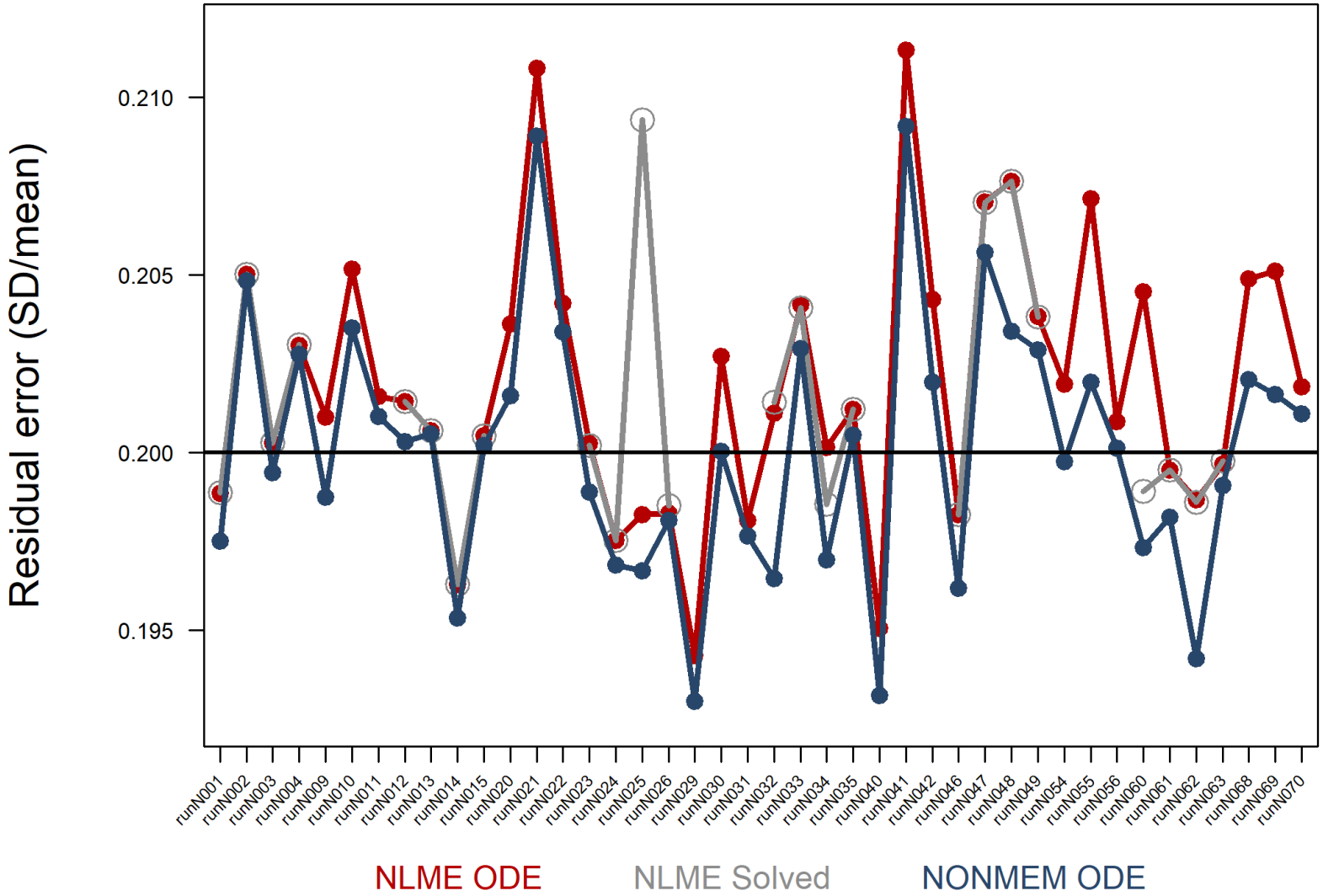


SE of theta estimates for Vc are very comparable

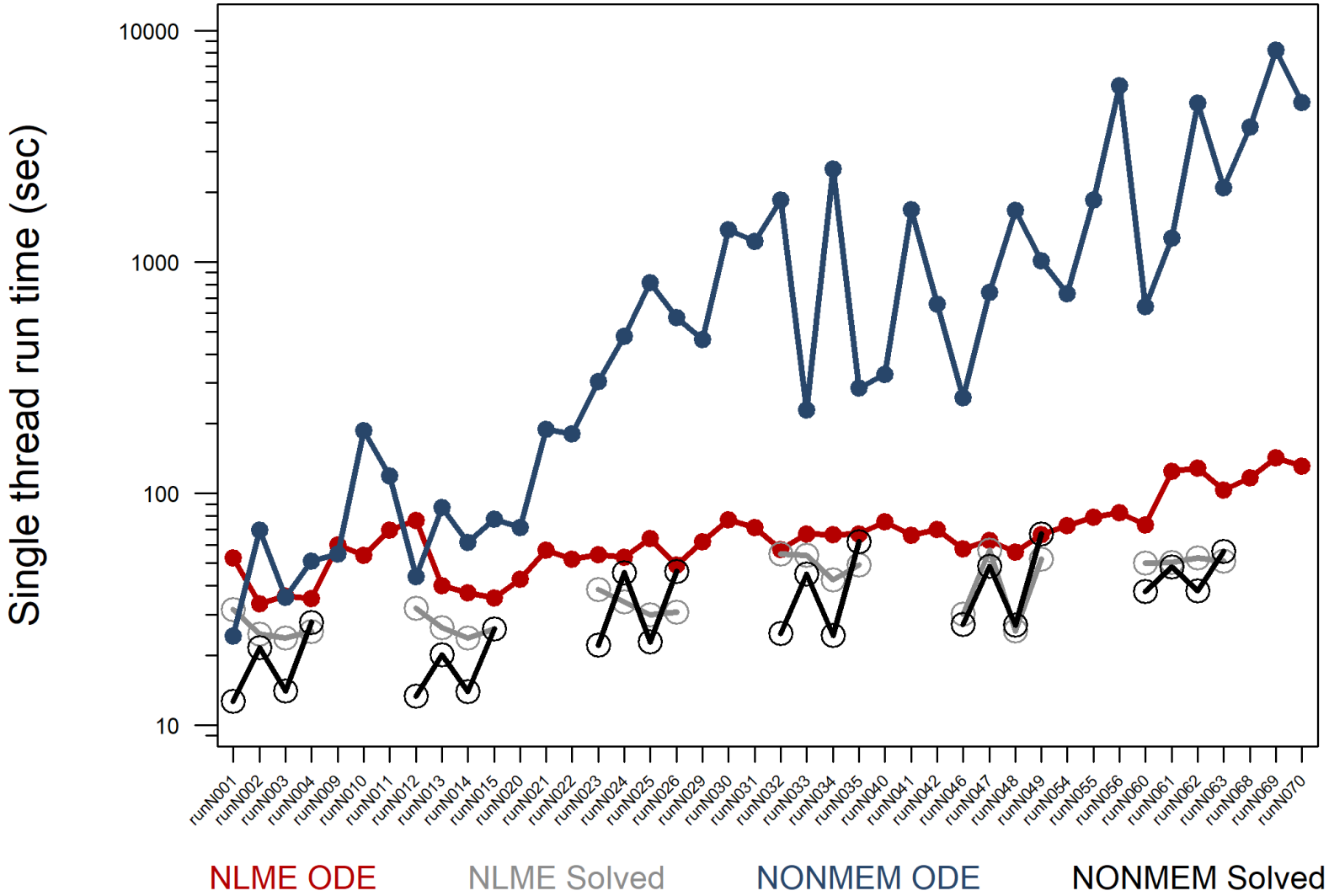


Residual error is well-estimated

Horizontal black line: value used for simulation

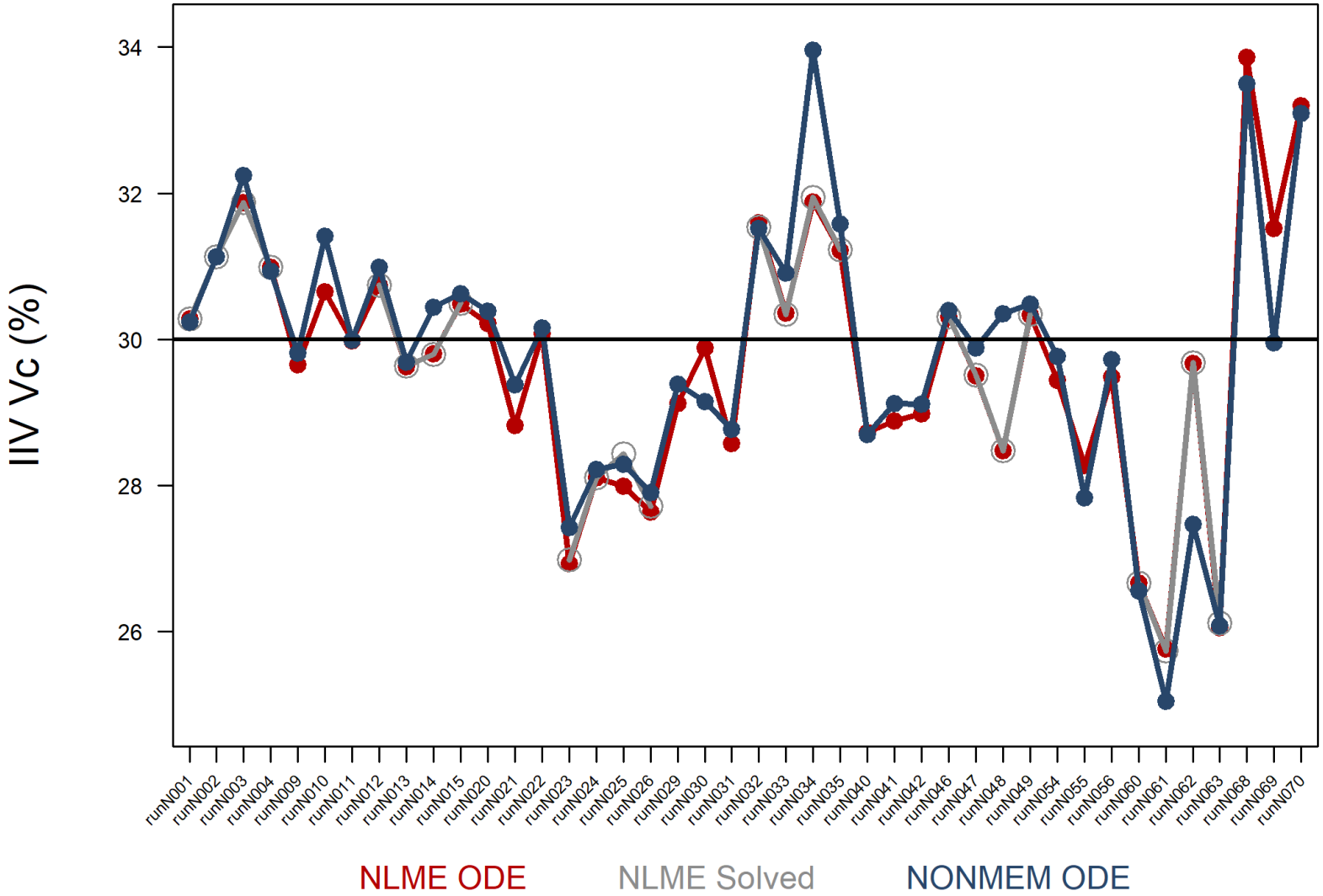


Run times are perfectly acceptable, and often lower than NONMEM …but currently only single-threaded…

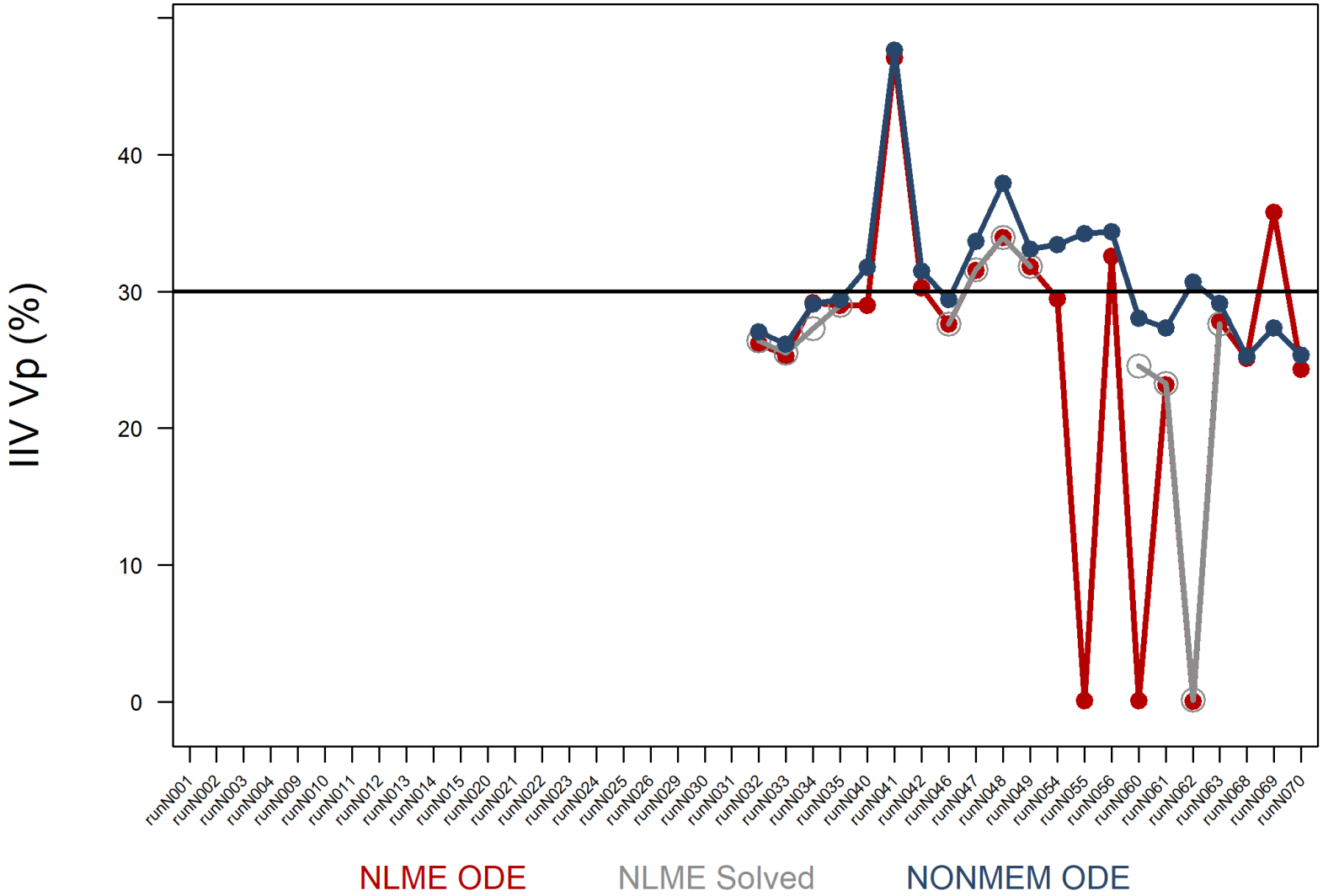


For Vc, Omega (IIV) estimates are also very comparable

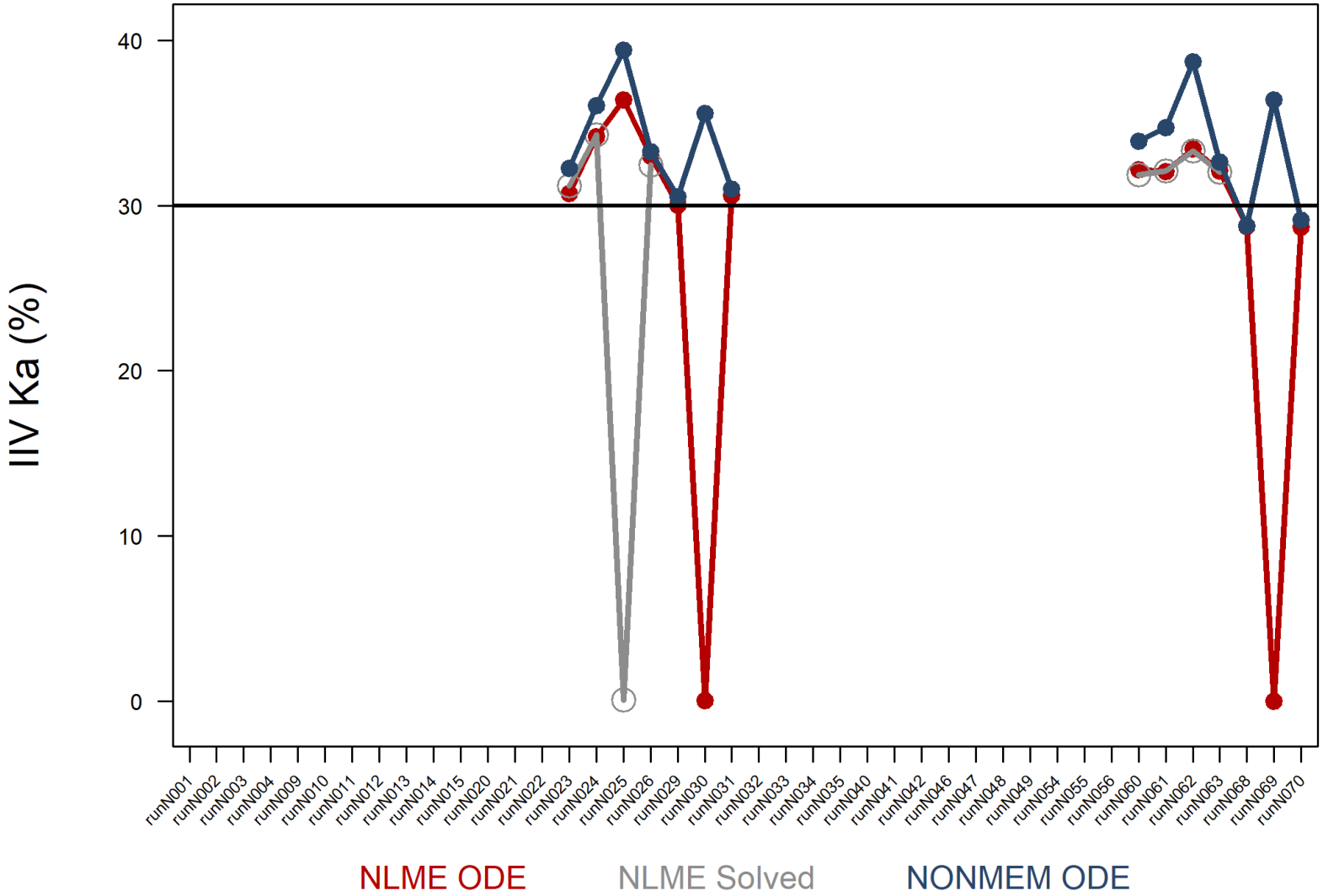
Horizontal black line: value used for simulation



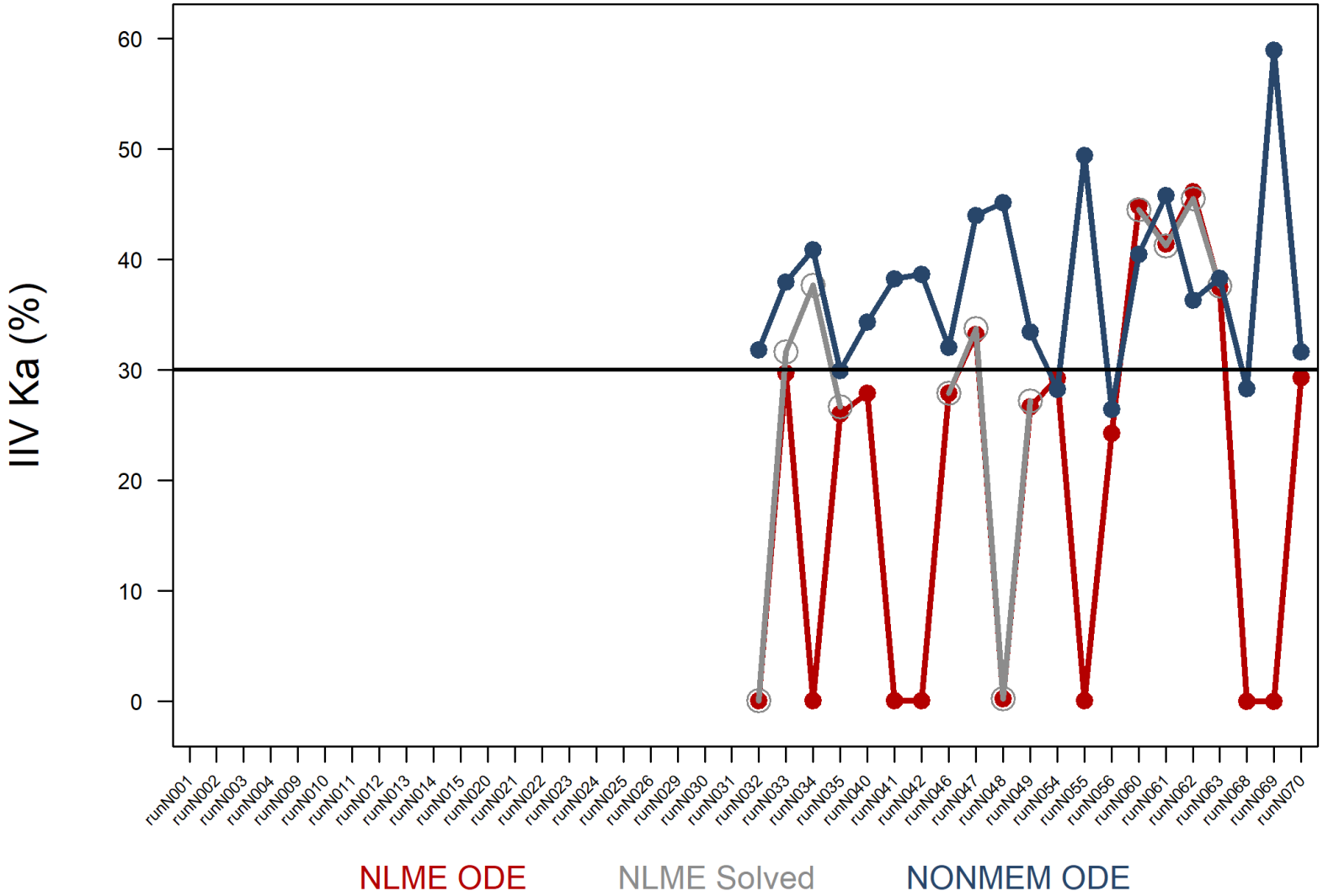
But if we examineVp…
the IIVs are often estimated close to zero



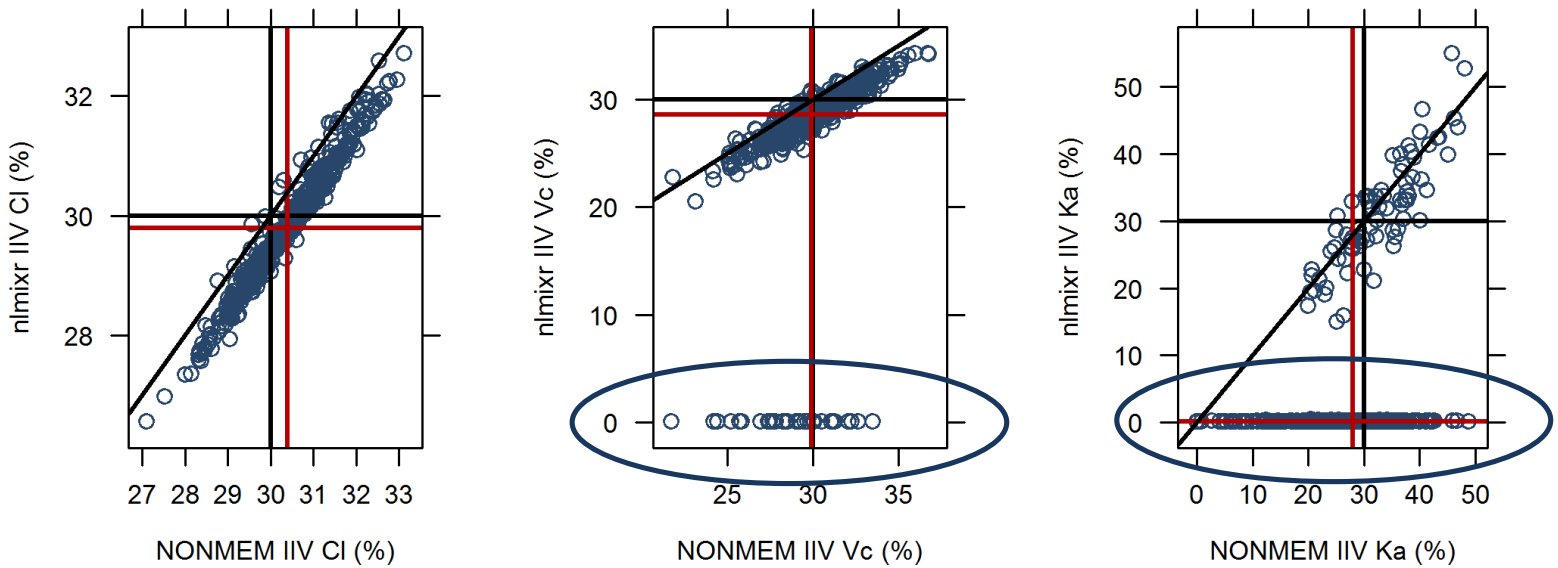
…same with Ka…


IIV Ka (%)
40
30
20
10
0
runN001
runN002
runN003
runN004
runN009
runN010
runN011
runN012
runN013
runN014
runN015
runN020
runN021
runN022
runN023
runN024
runN025
runN026
runN029
runN030
runN031
runN032
runN033
runN034
runN035
runN040
runN041
runN042
runN046
runN047
runN048
runN049
runN054
runN055
runN056
runN060
runN061
runN062
runN063
runN068
runN069
runN070
NLME ODE
NLME Solved
NONMEM ODE

…and Q


IIV Ka (%)
60
50
40
30
20
10
0
runN001
runN002
runN003
runN004
runN009
runN010
runN011
runN012
runN013
runN014
runN015
runN020
runN021
runN022
runN023
runN024
runN025
runN026
runN029
runN030
runN031
runN032
runN033
runN034
runN035
runN040
runN041
runN042
runN046
runN047
runN048
runN049
runN054
runN055
runN056
runN060
runN061
runN062
runN063
runN068
runN069
runN070
NLME ODE
NLME Solved
NONMEM ODE

A large fraction of runs with IIV=0 for Ka for 500 sparse datasets: 91.1% for nlmixr/nlme *vs.* 2.2% for NONMEM FOCE-I



Disappointing results?

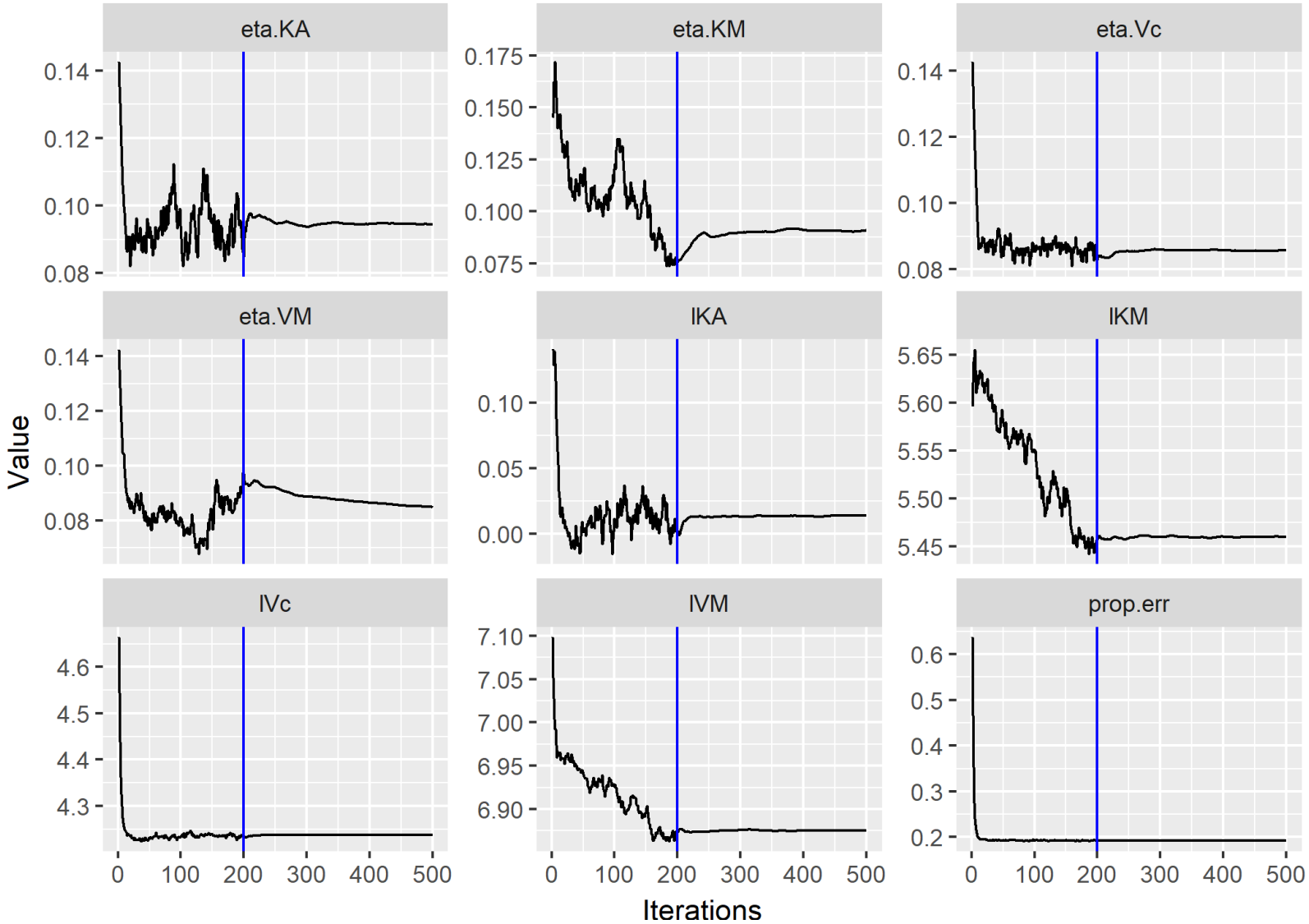
- Findings are in line with earlier experience with nlme
- Bob Bauer claims nlme is somewhere beween ITS and FOCE (personal communication)

- However, nlme in nlmixr provides a gateway into nonlinear mixed effect modelling for statisticians…

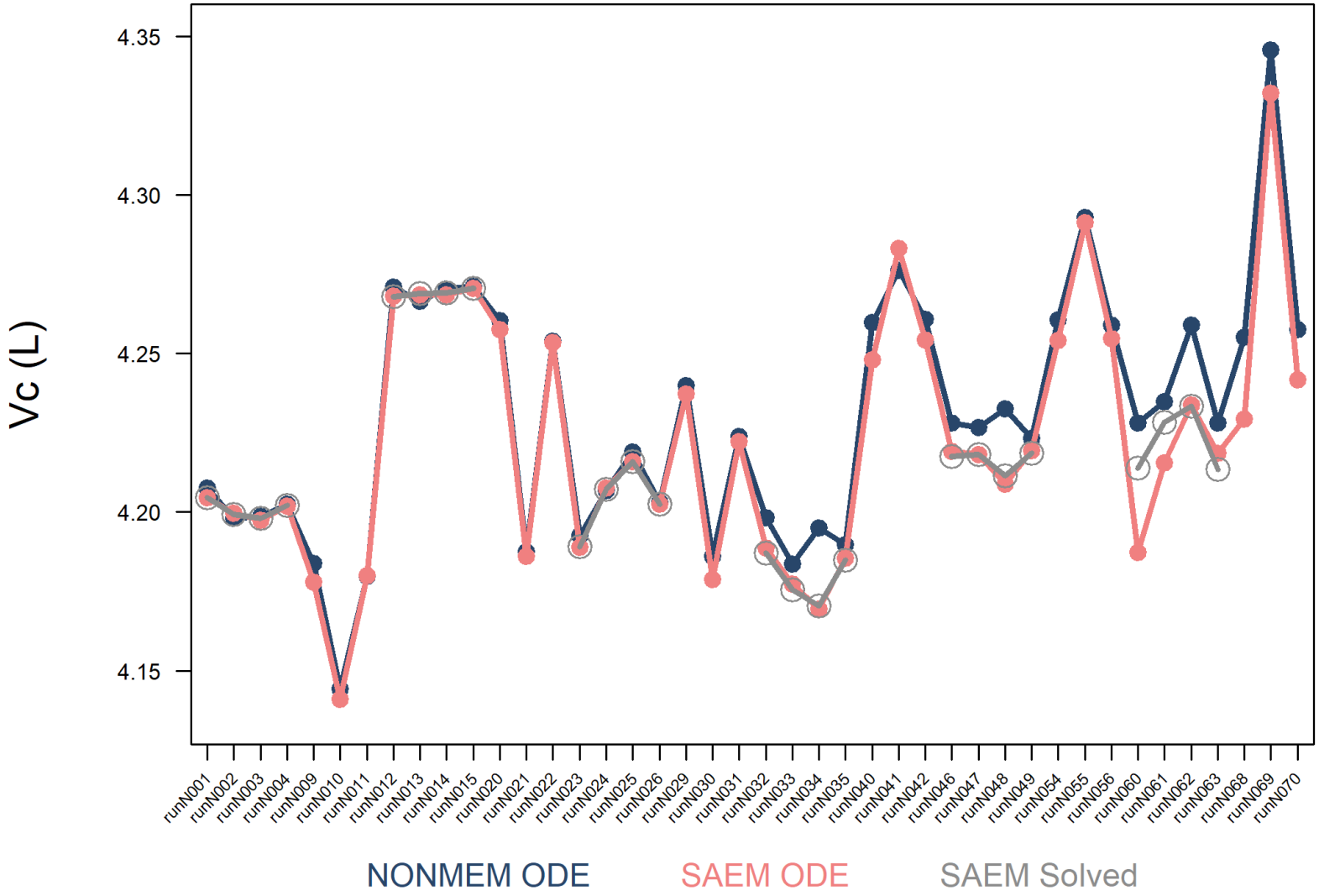
- With the machinery in place, the groundwork is laid for other/better estimation routines, like SAEM or FOCE-I…

- SAEM currently also available in nlmixr: so how does SAEM perform?

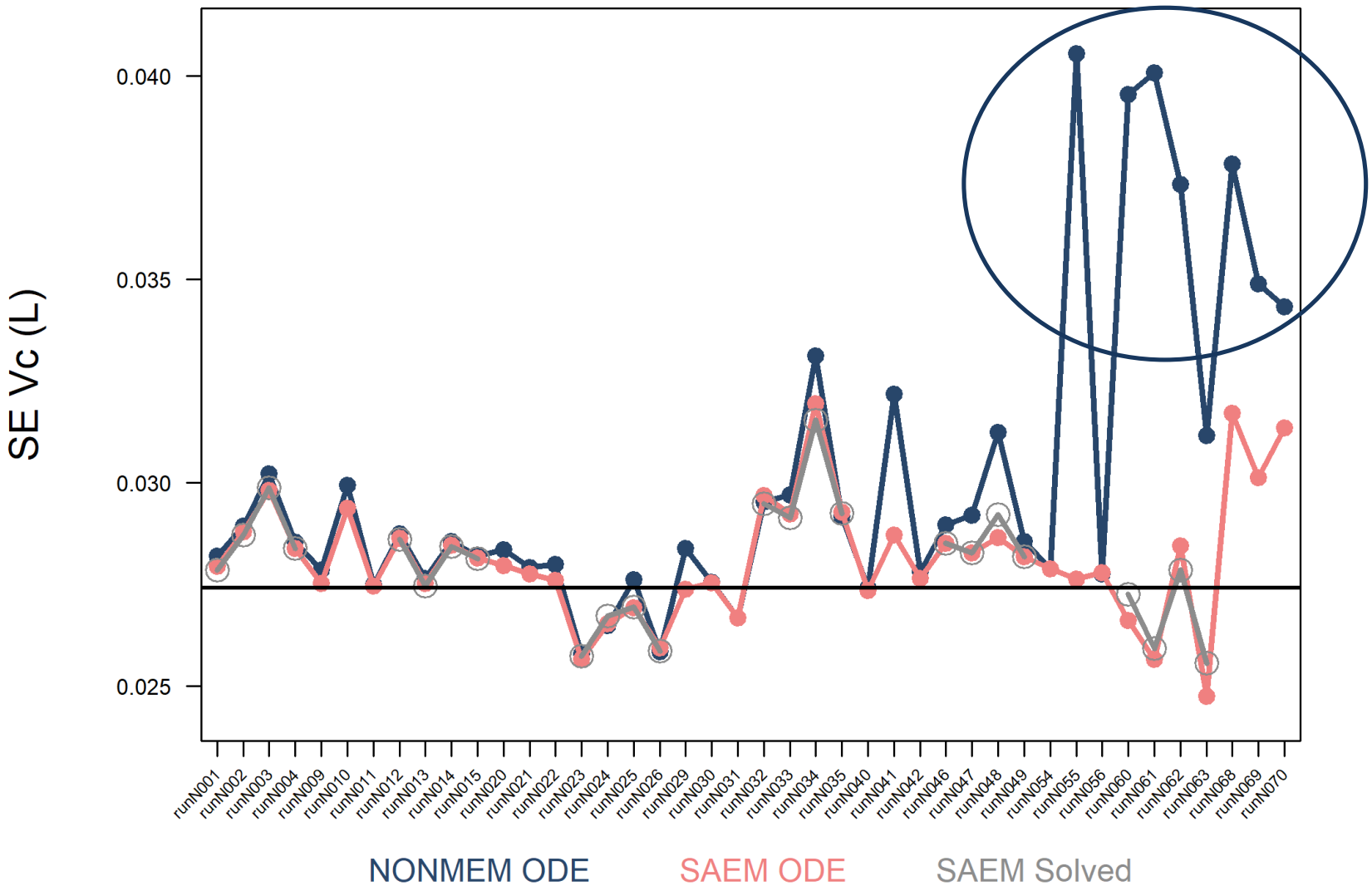
Traceplot for parameters from one of the nlmixr/SAEM models



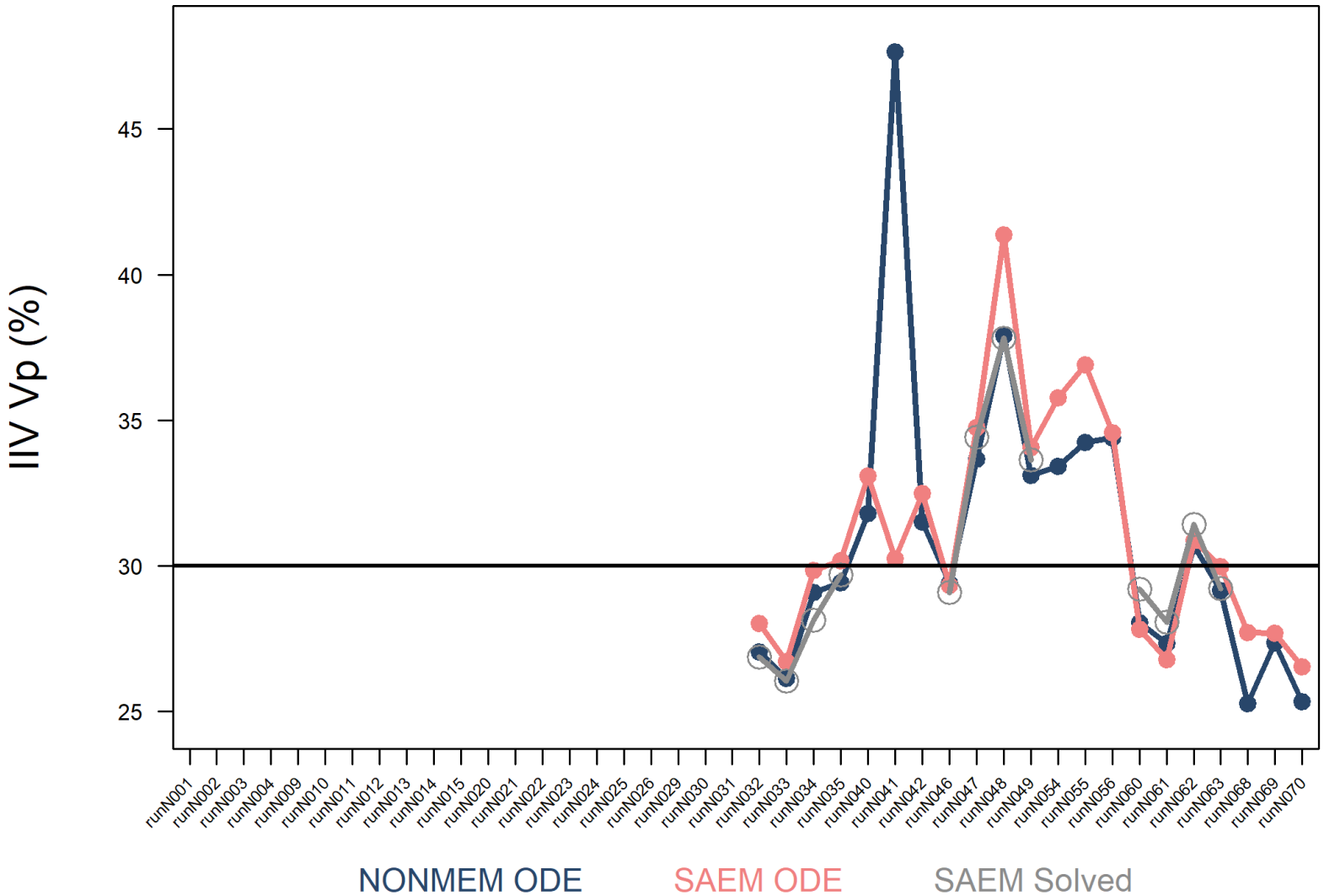
Thetas for nlmixr/SAEM for Vc behave very nicely compared to NONMEM…



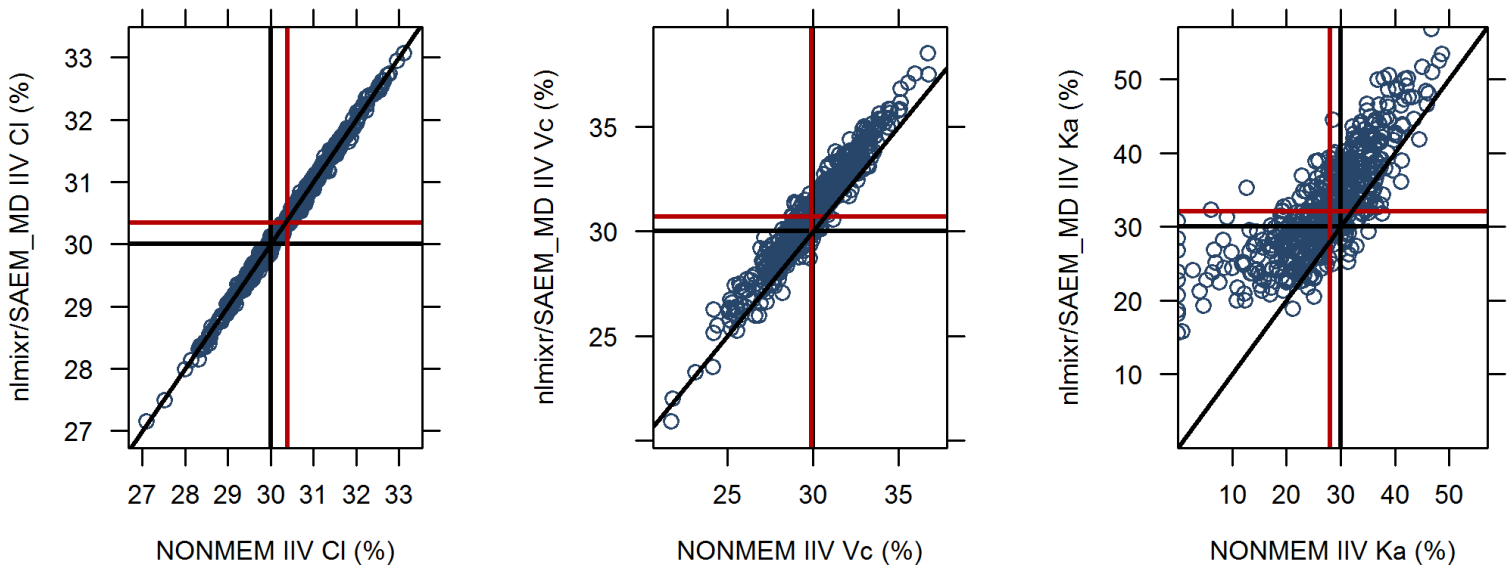
… and SEs for Vc seem to be even better estimated with nlmixr/SAEM than using NONMEM…



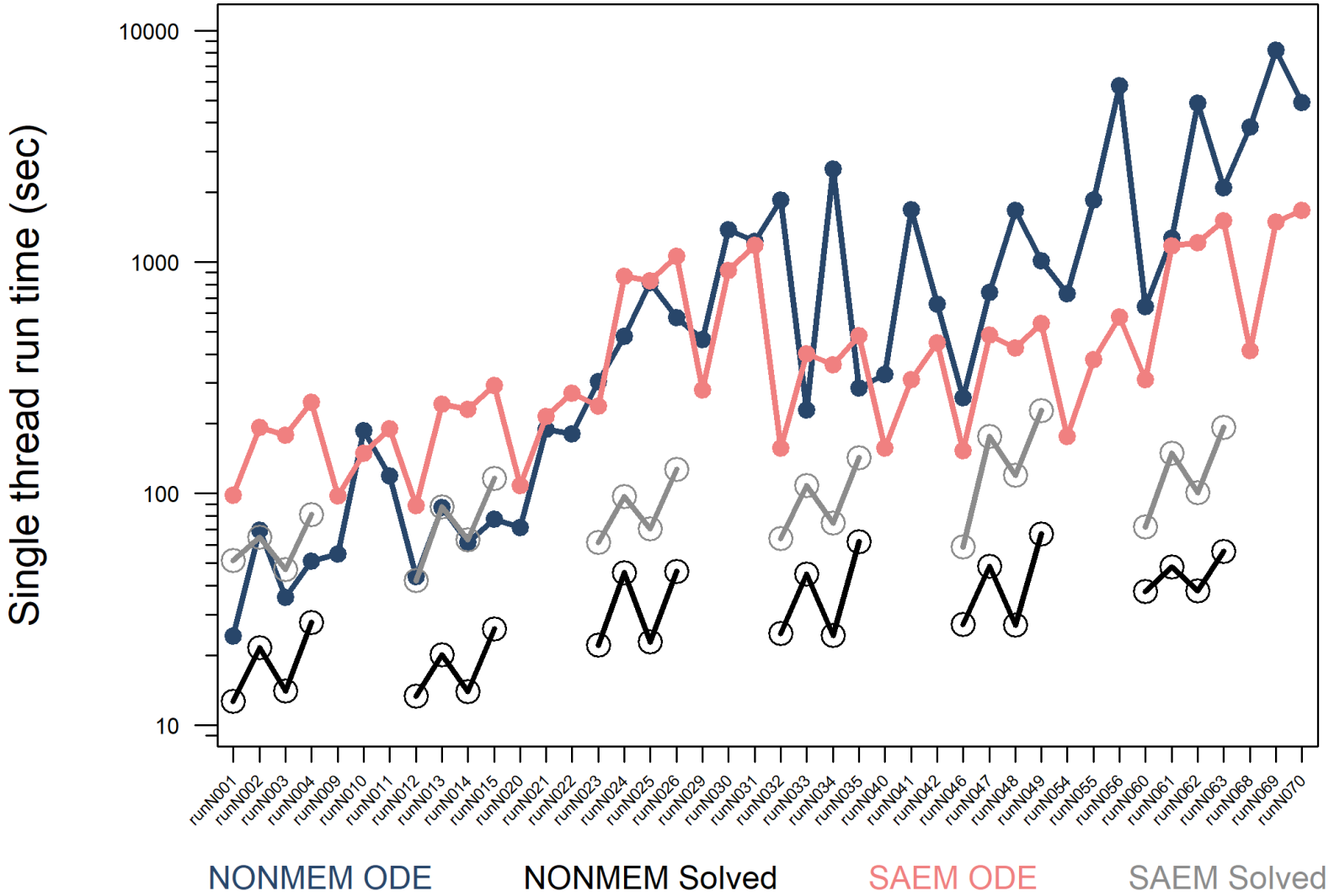
…and IIVs for nlmixr/SAEM for Vp show none of the close to zero behaviour observed with nlme



And no IIVs of zero with nlmixr/SAEM with sparse data



nlmixr/SAEM is slower than nlmixr/nlme but still workable



More good news?

- nlmixr also has an adaptive Gaussian quadrature algorithm (like NONMEM's Laplace and higher) allowing fancy models
- nlmixr also has single subject dynamic models e.g. for complex system simulation and estimation (mcmc algorithm)
- Steps to implement ordered categorical models and count models in the SAEM algorithm
- Elementary implementation of VPC and bootstrap functionality
- Serious progress into multi-threaded simulation that will lead to multi-threaded estimation
- Implementation of FOCE-I under construction

What's next?

- We need you!

- Field-testing: real-life examples

- Improving computational efficiency of estimation algorithms (e.g. within-problem parallelisation)
- Error-trapping
- New features implementation
- Etc, etc…

- This presentation will be made available on the bookdown site nlmixr.github.io