

# Package ‘nlmixr’

September 23, 2018

**Type** Package

**Title** Nonlinear Mixed Effects Models in Population Pharmacokinetics and Pharmacodynamics

**Depends** R (>= 3.2)

**Imports** Rcpp (>= 0.12.3), brew, lattice, parallel, lbfgs, dparser, methods, ggplot2, memoise, rex, minqa, Matrix, numDeriv, R.utils, n1qn1, PreciseSums, fastGHQuad, crayon, cli, RcppArmadillo (>= 0.5.600.2.0), vpc (>= 1.0.0), RxODE (>= 0.8.0-4), nlme, magrittr, tidyr

**Suggests** knitr, rmarkdown, dplyr, data.table, lbfgsb3, testthat, madness, devtools, expm, matrixcalc, reshape2, nloptr, ucminf

**Version** 1.0.0-7

**Description** Fit and compare nonlinear mixed-effects models in differential equations with flexible dosing information commonly seen in pharmacokinetics and pharmacodynamics (Almquist, Leander, and Jirstrand 2015 <doi:10.1007/s10928-015-9409-1>). Differential equation solving is by compiled C code provided in the 'RxODE' package (Wang, Hallow, and James 2015 <doi:10.1002/psp4.12052>).

**License** GPL (>= 2)

**NeedsCompilation** yes

**LinkingTo** dparser (>= 0.1.8), Rcpp (>= 0.12.3), RcppEigen (>= 0.3.3.3.0), StanHeaders, BH

**URL** <https://github.com/nlmixrdevelopment/nlmixr>

**LazyData** true

**VignetteBuilder** knitr

**RoxygenNote** 6.1.0

**Maintainer** Wenping Wang <wwang8198@gmail.com>

**Author** Matthew Fidler [aut],  
Yuan Xiong [aut],  
Rik Schoemaker [aut],  
Justin Wilkins [aut],

Mirjam Trame [aut],  
 Teun Post [aut],  
 Robert Leary [ctb],  
 Wenping Wang [aut, cre],  
 Hadley Wickham [ctb],  
 Dirk Eddelbuettel [cph],  
 David Ardia [cph],  
 Katharine Mullen [cph]

**Repository** CRAN

**Date/Publication** 2018-09-23 14:00:03 UTC

## R topics documented:

.collectWarnings	4
addCwres	5
addNpde	5
as.focei	6
Bolus_1CPT	7
Bolus_1CPTMM	8
Bolus_2CPT	9
Bolus_2CPTMM	10
bootdata	11
calcCov	12
configsaeM	13
dynmodel	14
dynmodel.mcmc	16
focei.eta	17
focei.theta	18
foceiControl	18
foceiFit	24
frwd_selection	31
gauss.quad	32
gen_saem_user_fn	32
getOMEGA	33
gnlmm	33
Infusion_1CPT	35
ini	36
instant.stan.extension	37
invgaussian	37
lincmt	38
lin_cmt	38
metabolite	39
model	40
nlme_gof	40
nlme_lin_cmpt	41
nlme_ode	42
nlmixr	44

<code>nlmixrAugPred</code>	54
<code>nlmixrBounds.eta.names</code>	56
<code>nlmixrBounds.focei.upper.lower</code>	56
<code>nlmixrLogo</code>	57
<code>nlmixrPred</code>	57
<code>nlmixrSim</code>	58
<code>nlmixrUI.focei.fixed</code>	62
<code>nlmixrUI.focei.inits</code>	62
<code>nlmixrUI.nlme.specs</code>	63
<code>nlmixrUI.rxode.pred</code>	63
<code>nlmixrUI.saem.ares</code>	64
<code>nlmixrUI.saem.bres</code>	64
<code>nlmixrUI.saem.distribution</code>	65
<code>nlmixrUI.saem.eta.trans</code>	65
<code>nlmixrUI.saem.fit</code>	66
<code>nlmixrUI.saem.fixed</code>	66
<code>nlmixrUI.saem.init</code>	67
<code>nlmixrUI.saem.init.omega</code>	67
<code>nlmixrUI.saem.init.theta</code>	68
<code>nlmixrUI.saem.log.eta</code>	68
<code>nlmixrUI.saem.model</code>	69
<code>nlmixrUI.saem.model.omega</code>	69
<code>nlmixrUI.saem.res.mod</code>	70
<code>nlmixrUI.saem.res.name</code>	70
<code>nlmixrUI.saem.theta.name</code>	71
<code>nlmixrUI.theta.pars</code>	71
<code>nlmixrUILinCmt</code>	72
<code>nlmixrValidate</code>	72
<code>nlmixrVersion</code>	73
<code>nlmixr_fit</code>	73
<code>nmDataConvert</code>	74
<code>nmsimplex</code>	74
<code>Oral_1CPT</code>	75
<code>plot.dyn.ID</code>	76
<code>plot.dyn.mcmc</code>	76
<code>plot.nlmixrFitData</code>	77
<code>plot.saemFit</code>	77
<code>prediction</code>	78
<code>print.dyn.ID</code>	79
<code>print.dyn.mcmc</code>	79
<code>print.gnlmm.fit</code>	80
<code>print.nlmixrUI</code>	80
<code>print.saemFit</code>	81
<code>pump</code>	81
<code>rats</code>	82
<code>residuals.nlmixrFitData</code>	83
<code>saem.cleanup</code>	84
<code>saem.fit</code>	84

saemControl . . . . .	86
summary.dyn.ID . . . . .	87
summary.dyn.mcmc . . . . .	88
summary.saemFit . . . . .	88
tableControl . . . . .	89
theo_md . . . . .	90
theo_sd . . . . .	91
traceplot . . . . .	91
vpc . . . . .	92
vpc_nlmixr_nlme . . . . .	92
vpc_saemFit . . . . .	93
vpc_ui . . . . .	94
Wang2007 . . . . .	96
<b>Index</b>	<b>97</b>

---

<code>.collectWarnings</code>	<i>Collect warnings and just warn once.</i>
-------------------------------	---

---

### Description

Collect warnings and just warn once.

### Usage

```
.collectWarnings(expr)
```

### Arguments

<code>expr</code>	R expression
-------------------	--------------

### Value

The value of the expression

### Author(s)

Matthew L. Fidler

---

addCwres	<i>Add CWRES</i>
----------	------------------

---

**Description**

This returns a new fit object with CWRES attached

**Usage**

```
addCwres(fit, updateObject = TRUE)
```

**Arguments**

fit	nlmixr fit without WRES/CWRES
updateObject	Boolean indicating if the original fit object should be updated. By default this is true.

**Value**

fit with CWRES

**Author(s)**

Matthew L. Fidler

---

addNpde	<i>NPDE calculation for nlmixr</i>
---------	------------------------------------

---

**Description**

NPDE calculation for nlmixr

**Usage**

```
addNpde(object, nsim = 300, ties = TRUE, seed = 1009,
         updateObject = TRUE, ...)
```

**Arguments**

object	nlmixr fit object
nsim	Number of simulations. By default this is 300
ties	When TRUE, the npde distribution can have ties. When FALSE, the npde distribution uses uniform random numbers to prevent ties.
seed	Seed for running nlmixr simulation. By default 1009
updateObject	Boolean indicating if original object should be updated. By default this is TRUE.
...	Other ignored parameters.

**Value**

New nlmixr fit object

**Author(s)**

Matthew L. Fidler

---

as.focei

*Convert fit to FOCEi style fit*

---

**Description**

Convert fit to FOCEi style fit

**Usage**

```
as.focei(object, uif, pt = proc.time(), ..., data, calcResid = TRUE)
```

```
## S3 method for class 'nlmixrNlme'  
as.focei(object, uif, pt = proc.time(), ..., data,  
  calcResid = TRUE)
```

**Arguments**

object	Fit object to convert to FOCEi-style fit.
uif	Unified Interface Function
pt	Proc time object
...	Other Parameters
data	The data to pass to the FOCEi translation.
calcResid	A boolean to indicate if the CWRES residuals should be calculated

**Value**

A FOCEi fit style object.

**Author(s)**

Matthew L. Fidler

---

Bolus_1CPT	<i>Bolus_1CPT – 1 Compartment Model Simulated Data from ACOP 2016</i>
------------	---

---

**Description**

This is a simulated dataset from the ACOP 2016 poster. All Datasets were simulated with the following methods.

**Usage**

Bolus\_1CPT

**Format**

A data frame with 7,920 rows and 14 columns

**ID** Simulated Subject ID

**TIME** Simulated Time

**DV** Simulated Dependent Variable

**LNDV** Simulated log(Dependent Variable)

**MDV** Missing DV data item

**AMT** Dosing AMT

**EVID** NONMEM Event ID

**DOSE** Dose

**V** Individual Simulated Volume

**CL** Individual Clearance

**SS** Steady State

**II** Interdose Interval

**SD** Single Dose Flag

**CMT** Compartment

**Details**

Richly sampled profiles were simulated for 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, for a range of test models: 1- and 2-compartment disposition, with and without 1st order absorption, with either linear or Michaelis-Menten (MM) clearance (MM without steady state dosing). This provided a total of 42 test cases. All inter-individual variabilities (IIVs) were set at 30 were the same for all models. A similar set of models was previously used to compare NONMEM and Monolix4. Estimates of population parameters, standard errors for fixed-effect parameters, and run times were compared both for closed-form solutions and using ODEs. Additionally, a sparse data estimation situation was investigated where 500 datasets of 600 subjects each (150 per dose) were generated consisting of 4 random time point samples in 24 hours per subject, using a first-order absorption, 1-compartment disposition, linear elimination model.

**Source**

Schoemaker R, Xiong Y, Wilkins J, Laveille C, Wang W. nlmixr: an open-source package for pharmacometric modelling in R. ACOP 2016

---

Bolus\_1CPTMM

*1 Compartment Model w/ Michaelis Menton Elimination*

---

**Description**

This is a simulated dataset from the ACOP 2016 poster. All Datasets were simulated with the following methods.

**Usage**

Bolus\_1CPTMM

**Format**

A data frame with 7,920 rows and 14 columns

**ID** Simulated Subject ID

**TIME** Simulated Time

**DV** Simulated Dependent Variable

**LNDV** Simulated log(Dependent Variable)

**MDV** Missing DV data item

**AMT** Dosing AMT

**EVID** NONMEM Event ID

**DOSE** Dose

**V** Individual Simulated Volume

**VM** Individual Vm constant

**KM** Individual Km constant

**SD** Single Dose Flag

**CMT** Compartment

**Details**

Richly sampled profiles were simulated for 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, for a range of test models: 1- and 2-compartment disposition, with and without 1st order absorption, with either linear or Michaelis-Menten (MM) clearance(MM without steady state dosing). This provided a total of 42 test cases. All inter-individual variabilities (IIVs) were set at 30 were the same for all models. A similar set of models was previously used to



compare NONMEM and Monolix4. Estimates of population parameters, standard errors for fixed-effect parameters, and run times were compared both for closed-form solutions and using ODEs. Additionally, a sparse data estimation situation was investigated where 500 datasets of 600 subjects each (150 per dose) were generated consisting of 4 random time point samples in 24 hours per subject, using a first-order absorption, 1-compartment disposition, linear elimination model.

### Source

Schoemaker R, Xiong Y, Wilkins J, Laveille C, Wang W. nlmixr: an open-source package for pharmacometric modelling in R. ACOP 2016

---

Bolus_2CPT	<i>2 Compartment Model</i>
------------	----------------------------

---

### Description

This is a simulated dataset from the ACOP 2016 poster. All Datasets were simulated with the following methods.

### Usage

Bolus\_2CPT

### Format

A data frame with 7,920 rows and 16 columns

**ID** Simulated Subject ID

**TIME** Simulated Time

**DV** Simulated Dependent Variable

**LNDV** Simulated log(Dependent Variable)

**MDV** Missing DV data item

**AMT** Dosing AMT

**EVID** NONMEM Event ID

**DOSE** Dose

**V1** Individual Central Compartment Volume

**CL** Individual Clearance

**Q** Individual Between Compartment Clearance

**V2** Peripherial Volume

**SS** Steady State Flag

**II** Interdose interval

**SD** Single Dose Flag

**CMT** Compartment Indicator

**Details**

Richly sampled profiles were simulated for 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, for a range of test models: 1- and 2-compartment disposition, with and without 1st order absorption, with either linear or Michaelis-Menten (MM) clearance (MM without steady state dosing). This provided a total of 42 test cases. All inter-individual variabilities (IIVs) were set at 30 were the same for all models. A similar set of models was previously used to compare NONMEM and Monolix4. Estimates of population parameters, standard errors for fixed-effect parameters, and run times were compared both for closed-form solutions and using ODEs. Additionally, a sparse data estimation situation was investigated where 500 datasets of 600 subjects each (150 per dose) were generated consisting of 4 random time point samples in 24 hours per subject, using a first-order absorption, 1-compartment disposition, linear elimination model.

**Source**

Schoemaker R, Xiong Y, Wilkins J, Laveille C, Wang W. nlmixr: an open-source package for pharmacometric modelling in R. ACOP 2016

---

 Bolus\_2CPTMM

---

*2 Compartment Model with Michaelis Menton Clearance*


---

**Description**

This is a simulated dataset from the ACOP 2016 poster. All Datasets were simulated with the following methods.

**Usage**

Bolus\_2CPTMM

**Format**

A data frame with 7,920 rows and 15 columns

**ID** Simulated Subject ID

**TIME** Simulated Time

**DV** Simulated Dependent Variable

**LNDV** Simulated log(Dependent Variable)

**MDV** Missing DV data item

**AMT** Dosing AMT

**EVID** NONMEM Event ID

**DOSE** Dose

**V** Individual Central Compartment Volume

**VM** Individual Vmax

**KM** Individual Km  
**Q** Individual Q  
**V2** Individual Peripheral Compartment Volume  
**SD** Single Dose Flag  
**CMT** Compartment Indicator

### Details

Richly sampled profiles were simulated for 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, for a range of test models: 1- and 2-compartment disposition, with and without 1st order absorption, with either linear or Michaelis-Menten (MM) clearance (MM without steady state dosing). This provided a total of 42 test cases. All inter-individual variabilities (IIVs) were set at 30 were the same for all models. A similar set of models was previously used to compare NONMEM and Monolix4. Estimates of population parameters, standard errors for fixed-effect parameters, and run times were compared both for closed-form solutions and using ODEs. Additionally, a sparse data estimation situation was investigated where 500 datasets of 600 subjects each (150 per dose) were generated consisting of 4 random time point samples in 24 hours per subject, using a first-order absorption, 1-compartment disposition, linear elimination model.

### Source

Schoemaker R, Xiong Y, Wilkins J, Laveille C, Wang W. nlmixr: an open-source package for pharmacometric modelling in R. ACOP 2016

---

bootdata

*Bootstrap data*

---

### Description

Bootstrap data by sampling the same number of subjects from the original dataset by sampling with replacement.

### Usage

```
bootdata(dat)
```

### Arguments

dat                    model data to be bootstrapped

### Value

Bootstrapped data

**Examples**

```

specs <- list(fixed=lKA+lCL+lV~1, random = pdDiag(lKA+lCL~1), start=c(lKA=0.5, lCL=-3.2, lV=-1))
set.seed(99); nboot = 5;

cat("generating", nboot, "bootstrap samples...\n")
cmat <- matrix(NA, nboot, 3)
for (i in 1:nboot)
{
  #print(i)
  bd <- bootdata(theo_md)
  fit <- nlme_lin_cmpt(bd, par_model=specs, ncmt=1)
  cmat[i,] = fit$coefficients$fixed
}
dimnames(cmat)[[2]] <- names(fit$coefficients$fixed)
print(head(cmat))

```

---

calcCov

*Calculate gnlmm variance-covariance matrix of fixed effects*


---

**Description**

Calculate variance-covariance matrix of fixed effects after a gnlmm() fit

**Usage**

```
calcCov(fit, method = 1, trace = FALSE)
```

**Arguments**

fit	a gnlmm fit object
method	method for calculating variance-covariance matrix
trace	logical whether to trace the iterations

**Value**

variance-covariance matrix of model parameters

---

 configsaem

*Configure an SAEM model*


---

**Description**

Configure an SAEM model by generating an input list to the SAEM model function

**Usage**

```
configsaem(model, data, inits, mcmc = list(niter = c(200, 300), nmc = 3,
  nu = c(2, 2, 2)), ODEopt = list(atol = 1e-06, rtol = 1e-04, stiff = 1,
  transitAbs = 0), distribution = c("normal", "poisson", "binomial"),
  seed = 99, fixed = NULL, DEBUG = 0)
```

**Arguments**

model	a compiled saem model by gen_saem_user_fn()
data	input data
inits	initial values
mcmc	a list of various mcmc options
ODEopt	optional ODE solving options
distribution	one of c("normal","poisson","binomial")
seed	seed for random number generator
fixed	a character vector of fixed effect only parameters (no random effects attached) to be fixed
DEBUG	Integer determining if debugging is enabled.

**Details**

Fit a generalized nonlinear mixed-effect model by the Stochastic Approximation Expectation-Maximization (SAEM) algorithm

**Author(s)**

Wenping Wang

**Examples**

```
## Not run:
library(nlmixr)

#ode <- "d/dt(depot) = -KA*depot;
#      d/dt(centr) = KA*depot - KE*centr;"
#m1 = RxODE(ode, modName="m1")
#ode <- "C2 = centr/V;
#      d/dt(depot) = -KA*depot;
```

```

#      d/dt(centr) = KA*depot - KE*centr;"
#m2 = RxODE(ode, modName="m2")

#Note: only use the '=' assignment, not the '<-' at this point

PKpars = function()
{
  CL = exp(lCL)
  V = exp(lV)
  KA = exp(lKA)
  KE = CL / V
  #initCondition = c(0, KE - CL/V)
}
PRED = function() centr / V
PRED2 = function() C2

saem_fit <- gen_saem_user_fn(model=lincmt(ncmt=1, oral=T))
#saem_fit <- gen_saem_user_fn(model=m1, PKpars, pred=PRED)
#saem_fit <- gen_saem_user_fn(model=m2, PKpars, pred=PRED2)

#--- saem cfg
nmdat = theo_sd
model = list(saem_mod=saem_fit, covars="WT")
inits = list(theta=c(.05, .5, 2))
cfg = configsaem(model, nmdat, inits)
cfg$print = 50

#cfg$Rfn = nlmixr:::Ruser_function_cmt
#dyn.load("m1.d/m1.so");cfg$Rfn = nlmixr:::Ruser_function_ode
fit = saem_fit(cfg)
df = simple.gof(fit)
xyplot(DV~TIME|ID, df, type=c("p", "l"), lwd=c(NA,1), pch=c(1,NA), groups=grp)
fit

## End(Not run)

```

---

dynmodel

*Fit a non-population dynamic model*

---

## Description

Fit a non-population dynamic model

## Usage

```

dynmodel(system, model, evTable, inits, data, fixPars = NULL,
  method = c("Nelder-Mead", "L-BFGS-B", "PORT"),
  control = list(ftol_rel = 1e-06, maxeval = 999), squared = T)

```

**Arguments**

system	an RxODE object
model	a list of statistical measurement models
evTable	an Event Table object
inits	initial values of system parameters
data	input data
fixPars	fixed system parameters
method	estimation method: choice of Nelder-Mead, L-BFGS-B, and PORT.
control	optional minimization control parameters
squared	if parameters be squared during estimation

**Author(s)**

Wenping Wang

**Examples**

```
ode <- "
  dose=200;
  pi = 3.1415926535897931;

  if (t<=0) {
    fI = 0;
  } else {
    fI = F*dose*sqrt(MIT/(2.0*pi*CVI2*t^3))*exp(-(t-MIT)^2/(2.0*CVI2*MIT*t));
  }

  C2 = centr/V2;
  C3 = peri/V3;
  d/dt(centr) = fI - CL*C2 - Q*C2 + Q*C3;
  d/dt(peri) = Q*C2 - Q*C3;
"
sys1 <- RxODE(model = ode)

## -----
dat <- invgaussian
mod <- cp ~ C2 + prop(.1)
inits <- c(MIT=190, CVI2=.65, F=.92)
fixPars <- c(CL=.0793, V2=.64, Q=.292, V3=9.63)
ev <- eventTable()
ev$add.sampling(c(0, dat$time))
(fit <- dynmodel(sys1, mod, ev, inits, dat, fixPars))
```

---

 dynmodel.mcmc

*Fit a non-population dynamic model using mcmc*


---

**Description**

Fit a non-population dynamic model using mcmc

**Usage**

```
dynmodel.mcmc(system, model, evTable, inits, data, fixPars = NULL,
  nsim = 500, squared = T, seed = NULL)
```

**Arguments**

system	an RxODE object
model	a list of statistical measurement models
evTable	an Event Table object
inits	initial values of system parameters
data	input data
fixPars	fixed system parameters
nsim	number of mcmc interactions
squared	if parameters be squared during estimation
seed	random number seed

**Author(s)**

Wenping Wang

**Examples**

```
## Not run:

ode <- "
  dose=200;
  pi = 3.1415926535897931;

  if (t<=0) {
    fI = 0;
  } else {
    fI = F*dose*sqrt(MIT/(2.0*pi*CVI2*t^3))*exp(-(t-MIT)^2/(2.0*CVI2*MIT*t));
  }

  C2 = centr/V2;
  C3 = peri/V3;
  d/dt(centr) = fI - CL*C2 - Q*C2 + Q*C3;
  d/dt(peri) = Q*C2 - Q*C3;
```



```

"
sys1 <- RxODE(model = ode)

## -----
dat <- read.table("invgaussian.txt", header=TRUE)
mod <- cp ~ C2 + prop(.1)
inits <- c(MIT=190, CVI2=.65, F=.92)
fixPars <- c(CL=.0793, V2=.64, Q=.292, V3=9.63)
ev <- eventTable()
ev$add.sampling(c(0, dat$time))
(fit <- dynmodel.mcmc(sys1, mod, ev, inits, dat, fixPars))

## End(Not run)

```

focei.eta

*Get the FOCEi theta or eta specification for model.***Description**

Get the FOCEi theta or eta specification for model.

**Usage**

```

focei.eta(object, uif, ...)

## S3 method for class 'nlmixrNlme'
focei.eta(object, ...)

```

**Arguments**

object	Fit object
uif	User interface function or object
...	Other parameters

**Value**

List for the OMGA list in FOCEi

**Author(s)**

Matthew L. Fidler

---

focei.theta	<i>Get the FOCEi theta specification for the model</i>
-------------	--

---

**Description**

Get the FOCEi theta specification for the model

**Usage**

```
focei.theta(object, uif, ...)

## S3 method for class 'nlmixrNlme'
focei.theta(object, uif, ...)
```

**Arguments**

object	Fit object
uif	User interface function or object
...	Other parameters

**Value**

Parameter estimates for Theta

---

foceiControl	<i>Control Options for FOCEi</i>
--------------	----------------------------------

---

**Description**

Control Options for FOCEi

**Usage**

```
foceiControl(sigdig = 4, epsilon = NULL, maxInnerIterations = 1000,
  maxOuterIterations = 5000, n1qn1nsim = NULL, method = c("liblsoda",
  "lsoda", "dop853"), transitAbs = NULL, atol = NULL, rtol = NULL,
  maxstepsOde = 5000L, hmin = 0L, hmax = NULL, hini = 0,
  maxordn = 12L, maxords = 5L, cores, covsInterpolation = c("locf",
  "linear", "nocb", "midpoint"), printInner = 0L, print = 1L,
  printNcol = floor((getOption("width") - 23)/12), scaleTo = 1,
  scaleObjective = 1, derivEps = rep(20 * sqrt(.Machine$double.eps),
  2), derivMethod = c("switch", "forward", "central"),
  derivSwitchTol = NULL, covDerivMethod = c("central", "forward"),
  covMethod = c("r,s", "r", "s", ""), hessEps = 0.001,
  covDerivEps = c(0, 0.001), lbfgsLmm = 7L, lbfgsPgtol = 0,
```

```

lbfgsFctr = NULL, eigen = TRUE, addPosthoc = TRUE,
diagXform = c("sqrt", "log", "identity"), sumProd = FALSE,
optExpression = TRUE, ci = 0.95, useColor = crayon::has_color(),
boundTol = NULL, calcTables = TRUE, noAbort = TRUE,
interaction = TRUE, cholSEtol = (.Machine$double.eps)^(1/3),
cholAccept = 0.001, resetEtaP = 0.1, diagOmegaBoundUpper = 5,
diagOmegaBoundLower = 100, cholSEOpt = FALSE, cholSECov = FALSE,
fo = FALSE, covTryHarder = FALSE, outerOpt = c("bobyqa",
"L-BFGS-B", "lbfgsb3", "nlnminb", "mma", "lbfgsbLG", "slsqp"),
innerOpt = c("n1qn1", "BFGS"), rhobeg = 0.2, rhoend = NULL,
npt = NULL, rel.tol = NULL, x.tol = NULL, eval.max = 4000,
iter.max = 2000, abstol = NULL, reltol = NULL,
resetHessianAndEta = FALSE, stateTrim = Inf, ..., stiff)

```

## Arguments

sigdig	Optimization significant digits. This controls: <ul style="list-style-type: none"> <li>• The tolerance of the inner and outer optimization is <math>10^{-\text{sigdig}}</math></li> <li>• The tolerance of the ODE solvers is <math>10^{-(\text{sigdig}-1)}</math></li> <li>• The tolerance of the boundary check is <math>5 \times 10^{-(\text{sigdig} + 1)}</math></li> <li>• The significant figures that some tables are rounded to.</li> </ul>
epsilon	Precision of estimate for n1qn1 optimization.
maxInnerIterations	Number of iterations for n1qn1 optimization.
maxOuterIterations	Maximum number of L-BFGS-B optimization for outer problem.
n1qn1nsim	Number of function evaluations for n1qn1 optimization.
method	The method for solving ODEs. Currently this supports: <ul style="list-style-type: none"> <li>• "liblsoda" thread safe lsoda. This supports parallel thread-based solving, and ignores user Jacobian specification.</li> <li>• "lsoda" – LSODA solver. Does not support parallel thread-based solving, but allows user Jacobian specification.</li> <li>• "dop853" – DOP853 solver. Does not support parallel thread-based solving nor user Jacobain specification</li> </ul>
transitAbs	boolean indicating if this is a transit compartment absorption
atol	a numeric absolute tolerance (1e-8 by default) used by the ODE solver to determine if a good solution has been achieved; This is also used in the solved linear model to check if prior doses do not add anything to the solution.
rtol	a numeric relative tolerance (1e-6 by default) used by the ODE solver to determine if a good solution has been achieved. This is also used in the solved linear model to check if prior doses do not add anything to the solution.
maxstepsOde	Maximum number of steps for ODE solver.
hmin	The minimum absolute step size allowed. The default value is 0.

hmax	The maximum absolute step size allowed. The default checks for the maximum difference in times in your sampling and events, and uses this value. The value 0 is equivalent to infinite maximum absolute step size.
hini	The step size to be attempted on the first step. The default value is determined by the solver (when hini = 0)
maxordn	The maximum order to be allowed for the nonstiff (Adams) method. The default is 12. It can be between 1 and 12.
maxords	The maximum order to be allowed for the stiff (BDF) method. The default value is 5. This can be between 1 and 5.
cores	Number of cores used in parallel ODE solving. This defaults to the number or system cores determined by <code>rxCores</code> for methods that support parallel solving (ie thread-safe methods like "libsoda").
covsInterpolation	<p>specifies the interpolation method for time-varying covariates. When solving ODEs it often samples times outside the sampling time specified in events. When this happens, the time varying covariates are interpolated. Currently this can be:</p> <ul style="list-style-type: none"> <li>• "linear" interpolation (the default), which interpolates the covariate by solving the line between the observed covariates and extrapolating the new covariate value.</li> <li>• "constant" – Last observation carried forward.</li> <li>• "NOCB" – Next Observation Carried Backward. This is the same method that NONMEM uses.</li> <li>• "midpoint" Last observation carried forward to midpoint; Next observation carried backward to midpoint.</li> </ul>
printInner	Integer representing when the inner step is printed. By default this is 0 or do not print. 1 is print every function evaluation, 5 is print every 5 evaluations.
print	Integer representing when the outer step is printed. When this is 0 or do not print the iterations. 1 is print every function evaluation (default), 5 is print every 5 evaluations.
printNcol	Number of columns to printout before wrapping parameter estimates/gradient
scaleTo	Scale the initial parameter estimate to this value. By default this is 1. When zero or below, no scaling is performed.
scaleObjective	<p>Scale the initial objective function to this value. By default this is 1. When scaleObjective is greater than zero, this scaling is performed by:</p> $\text{scaledObj} = \text{currentObj} / \text{abs}(\text{initialObj}) * \text{scaleObjective}$ <p>Therefore, if the initial objective function is negative, the initial scaled objective function would be negative as well. When scaleObjective is less than zero, no scaling is performed.</p>
derivEps	<p>Central/Forward difference tolerances, which is a vector of relative difference and absolute difference. The central/forward difference step size h is calculated as:</p> $h = \text{abs}(x) * \text{derivEps}[1] + \text{derivEps}[2]$

derivMethod	indicates the method for calculating derivatives of the outer problem. Currently supports "switch", "central" and "forward" difference methods. Switch starts with forward differences. This will switch to central differences when $\text{abs}(\text{delta}(\text{OFV})) \leq \text{derivSwitchTol}$ and switch back to forward differences when $\text{abs}(\text{delta}(\text{OFV})) > \text{derivSwitchTol}$ .
derivSwitchTol	The tolerance to switch forward to central differences.
covDerivMethod	indicates the method for calculating the derivatives while calculating the covariance components (Hessian and S).
covMethod	Method for calculating covariance. In this discussion, R is the Hessian matrix of the objective function. The S matrix is the sum of individual gradient cross-product (evaluated at the individual empirical Bayes estimates). <ul style="list-style-type: none"> <li>• "r,s" Uses the sandwich matrix to calculate the covariance, that is: <math>\text{solve}(R) \%*\% S \%*\% \text{solve}(R)</math></li> <li>• "r" Uses the Hessian matrix to calculate the covariance as <math>2 \%*\% \text{solve}(R)</math></li> <li>• "s" Uses the crossproduct matrix to calculate the covariance as <math>4 \%*\% \text{solve}(S)</math></li> <li>• "" Does not calculate the covariance step.</li> </ul>
hessEps	is a double value representing the epsilon for the Hessian calculation.
covDerivEps	Central/Forward difference tolerances while calculating the covariance matrices. This is a numeric vector of relative difference and absolute difference. The central/forward difference step size h is calculated as: $h = \text{abs}(x) * \text{derivEps}[1] + \text{derivEps}[2]$
lbfgsLmm	An integer giving the number of BFGS updates retained in the "L-BFGS-B" method, It defaults to 7.
lbfgsPgtol	is a double precision variable. On entry pgtol $\geq 0$ is specified by the user. The iteration will stop when: $\max(\backslash   \text{proj } g_i \backslash   i = 1, \dots, n) \leq \text{lbfgsPgtol}$ where $pg_i$ is the $i$ th component of the projected gradient. On exit pgtol is unchanged. This defaults to zero, when the check is suppressed.
lbfgsFactr	Controls the convergence of the "L-BFGS-B" method. Convergence occurs when the reduction in the objective is within this factor of the machine tolerance. Default is $1e10$ , which gives a tolerance of about $2e-6$ , approximately 4 sigdigs. You can check your exact tolerance by multiplying this value by <code>.Machine\$double.eps</code>
eigen	A boolean indicating if eigenvectors are calculated to include a condition number calculation.
addPosthoc	Boolean indicating if posthoc parameters are added to the table output.
diagXform	This is the transformation used on the diagonal of the $\text{chol}(\text{solve}(\text{omega}))$ . This matrix and values are the parameters estimated in FOCEi. The possibilities are: <ul style="list-style-type: none"> <li>• sqrt Estimates the sqrt of the diagonal elements of <math>\text{chol}(\text{solve}(\text{omega}))</math>. This is the default method.</li> <li>• log Estimates the log of the diagonal elements of <math>\text{chol}(\text{solve}(\text{omega}))</math></li> <li>• identity Estimates the diagonal elements without any transformations</li> </ul>

sumProd	Is a boolean indicating if the model should change multiplication to high precision multiplication and sums to high precision sums using the PreciseSums package. By default this is FALSE.
optExpression	Optimize the RxODE expression to speed up calculation. By default this is turned on.
ci	Confidence level for some tables. By default this is 0.95 or 95% confidence.
useColor	Boolean indicating if focei can use ASCII color codes
boundTol	Tolerance for boundary issues.
calcTables	This boolean is to determine if the foceiFit will calculate tables. By default this is TRUE
noAbort	Boolean to indicate if you should abort the FOCEi evaluation if it runs into troubles. (default TRUE)
interaction	Boolean indicate FOCEi should be used (TRUE) instead of FOCE (FALSE)
cholSEtol	tolerance for Generalized Cholesky Decomposition. Defaults to suggested $(.Machine$double.eps)^{(1/3)}$
cholAccept	Tolerance to accept a Generalized Cholesky Decomposition for a R or S matrix.
resetEtaP	represents the p-value for resetting the individual ETA to 0 during optimization (instead of the saved value). The two test statistics used in the z-test are either $\text{chol}(\omega^{-1})$ indicates the ETAs never reset. A p-value of 1 indicates the ETAs always reset.
diagOmegaBoundUpper	This represents the upper bound of the diagonal omega matrix. The upper bound is given by $\text{diag}(\omega) * \text{diagOmegaBoundUpper}$ . If $\text{diagOmegaBoundUpper}$ is 1, there is no upper bound on Omega.
diagOmegaBoundLower	This represents the lower bound of the diagonal omega matrix. The lower bound is given by $\text{diag}(\omega) / \text{diagOmegaBoundUpper}$ . If $\text{diagOmegaBoundLower}$ is 1, there is no lower bound on Omega.
cholSEOpt	Boolean indicating if the generalized Cholesky should be used while optimizing.
cholSECov	Boolean indicating if the generalized Cholesky should be used while calculating the Covariance Matrix.
fo	is a boolean indicating if this is a fo approximation routine.
covTryHarder	If the R matrix is non-positive definite and cannot be corrected to be non-positive definite try estimating the Hessian on the unscaled parameter space.
outerOpt	optimization method for the outer problem
innerOpt	optimization method for the inner problem (not implemented yet.)
rhobeg	Beginning change in parameters for bobyqa algorithm (trust region). By default this is 0.2 or 20 parameters when the parameters are scaled to 1. rhobeg and rhoend must be set to the initial and final values of a trust region radius, so both must be positive with $0 < \text{rhoend} < \text{rhobeg}$ . Typically rhobeg should be about one tenth of the greatest expected change to a variable. Note also that smallest difference $\text{abs}(\text{upper} - \text{lower})$ should be greater than or equal to $\text{rhobeg} * 2$ . If this is not the case then rhobeg will be adjusted.

rhoend	The smallest value of the trust region radius that is allowed. If not defined, then $10^{-(\text{sigdig}-1)}$ will be used.
npt	The number of points used to approximate the objective function via a quadratic approximation for bobyqa. The value of npt must be in the interval $[n+2, (n+1)(n+2)/2]$ where n is the number of parameters in par. Choices that exceed $2*n+1$ are not recommended. If not defined, it will be set to $2*n + 1$
rel.tol	Relative tolerance before nlmixr stops.
x.tol	X tolerance
eval.max	Number of maximum evaluations of the objective function
iter.max	Maximum number of iterations allowed.
abstol	Absolute tolerance for nlmixr
reltol	tolerance for nlmixr
resetHessianAndEta	is a boolean representing if the individual Hessian is reset when ETAs are reset using the option resetEtaP.
stateTrim	Trim state amounts/concentrations to this value.
...	Ignored parameters
stiff	a logical (TRUE by default) indicating whether the ODE system is stiff or not. For stiff ODE systems ( <code>stiff = TRUE</code> ), R <sub>x</sub> ODE uses the LSODA (Livermore Solver for Ordinary Differential Equations) Fortran package, which implements an automatic method switching for stiff and non-stiff problems along the integration interval, authored by Hindmarsh and Petzold (2003). For non-stiff systems ( <code>stiff = FALSE</code> ), R <sub>x</sub> ODE uses DOP853, an explicit Runge-Kutta method of order 8(5, 3) of Dormand and Prince as implemented in C by Hairer and Wanner (1993).

## Details

Note this uses the R's L-BFGS-B in `optim` for the outer problem and the BFGS `n1qn1` with that allows restoring the prior individual Hessian (for faster optimization speed).

By default FOCEi scales the outer problem parameters to 1.0 for the initial parameter estimates and scales the objective function to 1.0, as suggested by the [NAG library](https://www.scipy-lectures.org/advanced/mathematical_optimization/) and [https://www.scipy-lectures.org/advanced/mathematical\\_optimization/](https://www.scipy-lectures.org/advanced/mathematical_optimization/).

However the inner problem is not scaled. Since most eta estimates start near zero, scaling for these parameters do not make sense.

This process of scaling can fix some ill conditioning for the unscaled problem. The covariance step is performed on the unscaled problem, so the condition number of that matrix may not be reflective of the scaled problem's condition-number.

## Author(s)

Matthew L. Fidler

**See Also**

[optim](#)  
[n1qn1](#)  
[rxSolve](#)

foceiFit

*FOCEi fit***Description**

FOCEi fit

**Usage**

```
foceiFit(data, ...)

## S3 method for class 'data.frame'
foceiFit(data, ...)

## S3 method for class 'data.frame0'
foceiFit(data, inits, PKpars, model = NULL,
  pred = NULL, err = NULL, lower = -Inf, upper = Inf,
  fixed = NULL, skipCov = NULL, control = foceiControl(),
  thetaNames = NULL, etaNames = NULL, etaMat = NULL, ...,
  env = NULL)
```

**Arguments**

data	Data to fit; Needs to be RxODE compatible and have DV, AMT, EVID in the dataset.
...	Ignored parameters
inits	Initialization list
PKpars	Pk Parameters function
model	The RxODE model to use
pred	The Prediction function
err	The Error function
lower	Lower bounds
upper	Upper Bounds
fixed	Boolean vector indicating what parameters should be fixed.
skipCov	Boolean vector indicating what parameters should be fixed when calculating covariances
control	FOCEi options Control list. See <a href="#">foceiControl</a>



thetaNames	Names of the thetas to be used in the final object.
etaNames	Eta names to be used in the final object.
etaMat	Eta matrix for initial estimates or final estimates of the ETAs.
env	An environment used to build the FOCEi or nlmixr object.

**Value**

A focei fit or nlmixr fit object  
FOCEi fit object

**Author(s)**

Matthew L. Fidler and Wenping Wang

**Examples**

```
## Not run:
## Comparison to Wang2007 objective functions

mypar2 = function ()
{
  k = theta[1] * exp(eta[1]);
}

mod <- RxODE({
  ipre = 10 * exp(-k * t)
})
pred <- function() ipre

errProp <- function(){
  return(prop(0.1))
}

inits <- list(THTA=c(0.5),
             OMGA=list(ETA[1] ~ 0.04));

w7 <- Wang2007

w7$DV <- w7$Y
w7$EVID <- 0
w7$AMT <- 0

## Wang2007 prop error OBF 39.458 for NONMEM FOCEi, nlmixr matches.
fitPi <- foceiFit(w7, inits, mypar2,mod,pred,errProp,
                 control=foceiControl(maxOuterIterations=0,covMethod=""))

print(fitPi$objective)

## Wang2007 prop error OBF 39.207 for NONMEM FOCE; nlmixr matches.
fitP <- foceiFit(w7, inits, mypar2,mod,pred,errProp,
                 control=foceiControl(maxOuterIterations=0,covMethod=""),
```

```

        interaction=FALSE))

print(fitP$objective)

## Wang 2007 prop error OBF 39.213 for NONMEM FO; nlmixr matches
fitPfo <- foceiFit(w7, inits, mypar2,mod,pred,errProp,
  control=foceiControl(maxOuterIterations=0,covMethod="",
  fo=TRUE))

print(fitPfo$objective)

## Note if you have the etas you can evaluate the likelihood
## of an arbitrary model. It doesn't have to be solved by
## FOCEi

etaMat <- matrix(fitPi$eta[,-1])

fitP2 <- foceiFit(w7, inits, mypar2,mod,pred,errProp, etaMat=etaMat,
  control=foceiControl(maxOuterIterations=0,maxInnerIterations=0,
  covMethod=""))

errAdd <- function(){
  return(add(0.1))
}

## Wang2007 add error of -2.059 for NONMEM FOCE=NONMEM FOCEi;
## nlmixr matches.
fitA <- foceiFit(w7, inits, mypar2,mod,pred,errAdd,
  control=foceiControl(maxOuterIterations=0,covMethod=""))

## Wang2007 add error of 0.026 for NONMEM FO; nlmixr matches

fitAfo <- foceiFit(w7, inits, mypar2,mod,pred,errAdd,
  control=foceiControl(maxOuterIterations=0,fo=TRUE,covMethod=""))

## Extending Wang2007 to add+prop with same dataset
errAddProp <- function(){
  return(add(0.1) + prop(0.1));
}

fitAP <- foceiFit(w7, inits, mypar2,mod,pred,errAddProp,
  control=foceiControl(maxOuterIterations=0,covMethod=""))

## Checking lognormal

errLogn <- function(){
  return(lnorm(0.1));
}

## First run the fit with the nlmixr lnorm error

fitLN <- foceiFit(w7, inits, mypar2,mod,pred,errLogn,

```

```

control=foceiControl(maxOuterIterations=0,covMethod="")

## Next run on the log-transformed space
w72 <- w7; w72$DV <- log(w72$DV)

predL <- function() log(ipre)

fitLN2 <- foceiFit(w72, inits, mypar2,mod,predL,errAdd,
  control=foceiControl(maxOuterIterations=0,covMethod=""))

## Correct the fitLN2's objective function to be on the normal scale
print(fitLN2$objective + 2*sum(w72$DV))

## Note the objective function of the lognormal error is on the normal scale.
print(fitLN2$objective)

mypar2 <- function ()
{
  ka <- exp(THETA[1] + ETA[1])
  cl <- exp(THETA[2] + ETA[2])
  v <- exp(THETA[3] + ETA[3])
}

mod <- RxODE({
  d/dt(depot) <- -ka * depot
  d/dt(center) <- ka * depot - cl / v * center
  cp <- center / v
})

pred <- function() cp

err <- function(){
  return(add(0.1))
}

inits <- list(THETA=c(0.5, -3.2, -1),
  OMGA=list(ETA[1] ~ 1, ETA[2] ~ 2, ETA[3] ~ 1));

## Remove 0 concentrations (should be lloq)

d <- theo_sd[theo_sd$EVID==0 & theo_sd$DV>0 | theo_sd$EVID>0,];

fit1 <- foceiFit(d, inits, mypar2,mod,pred,err)

## you can also fit lognormal data with the objective function on the same scale

err1 <- function(){
  return(lnorm(0.1))
}

fit2 <- foceiFit(d, inits, mypar2,mod,pred,err1)

```

```

## You can also use the standard nlmixr functions to run FOCEi

library(data.table);
datr <- Infusion_1CPT;
datr$EVID<-ifelse(datr$EVID==1,10101,datr$EVID)
datr<-data.table(datr)
datr<-datr[EVID!=2]
datro<-copy(datr)
datIV<-datr[AMT>0][,TIME:=TIME+AMT/RATE][,AMT:=-1*AMT]
datr<-rbind(datr,datIV)

one.compartment.IV.model <- function(){
  ini({ # Where initial conditions/variables are specified
    # '<-' or '=' defines population parameters
    # Simple numeric expressions are supported
    lCl <- 1.6      #log Cl (L/hr)
    lVc <- log(90) #log V (L)
    # 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)
    # Between subject variability estimates are specified by '~'
    # Semicolons are optional
    eta.Vc ~ 0.1 #IIV V
    eta.Cl ~ 0.1; #IIV Cl
  })
  model({ # Where the model is specified
    # The model uses the ini-defined variable names
    Vc <- exp(lVc + eta.Vc)
    Cl <- exp(lCl + eta.Cl)
    # RxODE-style differential equations are supported
    d / dt(centr) = -(Cl / Vc) * centr;
    ## Concentration is calculated
    cp = centr / Vc;
    # And is assumed to follow proportional error estimated by prop.err
    cp ~ prop(prop.err)
  })
}

fitIVp <- nlmixr(one.compartment.IV.model, datr, "focei");

## You can also use the Cox-Box Transform of both sides with
## proportional error (Donse 2016)

one.compartment.IV.model <- function(){
  ini({ # Where initial conditions/variables are specified
    ## '<-' or '=' defines population parameters
    ## Simple numeric expressions are supported
    lCl <- 1.6      #log Cl (L/hr)
    lVc <- log(90) #log V (L)
    ## 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, 0.001)
    lambda <- c(-2, 1, 2)
  })
}

```

```

    zeta <- c(0.1, 1, 10)
    ## Between subject variability estimates are specified by '~'
    ## Semicolons are optional
    eta.Vc ~ 0.1 #IIV V
    eta.Cl ~ 0.1; #IIV Cl
  })
model({ ## Where the model is specified
  ## The model uses the ini-defined variable names
  Vc <- exp(lVc + eta.Vc)
  Cl <- exp(lCl + eta.Cl)
  ## RxODE-style differential equations are supported
  d / dt(centr) = -(Cl / Vc) * centr;
  ## Concentration is calculated
  cp = centr / Vc;
  ## And is assumed to follow proportional error estimated by prop.err
  cp ~ pow(prop.err, zeta) + add(add.err) + boxCox(lambda)
}})

fitIVtbs <- nlmixr(one.compartment.IV.model, datr, "focei")

## If you want to use a variance normalizing distribution with
## negative/positive data you can use the Yeo-Johnson transformation
## as well. This is implemented by the yeoJohnson(lambda) function.
one.compartment.IV.model <- function(){
  ini({ # Where initial conditions/variables are specified
    ## '<-' or '=' defines population parameters
    ## Simple numeric expressions are supported
    lCl <- 1.6 #log Cl (L/hr)
    lVc <- log(90) #log V (L)
    ## 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)
    delta <- c(0.1, 1, 10)
    add.err <- c(0, 0.001)
    lambda <- c(-2, 1, 2)
    ## Between subject variability estimates are specified by '~'
    ## Semicolons are optional
    eta.Vc ~ 0.1 #IIV V
    eta.Cl ~ 0.1; #IIV Cl
  })
  model({ ## Where the model is specified
    ## The model uses the ini-defined variable names
    Vc <- exp(lVc + eta.Vc)
    Cl <- exp(lCl + eta.Cl)
    ## RxODE-style differential equations are supported
    d / dt(centr) = -(Cl / Vc) * centr;
    ## Concentration is calculated
    cp = centr / Vc;
    ## And is assumed to follow proportional error estimated by prop.err
    cp ~ pow(prop.err, delta) + add(add.err) + yeoJohnson(lambda)
  })
}})

fitIVyj <- nlmixr(one.compartment.IV.model, datr, "focei")

```

```
## In addition to using L-BFGS-B for FOCEi (outer problem) you may
## use other optimizers. An example is below
```

```
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)
  })
}
```

```
fit <- nlmixr(one.cmt, theo_sd, "focei", foceiControl(outerOpt="bobyqa"))
```

```
## You may also make an arbitrary optimizer work by adding a wrapper function:
```

```
newuoa0 <- function(par, fn, gr, lower = -Inf, upper = Inf, control = list(), ...){
  ## The function requires par, fn, gr, lower, upper and control
  ##
  ## The par, fn, gr, lower and upper and sent to the function from nlmixr's focei.
  ## The control is the foceiControl list
  ##
  ## The following code modifies the list control list for no warnings.
  .ctl <- control;
  if (is.null(.ctl$npt)) .ctl$npt <- length(par) * 2 + 1
  ## nlmixr will print information this is to suppress the printing from the
  ## optimizer
  .ctl$iprint <- 0L;
  .ctl <- .ctl[names(.ctl) %in% c("npt", "rhobeg", "rhoend", "iprint", "maxfun")];
  ## This does not require gradient and is an unbounded optimization:
  .ret <- minqa::newuoa(par, fn, control=.ctl);
  ## The return statement must be a list with:
  ## - x for the final parameter message
  ## - message for a minimization message
  ## - convergence for a convergence code
  .ret$x <- .ret$par;
  .ret$message <- .ret$msg;
  .ret$convergence <- .ret$ ierr
  ## you can access the final list from the optimization by fit$optReturn
  return(.ret);
}
```

```
fit <- nlmixr(one.cmt, theo_sd, "focei", foceiControl(outerOpt=newuoa0))
```

```
## End(Not run)
```

---

frwd_selection	<i>Forward covariate selection for nlme-base non-linear mixed effect models</i>
----------------	---

---

### Description

Implements forward covariate selection for nlme-based non-linear mixed effect models

### Usage

```
frwd_selection(base, cv, dat, cutoff = 0.05)
```

### Arguments

base	base model
cv	a list of candidate covariate to model parameters
dat	model data
cutoff	significance level

### Value

an nlme object of the final model

### Examples

```
dat <- theo_md
dat$LOGWT <- log(dat$WT)
dat$TG <- (dat$ID < 6) + 0 #dummy covariate

specs <- list(
  fixed=list(lKA=lKA~1, lCL=lCL~1, lV=lV~1),
  random = pdDiag(lKA+lCL~1),
  start=c(0.5, -3.2, -1))
fit0 <- nlme_lin_cmpt(dat, par_model=specs, ncmt=1)
cv <- list(lCL=c("WT", "TG"), lV=c("WT"))
fit <- frwd_selection(fit0, cv, dat)
print(summary(fit))
```

---

gauss.quad	<i>Sets nodes and weigts of Gauss-Hermite quadrature</i>
------------	--

---

**Description**

Sets nodes and weigts of Gauss-Hermite quadrature

**Usage**

```
gauss.quad(n)
```

**Arguments**

n	numbr of nodes
---	----------------

**Value**

a list of nodes and weigts of Gauss-Hermite quadrature

**Examples**

```
gauss.quad(5)
```

---

gen_saem_user_fn	<i>Generate an SAEM model</i>
------------------	-------------------------------

---

**Description**

Generate an SAEM model using either closed-form solutions or ODE-based model definitions

**Usage**

```
gen_saem_user_fn(model, PKpars = attr(model, "default.pars"),
  pred = NULL, inPars = NULL)
```

**Arguments**

model	a compiled SAEM model by gen_saem_user_fn()
PKpars	PKpars function
pred	pred function
inPars	a character vector of parameters to be read from the input dataset

**Details**

Fit a generalized nonlinear mixed-effect model using the Stochastic Approximation Expectation-Maximization (SAEM) algorithm



**Author(s)**

Wenping Wang

getOMEGA

*Calculate gnltmm variance-covariance matrix of random effects***Description**

Calculate variance-covariance matrix of random effects after a gnltmm() fit

**Usage**

getOMEGA(fit)

**Arguments**

fit                    a gnltmm fit object

**Value**

variance-covariance matrix of random effects

gnltmm

*Fit a generalized nonlinear mixed-effect model***Description**

Fit a generalized nonlinear mixed-effect model by adaptive Gaussian quadrature (AQD)

**Usage**

```
gnltmm(llik, data, inits, syspar = NULL, system = NULL,
       diag.xform = c("sqrt", "log", "identity"), ..., control = list())
```

```
gnltmm2(llik, data, inits, syspar = NULL, system = NULL,
        diag.xform = c("sqrt", "log", "identity"), ..., control = list())
```

**Arguments**

llik	log-likelihood function
data	data to be fitted
inits	initial values
syspar	function: calculation of PK parameters
system	an optional (compiled) RxODE object
diag.xform	transformation to diagonal elements of OMEGA during fitting
...	additional options
control	additional optimization options

**Details**

Fit a generalized nonlinear mixed-effect model by adaptive Gaussian quadrature (AGQ)

**Author(s)**

Wenping Wang

**Examples**

```
llik <- function()
{
lp = THETA[1]*x1+THETA[2]*x2+(x1+x2*THETA[3])*ETA[1]
p = pnorm(lp)
dbinom(x, m, p, log=TRUE)
}
inits = list(THTA=c(1,1,1), OMGA=list(ETA[1]~1))

gnlmm(llik, rats, inits, control=list(nAQD=3))
```

```
## Not run:
llik <- function()
{
if (group==1) lp = THETA[1]+THETA[2]*logtstd+ETA[1]
else          lp = THETA[3]+THETA[4]*logtstd+ETA[1]
lam = exp(lp)
dpois(y, lam, log=TRUE)
}
inits = list(THTA=c(1,1,1,1), OMGA=list(ETA[1]~1))
```

```
fit = gnlmm(llik, pump, inits,
control=list(
  reltol.outer=1e-4,
  optim.outer="nmsimplex",
  nAQD=5
)
)
```

```
ode <- "
d/dt(depot) =-KA*depot;
d/dt(centr) = KA*depot - KE*centr;
"
sys1 = RxODE(ode)
```

```
pars <- function()
{
CL = exp(THETA[1] + ETA[1])#; if (CL>100) CL=100
KA = exp(THETA[2] + ETA[2])#; if (KA>20) KA=20
KE = exp(THETA[3])
V = CL/KE
```

```

sig2 = exp(THETA[4])
}
llik <- function() {
pred = centr/V
dnorm(DV, pred, sd=sqrt(sig2), log=TRUE)
}
inits = list(THTA=c(-3.22, 0.47, -2.45, 0))
inits$OMGA=list(ETA[1]~.027, ETA[2]~.37)
#inits$OMGA=list(ETA[1]+ETA[2]~c(.027, .01, .37))
theo <- read.table("theo_md.txt", head=TRUE)

fit = gnlmm(llik, theo, inits, pars, sys1,
control=list(trace=TRUE, nAQD=5))

cv = calcCov(fit)
cbind(fit$par[fit$nsplt==1], sqrt(diag(cv)))

## End(Not run)

```

---

Infusion\_1CPT

*Infusion\_1CPT – 1 Compartment Model Simulated Data from ACOP  
2016*


---

### Description

This is a simulated dataset from the ACOP 2016 poster. All Datasets were simulated with the following methods.

### Usage

```
Infusion_1CPT
```

### Format

A data frame with 7,920 rows and 14 columns

**ID** Simulated Subject ID

**TIME** Simulated Time

**DV** Simulated Dependent Variable

**LNDV** Simulated log(Dependent Variable)

**MDV** Missing DV data item

**AMT** Dosing AMT

**EVID** NONMEM Event ID

**DOSE** Dose

**V** Individual Simulated Volume

**CL** Individual Clearance

**SS** Steady State

**II** Interdose Interval

**SD** Single Dose Flag

**RATE NONMEM** Rate

**CMT** Compartment

### Details

Richly sampled profiles were simulated for 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, for a range of test models: 1- and 2-compartment disposition, with and without 1st order absorption, with either linear or Michaelis-Menten (MM) clearance (MM without steady state dosing). This provided a total of 42 test cases. All inter-individual variabilities (IIVs) were set at 30 and were the same for all models. A similar set of models was previously used to compare NONMEM and Monolix4. Estimates of population parameters, standard errors for fixed-effect parameters, and run times were compared both for closed-form solutions and using ODEs. Additionally, a sparse data estimation situation was investigated where 500 datasets of 600 subjects each (150 per dose) were generated consisting of 4 random time point samples in 24 hours per subject, using a first-order absorption, 1-compartment disposition, linear elimination model.

### Source

Schoemaker R, Xiong Y, Wilkins J, Laveille C, Wang W. nlmixr: an open-source package for pharmacometric modelling in R. ACOP 2016

---

ini	<i>nlmixr ini block handling</i>
-----	----------------------------------

---

### Description

nlmixr ini block handling

### Usage

```
ini(ini, ...)
```

### Arguments

ini	Ini block or nlmixr model object
...	Other arguments parsed by nlmixr

### Value

bounds expression or parsed ui object

**Author(s)**

Matthew L. Fidler

---

instant.stan.extension  
*instant.stan.extension.*

---

**Description**

instant.stan.extension

**Usage**

instant.stan.extension(ode\_str = NULL, covar = NULL)

**Arguments**

ode_str	ODE equations in a string
covar	a character vector of covariates

---

invgaussian                      *Inverse Guassian absorption model*

---

**Description**

Inverse Guassian absorption model

**Usage**

invgaussian

**Format**

A data frame with 32 rows and 6 columns

**time** Time of observation**cp** Concentration**Source**

D'Argenio DZ, Schumitzky A, and Wang X (2009). "ADAPT 5 User's Guide: Pharmacokinetic/Pharmacodynamic Systems Analysis Software".

---

lincmt *Parameters for a linear compartment model for SAEM*

---

### Description

Parameters for a linear compartment model using closed-form solutions and the SAEM algorithm.

### Usage

```
lincmt(ncmt, oral = T, tlag = F, infusion = F,
       parameterization = 1)
```

### Arguments

ncmt	number of compartments
oral	logical, whether oral absorption is true
tlag	logical, whether lag time is present
infusion	logical, whether infusion is true
parameterization	type of parameterization, 1=clearance/volume, 2=micro-constants

### Value

parameters for a linear compartment model

### Author(s)

Wenping Wang

---

lin\_cmt *concentrations from a linear compartment model*

---

### Description

concentrations from a linear compartment model by close-form solutions

### Usage

```
lin_cmt(obs_time, dose_time, dose, Tinf, params, oral, infusion, ncmt,
        parameterization)
```

**Arguments**

obs_time	times at which an observation is desired
dose_time	times at which doses are given
dose	a vector of doses
Tinf	a vector of infusion duration
params	model-appropriate parameters per parameterization
oral	logical, whether oral absorption is true
infusion	logical, whether infusion is true
ncmt	number of compartments
parameterization	type of parameterization, 1=clearance/volumn, 2=micro-constants

**Value**

calculated concentrations

---

metabolite	<i>Parent/Metabolite dataset</i>
------------	----------------------------------

---

**Description**

Parent/Metabolite dataset

**Usage**

metabolite

**Format**

A data frame with 32 rows and 6 columns

**time** Time of observation

**y1** Parent Concentration

**y2** Metabolite Concentration

**Source**

D'Argenio DZ, Schumitzky A, and Wang X (2009). "ADAPT 5 User's Guide: Pharmacokinetic/Pharmacodynamic Systems Analysis Software".

---

model	<i>nlmixr model block</i>
-------	---------------------------

---

**Description**

nlmixr model block

**Usage**

```
model(model, ...)
```

**Arguments**

model	Model specification
...	Other arguments to model object parsed by nlmixr

**Value**

Parsed UI object

**Author(s)**

Matthew L. Fidler

---

nlme_gof	<i>GOF plots for nlme-based mixed-effect models</i>
----------	---

---

**Description**

Generates basic goodness-of-fit plots for nlme-based mixed-effect models

**Usage**

```
nlme_gof(fit, ...)
```

**Arguments**

fit	nlme fit object
...	optional additional arguments



---

nlme_lin_cmpt	<i>Fit nlme-based linear compartment mixed-effect model using closed form solution</i>
---------------	--

---

## Description

'nlme\_lin\_cmpt' fits a linear one to three compartment model with either first order absorption, or i.v. bolus, or i.v. infusion. A user specifies the number of compartments, route of drug administrations, and the model parameterization. 'nlmixr' supports the clearance/volume parameterization and the micro constant parameterization, with the former as the default. Specification of fixed effects, random effects and initial values follows the standard nlme notations.

## Usage

```
nlme_lin_cmpt(dat, parModel, ncmt, oral = TRUE, infusion = FALSE,
  tlag = FALSE, parameterization = 1, parTrans = .getParfn(oral,
  ncmt, parameterization, tlag), mcCores = 1, ...)
```

```
nlmeLinCmpt(dat, parModel, ncmt, oral = TRUE, infusion = FALSE,
  tlag = FALSE, parameterization = 1, parTrans = .getParfn(oral,
  ncmt, parameterization, tlag), mcCores = 1, ...)
```

```
nlmeLinCmt(dat, parModel, ncmt, oral = TRUE, infusion = FALSE,
  tlag = FALSE, parameterization = 1, parTrans = .getParfn(oral,
  ncmt, parameterization, tlag), mcCores = 1, ...)
```

## Arguments

dat	data to be fitted
parModel	list: model for fixed effects, randoms effects and initial values using nlme-type syntax.
ncmt	numerical: number of compartments: 1-3
oral	logical
infusion	logical
tlag	logical
parameterization	numerical: type of parameterization, 1=clearance/volume, 2=micro-constants
parTrans	function: calculation of PK parameters
mcCores	number of cores used in fitting (only for Linux)
...	additional nlme options

## Author(s)

Wenping Wang

**Examples**

```
library(nlmixr)

specs <- list(fixed=lKA+lCL+lV~1, random = pdDiag(lKA+lCL~1), start=c(lKA=0.5, lCL=-3.2, lV=-1))
fit <- nlme_lin_cmpt(theo_md, par_model=specs, ncmt=1, verbose=TRUE)
#plot(augPred(fit,level=0:1))
summary(fit)
```

nlme\_ode

*Fit nlme-based mixed-effect model using ODE implementation***Description**

'nlme\_ode' fits a mixed-effect model described using ordinary differential equation (ODEs). The ODE-definition follows RxODE syntax. Specification of fixed effects, random effects and initial values follows the standard nlme notations.

**Usage**

```
nlme_ode(dat.o, model, parModel, parTrans, response,
         responseScaler = NULL, transitAbs = FALSE, atol = 1e-06,
         rtol = 1e-04, maxsteps = 5000, hmin = 0, hmax = NULL, hini = 0,
         maxordn = 12, maxords = 5, debugODE = FALSE, mcCores = 1, ...)

nlmeOde(dat.o, model, parModel, parTrans, response,
         responseScaler = NULL, transitAbs = FALSE, atol = 1e-06,
         rtol = 1e-04, maxsteps = 5000, hmin = 0, hmax = NULL, hini = 0,
         maxordn = 12, maxords = 5, debugODE = FALSE, mcCores = 1, ...)
```

**Arguments**

dat.o	data to be fitted
model	a string containing the set of ordinary differential equations (ODE) and other expressions defining the changes in the dynamic system. For details, see the sections "Details" and "RxODE Syntax" below.
parModel	list: model for fixed effects, randoms effects and initial values.
parTrans	function: calculation of PK parameters
response	names of the response variable
responseScaler	optional response variable scaler. default is NULL
transitAbs	boolean indicating if this is a transit compartment absorption
atol	a numeric absolute tolerance (1e-8 by default) used by the ODE solver to determine if a good solution has been achieved; This is also used in the solved linear model to check if prior doses do not add anything to the solution.

rtol	a numeric relative tolerance (1e-6 by default) used by the ODE solver to determine if a good solution has been achieved. This is also used in the solved linear model to check if prior doses do not add anything to the solution.
maxsteps	maximum number of (internally defined) steps allowed during one call to the solver. (5000 by default)
hmin	The minimum absolute step size allowed. The default value is 0.
hmax	The maximum absolute step size allowed. The default checks for the maximum difference in times in your sampling and events, and uses this value. The value 0 is equivalent to infinite maximum absolute step size.
hini	The step size to be attempted on the first step. The default value is determined by the solver (when hini = 0)
maxordn	The maximum order to be allowed for the nonstiff (Adams) method. The default is 12. It can be between 1 and 12.
maxords	The maximum order to be allowed for the stiff (BDF) method. The default value is 5. This can be between 1 and 5.
debugODE	a logical if debugging is enabled
mcCores	number of cores used in fitting (only for Linux)
...	additional nlme options

## Details

The ODE-based model specification may be coded inside a character string or in a text file, see Section *RxODE Syntax* below for coding details. An internal RxODE compilation manager object translates the ODE system into C, compiles it, and dynamically loads the object code into the current R session. The call to RxODE produces an object of class RxODE which consists of a list-like structure (closure) with various member functions (see Section *Value* below).

## RxODE Syntax

An RxODE model specification consists of one or more statements terminated by semi-colons, ‘;’, and optional comments (comments are delimited by # and an end-of-line marker). **NB:** Comments are not allowed inside statements.

A block of statements is a set of statements delimited by curly braces, ‘{ ... }’. Statements can be either assignments or conditional if statements. Assignment statements can be either “simple” assignments, where the left hand is an identifier (i.e., variable), or special “time-derivative” assignments, where the left hand specifies the change of that variable with respect to time e.g.,  $d/dt(\text{depot})$ .

Expressions in assignment and ‘if’ statements can be numeric or logical (no character expressions are currently supported). Numeric expressions can include the following numeric operators (+, -, \*, /, ^), and those mathematical functions defined in the C or the R math libraries (e.g., fabs, exp, log, sin). (Note that the modulo operator % is currently not supported.)

Identifiers in an RxODE model specification can refer to:

- state variables in the dynamic system (e.g., compartments in a pharmacokinetic/pharmacodynamic model);

- implied input variable,  $t$  (time),  $podo$  (oral dose, for absorption models), and  $tlast$  (last time point);
- model parameters, ( $k_a$  rate of absorption,  $CL$  clearance, etc.);
- others, as created by assignments as part of the model specification.

Identifiers consist of case-sensitive alphanumeric characters, plus the underscore ‘`_`’ character. **NB:** the dot ‘`.`’ character is **not** a valid character identifier.

The values of these variables at pre-specified time points are saved as part of the fitted/integrated/solved model (see [eventTable](#), in particular its member function `add.sampling` that defines a set of time points at which to capture a snapshot of the system via the values of these variables).

The ODE specification mini-language is parsed with the help of the open source tool *DParser*, Plevyak (2015).

### Author(s)

Wenping Wang, Mathew Fidler

### Examples

```
## Not run:
library(nlmixr)
ode <- "
d/dt(depot) ==-KA*depot;
d/dt(centr) = KA*depot - KE*centr;
"
dat <- read.table(system.file("examples/theo_md.txt", package = "nlmixr"), head=TRUE)
mypar <- function(lKA, lKE, lCL)
{
  KA=exp(lKA)
  KE=exp(lKE)
  CL=exp(lCL)
  V = CL/KE
}

specs <- list(fixed=lKA+lKE+lCL~1, random = pdDiag(lKA+lCL~1),
             start=c(lKA=0.5, lKE=-2.5, lCL=-3.2))

fit <- nlme_ode(theo_md, model=ode, par_model=specs, par_trans=mypar,
              response="centr", response.scaler="V", control=nlmeControl(pnlstol=0.1))

## End(Not run)
```

**Description**

nlmixr is an R package for fitting population pharmacokinetic (PK) and pharmacokinetic-pharmacodynamic (PKPD) models.

**Usage**

```
nlmixr(object, data, est = NULL, control = list(),
       table = tableControl(), ...)

## S3 method for class 'function'
nlmixr(object, data, est = NULL, control = list(),
       table = tableControl(), ...)

## S3 method for class 'nlmixrUI'
nlmixr(object, data, est = NULL, control = list(),
       ...)
```

**Arguments**

object	Fitted object or function specifying the model.
data	Dataset to estimate. Needs to be RxODE compatible in EVIDs.
est	Estimation method
control	Estimation control options. They could be <a href="#">nlmeControl</a> , <a href="#">saemControl</a> or <a href="#">foceiControl</a>
table	A list controlling the table options (i.e. CWRES, NPDE etc). See <a href="#">tableControl</a> .
...	Other parameters

**Details**

The nlmixr generalized function allows common access to the nlmixr estimation routines.

**Value**

Either a nlmixr model or a nlmixr fit object

**nlmixr modeling mini-language****Rationale**

nlmixr estimation routines each have their own way of specifying models. Often the models are specified in ways that are most intuitive for one estimation routine, but do not make sense for another estimation routine. Sometimes, legacy estimation routines like [nlme](#) have their own syntax that is outside of the control of the nlmixr package.

The unique syntax of each routine makes the routines themselves easier to maintain and expand, and allows interfacing with existing packages that are outside of nlmixr (like [nlme](#)). However, a model definition language that is common between estimation methods, and an output object that is uniform, will make it easier to switch between estimation routines and will facilitate interfacing output with external packages like Xpose.

The nlmixr mini-modeling language, attempts to address this issue by incorporating a common language. This language is inspired by both R and NONMEM, since these languages are familiar to many pharmacometricians.

### Initial Estimates and boundaries for population parameters

nlmixr models are contained in a R function with two blocks: ini and model. This R function can be named anything, but is not meant to be called directly from R. In fact if you try you will likely get an error such as Error: could not find function "ini".

The ini model block is meant to hold the initial estimates for the model, and the boundaries of the parameters for estimation routines that support boundaries (note nlmixr's [saem](#) and [nlme](#) do not currently support parameter boundaries).

To explain how these initial estimates are specified we will start with an annotated example:

```
f <- function(){ ## Note the arguments to the function are currently
  ## ignored by nlmixr
  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 <- 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)
  })
  ## The model block will be discussed later
  model({})
}
```

As shown in the above examples:

- Simple parameter values are specified as a R-compatible assignment
- Boundaries may be specified by `c(lower, est, upper)`.
- Like NONMEM, `c(lower, est)` is equivalent to `c(lower, est, Inf)`
- Also like NONMEM, `c(est)` does not specify a lower bound, and is equivalent to specifying the parameter without R's 'c' function.
- The initial estimates are specified on the variance scale, and in analogy with NONMEM, the square roots of the diagonal elements correspond to coefficients of variation when used in the exponential IIV implementation

These parameters can be named almost any R compatible name. Please note that:

- Residual error estimates should be coded as population estimates (i.e. using an '=' or '<-' statement, not a '~').

- Naming variables that start with "\_" are not supported. Note that R does not allow variable starting with "\_" to be assigned without quoting them.
- Naming variables that start with "rx\_" or "nlmixr\_" is not supported since [R<sub>x</sub>ODE](#) and nlmixr use these prefixes internally for certain estimation routines and calculating residuals.
- Variable names are case sensitive, just like they are in R. "CL" is not the same as "Cl".

### Initial Estimates for between subject error distribution (NONMEM's \$OMEGA)

In mixture models, multivariate normal individual deviations from the population parameters are estimated (in NONMEM these are called eta parameters). Additionally the variance/covariance matrix of these deviations is also estimated (in NONMEM this is the OMEGA matrix). These also have initial estimates. In nlmixr these are specified by the '~' operator that is typically used in R for "modeled by", and was chosen to distinguish these estimates from the population and residual error parameters.

Continuing the prior example, we can annotate the estimates for the between subject error distribution

```
f <- 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)
    ## Initial estimate for ka IIV variance
    ## Labels work for single parameters
    eta.ka ~ 0.1 # BSV Ka

    ## 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.cl + eta.vc ~ c(0.1,
                       0.005, 0.1)
    ## Note that labels do not currently work for correlated
    ## parameters. Also do not put comments inside the lower
    ## triangular matrix as this will currently break the model.
  })
  ## The model block will be discussed later
  model({})
}
```

As shown in the above examples:

- Simple variances are specified by the variable name and the estimate separated by '~'.
- Correlated parameters are specified by the sum of the variable labels and then the lower triangular matrix of the covariance is specified on the left handed side of the equation. This is also separated by '~'.

Currently the model syntax does not allow comments inside the lower triangular matrix.

### Model Syntax for ODE based models (NONMEM's \$PK, \$PRED, \$DES and \$ERROR)

Once the initialization block has been defined, you can define a model in terms of the defined variables in the ini block. You can also mix in RxODE blocks into the model.

The current method of defining a nlmixr model is to specify the parameters, and then possibly the RxODE lines:

Continuing describing the syntax with an annotated example:

```
f <- function(){
  ini({
    lCl <- 1.6      #log Cl (L/hr)
    lVc <- log(90)  #log Vc (L)
    lKA <- 0.1     #log Ka (1/hr)
    prop.err <- c(0, 0.2, 1)
    eta.Cl ~ 0.1 ## BSV Cl
    eta.Vc ~ 0.1 ## BSV Vc
    eta.KA ~ 0.1 ## BSV Ka
  })
  model({
    ## First parameters are defined in terms of the initial estimates
    ## parameter names.
    Cl <- exp(lCl + eta.Cl)
    Vc = exp(lVc + eta.Vc)
    KA <- exp(lKA + eta.KA)
    ## After 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 proportional error (estimated by the parameter prop.err)
    cp ~ prop(prop.err)
  })
}
```

A few points to note:

- Parameters are defined before the differential equations. Currently directly defining the differential equations in terms of the population parameters is not supported.
- The differential equations, parameters and error terms are in a single block, instead of multiple sections.
- State names, calculated variables cannot start with either "rx\_" or "nlmixr\_" since these are used internally in some estimation routines.
- Errors are specified using the '~'. Currently you can use either add(parameter) for additive error, prop(parameter) for proportional error or add(parameter1) + prop(parameter2) for additive plus proportional error. You can also specify norm(parameter) for the additive error, since it follows a normal distribution.



- Some routines, like [saem](#) require parameters in terms of `Pop.Parameter + Individual.Deviation.Parameter + Covariate`. The order of these parameters do not matter. This is similar to NONMEM's mu-referencing, though not quite so restrictive.
- The type of parameter in the model is determined by the initial block; Covariates used in the model are missing in the `ini` block. These variables need to be present in the modeling dataset for the model to run.

### Model Syntax for solved PK systems

Solved PK systems are also currently supported by nlmixr with the `'linCmt()'` pseudo-function. An annotated example of a solved system is below:

```
##
f <- function(){
  ini({
    lCl <- 1.6      #log Cl (L/hr)
    lVc <- log(90)  #log Vc (L)
    lKA <- 0.1      #log Ka (1/hr)
    prop.err <- c(0, 0.2, 1)
    eta.Cl ~ 0.1 ## BSV Cl
    eta.Vc ~ 0.1 ## BSV Vc
    eta.KA ~ 0.1 ## BSV Ka
  })
  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 the solved system.
    ##
    ## 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)
  })
}
```

A few things to keep in mind:

- Currently the solved systems support either oral dosing, IV dosing or IV infusion dosing and does not allow mixing the dosing types.
- While RxODE allows mixing of solved systems and ODEs, this has not been implemented in nlmixr yet.
- The solved systems implemented are the one, two and three compartment models with or without first-order absorption. Each of the models support a lag time with a `tlag` parameter.
- In general the linear compartment model figures out the model by the parameter names. nlmixr currently knows about numbered volumes, `Vc/Vp`, Clearances in terms of both `Cl` and `Q/CLD`.

Additionally nlmixr knows about elimination micro-constants (ie K12). Mixing of these parameters for these models is currently not supported.

### Checking model syntax

After specifying the model syntax you can check that nlmixr is interpreting it correctly by using the `nlmixr` function on it.

Using the above function we can get:

```
> nlmixr(f)
## 1-compartment model with first-order absorption in terms of Cl
## Initialization:
#####
Fixed Effects ($theta):
      lCl      lVc      lKA
1.60000 4.49981 0.10000

Omega ($omega):
      [,1] [,2] [,3]
[1,] 0.1 0.0 0.0
[2,] 0.0 0.1 0.0
[3,] 0.0 0.0 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 the solved system.
##
## 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)
```

In general this gives you information about the model (what type of solved system/RxODE), initial estimates as well as the code for the model block.

### Using the model syntax for estimating a model

Once the model function has been created, you can use it and a dataset to estimate the parameters for a model given a dataset.

This dataset has to have RxODE compatible events IDs. Both Monolix and NONMEM use a different dataset description. You may convert these datasets to RxODE-compatible datasets with the [nmDataConvert](#) function. Note that steady state doses are not supported by RxODE, and therefore not supported by the conversion function.

As an example, you can use a simulated rich 1-compartment dataset.

```
d <- Oral_1CPT
d <- d[,names(d) != "SS"];
d <- nmDataConvert(d);
```

Once the data has been converted to the appropriate format, you can use the `nlmixr` function to run the appropriate code.

The method to estimate the model is:

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

Currently `nlme` and `saem` are implemented. For example, to run the above model with `saem`, we could have the following:

```
> f <- function(){
  ini({
    lCl <- 1.6      #log Cl (L/hr)
    lVc <- log(90)  #log Vc (L)
    lKA <- 0.1     #log Ka (1/hr)
    prop.err <- c(0, 0.2, 1)
    eta.Cl ~ 0.1 ## BSV Cl
    eta.Vc ~ 0.1 ## BSV Vc
    eta.KA ~ 0.1 ## BSV Ka
  })
  model({
    ## First parameters are defined in terms of the initial estimates
    ## parameter names.
    Cl <- exp(lCl + eta.Cl)
    Vc = exp(lVc + eta.Vc)
    KA <- exp(lKA + eta.KA)
    ## After 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 proportional error (estimated by the parameter prop.err)
    cp ~ prop(prop.err)
  })
}
> fit.s <- nlmixr(f,d,est="saem",control=saemControl(n.burn=50,n.em=100,print=50));
Compiling RxODE differential equations...done.
c:/Rtools/mingw_64/bin/g++ -I"c:/R/R-34~1.1/include" -DNDEBUG -I"d:/Compiler/gcc-4.9.3/local330/
In file included from c:/R/R-34~1.1/library/RCPPAR~1/include/armadillo:52:0,
      from c:/R/R-34~1.1/library/RCPPAR~1/include/RcppArmadilloForward.h:46,
      from c:/R/R-34~1.1/library/RCPPAR~1/include/RcppArmadillo.h:31,
      from saem3090757b4bd1x64.cpp:1:
c:/R/R-34~1.1/library/RCPPAR~1/include/armadillo_bits/compiler_setup.hpp:474:96: note: #pragma messa
```

```

#pragma message ("WARNING: use of OpenMP disabled; this compiler doesn't support OpenMP 3.0+")
^
c:/Rtools/mingw_64/bin/g++ -shared -s -static-libgcc -o saem3090757b4bd1x64.dll tmp.def saem3090757b4
done.
1:   1.8174   4.6328   0.0553   0.0950   0.0950   0.0950   0.6357
50:   1.3900   4.2039   0.0001   0.0679   0.0784   0.1082   0.1992
100:  1.3894   4.2054   0.0107   0.0686   0.0777   0.1111   0.1981
150:  1.3885   4.2041   0.0089   0.0683   0.0778   0.1117   0.1980
Using sympy via SnakeCharmR
## Calculate ETA-based prediction and error derivatives:
Calculate Jacobian.....done.
Calculate sensitivities.....
done.
## Calculate d(f)/d(eta)
## ...
## done
## ...
## done
The model-based sensitivities have been calculated
Calculating Table Variables...
done

```

The options for `saem` are controlled by `saemControl`. You may wish to make sure the minimization is complete in the case of `saem`. You can do that with `traceplot` which shows the iteration history with the divided by burn-in and EM phases. In this case, the burn in seems reasonable; you may wish to increase the number of iterations in the EM phase of the estimation. Overall it is probably a semi-reasonable solution.

### nlmixr output objects

In addition to unifying the modeling language sent to each of the estimation routines, the outputs currently have a unified structure.

You can see the fit object by typing the object name:

```

> fit.s
-- nlmixr SAEM fit (ODE); OBJF calculated from FOCEi approximation -----
      OBJF      AIC      BIC Log-likelihood Condition Number
62337.09 62351.09 62399.01      -31168.55      82.6086

-- Time (sec; fit.s$time): -----
      saem setup Likelihood Calculation covariance table
elapsed 430.25 31.64      1.19      0 3.44

-- Parameters (fit.s$par.fixed): -----
      Parameter Estimate      SE
lCl    log Cl (L/hr)      1.39 0.0240 1.73      4.01 (3.83, 4.20) 26.6
lVc    log Vc (L)      4.20 0.0256 0.608      67.0 (63.7, 70.4) 28.5
lKA    log Ka (1/hr) 0.00924 0.0323 349.      1.01 (0.947, 1.08) 34.3
prop.err    prop.err    0.198      19.8
Shrink(SD)

```

```
lCl          0.248
lVc          1.09
lKA          4.19
prop.err     1.81
```

No correlations in between subject variability (BSV) matrix  
 Full BSV covariance (fit.s\$omega) or correlation (fit.s\$omega.R; diagonals=SDs)  
 Distribution stats (mean/skewness/kurtosis/p-value) available in fit.s\$shrink

```
-- Fit Data (object fit.s is a modified data.frame): -----
# A tibble: 6,947 x 22
  ID    TIME    DV  PRED    RES    WRES IPRED  IRES  IWRES CPRED  CRES
* <fct> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 1      0.25  205.  198.   6.60  0.0741  189.  16.2  0.434  198.   6.78
2 1      0.5   311.  349. -38.7 -0.261  330. -19.0 -0.291  349. -38.3
3 1      0.75  389.  464. -74.5 -0.398  434. -45.2 -0.526  463. -73.9
# ... with 6,944 more rows, and 11 more variables: CWRES <dbl>, eta.Cl <dbl>,
#   eta.Vc <dbl>, eta.KA <dbl>, depot <dbl>, centr <dbl>, Cl <dbl>, Vc <dbl>,
```

This example shows what is typical printout of a nlmixr fit object. The elements of the fit are:

- The type of fit ([nlme](#), [saem](#), etc)
- Metrics of goodness of fit ([AIC](#), [BIC](#), and [logLik](#)).
  - To align the comparison between methods, the FOCEi likelihood objective is calculated regardless of the method used and used for goodness of fit metrics.
  - This FOCEi likelihood has been compared to NONMEM's objective function and gives the same values (based on the data in Wang 2007)
  - Also note that [saem](#) does not calculate an objective function, and the FOCEi is used as the only objective function for the fit.
  - Even though the objective functions are calculated in the same manner, caution should be used when comparing fits from various estimation routines.
- The next item is the timing of each of the steps of the fit.
  - These can be also accessed by (fit.s\$time).
  - As a mnemonic, the access for this item is shown in the printout. This is true for almost all of the other items in the printout.
- After the timing of the fit, the parameter estimates are displayed (can be accessed by fit.s\$par.fixed)
  - While the items are rounded for R printing, each estimate without rounding is still accessible by the '\$' syntax. For example, the '\$Untransformed' gives the untransformed parameter values.
  - The Untransformed parameter takes log-space parameters and back-transforms them to normal parameters. Not the CIs are listed on the back-transformed parameter space.
  - Proportional Errors are converted to
- Omega block (accessed by fit.s\$omega)
- The table of fit data. Please note:

- A nlmixr fit object is actually a data frame. Saving it as a Rdata object and then loading it without nlmixr will just show the data by itself. Don't worry; the fit information has not vanished, you can bring it back by simply loading nlmixr, and then accessing the data.
- Special access to fit information (like the `$omega`) needs nlmixr to extract the information.
- If you use the `$` to access information, the order of precedence is:
  - \* Fit data from the overall data.frame
  - \* Information about the parsed nlmixr model (via `$uif`)
  - \* Parameter history if available (via `$par.hist` and `$par.hist.stacked`)
  - \* Fixed effects table (via `$par.fixed`)
  - \* Individual differences from the typical population parameters (via `$eta`)
  - \* Fit information from the list of information generated during the post-hoc residual calculation.
  - \* Fit information from the environment where the post-hoc residual were calculated
  - \* Fit information about how the data and options interacted with the specified model (such as estimation options or if the solved system is for an infusion or an IV bolus).
- While the printout may displays the data as a `data.table` object or `tbl` object, the data is NOT any of these objects, but rather a derived data frame.
- Since the object *is* a data.frame, you can treat it like one.

In addition to the above properties of the fit object, there are a few additional that may be helpful for the modeler:

- `$theta` gives the fixed effects parameter estimates (in NONMEM the thetas). This can also be accessed in `fixed.effects` function. Note that the residual variability is treated as a fixed effect parameter and is included in this list.
- `$eta` gives the random effects parameter estimates, or in NONMEM the etas. This can also be accessed in using the `random.effects` function.

### Author(s)

Matthew L. Fidler, Rik Schoemaker

---

nlmixrAugPred

*Augmented Prediction for nlmixr fit*

---

### Description

Augmented Prediction for nlmixr fit

**Usage**

```
nlmixrAugPred(object, ..., covsInterpolation = c("linear", "locf",
  "nocb", "midpoint"), primary = NULL, minimum = NULL,
  maximum = NULL, length.out = 51L)

## S3 method for class 'nlmixrFitData'
augPred(object, primary = NULL,
  minimum = min(primary), maximum = max(primary), length.out = 51,
  ...)
```

**Arguments**

object	Nlmixr fit object
...	some methods for the generic may require additional arguments.
covsInterpolation	specifies the interpolation method for time-varying covariates. When solving ODEs it often samples times outside the sampling time specified in events. When this happens, the time varying covariates are interpolated. Currently this can be: <ul style="list-style-type: none"> <li>• "linear" interpolation (the default), which interpolates the covariate by solving the line between the observed covariates and extrapolating the new covariate value.</li> <li>• "constant" – Last observation carried forward.</li> <li>• "NOCB" – Next Observation Carried Backward. This is the same method that NONMEM uses.</li> <li>• "midpoint" Last observation carried forward to midpoint; Next observation carried backward to midpoint.</li> </ul>
primary	an optional one-sided formula specifying the primary covariate to be used to generate the augmented predictions. By default, if a covariate can be extracted from the data used to generate object (using getCovariate), it will be used as primary.
minimum	an optional lower limit for the primary covariate. Defaults to min(primary).
maximum	an optional upper limit for the primary covariate. Defaults to max(primary).
length.out	an optional integer with the number of primary covariate values at which to evaluate the predictions. Defaults to 51.

**Value**

Stacked data.frame with observations, individual/population predictions.

**Author(s)**

Matthew L. Fidler

nlmixrBounds.eta.names

*Get ETA names*

---

**Description**

Get ETA names

**Usage**

```
nlmixrBounds.eta.names(obj)
```

**Arguments**

obj            UI object

**Value**

ETA names

**Author(s)**

Matthew L. Fidler

---

nlmixrBounds.focei.upper.lower

*Get upper/lower/names for THETAs*

---

**Description**

Get upper/lower/names for THETAs

**Usage**

```
nlmixrBounds.focei.upper.lower(obj, type = c("upper", "lower", "name",  
"err"))
```

**Arguments**

obj            Bounds object  
type           type of object extracted

**Value**

lower/upper/name vector



**Author(s)**

Matthew L. Fidler

---

nlmixrLogo                      *Messages the nlmixr logo...*


---

**Description**

Messages the nlmixr logo...

**Usage**

```
nlmixrLogo(str = "", version = sessionInfo()$otherPkgs$nlmixr$Version)
```

**Arguments**

```
str                      String to print
version                  Version information (by default use package version)
```

**Author(s)**

Matthew L. Fidler

---

nlmixrPred                      *Predict a nlmixr solved system*


---

**Description**

Predict a nlmixr solved system

**Usage**

```
nlmixrPred(object, ..., ipred = FALSE)
```

```
## S3 method for class 'nlmixrFitData'
predict(object, ...)
```

**Arguments**

```
object                  is a either a RxODE family of objects, or a file-name with a RxODE model
                        specification, or a string with a RxODE model specification.
...                      Other arguments including scaling factors for each compartment. This includes
                        S# = numeric will scale a compartment # by a dividing the compartment amount
                        by the scale factor, like NONMEM.
ipred                    Flag to calculate individual predictions. When ipred is TRUE, calculate individ-
                        ual predictions. When ipred is FALSE, set calculate typical population preda-
                        tions. When ipred is NA, calculate both individual and population predictions.
```

nlmixrSim

*Simulate a nlmixr solved system***Description**

This takes the uncertainty in the model parameter estimates and to simulate a number of theoretical studies. Each study simulates a realization of the parameters from the uncertainty in the fixed parameter estimates. In addition the omega and sigma matrices are simulated from the uncertainty in the Omega/Sigma matrices based on the number of subjects and observations the model was based on.

**Usage**

```
nlmixrSim(object, ...)

## S3 method for class 'nlmixrFitData'
rxSolve(object, params = NULL, events = NULL,
  inits = NULL, scale = NULL, covs = NULL, method = c("liblsoda",
  "lsoda", "dop853"), transitAbs = NULL, atol = 1e-06, rtol = 1e-04,
  maxsteps = 5000L, hmin = 0L, hmax = NULL, hini = 0L,
  maxordn = 12L, maxords = 5L, ..., cores,
  covsInterpolation = c("linear", "locf", "nocb", "midpoint"),
  addCov = FALSE, matrix = FALSE, sigma = NULL, sigmaDf = NULL,
  nCoresRV = 1L, sigmaIsChol = FALSE, nDisplayProgress = 10000L,
  amountUnits = NA_character_, timeUnits = "hours", stiff,
  theta = NULL, eta = NULL, addDosing = FALSE,
  updateObject = FALSE, doSolve = TRUE, omega = NULL,
  omegaDf = NULL, omegaIsChol = FALSE, nSub = 1L, thetaMat = NULL,
  thetaDf = NULL, thetaIsChol = FALSE, nStud = 1L, dfSub = 0,
  dfObs = 0, returnType = c("rxSolve", "matrix", "data.frame"),
  seed = NULL, nsim = NULL)

## S3 method for class 'nlmixrFitData'
simulate(object, nsim = 1, seed = NULL, ...)

## S3 method for class 'nlmixrFitData'
solve(a, b, ...)
```

**Arguments**

object	nlmixr object
...	Other arguments sent to rxSolve
params	a numeric named vector with values for every parameter in the ODE system; the names must correspond to the parameter identifiers used in the ODE specification;

events	an eventTable object describing the input (e.g., doses) to the dynamic system and observation sampling time points (see <a href="#">eventTable</a> );
inits	a vector of initial values of the state variables (e.g., amounts in each compartment), and the order in this vector must be the same as the state variables (e.g., PK/PD compartments);
scale	a numeric named vector with scaling for ode parameters of the system. The names must correspond to the parameter identifiers in the ODE specification. Each of the ODE variables will be divided by the scaling factor. For example <code>scale=(center=2)</code> will divide the center ODE variable by 2.
covs	a matrix or dataframe the same number of rows as the sampling points defined in the events eventTable. This is for time-varying covariates.
method	The method for solving ODEs. Currently this supports: <ul style="list-style-type: none"> <li>• "liblsoda" thread safe lsoda. This supports parallel thread-based solving, and ignores user Jacobian specification.</li> <li>• "lsoda" – LSODA solver. Does not support parallel thread-based solving, but allows user Jacobian specification.</li> <li>• "dop853" – DOP853 solver. Does not support parallel thread-based solving nor user Jacobian specification</li> </ul>
transitAbs	boolean indicating if this is a transit compartment absorption
atol	a numeric absolute tolerance (1e-8 by default) used by the ODE solver to determine if a good solution has been achieved; This is also used in the solved linear model to check if prior doses do not add anything to the solution.
rtol	a numeric relative tolerance (1e-6 by default) used by the ODE solver to determine if a good solution has been achieved. This is also used in the solved linear model to check if prior doses do not add anything to the solution.
maxsteps	maximum number of (internally defined) steps allowed during one call to the solver. (5000 by default)
hmin	The minimum absolute step size allowed. The default value is 0.
hmax	The maximum absolute step size allowed. The default checks for the maximum difference in times in your sampling and events, and uses this value. The value 0 is equivalent to infinite maximum absolute step size.
hini	The step size to be attempted on the first step. The default value is determined by the solver (when hini = 0)
maxordn	The maximum order to be allowed for the nonstiff (Adams) method. The default is 12. It can be between 1 and 12.
maxords	The maximum order to be allowed for the stiff (BDF) method. The default value is 5. This can be between 1 and 5.
cores	Number of cores used in parallel ODE solving. This defaults to the number of system cores determined by <a href="#">rxCores</a> for methods that support parallel solving (ie thread-safe methods like "liblsoda").
covsInterpolation	specifies the interpolation method for time-varying covariates. When solving ODEs it often samples times outside the sampling time specified in events. When this happens, the time varying covariates are interpolated. Currently this can be:

- "linear" interpolation (the default), which interpolates the covariate by solving the line between the observed covariates and extrapolating the new covariate value.
- "constant" – Last observation carried forward.
- "NOCB" – Next Observation Carried Backward. This is the same method that NONMEM uses.
- "midpoint" Last observation carried forward to midpoint; Next observation carried backward to midpoint.

addCov	A boolean indicating if covariates should be added to the output matrix or data frame. By default this is disabled.
matrix	A boolean indicating if a matrix should be returned instead of the RxODE's solved object.
sigma	Named sigma covariance or Cholesky decomposition of a covariance matrix. The names of the columns indicate parameters that are simulated. These are simulated for every observation in the solved system.
sigmaDf	Degrees of freedom of the sigma t-distribution. By default it is equivalent to Inf, or a normal distribution.
nCoresRV	Number of cores used for the simulation of the sigma variables. By default this is 1. This uses the package <code>rmvn</code> and <code>rmvt</code> . To reproduce the results you need to run on the same platform with the same number of cores. This is the reason this is set to be one, regardless of what the number of cores are used in threaded ODE solving.
sigmaIsChol	Boolean indicating if the sigma is in the Cholesky decomposition instead of a symmetric covariance
nDisplayProgress	An integer indicating the minimum number of c-based solves before a progress bar is shown. By default this is 10,000.
amountUnits	This supplies the dose units of a data frame supplied instead of an event table. This is for importing the data as an RxODE event table.
timeUnits	This supplies the time units of a data frame supplied instead of an event table. This is for importing the data as an RxODE event table.
stiff	a logical (TRUE by default) indicating whether the ODE system is stiff or not. For stiff ODE systems ( <code>stiff = TRUE</code> ), RxODE uses the LSODA (Livermore Solver for Ordinary Differential Equations) Fortran package, which implements an automatic method switching for stiff and non-stiff problems along the integration interval, authored by Hindmarsh and Petzold (2003). For non-stiff systems ( <code>stiff = FALSE</code> ), RxODE uses DOP853, an explicit Runge-Kutta method of order 8(5, 3) of Dormand and Prince as implemented in C by Hairer and Wanner (1993).
theta	A vector of parameters that will be named THETA[#] and added to parameters
eta	A vector of parameters that will be named ETA[#] and added to parameters
addDosing	Boolean indicating if the solve should add RxODE evid and amt columns. This will also include dosing information and estimates at the doses. By default, RxODE only includes estimates at the observations. (default FALSE).

updateObject	This is an internally used flag to update the RxODE solved object (when supplying an RxODE solved object) as well as returning a new object. You probably should not modify it's FALSE default unless you are willing to have unexpected results.
doSolve	Internal flag. By default this is TRUE, when FALSE a list of solving options is returned.
omega	Named omega matrix.
omegaDf	The degrees of freedom of a t-distribution for simulation. By default this is NULL which is equivalent to Inf degrees, or to simulate from a normal distribution instead of a t-distribution.
omegaIsChol	Indicates if the omega supplied is a Cholesky decomposed matrix instead of the traditional symmetric matrix.
nSub	Number between subject variabilities (ETAs) simulated for every realization of the parameters.
thetaMat	Named theta matrix.
thetaDf	The degrees of freedom of a t-distribution for simulation. By default this is NULL which is equivalent to Inf degrees, or to simulate from a normal distribution instead of a t-distribution.
thetaIsChol	Indicates if the theta supplied is a Cholesky decomposed matrix instead of the traditional symmetric matrix.
nStud	Number virtual studies to characterize uncertainty in estimated parameters.
dfSub	Degrees of freedom to sample the between subject variability matrix from the inverse Wishart distribution (scaled) or scaled inverse chi squared distribution.
dfObs	Degrees of freedom to sample the unexplained variability matrix from the inverse Wishart distribution (scaled) or scaled inverse chi squared distribution.
returnType	This tells what type of object is returned. The currently supported types are: <ul style="list-style-type: none"> <li>• "rxSolve" (default) will return a reactive data frame that can change easily change different pieces of the solve and update the data frame. This is the currently standard solving method in RxODE, is used for rxSolve(object, ...), solve(object, ...),</li> <li>• "data.frame" – returns a plain, non-reactive data frame; Currently very slightly Faster than returnType="matrix"</li> <li>• "matrix" – returns a plain matrix with column names attached to the solved object. This is what is used object\$run as well as object\$solve</li> </ul>
seed	an object specifying if and how the random number generator should be initialized
nsim	represents the number of simulations. For RxODE, if you supply single subject event tables (created with eventTable)
a	when using solve, this is equivalent to the object argument. If you specify object later in the argument list it overwrites this parameter.
b	when using solve, this is equivalent to the params argument. If you specify params as a named argument, this overwrites the output

nlmixrUI.focei.fixed *Get parameters that are fixed*

---

**Description**

Get parameters that are fixed

**Usage**

```
nlmixrUI.focei.fixed(obj)
```

**Arguments**

obj                    UI object

**Value**

logical vector of fixed THETA parameters

**Author(s)**

Matthew L. Fidler

---

nlmixrUI.focei.inits *Get the FOCEi initializations*

---

**Description**

Get the FOCEi initializations

**Usage**

```
nlmixrUI.focei.inits(obj)
```

**Arguments**

obj                    UI object

**Value**

list with FOCEi style initializations

**Author(s)**

Matthew L. Fidler

---

nlmixrUI.nlme.specs    *Create the nlme specs list for nlmixr nlme solving*

---

**Description**

Create the nlme specs list for nlmixr nlme solving

**Usage**

```
nlmixrUI.nlme.specs(object, mu.type = c("thetas", "covariates", "none"))
```

**Arguments**

object	UI object
mu.type	is the mu-referencing type of model hat nlme will be using.

**Value**

specs list for nlme

**Author(s)**

Matthew L. Fidler

---

nlmixrUI.rxode.pred    *Return RxODE model with predictions appended*

---

**Description**

Return RxODE model with predictions appended

**Usage**

```
nlmixrUI.rxode.pred(object)
```

**Arguments**

object	UI object
--------	-----------

**Value**

String or NULL if RxODE is not specified by UI.

**Author(s)**

Matthew L. Fidler

nlmixrUI.saem.ares     *Get initial estimate for ares SAEM.*

---

**Description**

Get initial estimate for ares SAEM.

**Usage**

```
nlmixrUI.saem.ares(obj)
```

**Arguments**

obj                    UI model

**Value**

SAEM model\$ares spec

**Author(s)**

Matthew L. Fidler

---

nlmixrUI.saem.bres     *Get initial estimate for bres SAEM.*

---

**Description**

Get initial estimate for bres SAEM.

**Usage**

```
nlmixrUI.saem.bres(obj)
```

**Arguments**

obj                    UI model

**Value**

SAEM model\$ares spec

**Author(s)**

Matthew L. Fidler



---

`nlmixrUI.saem.distribution`  
*Get SAEM distribution*

---

**Description**

Get SAEM distribution

**Usage**

`nlmixrUI.saem.distribution(obj)`

**Arguments**

`obj`            UI object

**Value**

Character of distribution

**Author(s)**

Matthew L. Fidler

---

`nlmixrUI.saem.eta.trans`  
*Get the eta->eta.trans for SAEM*

---

**Description**

Get the eta->eta.trans for SAEM

**Usage**

`nlmixrUI.saem.eta.trans(obj)`

**Arguments**

`obj`            ui object

**Value**

list of eta to eta.trans

**Author(s)**

Matthew L. Fidler

nlmixrUI.saem.fit      *Generate saem.fit user function.*

---

**Description**

Generate saem.fit user function.

**Usage**

```
nlmixrUI.saem.fit(obj)
```

**Arguments**

obj                      UI object

**Value**

saem user function

**Author(s)**

Matthew L. Fidler

---

nlmixrUI.saem.fixed      *Get parameters that are fixed for SAEM*

---

**Description**

Get parameters that are fixed for SAEM

**Usage**

```
nlmixrUI.saem.fixed(obj)
```

**Arguments**

obj                      UI object

**Value**

List of parameters that are fixed.

**Author(s)**

Matthew L. Fidler

---

nlmixrUI.saem.init      *Get saem initialization list*

---

**Description**

Get saem initialization list

**Usage**

```
nlmixrUI.saem.init(obj)
```

**Arguments**

obj                      nlmixr UI object

**Value**

Return SAEM inits list.

**Author(s)**

Matthew L. Fidler

---

nlmixrUI.saem.init.omega  
*SAEM's init\$omega*

---

**Description**

SAEM's init\$omega

**Usage**

```
nlmixrUI.saem.init.omega(obj, names = FALSE)
```

**Arguments**

obj                      nlmixr UI object  
names                    When TRUE return the omega names. By default this is FALSE.

**Value**

Return initial matrix

**Author(s)**

Matthew L. Fidler

---

```
nlmixrUI.saem.init.theta
```

*Generate SAEM initial estimates for THETA.*

---

**Description**

Generate SAEM initial estimates for THETA.

**Usage**

```
nlmixrUI.saem.init.theta(obj)
```

**Arguments**

obj                    nlmixr UI object

**Value**

SAEM theta initial estimates

**Author(s)**

Matthew L. Fidler

---

```
nlmixrUI.saem.log.eta
```

*Get model\$log.eta for SAEM*

---

**Description**

Get model\$log.eta for SAEM

**Usage**

```
nlmixrUI.saem.log.eta(obj)
```

**Arguments**

obj                    UI model

**Value**

SAEM model\$log.eta

**Author(s)**

Matthew L. Fidler

---

`nlmixrUI.saem.model`    *Generate SAEM model list*

---

**Description**

Generate SAEM model list

**Usage**

`nlmixrUI.saem.model(obj)`

**Arguments**

`obj`                    `nlmixr` UI object

**Value**

SAEM model list

**Author(s)**

Matthew L. Fidler

---

`nlmixrUI.saem.model.omega`  
*Get the SAEM model Omega*

---

**Description**

Get the SAEM model Omega

**Usage**

`nlmixrUI.saem.model.omega(obj)`

**Arguments**

`obj`                    UI model

**Value**

SAEM model\$omega spec

**Author(s)**

Matthew L. Fidler

nlmixrUI.saem.res.mod *Get the SAEM model\$res.mod code*

---

**Description**

Get the SAEM model\$res.mod code

**Usage**

```
nlmixrUI.saem.res.mod(obj)
```

**Arguments**

obj                   UI model

**Value**

SAEM model\$res.mod spec

**Author(s)**

Matthew L. Fidler

---

nlmixrUI.saem.res.name  
*Get error names for SAEM*

---

**Description**

Get error names for SAEM

**Usage**

```
nlmixrUI.saem.res.name(obj)
```

**Arguments**

obj                   SAEM user interface function.

**Value**

Names of error estimates for SAEM

**Author(s)**

Matthew L. Fidler

---

`nlmixrUI.saem.theta.name`

*Get THETA names for nlmixr's SAEM*

---

**Description**

Get THETA names for nlmixr's SAEM

**Usage**

`nlmixrUI.saem.theta.name(uif)`

**Arguments**

`uif`                    nlmixr UI object

**Value**

SAEM theta names

**Author(s)**

Matthew L. Fidler

---

`nlmixrUI.theta.pars`

*Get the Parameter function with THETA/ETAs defined*

---

**Description**

Get the Parameter function with THETA/ETAs defined

**Usage**

`nlmixrUI.theta.pars(obj)`

**Arguments**

`obj`                    UI object

**Value**

parameters function defined in THETA[#] and ETA[#]s.

**Author(s)**

Matthew L. Fidler

---

nlmixrUILinCmt	<i>Build linear solved information based on defined parameters.</i>
----------------	---

---

**Description**

Build linear solved information based on defined parameters.

**Usage**

```
nlmixrUILinCmt(lhs)
```

**Arguments**

lhs	List of defined parameters
-----	----------------------------

**Value**

List containing the translation parameters (extra lines) to give what nlmixr expects, and model properties (parameterization, ncmt, oral, and tlag). The infusion needs to be guessed from the data and is not included in this function's output

**Author(s)**

Matthew L. Fidler

---

nlmixrValidate	<i>Validate nlmixr</i>
----------------	------------------------

---

**Description**

This allows easy validation/qualification of nlmixr by running the testing suite on your system.

**Usage**

```
nlmixrValidate(full = FALSE)
```

**Arguments**

full	Should a full validation be performed? (By default FALSE)
------	---

**Author(s)**

Matthew L. Fidler



---

nlmixrVersion	<i>Dispaly nlmixr's version</i>
---------------	---------------------------------

---

**Description**

Dispaly nlmixr's version

**Usage**

```
nlmixrVersion()
```

**Author(s)**

Matthew L. Fidler

---

nlmixr_fit	<i>Fit a nlmixr model</i>
------------	---------------------------

---

**Description**

Fit a nlmixr model

**Usage**

```
nlmixr_fit(uif, data, est = NULL, control = list(), ...,
  sum.prod = FALSE, table = tableControl())
```

**Arguments**

uif	Parsed nlmixr model (by <code>nlmixr(mod.fn)</code> ).
data	Dataset to estimate. Needs to be RxODE compatible in EVIDs.
est	Estimation method
control	Estimation control options. They could be <a href="#">nlmeControl</a> , <a href="#">saemControl</a> or <a href="#">foceiControl</a>
...	Parameters passed to estimation method.
sum.prod	Take the RxODE model and use more precise products/sums. Increases solving accuracy and solving time.
table	A list controlling the table options (i.e. CWRES, NPDE etc). See <a href="#">tableControl</a> .

**Value**

nlmixr fit object

**Author(s)**

Matthew L. Fidler

---

nmDataConvert	<i>Convert NONMEM/Monolix data to RxODE/nlmixr style data</i>
---------------	---

---

**Description**

Convert NONMEM/Monolix data to RxODE/nlmixr style data

**Usage**

```
nmDataConvert(nonmem.data)
```

**Arguments**

nonmem.data      NONMEM dataset

**Value**

RxODE compatible data.

**Author(s)**

Yuan Xiong and Matthew L. Fidler

---

nmsimplex	<i>Nelder-Mead of simplex search</i>
-----------	--------------------------------------

---

**Description**

Nelder-Mead of simplex search

**Usage**

```
nmsimplex(start, fr, rho = NULL, control = list())
```

**Arguments**

start	initials
fr	objective function
rho	evaluation environment
control	additional optimization options

**Value**

a list of ...

---

Oral_1CPT	<i>Oral_1CPT – 1 Compartment Model with Oral Absorption Simulated Data from ACOP 2016</i>
-----------	---

---

### Description

This is a simulated dataset from the ACOP 2016 poster. All Datasets were simulated with the following methods.

### Usage

Oral\_1CPT

### Format

A data frame with 7,920 rows and 15 columns

**ID** Simulated Subject ID  
**TIME** Simulated Time  
**DV** Simulated Dependent Variable  
**LNDV** Simulated log(Dependent Variable)  
**MDV** Missing DV data item  
**AMT** Dosing AMT  
**EVID** NONMEM Event ID  
**DOSE** Dose  
**V** Individual Simulated Volume  
**CL** Individual Clearance  
**KA** Individual Ka  
**SS** Steady State  
**II** Interdose Interval  
**SD** Single Dose Flag  
**CMT** Compartment

### Details

Richly sampled profiles were simulated for 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, for a range of test models: 1- and 2-compartment disposition, with and without 1st order absorption, with either linear or Michaelis-Menten (MM) clearance (MM without steady state dosing). This provided a total of 42 test cases. All inter-individual variabilities (IIVs) were set at 30 were the same for all models. A similar set of models was previously used to compare NONMEM and Monolix4. Estimates of population parameters, standard errors for fixed-effect parameters, and run times were compared both for closed-form solutions and using ODEs. Additionally, a sparse data estimation situation was investigated where 500 datasets of 600 subjects each (150 per dose) were generated consisting of 4 random time point samples in 24 hours per subject, using a first-order absorption, 1-compartment disposition, linear elimination model.

**Source**

Schoemaker R, Xiong Y, Wilkins J, Laveille C, Wang W. nlmixr: an open-source package for pharmacometric modelling in R. ACOP 2016

---

plot.dyn.ID	<i>Plot of a non-population dynamic model fit</i>
-------------	---

---

**Description**

Plot of a non-population dynamic model fit

**Usage**

```
## S3 method for class 'dyn.ID'
plot(x, ...)
```

**Arguments**

x	a dynamodel fit object
...	additional arguments

---

plot.dyn.mcmc	<i>Plot of a non-population dynamic model fit using mcmc</i>
---------------	--

---

**Description**

Plot of a non-population dynamic model fit using mcmc

**Usage**

```
## S3 method for class 'dyn.mcmc'
plot(x, ...)
```

**Arguments**

x	a dynmodel fit object
...	additional arguments

---

plot.nlmixrFitData      *Plot a.nlmixr data object*

---

**Description**

Plot some standard goodness of fit plots for the focei fitted object

**Usage**

```
## S3 method for class '.nlmixrFitData'  
plot(x, ...)
```

**Arguments**

x                    a focei fit object  
...                  additional arguments

**Author(s)**

Wenping Wang & Matthew Fidler

---

plot.saemFit              *Plot an SAEM model fit*

---

**Description**

Plot an SAEM model fit

**Usage**

```
## S3 method for class 'saemFit'  
plot(x, ...)
```

**Arguments**

x                    a saemFit object  
...                  others

**Value**

a list

---

prediction	<i>Prediction after a gnlmm fit</i>
------------	-------------------------------------

---

### Description

Generate predictions after a generalized non-linear mixed effect model fit

### Usage

```
prediction(fit, pred, data = NULL, mc.cores = 1)
```

### Arguments

fit	a gnlmm fit object
pred	prediction function
data	new data
mc.cores	number of cores (for Linux only)

### Value

observed and predicted

### Examples

```
## Not run:

ode <- "
d/dt(depot) =-KA*depot;
d/dt(centr) = KA*depot - KE*centr;
"
sys1 = RxODE(ode)

pars <- function()
{
CL = exp(THETA[1] + ETA[1])#; if (CL>100) CL=100
KA = exp(THETA[2] + ETA[2])#; if (KA>20) KA=20
KE = exp(THETA[3])
V = CL/KE
sig2 = exp(THETA[4])
}
llik <- function() {
pred = centr/V
dnorm(DV, pred, sd=sqrt(sig2), log=TRUE)
}
inits = list(THTA=c(-3.22, 0.47, -2.45, 0))
inits$OMGA=list(ETA[1]~.027, ETA[2]~.37)
theo <- read.table("theo_md.txt", head=TRUE)
```

```

fit = glm(llik, theo, inits, pars, sys1,
control=list(trace=TRUE, nAQD=5))

pred = function() {
  pred = centr/V
}

s = prediction(fit, pred)
plot(s$p, s$dv); abline(0,1,col="red")

## End(Not run)

```

---

```

print.dyn.ID          Print a non-population dynamic model fit object

```

---

**Description**

Print a non-population dynamic model fit object

**Usage**

```

## S3 method for class 'dyn.ID'
print(x, ...)

```

**Arguments**

x	a dynmodel fit object
...	additional arguments

---

```

print.dyn.mcmc       Summary of a non-population dynamic model fit using mcmc

```

---

**Description**

Summary of a non-population dynamic model fit using mcmc

**Usage**

```

## S3 method for class 'dyn.mcmc'
print(x, ...)

```

**Arguments**

x	a dynmodel fit object
...	additional arguments

print.gnlmm.fit      *Print a gnlmm fit*

---

**Description**

Print a generalized non-linear mixed effect model fit

**Usage**

```
## S3 method for class 'gnlmm.fit'  
print(x, ...)
```

**Arguments**

x                    a gnlmm fit object  
...                   additional arguments

---

print.nlmixrUI      *Print UI function*

---

**Description**

Print UI function

**Usage**

```
## S3 method for class 'nlmixrUI'  
print(x, ...)
```

**Arguments**

x                    UI function  
...                   other arguments

**Author(s)**

Matthew L. Fidler



---

print.saemFit	<i>Print an SAEM model fit summary</i>
---------------	--

---

**Description**

Print an SAEM model fit summary

**Usage**

```
## S3 method for class 'saemFit'
print(x, ...)
```

**Arguments**

x	a saemFit object
...	others

**Value**

a list

---

pump	<i>Pump failure example dataset</i>
------	-------------------------------------

---

**Description**

This records the number of failures and operation time for groups of 10 pumps.

**Usage**

pump

**Format**

A data frame with 10 rows and 5 columns

**y** Number of pump failures

**t** Failure Time

**group** Continuous Operation (=1) or Intermittent Operation(=2)

**ID** ID for group of 10 pumps

**logstd** Centered operation times

**Source**

[https://support.sas.com/documentation/cdl/en/statug/63033/HTML/default/viewer.htm#statug\\_nlmixed\\_sect040.htm](https://support.sas.com/documentation/cdl/en/statug/63033/HTML/default/viewer.htm#statug_nlmixed_sect040.htm)

**References**

Gaver, D. P. and O'Muircheartaigh, I. G. (1987), "Robust Empirical Bayes Analysis of Event Rates," *Technometrics*, 29, 1-15.

---

rats

*Pregnant Rat Diet Experiment*

---

**Description**

16 pregnant rats have a control diet, and 16 have a chemically treated diet. The litter size for each rat is recorded after 4 and 21 days. This dataset is used in the SAS Probit-model with binomial data, and saved in the nlmixr package as rats.

**Usage**

rats

**Format**

A data frame with 32 rows and 6 columns

**trt** Treatment; c= control diet; t=treated diet

**m** Litter size after 4 days

**x** Litter size after 21 days

**x1** Indicator for trt=c

**x2** Indicator for trt=t

**ID** Rat ID

**Source**

[https://support.sas.com/documentation/cdl/en/statug/63033/HTML/default/viewer.htm#statug\\_nlmixed\\_sect040.htm](https://support.sas.com/documentation/cdl/en/statug/63033/HTML/default/viewer.htm#statug_nlmixed_sect040.htm)

## References

- Weil, C.S., 1970. Selection of the valid number of sampling units and a consideration of their combination in toxicological studies involving reproduction, teratogenesis or carcinogenesis. *Fd. Cosmet. Toxicol.* 8, 177-182.
- Williams, D.A., 1975. The analysis of binary responses from toxicological experiments involving reproduction and teratogenicity. *Biometrics* 31, 949-952.
- McCulloch, C. E. (1994), "Maximum Likelihood Variance Components Estimation for Binary Data," *Journal of the American Statistical Association*, 89, 330 - 335.
- Ochi, Y. and Prentice, R. L. (1984), "Likelihood Inference in a Correlated Probit Regression Model," *Biometrika*, 71, 531-543.

---

residuals.nlmixrFitData

*Extract residuals from the FOCEI fit*

---

## Description

Extract residuals from the FOCEI fit

## Usage

```
## S3 method for class '.nlmixrFitData'  
residuals(object, ..., type = c("ires", "res",  
  "iwres", "wres", "cwres", "cpred", "cres"))
```

## Arguments

object	focei.fit object
...	Additional arguments
type	Residuals type fitted.

## Value

residuals

## Author(s)

Matthew L. Fidler

---

saem.cleanup	<i>Cleanup saem_fit environment by removing dll after the object is no longer used by R.</i>
--------------	--

---

**Description**

Cleanup saem\_fit environment by removing dll after the object is no longer used by R.

**Usage**

```
## S3 method for class 'cleanup'
saem(env)
```

**Arguments**

env Environment where cleanup needs to occur.

**Author(s)**

Matthew L. Fidler

---

saem.fit	<i>Fit an SAEM model</i>
----------	--------------------------

---

**Description**

Fit an SAEM model using either closed-form solutions or ODE-based model definitions

**Usage**

```
saem.fit(model, data, inits, PKpars = NULL, pred = NULL,
  covars = NULL, mcmc = list(niter = c(200, 300), nmc = 3, nu = c(2, 2,
  2)), ODEopt = list(atol = 1e-06, rtol = 1e-04, stiff = 1, transitAbs =
  0), distribution = c("normal", "poisson", "binomial"), seed = 99)
```

```
saem(model, data, inits, PKpars = NULL, pred = NULL, covars = NULL,
  mcmc = list(niter = c(200, 300), nmc = 3, nu = c(2, 2, 2)),
  ODEopt = list(atol = 1e-06, rtol = 1e-04, stiff = 1, transitAbs = 0),
  distribution = c("normal", "poisson", "binomial"), seed = 99)
```

```
## S3 method for class 'fit.nlmixr.ui.nlmf'
saem(model, data, inits, PKpars = NULL,
  pred = NULL, covars = NULL, mcmc = list(niter = c(200, 300), nmc =
  3, nu = c(2, 2, 2)), ODEopt = list(atol = 1e-06, rtol = 1e-04, stiff =
  1, transitAbs = 0), distribution = c("normal", "poisson", "binomial"),
  seed = 99)
```

```

## S3 method for class 'fit.function'
saem(model, data, inits, PKpars = NULL,
      pred = NULL, covars = NULL, mcmc = list(niter = c(200, 300), nmc =
        3, nu = c(2, 2, 2)), ODEopt = list(atol = 1e-06, rtol = 1e-04, stiff =
        1, transitAbs = 0), distribution = c("normal", "poisson", "binomial"),
      seed = 99)

## S3 method for class 'fit.nlmixrUI'
saem(model, data, inits, PKpars = NULL,
      pred = NULL, covars = NULL, mcmc = list(niter = c(200, 300), nmc =
        3, nu = c(2, 2, 2)), ODEopt = list(atol = 1e-06, rtol = 1e-04, stiff =
        1, transitAbs = 0), distribution = c("normal", "poisson", "binomial"),
      seed = 99)

## S3 method for class 'fit.RxODE'
saem(model, data, inits, PKpars = NULL,
      pred = NULL, covars = NULL, mcmc = list(niter = c(200, 300), nmc =
        3, nu = c(2, 2, 2)), ODEopt = list(atol = 1e-06, rtol = 1e-04, stiff =
        1, transitAbs = 0), distribution = c("normal", "poisson", "binomial"),
      seed = 99)

## S3 method for class 'fit.default'
saem(model, data, inits, PKpars = NULL,
      pred = NULL, covars = NULL, mcmc = list(niter = c(200, 300), nmc =
        3, nu = c(2, 2, 2)), ODEopt = list(atol = 1e-06, rtol = 1e-04, stiff =
        1, transitAbs = 0), distribution = c("normal", "poisson", "binomial"),
      seed = 99)

```

## Arguments

model	an RxODE model or lincmt()
data	input data
inits	initial values
PKpars	PKpars function
pred	pred function
covars	Covariates in data
mcmc	a list of various mcmc options
ODEopt	optional ODE solving options
distribution	one of c("normal","poisson","binomial")
seed	seed for random number generator

## Details

Fit a generalized nonlinear mixed-effect model using the Stochastic Approximation Expectation-Maximization (SAEM) algorithm

**Author(s)**

Matthew Fidler &amp; Wenping Wang

saemControl

*Control Options for SAEM***Description**

Control Options for SAEM

**Usage**

```
saemControl(seed = 99, nBurn = 200, nEm = 300, nmc = 3, nu = c(2,
  2, 2), atol = 1e-06, rtol = 1e-04, stiff = TRUE,
  transitAbs = FALSE, print = 1, trace = 0, covMethod = c("fim",
  "r,s", "r", "s"), logLik = FALSE, optExpression = TRUE, ...)
```

**Arguments**

seed	Random Seed for SAEM step. (Needs to be set for reproducibility.) By default this is 99.
nBurn	Number of iterations in the Stochastic Approximation (SA), or burn-in step. This is equivalent to Monolix's $K_0$ or $K_b$ .
nEm	Number of iterations in the Expectation-Maximization (EM) Step. This is equivalent to Monolix's $K_1$ .
nmc	Number of Markov Chains. By default this is 3. When you increase the number of chains the numerical integration by MC method will be more accurate at the cost of more computation. In Monolix this is equivalent to L
nu	This is a vector of 3 integers. They represent the numbers of transitions of the three different kernels used in the Hasting-Metropolis algorithm. The default value is $c(2, 2, 2)$ , representing 40 for each transition initially (each value is multiplied by 20). The first value represents the initial number of multi-variate Gibbs samples are taken from a normal distribution. The second value represents the number of uni-variate, or multi- dimensional random walk Gibbs samples are taken. The third value represents the number of bootstrap/reshuffling or uni-dimensional random samples are taken.
atol	a numeric absolute tolerance (1e-8 by default) used by the ODE solver to determine if a good solution has been achieved; This is also used in the solved linear model to check if prior doses do not add anything to the solution.
rtol	a numeric relative tolerance (1e-6 by default) used by the ODE solver to determine if a good solution has been achieved. This is also used in the solved linear model to check if prior doses do not add anything to the solution.

stiff	a logical (TRUE by default) indicating whether the ODE system is stiff or not. For stiff ODE systems ( <code>stiff = TRUE</code> ), RxODE uses the LSODA (Livermore Solver for Ordinary Differential Equations) Fortran package, which implements an automatic method switching for stiff and non-stiff problems along the integration interval, authored by Hindmarsh and Petzold (2003). For non-stiff systems ( <code>stiff = FALSE</code> ), RxODE uses DOP853, an explicit Runge-Kutta method of order 8(5, 3) of Dormand and Prince as implemented in C by Hairer and Wanner (1993).
transitAbs	boolean indicating if this is a transit compartment absorption
print	The number of iterations that are completed before anything is printed to the console. By default, this is 1.
trace	An integer indicating if you want to trace(1) the SAEM algorithm process. Useful for debugging, but not for typical fitting.
covMethod	Method for calculating covariance. In this discussion, R is the Hessian matrix of the objective function. The S matrix is the sum of each individual's gradient cross-product (evaluated at the individual empirical Bayes estimates). "fim" Use the SAEM-calculated Fisher Information Matrix to calculate the covariance. "r, s" Uses the sandwich matrix to calculate the covariance, that is: $R^{-1} \times S \times R^{-1}$ "r" Uses the Hessian matrix to calculate the covariance as $2 \times R^{-1}$ "s" Uses the crossproduct matrix to calculate the covariance as $4 \times S^{-1}$ "" Does not calculate the covariance step.
logLik	boolean indicating that log-likelihood should be calculated by Gaussian quadrature.
optExpression	Optimize the RxODE expression to speed up calculation. By default this is turned on.
...	Other arguments to control SAEM.

**Value**

List of options to be used in `nlmixr` fit for SAEM.

**Author(s)**

Wenping Wang & Matthew L. Fidler

---

summary.dyn.ID

*Summary of a non-population dynamic model fit*

---

**Description**

Summary of a non-population dynamic model fit

**Usage**

```
## S3 method for class 'dyn.ID'
summary(object, ...)
```

**Arguments**

```
object      a dynmodel fit object
...         additional arguments
```

---

```
summary.dyn.mcmc      Summary of a non-population dynamic model fit using mcmc
```

---

**Description**

Summary of a non-population dynamic model fit using mcmc

**Usage**

```
## S3 method for class 'dyn.mcmc'
summary(object, ...)
```

**Arguments**

```
object      a dynmodel fit object
...         additional arguments
```

---

```
summary.saemFit      Print an SAEM model fit summary
```

---

**Description**

Print an SAEM model fit summary

**Usage**

```
## S3 method for class 'saemFit'
summary(object, ...)
```

**Arguments**

```
object      a saemFit object
...         others
```

**Value**

a list



---

tableControl	<i>Output table/data.frame options</i>
--------------	--

---

**Description**

Output table/data.frame options

**Usage**

```
tableControl(npde = NULL, cwres = NULL, saemNPDE = FALSE,
             saemCWRES = FALSE, nlmeNPDE = FALSE, nlmeCWRES = FALSE,
             foceiNPDE = FALSE, foceNPDE = FALSE, nsim = 300, ties = TRUE,
             seed = 1009)
```

**Arguments**

npde	When TRUE, request npde regardless of the algorithm used.
cwres	When TRUE, request CWRES and FOCEi likelihood regardless of the algorithm used.
saemNPDE	When TRUE and estimating with SAEM, adds NPDE metrics to fit including EPRED, ERES, and NPDE. (default TRUE);
saemCWRES	When TRUE and estimating with SAEM, adds CWRES metrics to the fit including CPRED, CRES and CWRES. It also evaluates the function with the FOCEi objective function to allow comparison between estimation methods. (default FALSE)
nlmeNPDE	When TRUE and estimating with nlme, adds NPDE metrics to fit including EPRED, ERES, and NPDE. (default TRUE);
nlmeCWRES	When TRUE and estimating with nlme, adds CWRES metrics to the fit including CPRED, CRES and CWRES. It also evaluates the function with the FOCEi objective function to allow comparison between estimation methods. (default FALSE)
foceiNPDE	When TRUE and estimating with FOCEi, adds NPDE metrics to fit including EPRED, ERES, and NPDE. (default TRUE);
foceNPDE	When TRUE and estimating with FOCEi, adds NPDE metrics to fit including EPRED, ERES, and NPDE. (default TRUE);
nsim	Number of simulations. By default this is 300
ties	When TRUE, the npde distribution can have ties. When FALSE, the npde distribution uses uniform random numbers to prevent ties.
seed	Seed for running nlmixr simulation. By default 1009

**Details**

If you ever want to add CWRES/FOCEi objective function you can use the [addCwres](#)

If you ever want to add NPDE/EPRED columns you can use the [addNpde](#)

**Value**

A list of table options for nlmixr

**Author(s)**

Matthew L. Fidler

---

theo\_md

*Multiple dose Theophiline PK data*

---

**Description**

This dataset starts with the day 1 concentrations of the Theophiline data that is included in the nlme/NONMEM. After day 7 concentrations were simulated with once a day regimen for 7 days (QD).

**Usage**

theo\_md

**Format**

A data frame with 348 rows by 7 columns

**ID** Subject ID

**TIME** Time (hrs)

**DV** Dependant Variable, Theophiline Concentration

**AMT** Dose Amount/kg

**EVID** RxODE/nlmixr event ID (not NONMEM's)

**CMT** Compartment number

**WT** Weight (kg)

**Source**

NONMEM/nlme

---

theo_sd	<i>Multiple dose Theophiline PK data</i>
---------	--

---

**Description**

This dataset is the day 1 concentrations of the Theophiline data that is included in the nlme/NONMEM.

**Usage**

```
theo_sd
```

**Format**

A data frame with 144 rows by 7 columns

**ID** Subject ID

**TIME** Time (hrs)

**DV** Dependant Variable, Theophiline Concentration

**AMT** Dose Amount/kg

**EVID** RxODE/nlmixr event ID (not NONMEM's)

**CMT** Compartment Number

**WT** Weight (kg)

**Source**

NONMEM/nlme

---

traceplot	<i>Produce trace-plot for fit if applicable</i>
-----------	---

---

**Description**

Produce trace-plot for fit if applicable

**Usage**

```
traceplot(x, ...)
```

```
## S3 method for class 'nlmixrFitCore'
traceplot(x, ...)
```

**Arguments**

x	fit object
...	other parameters

**Value**

Fit traceplot or nothing.

**Author(s)**

Rik Schoemaker, Wenping Wang & Matthew L. Fidler

---

vpc	<i>Vpc function for nlmixr</i>
-----	--------------------------------

---

**Description**

Vpc function for nlmixr

**Usage**

```
vpc(sim, ...)
```

## Default S3 method:

```
vpc(sim, ...)
```

**Arguments**

sim	Observed data frame or fit object
...	Other parameters

---

vpc_nlmixr_nlme	<i>Visual predictive check (VPC) for nlmixr nlme objects</i>
-----------------	--

---

**Description**

Do visual predictive check (VPC) plots for nlme-based non-linear mixed effect models

**Usage**

```
vpc_nlmixr_nlme(fit, nsim = 100, condition = NULL, ...)
```

```
vpcNlmixrNlme(fit, nsim = 100, condition = NULL, ...)
```

## S3 method for class 'nlmixrNlme'

```
vpc(sim, ...)
```

**Arguments**

fit	nlme fit object
nsim	number of simulations
condition	conditional variable
...	Additional arguments
sim	this is usually a data.frame with observed data, containing the independent and dependent variable, a column indicating the individual, and possibly covariates. E.g. load in from NONMEM using <a href="#">read_table_nm</a> . However it can also be an object like a nlmixr or xpose object

**Examples**

```
specs <- list(fixed=lKA+lCL+lV~1, random = pdDiag(lKA+lCL~1), start=c(lKA=0.5, lCL=-3.2, lV=-1))
fit <- nlme_lin_cmpt(theo_md, par_model=specs, ncmt=1, verbose=TRUE)
vpc(fit, nsim = 100, condition = NULL)
```

---

vpc\_saemFit                      *VPC for nlmixr saemFit objects*

---

**Description**

VPC for nlmixr saemFit objects

**Usage**

```
vpc_saemFit(fit, dat, nsim = 100, by = NULL, ...)
```

```
## S3 method for class 'saemFit'
vpc(sim, ...)
```

**Arguments**

fit	saemFit object
dat	Data to augment the saemFit vpc simulation
nsim	Number of simulations for the VPC
by	Variables to condition the VPC
...	Other arguments sent to <a href="#">vpc_vpc</a>
sim	this is usually a data.frame with observed data, containing the independent and dependent variable, a column indicating the individual, and possibly covariates. E.g. load in from NONMEM using <a href="#">read_table_nm</a> . However it can also be an object like a nlmixr or xpose object

**Value**

vpc object from the [vpc\\_vpc](#) package

**Author(s)**

Wenping Wang

vpc\_ui

*VPC based on ui model***Description**

VPC based on ui model

**Usage**

```
vpc_ui(fit, data = NULL, n = 100, bins = "jenks", n_bins = "auto",
  bin_mid = "mean", show = NULL, stratify = NULL,
  pred_corr = FALSE, pred_corr_lower_bnd = 0, pi = c(0.05, 0.95),
  ci = c(0.05, 0.95), uloq = NULL, lloq = NULL, log_y = FALSE,
  log_y_min = 0.001, xlab = NULL, ylab = NULL, title = NULL,
  smooth = TRUE, vpc_theme = NULL, facet = "wrap", labeller = NULL,
  vpcdb = FALSE, verbose = FALSE, ...)
```

```
## S3 method for class 'nlmixrFitData'
vpc(sim, ...)
```

```
## S3 method for class 'nlmixrVpc'
vpc(sim, ...)
```

```
## S3 method for class 'ui'
vpc(sim, ...)
```

**Arguments**

fit	nlmixr fit object
data	this is the data to use to augment the VPC fit. By default is the fitted data, (can be retrieved by <a href="#">getData</a> ), but it can be changed by specifying this argument.
n	Number of VPC simulations. By default 100
bins	either "density", "time", or "data", "none", or one of the approaches available in <a href="#">classInterval()</a> 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 aim for
bin_mid	either "mean" for the mean of all timepoints (default) or "middle" to use the average of the bin boundaries.
show	what to show in VPC (obs_dv, obs_ci, pi, pi_as_area, pi_ci, obs_median, sim_median, sim_median_ci)
stratify	character vector of stratification variables. Only 1 or 2 stratification variables can be supplied.

pred_corr	perform prediction-correction?
pred_corr_lower_bnd	lower bound for the prediction-correction
pi	simulated prediction interval to plot. Default is c(0.05, 0.95),
ci	confidence interval to plot. Default is (0.05, 0.95)
uloq	Number or NULL indicating upper limit of quantification. Default is NULL.
lloq	Number or NULL indicating lower limit of quantification. Default is NULL.
log_y	Boolean indicating whether y-axis should be shown as logarithmic. Default is FALSE.
log_y_min	minimal value when using log_y argument. Default is 1e-3.
xlab	label for x axis
ylab	label for y axis
title	title
smooth	"smooth" the VPC (connect bin midpoints) or show bins as rectangular boxes. Default is TRUE.
vpc_theme	theme to be used in VPC. Expects list of class vpc_theme created with function vpc_theme()
facet	either "wrap", "columns", or "rows"
labeller	ggplot2 labeller function to be passed to underlying ggplot object
vpcdb	Boolean whether to return the underlying vpcdb rather than the plot
verbose	show debugging information (TRUE or FALSE)
...	Args sent to <a href="#">rxSolve</a>
sim	this is usually a data.frame with observed data, containing the independent and dependent variable, a column indicating the individual, and possibly covariates. E.g. load in from NONMEM using <a href="#">read_table_nm</a> . However it can also be an object like a nlmixr or xpose object

**Value**

Simulated dataset (invisibly)

**Author(s)**

Matthew L. Fidler

---

Wang2007

*Simulated Data Set for comparing objective functions*

---

**Description**

This is a simulated dataset from Wang2007 where various NONMEM estimation methods (Laplace FO, FOCE with and without interaction) are described.

**Usage**

Wang2007

**Format**

A data frame with 20 rows and 3 columns

**ID** Simulated Subject ID

**Time** Simulated Time

**Y** Simulated Objective Function

**Source**

Table 1 from Wang, Y *Derivation of Various NONMEM estimation methods*. J Pharmacokinet Pharmacodyn (2007) 34:575-593.



# Index

## \*Topic **datasets**

- Bolus\_1CPT, 7
- Bolus\_1CPTMM, 8
- Bolus\_2CPT, 9
- Bolus\_2CPTMM, 10
- Infusion\_1CPT, 35
- invgaussian, 37
- metabolite, 39
- Oral\_1CPT, 75
- pump, 81
- rats, 82
- theo\_md, 90
- theo\_sd, 91
- Wang2007, 96
- .collectWarnings, 4
  
- addCwres, 5, 89
- addNpde, 5, 89
- AIC, 53
- as.focei, 6
- augPred.nlmixrFitData (nlmixrAugPred), 54
  
- BIC, 53
- Bolus\_1CPT, 7
- Bolus\_1CPTMM, 8
- Bolus\_2CPT, 9
- Bolus\_2CPTMM, 10
- bootdata, 11
  
- calcCov, 12
- configsaem, 13
  
- dynmodel, 14
- dynmodel.mcmc, 16
  
- eventTable, 44, 59
  
- fixed.effects, 54
- focei.eta, 17
- focei.theta, 18
  
- foceiControl, 18, 24, 45, 73
- foceiFit, 24
- frwd\_selection, 31
  
- gauss.quad, 32
- gen\_saem\_user\_fn, 32
- getData, 94
- getOMEGA, 33
- gnlmm, 33
- gnlmm2 (gnlmm), 33
  
- Infusion\_1CPT, 35
- ini, 36
- instant.stan.extension, 37
- invgaussian, 37
  
- lin\_cmt, 38
- lincmt, 38
- logLik, 53
  
- metabolite, 39
- model, 40
  
- n1qn1, 23, 24
- nlme, 45, 46, 51, 53
- nlme\_gof, 40
- nlme\_lin\_cmpt, 41
- nlme\_ode, 42
- nlmeControl, 45, 73
- nlmeLinCmpt (nlme\_lin\_cmpt), 41
- nlmeLinCmt (nlme\_lin\_cmpt), 41
- nlmeOde (nlme\_ode), 42
- nlmixr, 44, 87
- nlmixr\_fit, 73
- nlmixrAugPred, 54
- nlmixrBounds.eta.names, 56
- nlmixrBounds.focei.upper.lower, 56
- nlmixrLogo, 57
- nlmixrPred, 57
- nlmixrSim, 58
- nlmixrUI.focei.fixed, 62

- nlmixrUI.focei.inits, 62
- nlmixrUI.nlme.specs, 63
- nlmixrUI.rxode.pred, 63
- nlmixrUI.saem.ares, 64
- nlmixrUI.saem.bres, 64
- nlmixrUI.saem.distribution, 65
- nlmixrUI.saem.eta.trans, 65
- nlmixrUI.saem.fit, 66
- nlmixrUI.saem.fixed, 66
- nlmixrUI.saem.init, 67
- nlmixrUI.saem.init.omega, 67
- nlmixrUI.saem.init.theta, 68
- nlmixrUI.saem.log.eta, 68
- nlmixrUI.saem.model, 69
- nlmixrUI.saem.model.omega, 69
- nlmixrUI.saem.res.mod, 70
- nlmixrUI.saem.res.name, 70
- nlmixrUI.saem.theta.name, 71
- nlmixrUI.theta.pars, 71
- nlmixrUILinCmt, 72
- nlmixrValidate, 72
- nlmixrVersion, 73
- nmDataConvert, 50, 74
- nmsimplex, 74
  
- optim, 23, 24
- Oral\_1CPT, 75
  
- plot.dyn.ID, 76
- plot.dyn.mcmc, 76
- plot.nlmixrFitData, 77
- plot.saemFit, 77
- predict.nlmixrFitData (nlmixrPred), 57
- prediction, 78
- print.dyn.ID, 79
- print.dyn.mcmc, 79
- print.gnlmm.fit, 80
- print.nlmixrUI, 80
- print.saemFit, 81
- pump, 81
  
- random.effects, 54
- rats, 82
- read\_table\_nm, 93, 95
- residuals.nlmixrFitData, 83
- rmvn, 60
- rmvt, 60
- rxCores, 20, 59
- RxODE, 47
  
- rxSolve, 24, 95
- rxSolve.nlmixrFitData (nlmixrSim), 58
  
- saem, 46, 49, 51–53
- saem (saem.fit), 84
- saem.cleanup, 84
- saem.fit, 84
- saemControl, 45, 52, 73, 86
- simulate.nlmixrFitData (nlmixrSim), 58
- solve.nlmixrFitData (nlmixrSim), 58
- summary.dyn.ID, 87
- summary.dyn.mcmc, 88
- summary.saemFit, 88
  
- tableControl, 45, 73, 89
- theo\_md, 90
- theo\_sd, 91
- traceplot, 52, 91
  
- vpc, 92
- vpc.nlmixrFitData (vpc\_ui), 94
- vpc.nlmixrNlme (vpc\_nlmixr\_nlme), 92
- vpc.nlmixrVpc (vpc\_ui), 94
- vpc.saemFit (vpc\_saemFit), 93
- vpc.ui (vpc\_ui), 94
- vpc\_nlmixr\_nlme, 92
- vpc\_saemFit, 93
- vpc\_ui, 94
- vpc\_vpc, 93
- vpcNlmixrNlme (vpc\_nlmixr\_nlme), 92
  
- Wang2007, 96