Package ‘nlmixr2est’

June 22, 2022

Type Package

Title Nonlinear Mixed Effects Models in Population PK/PD, Estimation Routines

Version 2.0.8

Maintainer Matthew Fidler <matthew.fidler@gmail.com>


License GPL (>= 3)

URL https://github.com/nlmixr2/nlmixr2est

Depends nlmixr2data, R (>= 4.0)

Imports backports, checkmate, cli, graphics, knitr, lbfgsb3c, lotri, magrittr, Matrix, methods, minqa, n1qn1 (>= 6.0.1-10), nlme, Rcpp, rex, Rvmmin, rxode2 (>= 2.0.7), stats, symengine, ucminf, utils, vpc

Suggests broom.mixed, crayon, data.table, devtools, digest, dparser (>= 0.1.8), dplyr, generics, nloptr, qs, sys, testthat, tibble, withr, xgxr

LinkingTo BH, dparser (>= 0.1.8), lbfgsb3c, Rcpp, RcppArmadillo (>= 0.5.600.2.0), RcppEigen (>= 0.3.3.3.0), rxode2 (>= 2.0.7), StanHeaders (>= 2.18.0)

Biarch true

Config/testthat/edition 3

Encoding UTF-8

Language en-US

NeedsCompilation yes

RoxygenNote 7.2.0
Author  Matthew Fidler [aut, cre] (<https://orcid.org/0000-0001-8538-6691>),
Yuan Xiong [aut],
Rik Schoemaker [aut] (<https://orcid.org/0000-0002-7538-3005>),
Justin Wilkins [aut] (<https://orcid.org/0000-0002-7099-9396>),
Wenping Wang [aut],
Robert Leary [ctb],
Mason McComb [ctb] (<https://orcid.org/0000-0001-9871-8616>),
Vipul Mann [aut],
Mirjam Trame [ctb],
Mahmoud Abdelwahab [ctb],
Teun Post [ctb],
Richard Hooijmaijers [aut],
Hadley Wickham [ctb],
Dirk Eddelbuettel [cph],
Johannes Pfeifer [ctb],
Robert B. Schnabel [ctb],
Elizabeth Eskow [ctb],
Emmanuelle Comets [ctb],
Audrey Lavenu [ctb],
Marc Lavielle [ctb],
David Ardia [cph],
Katharine Mullen [cph],
Ben Goodrich [ctb]

Repository  CRAN

Date/Publication  2022-06-22 08:00:02 UTC

R topics documented:

addCwres ........................................... 3
addNpde ............................................... 5
addTable ............................................. 6
assertNlmixrFit ..................................... 8
assertNlmixrFitData ................................. 8
boxCox .............................................. 9
choiSE ............................................ 10
foceiControl ...................................... 11
getValidNlmixrControl ......................... 21
nlmixr2 ........................................... 22
nlmixr2AllEst ..................................... 24
nlmixr2AugPredSolve ......................... 34
nlmixr2CreateOutputFromUi .................. 35
nlmixr2Est.focei ................................ 36
nlmixr2Gill83 .................................. 38
nlmixr2Hess .................................... 40
nlmixr2Logo .................................... 42
nlmixr2NlmeControl ......................... 42
nlmixr2Validate ................................. 47
addCwres

Description

This returns a new fit object with CWRES attached

Usage

```
addCwres(fit, focei = TRUE, updateObject = TRUE, envir = parent.frame(1))
```

Arguments

- **fit**: `nlmixr2` fit without WRES/CWRES
- **focei**: Boolean indicating if the focei objective function is added. If not the focei objective function is added.
- **updateObject**: Boolean indicating if the original fit object should be updated. By default this is true.
- **envir**: Environment that should be checked for object to update. By default this is the global environment.
Value

fit with CWRES

Author(s)

Matthew L. Fidler

Examples

```r
one.cmt <- function() {
  ini(
    ## You may label each parameter with a comment
    tka <- 0.45 # Log Ka
    tcl <- log(c(0, 2.7, 100)) # Log Cl
    ## This works with interactive models
    ## You may also label the preceding line with label("label text")
    tv <- 3.45; label("log V")
    ## the label("Label name") works with all models
    eta.ka ~ 0.6
    eta.cl ~ 0.3
    eta.v ~ 0.1
    add.sd <- 0.7
  )
  model(
    ka <- exp(tka + eta.ka)
    cl <- exp(tcl + eta.cl)
    v <- exp(tv + eta.v)
    linCmt() ~ add(add.sd)
  )
}

f <- try(nlmixr2(one.cmt, theo_sd, "saem"))

print(f)

# even though you may have forgotten to add the cwres, you can add it to the data.frame:

if (!inherits(f, "try-error")) {
  f <- try(addCwres(f))
  print(f)
}

# Note this also adds the FOCEi objective function
```
Description

NPDE calculation for nlmixr2

Usage

```r
addNpde(
  object,
  updateObject = TRUE,
  table = tableControl(),
  ..., 
  envir = parent.frame(1)
)
```

Arguments

- **object**: nlmixr2 fit object
- **updateObject**: Boolean indicating if original object should be updated. By default this is TRUE.
- **table**: `tableControl()` list of options
- **...**: Other ignored parameters.
- **envir**: Environment that should be checked for object to update. By default this is the global environment.

Value

New nlmixr2 fit object

Author(s)

Matthew L. Fidler

Examples

```r
one.cmt <- function() {
  ini({
    ## You may label each parameter with a comment
    tka <- 0.45 # Log Ka
    tcl <- log(c(0, 2.7, 100)) # Log Cl
    ## This works with interactive models
    ## You may also label the preceding line with label("label text")
    tv <- 3.45; label("log V")
  })
```
## the label("Label name") works with all models
eta.ka ~ 0.6
eta.cl ~ 0.3
eta.v ~ 0.1
add.sd <- 0.7
})
model({
  ka <- exp(tka + eta.ka)
  cl <- exp(tcl + eta.cl)
  v <- exp(tv + eta.v)
  linCmt() ~ add(add.sd)
})
}
f <- nlmixr2(one.cmt, theo_sd, "saem")
# even though you may have forgotten to add the NPDE, you can add it to the data.frame:
f <- addNpde(f)

---

### addTable

Add table information to nlmixr2 fit object without tables

**Description**

Add table information to nlmixr2 fit object without tables

**Usage**

```r
addTable(
  object,
  updateObject = FALSE,
  data = object$dataSav,
  thetaEtaParameters = object$foceThetaEtaParameters,
  table = tableControl(),
  keep = NULL,
  drop = NULL,
  envir = parent.frame(1)
)
```

**Arguments**

- `object` nlmixr2 family of objects
- `updateObject` Update the object (default FALSE)
- `data` Saved data from
- `thetaEtaParameters` Internal theta/eta parameters
addTable

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table</td>
<td>a <code>tableControl()</code> list of options</td>
</tr>
<tr>
<td>keep</td>
<td>Character Vector of items to keep</td>
</tr>
<tr>
<td>drop</td>
<td>Character Vector of items to drop or NULL</td>
</tr>
<tr>
<td>envir</td>
<td>Environment to search for updating</td>
</tr>
</tbody>
</table>

**Value**

Fit with table information attached

**Author(s)**

Matthew Fidler

**Examples**

```r
one.cmt <- function() {
  ini(
    ## You may label each parameter with a comment
    tka <- 0.45 # Log Ka
    tcl <- log(c(0, 2.7, 100)) # Log Cl
    ## This works with interactive models
    ## You may also label the preceding line with label("label text")
    tv <- 3.45; label("log V")
    ## the label("Label name") works with all models
    eta.ka ~ 0.6
    eta.cl ~ 0.3
    eta.v ~ 0.1
    add.sd <- 0.7
  )
  model(
    ka <- exp(tka + eta.ka)
    cl <- exp(tcl + eta.cl)
    v <- exp(tv + eta.v)
    linCmt() ~ add(add.sd)
  )
}

# run without tables step
f <- nlmixr2(one.cmt, theo_sd, "saem", control=list(calcTables=FALSE))
print(f)

# Now add the tables
f <- addTable(f)
print(f)
```
assertNlmixrFit

Assert that this is a nlmixr2 fit object

Description
Will error without nlmixr2 fit object

Usage
assertNlmixrFit(fit)

Arguments
fit Fit object

Value
Nothing

Author(s)
Matthew L. Fidler

Examples
## Not run:

f <- 4
assertNlmixrFit(f) # throw error

## End(Not run)

assertNlmixrFitData

Assert that this is a nlmixr2 fit data object

Description
Will error without nlmixr2 fit data object

Usage
assertNlmixrFitData(fit)

Arguments
fit Fit object
Value

Nothing

Author(s)

Matthew L. Fidler

Examples

## Not run:

f <- 4
assertNlmixrFitData(f) # throw errors

## End(Not run)

---

Cox Box, Yeo Johnson and inverse transformation

Description

Cox Box, Yeo Johnson and inverse transformation

Usage

boxCox(x, lambda = 1)

iBoxCox(x, lambda = 1)

yeoJohnson(x, lambda = 1)

iYeoJohnson(x, lambda = 1)

Arguments

x data to transform

lambda Cox-box lambda parameter

Value

Cox-Box Transformed Data

Author(s)

Matthew L. Fidler
Examples

boxCox(1:3,1) ## Normal
iBoxCox(boxCox(1:3,1))

boxCox(1:3,0) ## Log-Normal
iBoxCox(boxCox(1:3,0),0)

boxCox(1:3,0.5) ## lambda=0.5
iBoxCox(boxCox(1:3,0.5),0.5)

yeoJohnson(seq(-3,3),1) ## Normal
iYeoJohnson(yeoJohnson(seq(-3,3),1))

yeoJohnson(seq(-3,3),0)
iYeoJohnson(yeoJohnson(seq(-3,3),0),0)

cholSE

Generalized Cholesky Matrix Decomposition

Description

Performs a (modified) Cholesky factorization of the form

Usage

cholSE(matrix, tol = (.Machine$double.eps)^(1/3))

Arguments

matrix  
Matrix to be Factorized.

tol  
Tolerance; Algorithm suggests (.Machine$double.eps) ^ (1 / 3), default

Details

t(P) %*% A %*% P + E = t(R) %*% R
As detailed in Schnabel/Eskow (1990)

Value

Generalized Cholesky decomposed matrix.

Note

This version does not pivot or return the E matrix

Author(s)

Matthew L. Fidler (translation), Johannes Pfeifer, Robert B. Schnabel and Elizabeth Eskow
References

matlab source: http://www.dynare.org/dynare-matlab-m2html/matlab/chol_SE.html; Slightly different return values


---

foceiControl

Control Options for FOCEi

Description

Control Options for FOCEi

Usage

```r
foceiControl(
  sigdig = 3,
  ..., 
  epsilon = NULL,
  maxInnerIterations = 1000,
  maxOuterIterations = 5000,
  nlq1nsim = NULL,
  print = 1L,
  printNcol = floor((getOption("width") - 23)/12),
  scaleTo = 1,
  scaleObjective = 0,
  normType = c("rescale2", "mean", "rescale", "std", "len", "constant"),
  scaleType = c("nlmixr2", "norm", "mult", "multAdd"),
  scaleCmax = 1e+05,
  scaleCmin = 1e-05,
  scaleC = NULL,
  scaleC0 = 1e+05,
  derivEps = rep(20 * sqrt(.Machine$double.eps), 2),
  derivMethod = c("switch", "forward", "central"),
  derivSwitchTol = NULL,
  covDerivMethod = c("central", "forward"),
  covMethod = c("r", "s", "r", "s", ""),
  hessEps = (.Machine$double.eps)^(1/3),
  eventFD = sqrt(.Machine$double.eps),
  eventType = c("gill", "central", "forward"),
  centralDerivEps = rep(20 * sqrt(.Machine$double.eps), 2),
  lbfgsLmm = 7L,
  lbfgsPgtol = 0,
  lbfgsFactr = 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$double.eps)^(1/3),
cholAccept = 0.001,
resetEtaP = 0.15,
resetThetaP = 0.05,
resetThetaFinalP = 0.15,
diagOmegaBoundUpper = 5,
diagOmegaBoundLower = 100,
cholSEOpt = FALSE,
cholSEcov = FALSE,
fo = FALSE,
covTryHarder = FALSE,
outerOpt = c("nlminb", "bobyqa", "lbfgsb3c", "L-BFGS-B", "mma", "lbfgsbLG", "slsqp",
          "Rvmmin"),
innerOpt = c("n1qn1", "BFGS"),
rhobeg = 0.2,
rhoend = NULL,
npt = NULL,
rel.tol = NULL,
x.tol = NULL,
eval.max = 4000,
itner.max = 2000,
abstol = NULL,
reltol = NULL,
resetHessianAndEta = FALSE,
stateTrim = Inf,
gillK = 10L,
gillStep = 4,
gillFtol = 0,
gillRtol = sqrt(.Machine$double.eps),
gillKcov = 10L,
gillStepCov = 2,
gillFtolCov = 0,
rmatNorm = TRUE,
smatNorm = TRUE,
covGillF = TRUE,
optGillF = TRUE,
covSmall = 1e-05,
```r
adjLik = TRUE,
ggradTrim = Inf,
maxOdeRecalc = 5,
odeRecalcFactor = 10^(0.5),
grgradCalcCentralSmall = 1e-04,
grgradCalcCentralLarge = 10000,
etaNudge = qnorm(1 - 0.05/2)/sqrt(3),
etaNudge2 = qnorm(1 - 0.05/2) * sqrt(3/5),
nRetries = 3,
seed = 42,
resetThetaCheckPer = 0.1,
etMat = NULL,
repeatGillMax = 3,
stickyRecalcN = 5,
grgradProgressOfvTime = 10,
addProp = c("combined2", "combined1"),
badSolveObjfAdj = 100,
compress = TRUE,
rxControl = NULL,
sigdigTable = NULL,
fallbackFD = FALSE
)
```

### Arguments

- **sigdig**: Optimization significant digits. This controls:
  - The tolerance of the inner and outer optimization is $10^{-\text{sigdig}}$
  - The tolerance of the ODE solvers is $0.5 \times 10^{(-\text{sigdig}-2)}$; For the sensitivity equations and steady-state solutions the default is $0.5 \times 10^{(-\text{sigdig}-1.5)}$ (sensitivity changes only applicable for liblsoda)
  - The tolerance of the boundary check is $5 \times 10^{(-\text{sigdig} + 1)}$

Ignored parameters:

- **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.
- **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**: Scale the initial objective function to this value. By default this is 0 (meaning do not scale)
This is the type of parameter normalization/scaling used to get the scaled initial values for nlmixr2. These are used with `scaleType` of.

With the exception of `rescale2`, these come from Feature Scaling. The `rescale2` rescaling is the same type described in the OptdesX software manual.

In general, all scaling formula can be described by:
\[
v_{\text{scaled}} = \frac{(v_{\text{unscaled}} - C_1)}{C_2}
\]

Where

The other data normalization approaches follow the following formula
\[
v_{\text{scaled}} = \frac{(v_{\text{unscaled}} - C_1)}{C_2};
\]

- `rescale2` This scales all parameters from (-1 to 1). The relative differences between the parameters are preserved with this approach and the constants are:
  \[
  C_1 = \frac{\text{max}(\text{all unscaled values}) + \text{min}(\text{all unscaled values})}{2}
  \]

- `rescale` or min-max normalization. This rescales all parameters from (0 to 1). As in the `rescale2` the relative differences are preserved. In this approach:
  \[
  C_1 = \text{min}(\text{all unscaled values})
  \]
  \[
  C_2 = \text{max}(\text{all unscaled values}) - \text{min}(\text{all unscaled values})
  \]

- `mean` or mean normalization. This rescales to center the parameters around the mean but the parameters are from 0 to 1. In this approach:
  \[
  C_1 = \text{mean}(\text{all unscaled values})
  \]
  \[
  C_2 = \text{max}(\text{all unscaled values}) - \text{min}(\text{all unscaled values})
  \]

- `std` or standardization. This standardizes by the mean and standard deviation. In this approach:
  \[
  C_1 = \text{mean}(\text{all unscaled values})
  \]
  \[
  C_2 = \text{sd}(\text{all unscaled values})
  \]

- `len` or unit length scaling. This scales the parameters to the unit length. For this approach we use the Euclidean length, that is:
  \[
  C_1 = 0
  \]
  \[
  C_2 = \sqrt{v_1^2 + v_2^2 + \ldots + v_n^2}
  \]

- `constant` which does not perform data normalization. That is
  \[
  C_1 = 0
  \]
  \[
  C_2 = 1
  \]

The scaling scheme for nlmixr2. The supported types are:

- `nlmixr2` In this approach the scaling is performed by the following equation:
  \[
  v_{\text{scaled}} = \frac{(v_{\text{current}} - v_{\text{init}})}{\text{scaleC}[i] + \text{scaleTo}}
  \]

  The `scaleTo` parameter is specified by the `normType`, and the scales are specified by `scaleC`.

- `norm` This approach uses the simple scaling provided by the `normType` argument.

- `mult` This approach does not use the data normalization provided by `normType`, but rather uses multiplicative scaling to a constant provided by the `scaleTo` argument.
In this case:
\[ v_{\text{scaled}} = \frac{v_{\text{current}}}{v_{\text{init}}} \times \text{scaleTo} \]

- **multAdd** This approach changes the scaling based on the parameter being specified. If a parameter is defined in an exponential block (ie \( \exp(\theta) \)), then it is scaled on a linearly, that is:
\[ v_{\text{scaled}} = (v_{\text{current}}-v_{\text{init}}) + \text{scaleTo} \]

Otherwise the parameter is scaled multiplicatively.
\[ v_{\text{scaled}} = \frac{v_{\text{current}}}{v_{\text{init}}} \times \text{scaleTo} \]

- **scaleCmax** Maximum value of the scaleC to prevent overflow.
- **scaleCmin** Minimum value of the scaleC to prevent underflow.
- **scaleC** The scaling constant used with scaleType=nlmixr2. When not specified, it is based on the type of parameter that is estimated. The idea is to keep the derivatives similar on a log scale to have similar gradient sizes. Hence parameters like \( \log(\exp(\theta)) \) would have a scaling factor of 1 and \( \log(\theta) \) would have a scaling factor of \( \text{ini}_\text{value} \) (to scale by \( 1/\text{value} \); ie \( \text{d/dt}(\log(\text{ini}_\text{value})) = 1/\text{ini}_\text{value} \) or scaleC=ini_value)

- For parameters in an exponential (ie \( \exp(\theta) \)) or parameters specifying powers, boxCox or yeoJohnson transformations, this is 1.
- For additive, proportional, lognormal error structures, these are given by \( 0.5 \times |\text{initial}_\text{estimate}| \)
- Factorials are scaled by \( |1/\text{digamma}(\text{initial}_\text{estimate}+1)| \)
- Parameters in a log scale (ie \( \log(\theta) \)) are transformed by \( \log(|\text{initial}_\text{estimate}|) \times |\text{initial}_\text{estimate}| \)

These parameter scaling coefficients are chose to try to keep similar slopes among parameters. That is they all follow the slopes approximately on a log scale.

While these are chosen in a logical manner, they may not always apply. You can specify each parameters scaling factor by this parameter if you wish.

- **scaleC0** Number to adjust the scaling factor by if the initial gradient is zero.
- **derivEps** Forward difference tolerances, which is a vector of relative difference and absolute difference. The central/forward difference step size \( h \) is calculated as:
\[ h = |x| \times \text{derivEps}[1] + \text{derivEps}[2] \]

- **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 \( |\text{delta}(\text{OFV})| \leq \text{derivSwitchTol} \) and switch back to forward differences when \( |\text{delta}(\text{OFV})| > \text{derivSwitchTol} \).

- **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-products (evaluated at the individual empirical Bayes estimates).

  - "r,s" Uses the sandwich matrix to calculate the covariance, that is: \( \text{solve}(R) \times S \times \text{solve}(R) \)
• "r" Uses the Hessian matrix to calculate the covariance as \(2 \times \text{solve}(R)\)
• "s" Uses the cross-product matrix to calculate the covariance as \(4 \times \text{solve}(S)\)
• "" Does not calculate the covariance step.

hessEps
is a double value representing the epsilon for the Hessian calculation.

eventFD
Finite difference step for forward or central difference estimation of event-based gradients.

eventType
Event gradient type for dosing events; Can be "gill", "central" or "forward"

centralDerivEps
Central difference tolerances. This is a numeric vector of relative difference and absolute difference. The central/forward difference step size \(h\) is calculated as:
\[ h = \text{abs}(x) \times \text{derivEps}[1] + \text{derivEps}[2] \]

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 >= 0 is specified by the user. The iteration will stop when:
\[ \max(\| \text{proj g}_i \| \text{ i = 1, ..., n} ) <= \text{lbfgsPgtol} \]
where \(\text{pg}_i\) is the ith 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 .Machine$double.eps

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 \(\text{chol} (\text{solve}(\omega))\). This matrix and values are the parameters estimated in FOCEi. The possibilities are:
• \(\text{sqrt}\) Estimates the sqrt of the diagonal elements of \(\text{chol} (\text{solve}(\omega))\). This is the default method.
• \(\text{log}\) Estimates the log of the diagonal elements of \(\text{chol} (\text{solve}(\omega))\)
• \(\text{identity}\) Estimates the diagonal elements without any transformations

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 rxode2 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 fociiFit 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).

deriveTable

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 chol(omega^-1) %*% eta or eta/sd(allEtas). A p-value of 0 indicates the ETAs never reset. A p-value of 1 indicates the ETAs always reset.

resetThetaP

represents the p-value for resetting the population mu-referenced THETA parameters based on ETA drift during optimization, and resetting the optimization. A p-value of 0 indicates the THETAs never reset. A p-value of 1 indicates the THETAs always reset and is not allowed. The theta reset is checked at the beginning and when nearing a local minima. The percent change in objective function where a theta reset check is initiated is controlled in resetThetaCheckPer.

resetThetaFinalP

represents the p-value for resetting the population mu-referenced THETA parameters based on ETA drift during optimization, and resetting the optimization one final time.

diagOmegaBoundUpper

This represents the upper bound of the diagonal omega matrix. The upper bound is given by diag(omega)*diagOmegaBoundUpper. If 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 diag(omega)/diagOmegaBoundUpper. If diagOmegaBoundLower is 1, there is no lower bound on Omega.

cholSE0pt

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 < rhoend < rhobeg. Typically rhobeg should be about one tenth of the greatest expected change to a variable. Note also that smallest difference abs(upper-lower) should be greater than or equal to rhobeg*2. If this is not the case then rhobeg will be adjusted. (bobyqa)
rhoend  The smallest value of the trust region radius that is allowed. If not defined, then 10^(-sigdig-1) will be used. (bobyqa)

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\(*n+1\). (bobyqa)

rel.tol  Relative tolerance before nlminb stops (nlmimb).

ox.tol  X tolerance for nlmixr2 optimizer

eval.max  Number of maximum evaluations of the objective function (nlmimb)

iter.max  Maximum number of iterations allowed (nlmimb)

abstol  Absolute tolerance for nlmixr2 optimizer (BFGS)

reltol  tolerance for nlmixr2 (BFGS)

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.

gillK  The total number of possible steps to determine the optimal forward/central difference step size per parameter (by the Gill 1983 method). If 0, no optimal step size is determined. Otherwise this is the optimal step size determined.

gillStep  When looking for the optimal forward difference step size, this is the step size to increase the initial estimate by. So each iteration the new step size = (prior step size)*gillStep

gillFtol  The gillFtol is the gradient error tolerance that is acceptable before issuing a warning/error about the gradient estimates.

gillRtol  The relative tolerance used for Gill 1983 determination of optimal step size.

gillKcov  The total number of possible steps to determine the optimal forward/central difference step size per parameter (by the Gill 1983 method) during the covariance step. If 0, no optimal step size is determined. Otherwise this is the optimal step size determined.

gillStepCov  When looking for the optimal forward difference step size, this is the step size to increase the initial estimate by. So each iteration during the covariance step is equal to the new step size = (prior step size)*gillStepCov

gillFtolCov  The gillFtol is the gradient error tolerance that is acceptable before issuing a warning/error about the gradient estimates during the covariance step.

rmatNorm  A parameter to normalize gradient step size by the parameter value during the calculation of the R matrix

smatNorm  A parameter to normalize gradient step size by the parameter value during the calculation of the S matrix

covGillF  Use the Gill calculated optimal Forward difference step size for the instead of the central difference step size during the central difference gradient calculation.

optGillF  Use the Gill calculated optimal Forward difference step size for the instead of the central difference step size during the central differences for optimization.
covSmall  The covSmall is the small number to compare covariance numbers before rejecting an estimate of the covariance as the final estimate (when comparing sandwich vs R/S matrix estimates of the covariance). This number controls how small the variance is before the covariance matrix is rejected.

adjLik  In nlmixr2, the objective function matches NONMEM’s objective function, which removes a 2*pi constant from the likelihood calculation. If this is TRUE, the likelihood function is adjusted by this 2*pi factor. When adjusted this number more closely matches the likelihood approximations of nlme, and SAS approximations. Regardless of if this is turned on or off the objective function matches NONMEM’s objective function.

gradTrim  The parameter to adjust the gradient to if the |gradient| is very large.

maxOdeRecalc  Maximum number of times to reduce the ODE tolerances and try to resolve the system if there was a bad ODE solve.

odeRecalcFactor  The ODE recalculation factor when ODE solving goes bad, this is the factor the rtol/atol is reduced.

gradCalcCentralSmall  A small number that represents the value where |grad| < gradCalcCentralSmall where forward differences switch to central differences.

gradCalcCentralLarge  A large number that represents the value where |grad| > gradCalcCentralLarge where forward differences switch to central differences.

etaNudge  By default initial ETA estimates start at zero; Sometimes this doesn’t optimize appropriately. If this value is non-zero, when the nlqnn1 optimization didn’t perform appropriately, reset the Hessian, and nudge the ETA up by this value; If the ETA still doesn’t move, nudge the ETA down by this value. By default this value is qnorm(1-0.05/2)*1/sqrt(3), the first of the Gauss Quadrature numbers times by the 0.95% normal region. If this is not successful try the second eta nudge number (below). If +-etaNudge2 is not successful, then assign to zero and do not optimize any longer.

etaNudge2  This is the second eta nudge. By default it is qnorm(1-0.05/2)*sqrt(3/5), which is the n=3 quadrature point (excluding zero) times by the 0.95% normal region.

nRetries  If FOCEi doesn’t fit with the current parameter estimates, randomly sample new parameter estimates and restart the problem. This is similar to ‘PsN’ resampling.

seed  an object specifying if and how the random number generator should be initialized.

resetThetaCheckPer  represents objective function % percentage below which resetThetaP is checked.

etaMat  Eta matrix for initial estimates or final estimates of the ETAs.

repeatGillMax  If the tolerances were reduced when calculating the initial Gill differences, the Gill difference is repeated up to a maximum number of times defined by this parameter.

stickyRecalcN  The number of bad ODE solves before reducing the atol/rtol for the rest of the problem.
gradProgressOfvTime
This is the time for a single objective function evaluation (in seconds) to start progress bars on gradient evaluations

addProp
specifies the type of additive plus proportional errors, the one where standard deviations add (combined1) or the type where the variances add (combined2).
The combined1 error type can be described by the following equation:
\[ y = f + (a + b f^c) \times \text{err} \]
The combined2 error model can be described by the following equation:
\[ y = f + \sqrt{a^2 + b^2 (f^c)^2} \times \text{err} \]
Where:
- \( y \) represents the observed value
- \( f \) represents the predicted value
- \( a \) is the additive standard deviation
- \( b \) is the proportional/poer standard deviation
- \( c \) is the power exponent (in the proportional case \( c=1 \))

badSolveObjfAdj
The objective function adjustment when the ODE system cannot be solved. It is based on each individual bad solve.

compress
Should the object have compressed items

rxControl
‘rxode2’ ODE solving options during fitting, created with ‘rxControl()’
sigdigTable
Significant digits in the final output table. If not specified, then it matches the significant digits in the ‘sigdig’ optimization algorithm. If ‘sigdig’ is NULL, use 3.
fallbackFD
Fallback to the finite differences if the sensitivity equations do not solve.

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).

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.

Value
The control object that changes the options for the FOCEi family of estimation methods

Author(s)
Matthew L. Fidler
getValidNlmixrControl

**See Also**

optim, n1qn1, rxSolve

---

getValidNlmixrControl  *Get valid nlmixr control object*

Description

Get valid nlmixr control object

Usage

getValidNlmixrControl(control, est)

genericNlmixrCtl(control)

## S3 method for class 'focei'
getValidNlmixrCtl(control)

## S3 method for class 'foce'
getValidNlmixrCtl(control)

## S3 method for class 'fo'
getValidNlmixrCtl(control)

## S3 method for class 'foi'
getValidNlmixrCtl(control)

## S3 method for class 'posthoc'
getValidNlmixrCtl(control)

## S3 method for class 'foce'
getValidNlmixrCtl(control)

## S3 method for class 'nlme'
getValidNlmixrCtl(control)

## S3 method for class 'saem'
getValidNlmixrCtl(control)

## S3 method for class 'rxSolve'
getValidNlmixrCtl(control)

## S3 method for class 'simulation'
getValidNlmixrCtl(control)

## S3 method for class 'tableControl'
getValidNlmixrCtl(control)

## Default S3 method:
getValidNlmixrCtl(control)

Arguments

control nlmixr control object
est Estimation routine

Details

This is based on running the S3 method ‘getValidNlmixrCtl()’ the ‘control’ object is put into a list and the class of this new list is ‘c(est, "getValidNlmixrControl")’

Value

Valid control object based on estimation method run.

Description

nlmixr2 is an R package for fitting population pharmacokinetic (PK) and pharmacokinetic-pharmacodynamic (PKPD) models.

Usage

```r
nlmixr2(
  object,
  data,
  est = NULL,
  control = list(),
  table = tableControl(),
  ...,
  save = NULL,
  envir = parent.frame()
)
```

```r
nlmixr(
  object,
  data,
```
est = NULL,
control = list(),
table = tableControl(),
..., 
save = NULL,
envir = parent.frame()
)

## S3 method for class `function`
nlmixr2(
  object,
data = NULL,
est = NULL,
control = NULL,
table = tableControl(),
..., 
save = NULL,
envir = parent.frame()
)

## S3 method for class `rxUi`
nlmixr2(
  object,
data = NULL,
est = NULL,
control = NULL,
table = tableControl(),
..., 
save = NULL,
envir = parent.frame()
)

## S3 method for class `nlmixr2FitCore`
nlmixr2(
  object,
data = NULL,
est = NULL,
control = NULL,
table = tableControl(),
..., 
save = NULL,
envir = parent.frame()
)

## S3 method for class `nlmixr2FitData`
nlmixr2(
  object,
data = NULL,
est = NULL,
control = NULL,
table = tableControl(),
..., 
save = NULL,
envir = parent.frame()
)

Arguments

- **object**: Fitted object or function specifying the model.
- **data**: nlmixr data
- **est**: estimation method (all methods are shown by ‘nlmixr2AllEst()’). Methods can be added for other tools
- **control**: The estimation control object. These are expected to be different for each type of estimation method
- **table**: The output table control object (like ‘tableControl()’)
- **...**: Other parameters
- **save**: Boolean to save a nlmixr2 object in a rds file in the working directory. If NULL, uses option "nlmixr2.save"
- **envir**: Environment where the nlmixr object/function is evaluated before running the estimation routine.

Details

The nlmixr2 generalized function allows common access to the nlmixr2 estimation routines.

Value

Either a nlmixr2 model or a nlmixr2 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 an 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:

```r
f <- function(){  
  ini({
    # Initial conditions for population parameters (sometimes called theta parameters) are defined by either '<- ' or '='
    lC1 <- 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:
    # Residual 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 R-compatible assignment
- Boundaries may be specified by `c(lower, est, upper)`. 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 rxode2 and nlmixr2 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

```r
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:

```r
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 often defined before the differential equations.
- 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*Covariate.Parameter. 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:

```r
##'
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:

• 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:
## 1-compartment model with first-order absorption in terms of Cl
## Initialization:

### Fixed Effects ($\theta$):

- lCl \quad 1.60000
- lVc \quad 4.49981
- lKA \quad 0.10000

### Omega ($\omega$):

\[
\begin{bmatrix}
[1,] & 0.1 & 0.0 & 0.0 \\
[2,] & 0.0 & 0.1 & 0.0 \\
[3,] & 0.0 & 0.0 & 0.1 \\
\end{bmatrix}
\]

### Model:

\[
\begin{align*}
Cl & \leftarrow \exp(lCl + \eta.Cl) \\
Vc & = \exp(lVc + \eta.Vc) \\
KA & \leftarrow \exp(lKA + \eta.KA)
\end{align*}
\]

Instead of specifying the ODEs, you can use the \texttt{linCmt()} function to use the solved system.

\[
\text{linCmt()} \sim \text{prop(prop.err)}
\]

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 very similar standard to what nlmixr can support.

Once the data has been converted to the appropriate format, you can use the \texttt{nlmixr} function to run the appropriate code.

The method to estimate the model is:

\[
\text{fit} <- \text{nlmixr(model.function, dataset, est="est", control=estControl(options))}
\]

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

\[
\begin{align*}
\text{> f} & \leftarrow \text{function()}{
\text{ini}({}
\text{\hspace{1em} lCl} & \leftarrow 1.6 \quad \# \text{log Cl (L/hr)}
\text{\})}
\end{align*}
\]
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.
In file included from c:/R/R-34~1.1/library/RcppArmadillo/include/RcppArmadillo.h:31,
  from c:/R/R-34~1.1/library/RcppArmadillo/include/RcppArmadilloForward.h:46,
  from c:/R/R-34~1.1/library/Rcpp/include/armadillo_bits/compiler_setup.hpp:474:96: note: #pragma message: WARNING: use of OpenMP disabled; this compiler doesn’t support OpenMP 3.0+

Using sympy via SnakeCharmR
## Calculate ETA-based prediction and error derivatives:
Calculate Jacobian...................done.
Calculate sensitivities............done.
## Calculate d(f)/d(eta)
## ...
## 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:

```r
> 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 <fct> TIME <dbl> DV PRED RES WRES IPRED IRES IWRES CPRED CRES
* <dbl> <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
```
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

Examples

```r
one.cmt <- function() {
  ini(
    ## You may label each parameter with a comment
    tka <- 0.45 # Ka
    tcl <- log(c(0, 2.7, 100)) # Log Cl
    ## This works with interactive models
    ## You may also label the preceding line with label("label text")
    tv <- 3.45; label("log V")
    ## the label("Label name") works with all models
    eta.ka ~ 0.6
    eta.cl ~ 0.3
    eta.v ~ 0.1
    add.sd <= 0.7
    prop.sd <= 0.01
  )
  model(
    ka <- exp(tka + eta.ka)
    cl <- exp(tcl + eta.cl)
    v <- exp(tv + eta.v)
    linCmt() ~ add(add.sd) + prop(prop.sd)
  )
}

fitF <- nlmixr(one.cmt, theo_sd, "focei")
fitS <- nlmixr(one.cmt, theo_sd, "saem")
```
nlmixr2AllEst  

*Show all the current estimation methods*

**Description**

Show all the current estimation methods

**Usage**

```r
nlmixr2AllEst()
```

**Value**

List of supported nlmixr2 estimation options (est=...)

**Examples**

```r
nlmixr2AllEst()
```

---

nlmixr2AugPredSolve  

*Augmented Prediction for nlmixr2 fit*

**Description**

Augmented Prediction for nlmixr2 fit

**Usage**

```r
nlmixr2AugPredSolve(
  fit,
  covsInterpolation = c("locf", "nocb", "linear", "midpoint"),
  minimum = NULL,
  maximum = NULL,
  length.out = 51L,
  ...
)
```

```r
# S3 method for class 'nlmixr2FitData'
augPred(
  object,
  primary = NULL,
  minimum = NULL,
  maximum = NULL,
  length.out = 51,
  ...
)
```
Arguments

- **fit**: Nlmixr2 fit object

- **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" - linear interpolation, which interpolates the covariate by solving the line between the observed covariates and extrapolating the new covariate value.
  - "constant" - Last observation carried forward (the default).
  - "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.

- **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.

- **...**: Some methods for the generic may require additional arguments.

- **object**: A fitted model object from which predictions can be extracted, using a predict method.

- **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.

Value

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

Author(s)

Matthew L. Fidler

**nlmixr2CreateOutputFromUi**

Create nlmixr output from the UI

Description

Create nlmixr output from the UI
Usage

```r
nlmixr2CreateOutputFromUi(
  ui,
  data = NULL,
  control = NULL,
  table = NULL,
  env = NULL,
  est = "none"
)
```

Arguments

- **ui**
  This is the UI that will be used for the translation.

- **data**
  This has the data.

- **control**
  focei control for data creation.

- **table**
  Table options.

- **env**
  Environment setup which needs the following:
  - "$table" for table options
  - "$origData" – Original Data
  - "$dataSav" – Processed data from .foceiPreProcessData
  - "$idLvl" – Level information for ID factor added
  - "$ui" for ui object
  - "$fullTheta" Full theta information
  - "$etaObf" data frame with ID, etas and OBJI
  - "$cov" For covariance
  - "$covMethod" for the method of calculating the covariance
  - "$adjObjf" Should the objective function value be adjusted
  - "$objective" objective function value
  - "$extra" Extra print information
  - "$method" Estimation method (for printing)
  - "$omega" Omega matrix
  - "$etaObf" Eta objective function data frame
  - "$theta" Is a theta data frame
  - "$model" a list of model information for table generation
  - "$message" Message for display
  - "$est" estimation method
  - "$ofvType" (optional) tells the type of ofv is currently being use
  There are some more details that need to be described here.

- **est**
  Estimation method.

Value

- nlmixr fit object

Author(s)

Matthew L. Fidler

---

**nlmixr2Est.focei**

Generic for nlmixr2 estimation methods

Description

Generic for nlmixr2 estimation methods
Usage

## S3 method for class 'focei'
nlmixr2Est(env, ...)

## S3 method for class 'foce'
nlmixr2Est(env, ...)

## S3 method for class 'posthoc'
nlmixr2Est(env, ...)

## S3 method for class 'foi'
nlmixr2Est(env, ...)

## S3 method for class 'fo'
nlmixr2Est(env, ...)

## S3 method for class 'output'
nlmixr2Est(env, ...)

## S3 method for class 'nlme'
nlmixr2Est(env, ...)

nlmixr2Est(env, ...)

## Default S3 method:
nlmixr2Est(env, ...)

## S3 method for class 'rxSolve'
nlmixr2Est(env, ...)

## S3 method for class 'simulate'
nlmixr2Est(env, ...)

## S3 method for class 'predict'
nlmixr2Est(env, ...)

## S3 method for class 'saem'
nlmixr2Est(env, ...)

Arguments

env Environment for the nlmixr2 estimation routines.
This needs to have:
- rxode2 ui object in `$ui`
- data to fit in the estimation routine in `$data`
- control for the estimation routine’s control options in `$ui`
...

... Other arguments provided to `nlmixr2Est()` provided for flexibility but not currently used inside nlmixr
Details

This is a S3 generic that allows others to use the nlmixr2 environment to do their own estimation routines.

Value

nlmixr2 fit object

Author(s)

Matthew Fidler

nlmixr2Gill83

Get the optimal forward difference interval by Gill83 method

Description

Get the optimal forward difference interval by Gill83 method

Usage

```
nlmixr2Gill83(
  what,
  args,
  envir = parent.frame(),
  which,
  gillRtol = sqrt(.Machine$double.eps),
  gillK = 10L,
  gillStep = 2,
  gillFtol = 0
)
```

Arguments

- **what**: either a function or a non-empty character string naming the function to be called.
- **args**: a list of arguments to the function call. The names attribute of args gives the argument names.
- **envir**: an environment within which to evaluate the call. This will be most useful if what is a character string and the arguments are symbols or quoted expressions.
- **which**: Which parameters to calculate the forward difference and optimal forward difference interval.
- **gillRtol**: The relative tolerance used for Gill 1983 determination of optimal step size.
- **gillK**: The total number of possible steps to determine the optimal forward/central difference step size per parameter (by the Gill 1983 method). If 0, no optimal step size is determined. Otherwise this is the optimal step size determined.
When looking for the optimal forward difference step size, this is the step size to increase the initial estimate by. So each iteration the new step size = (prior step size)*gillStep

The gillFtol is the gradient error tolerance that is acceptable before issuing a warning/error about the gradient estimates.

Value

A data frame with the following columns:

- info: Gradient evaluation/forward difference information
- hf: Forward difference final estimate
- df: Derivative estimate
- df2: 2nd Derivative Estimate
- err: Error of the final estimate derivative
- aEps: Absolute difference for forward numerical differences
- rEps: Relative Difference for backward numerical differences
- aEpsC: Absolute difference for central numerical differences
- rEpsC: Relative difference for central numerical differences

The info returns one of the following:

- Not Assessed: Gradient wasn’t assessed
- Good: Success in Estimating optimal forward difference interval
- High Grad Error: Large error; Derivative estimate error fTol or more of the derivative
- Constant Grad: Function constant or nearly constant for this parameter
- Odd/Linear Grad: Function odd or nearly linear, df = K, df2 ~ 0
- Grad changes quickly: df2 increases rapidly as h decreases

Author(s)

Matthew Fidler

Examples

```r
## These are taken from the numDeriv::grad examples to show how
## simple gradients are assessed with nlmixr2Gill83

nlmixr2Gill83(sin, pi)
nlmixr2Gill83(sin, (0:10)*2*pi/10)

func0 <- function(x){ sum(sin(x)) }
nlmixr2Gill83(func0 , (0:10)*2*pi/10)

func1 <- function(x){ sin(10*x) - exp(-x) }
```
curve(func1, from=0, to=5)

x <- 2.04
numd1 <- nlmixr2Gill83(func1, x)
extact <- 10*cos(10*x) + exp(-x)
c(numd1$df, exact, (numd1$df - exact)/exact)

x <- c(1:10)
umd1 <- nlmixr2Gill83(func1, x)
extact <- 10*cos(10*x) + exp(-x)
cbind(numd1=numd1$df, exact, err=(numd1$df - exact)/exact)

sc2.f <- function(x){
  n <- length(x)
  sum((1:n) * (exp(x) - x)) / n
}

sc2.g <- function(x){
  n <- length(x)
  (1:n) * (exp(x) - 1) / n
}

x0 <- rnorm(100)
extact <- sc2.g(x0)
g <- nlmixr2Gill83(sc2.f, x0)
max(abs(exact - g$df)/(1 + abs(exact)))

---

**nlmixr2Hess**: Calculate Hessian

**Description**

Unlike `stats::optimHess` which assumes the gradient is accurate, nlmixr2Hess does not make as strong an assumption that the gradient is accurate but takes more function evaluations to calculate the Hessian. In addition, this procedure optimizes the forward difference interval by nlmixr2Gill83.

**Usage**

`nlmixr2Hess(par, fn, ..., envir = parent.frame())`

**Arguments**

- **par**: Initial values for the parameters to be optimized over.
- **fn**: A function to be minimized (or maximized), with first argument the vector of parameters over which minimization is to take place. It should return a scalar result.
Extra arguments sent to `nlmixr2Gill83` an environment within which to evaluate the call. This will be most useful if what is a character string and the arguments are symbols or quoted expressions.

Details

If you have an analytical gradient function, you should use `stats::optimHess`.

Value

Hessian matrix based on Gill83

Author(s)

Matthew Fidler

References


See Also

`nlmixr2Gill83`, `optimHess`

Examples

```r
func0 <- function(x){ sum(sin(x)) } 
x <- (0:10)*2*pi/10
nlmixr2Hess(x, func0)

fr <- function(x) { ## Rosenbrock Banana function
  x1 <- x[1]
  x2 <- x[2]
  100 * (x2 - x1 * x1)^2 + (1 - x1)^2
}
grr <- function(x) { ## Gradient of 'fr'
  x1 <- x[1]
  x2 <- x[2]
  c(-400 * x1 * (x2 - x1 * x1) - 2 * (1 - x1),
      200 * (x2 - x1 * x1))
}

h1 <- optimHess(c(1.2,1.2), fr, grr)

h2 <- optimHess(c(1.2,1.2), fr)

## in this case h3 is closer to h1 where the gradient is known

h3 <- nlmixr2Hess(c(1.2,1.2), fr)
```
nlmixr2Logo

*Messages the nlmixr2 logo...*

Description

Messages the nlmixr2 logo...

Usage

```r
nlmixr2Logo(str = "", version = sessionInfo()$otherPkgs$nlmixr2$Version)
```

Arguments

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

Value

nothing; Called to display version information

Author(s)

Matthew L. Fidler

nlmixr2NlmeControl

*Control Values for nlme Fit with extra options for nlmixr*

Description

The values supplied in the function call replace the defaults and a list with all possible arguments is returned. The returned list is used as the ‘control’ argument to the ‘nlme’ function.

Usage

```r
nlmixr2NlmeControl(
  maxIter = 100,
  pnlsMaxIter = 100,
  msMaxIter = 100,
  minScale = 0.001,
  tolerance = 1e-05,
  niterEM = 25,
  pnlsTol = 0.001,
  msTol = 1e-06,
  returnObject = FALSE,
  msVerbose = FALSE,
)```
msWarnNoConv = TRUE,
gradiHess = TRUE,
apVar = TRUE,
.relStep = .Machine$double.eps^(1/3),
minAbsParApVar = 0.05,
opt = c("nlminb", "nlm"),
natural = TRUE,
sigma = NULL,
optExpression = TRUE,
sumProd = FALSE,
rxCntrl = NULL,
method = c("ML", "REML"),
random = NULL,
fixed = NULL,
weights = NULL,
verbose = TRUE,
returnNlme = FALSE,
addProp = c("combined2", "combined1"),
calcTables = TRUE,
compress = TRUE,
adjObf = TRUE,
c1 = 0.95,
sigdig = 4,
sigdigTable = NULL,

nlmeControl(
  maxIter = 100,
pnlsMaxIter = 100,
msMaxIter = 100,
minScale = 0.001,
tolerance = 1e-05,
niterEM = 25,
pnlsTol = 0.001,
msTol = 1e-06,
returnObject = FALSE,
msVerbose = FALSE,
msWarnNoConv = TRUE,
gradiHess = TRUE,
apVar = TRUE,
.relStep = .Machine$double.eps^(1/3),
minAbsParApVar = 0.05,
opt = c("nlminb", "nlm"),
natural = TRUE,
sigma = NULL,
optExpression = TRUE,
sumProd = FALSE,
Arguments

maxIter maximum number of iterations for the \texttt{nlme} optimization algorithm. Default is 50.

pnlsMaxIter maximum number of iterations for the PNLS optimization step inside the \texttt{nlme} optimization. Default is 7.

msMaxIter maximum number of iterations for \texttt{nlminb} (\texttt{iter.max}) or the \texttt{nlm} (\texttt{iterlim}, from the 10-th step) optimization step inside the \texttt{nlme} optimization. Default is 50 (which may be too small for e.g. for overparametrized cases).

minScale minimum factor by which to shrink the default step size in an attempt to decrease the sum of squares in the PNLS step. Default \(0.001\).

tolerance tolerance for the convergence criterion in the \texttt{nlme} algorithm. Default is \(1e^{-6}\).

niterEM number of iterations for the EM algorithm used to refine the initial estimates of the random effects variance-covariance coefficients. Default is 25.

pnlsTol tolerance for the convergence criterion in PNLS step. Default is \(1e^{-3}\).

msTol tolerance for the convergence criterion in \texttt{nlm}, passed as the \texttt{gradtol} argument to the function (see documentation on \texttt{nlm}). Default is \(1e^{-7}\).

returnObject a logical value indicating whether the fitted object should be returned when the maximum number of iterations is reached without convergence of the algorithm. Default is FALSE.

msVerbose a logical value passed as the trace to \texttt{nlminb}(...) or as argument \texttt{print.level} to \texttt{nlm}(). Default is FALSE.

msWarnNoConv logical indicating if a \texttt{warning} should be signalled whenever the minimization by (\texttt{opt}) in the LME step does not converge; defaults to TRUE.

gradHess a logical value indicating whether numerical gradient vectors and Hessian matrices of the log-likelihood function should be used in the \texttt{nlm} optimization. This option is only available when the correlation structure (\texttt{corStruct}) and the variance function structure (\texttt{varFunc}) have no "varying" parameters and the
pdMat classes used in the random effects structure are `pdSymm` (general positive-definite), `pdDiag` (diagonal), `pdIdent` (multiple of the identity), or `pdCompSymm` (compound symmetry). Default is TRUE.

**apVar**
a logical value indicating whether the approximate covariance matrix of the variance-covariance parameters should be calculated. Default is TRUE.

**relStep**
the relative step for numerical derivatives calculations. Default is `.Machine$double.eps^*(1/3)`.

**minAbsParApVar**
numeric value - minimum absolute parameter value in the approximate variance calculation. The default is 0.05.

**opt**
the optimizer to be used, either "nlminb" (the default) or "nlm".

**natural**
a logical value indicating whether the `pdNatural` parametrization should be used for general positive-definite matrices (`pdSymm`) in reStruct, when the approximate covariance matrix of the estimators is calculated. Default is TRUE.

**sigma**
optionally a positive number to fix the residual error at. If NULL, as by default, or 0, sigma is estimated.

**optExpression**
Optimize the rxode2 expression to speed up calculation. By default this is turned on.

**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.

**rxControl**
'rxode2' ODE solving options during fitting, created with 'rxControl()'

**method**
a character string. If "REML" the model is fit by maximizing the restricted log-likelihood. If "ML" the log-likelihood is maximized. Defaults to "ML".

**random**
optionally, any of the following: (i) a two-sided formula of the form \( r_1 + \ldots + r_n \sim x_1 + \ldots + x_m \mid g_1 / \ldots / g_Q \), with \( r_1, \ldots, r_n \) naming parameters included on the right hand side of model, \( x_1 + \ldots + x_m \) specifying the random-effects model for these parameters and \( g_1 / \ldots / g_Q \) the grouping structure (\( Q \) may be equal to 1, in which case no \( / \) is required). The random effects formula will be repeated for all levels of grouping, in the case of multiple levels of grouping; (ii) a two-sided formula of the form \( r_1 + \ldots + r_n \sim x_1 + \ldots + x_m \), a list of two-sided formulas of the form \( r_1 \sim x_1 + \ldots + x_m \), with possibly different random-effects models for different parameters, a pdMat object with a two-sided formula, or list of two-sided formulas (i.e. a non-NULL value for formula(random)), or a list of pdMat objects with two-sided formulas, or lists of two-sided formulas. In this case, the grouping structure formula will be given in groups, or derived from the data used to fit the nonlinear mixed-effects model, which should inherit from class groupedData; (iii) a named list of formulas, lists of formulas, or pdMat objects as in (ii), with the grouping factors as names. The order of nesting will be assumed the same as the order of the order of the elements in the list; (iv) an reStruct object. See the documentation on pdClasses for a description of the available pdMat classes. Defaults to fixed, resulting in all fixed effects having also random effects.

**fixed**
a two-sided linear formula of the form \( f_1 + \ldots + f_n \sim x_1 + \ldots + x_m \), or a list of two-sided formulas of the form \( f_1 \sim x_1 + \ldots + x_m \), with possibly different models for different parameters. The \( f_1, \ldots, f_n \) are the names of parameters included on the right hand side of model and the \( x_1 + \ldots + x_m \) expressions define linear models
for these parameters (when the left hand side of the formula contains several parameters, they all are assumed to follow the same linear model, described by the right hand side expression). A 1 on the right hand side of the formula(s) indicates a single fixed effects for the corresponding parameter(s).

weights

an optional varFunc object or one-sided formula describing the within-group heteroscedasticity structure. If given as a formula, it is used as the argument to varFixed, corresponding to fixed variance weights. See the documentation on varClasses for a description of the available varFunc classes. Defaults to NULL, corresponding to homoscedastic within-group errors.

verbose

an optional logical value. If TRUE information on the evolution of the iterative algorithm is printed. Default is FALSE.

returnNlme

Returns the nlme object instead of the nlmixr object (by default FALSE). If any of the nlme specific options of 'random', 'fixed', 'sens', the nlme object is returned.

addProp

specifies the type of additive plus proportional errors, the one where standard deviations add (combined1) or the type where the variances add (combined2). The combined1 error type can be described by the following equation:

\[ y = f + (a + b*f^c)*\text{err} \]

The combined2 error model can be described by the following equation:

\[ y = f + \sqrt{a^2 + b^2*(f^c)^2} * \text{err} \]

Where:
- \( y \) represents the observed value
- \( f \) represents the predicted value
- \( a \) is the additive standard deviation
- \( b \) is the proportional/power standard deviation
- \( c \) is the power exponent (in the proportional case c=1)

calcTables

This boolean is to determine if the focciFit will calculate tables. By default this is TRUE.

compress

Should the object have compressed items.

adjObf

is a boolean to indicate if the objective function should be adjusted to be closer to NONMEM’s default objective function. By default this is TRUE.

ci

Confidence level for some tables. By default this is 0.95 or 95% confidence.

sigdig

Optimization significant digits. This controls:

- The tolerance of the inner and outer optimization is \( 10^{-\text{sigdig}} \)
- The tolerance of the ODE solvers is \( 0.5*10^{(-\text{sigdig}-2)} \); For the sensitivity equations and steady-state solutions the default is \( 0.5*10^{(-\text{sigdig}-1.5)} \) (sensitivity changes only applicable for liblsoda)
- The tolerance of the boundary check is \( 5*10^{(-\text{sigdig} + 1)} \)

sigdigTable

Significant digits in the final output table. If not specified, then it matches the significant digits in the 'sigdig' optimization algorithm. If 'sigdig' is NULL, use 3.

Further, named control arguments to be passed to nlminb (apart from trace and iter.max mentioned above), where used (eval.max and those from abs.tol down).
nlmixr2Validate

Value

a nlmixr-nlme list

Examples

nlmixr2est::nlmeControl()
nlmixr2NlmeControl()

Description

This allows easy validation/qualification of nlmixr2 by running the testing suite on your system.

Usage

nlmixr2Validate(type = NULL, skipOnCran = TRUE)
nmTest(type = NULL, skipOnCran = TRUE)

Arguments

type of test to be run
skipOnCran when ‘TRUE’ skip the test on CRAN.

Value

Nothing, called for its side effects

Author(s)

Matthew L. Fidler

nlmixr2Version

Display nlmixr2’s version

Description

Display nlmixr2’s version

Usage

nlmixr2Version()
Value

Nothing, called for its side effects

Author(s)

Matthew L. Fidler
nlmixrAddTiming

Manually add time to a nlmixr2 object

Description

Manually add time to a nlmixr2 object

Usage

nlmixrAddTiming(object, name, time)

Arguments

object nlmixr2 object
name string of the timing name
time time (in seconds)

Value

Nothing, called for side effects

Author(s)

Matthew L. Fidler

Examples

one.cmt <- function() {
  ini(
    ## You may label each parameter with a comment
    tka <- 0.45 # Ka
    tcl <- log(c(0, 2.7, 100)) # Log Cl
    ## This works with interactive models
    ## You may also label the preceding line with label("label text")
    tv <- 3.45; label("log V")
    ## the label("Label name") works with all models
    eta.ka ~ 0.6
    eta.cl ~ 0.3
    eta.v ~ 0.1
    add.sd <- 0.7
  )
  model(
    ka <- exp(tka + eta.ka)
    cl <- exp(tcl + eta.cl)
    v <- exp(tv + eta.v)
    linCmt() ~ add(add.sd)
```r
})

fit <- nlmixr(one.cmt, theo_sd, est="saem")

# will add to the current setup
nlmixrAddTiming(fit, "setup", 3)

# Add a new item to the timing dataframe
nlmixrAddTiming(fit, "new", 3)
```

---

**nlmixrCbind**

**Description**

'cbind' for 'nlmixr' objects that preserve the fit information

**Usage**

```
nlmixrCbind(fit, extra)
```

**Arguments**

- `fit`: nlmixr fit
- `extra`: data to cbind to nlmixr fit

**Value**

fit expanded with extra values, without disturbing the fit information

**Author(s)**

Matthew L. Fidler
nlmixrClone

Clone nlmixr environment

Description

Clone nlmixr environment

Usage

nlmixrClone(x)

Arguments

x

nlmixr fit

Value

cloned nlmixr environment

Author(s)

Matthew L. Fidler

Examples

## Not run:

one.cmt <- function() {
  ini({
    ## You may label each parameter with a comment
    tka <- 0.45 # Log Ka
    tcl <- log(c(0, 2.7, 100)) # Log Cl
    ## This works with interactive models
    ## You may also label the preceding line with label("label text")
    tv <- 3.45; label("log V")
    ## the label("Label name") works with all models
    eta.ka ~ 0.6
    eta.cl ~ 0.3
    eta.v ~ 0.1
    add.sd <- 0.7
  })
  model({
    ka <- exp(tka + eta.ka)
    cl <- exp(tcl + eta.cl)
    v <- exp(tv + eta.v)
    linCmt() ~ add(add.sd)
  })
}

f <- nlmixr2(one.cmt, theo_sd, "saem")
nlmixrClone(f)

## End(Not run)

---

### nlmixrWithTiming

**Time a part of a nlmixr operation and add to nlmixr object**

---

#### Description

Time a part of a nlmixr operation and add to nlmixr object

#### Usage

```r
nlmixrWithTiming(name, code, envir = NULL)
```

#### Arguments

- `name`: Name of the timing to be integrated
- `code`: Code to be evaluated and timed
- `envir`: can be either the nlmixr2 fit data, the nlmixr2 fit environment or NULL, which implies it is going to be added to the nlmixr fit when it is finalized. If the function is being called after a fit is created, please supply this environmental variable

#### Value

Result of code

#### Author(s)

Matthew L. Fidler

#### Examples

```r
one.cmt <- function() {
  ini(
    ## You may label each parameter with a comment
    tka <- 0.45 # Ka
    tcl <- log(c(0, 2.7, 100)) # Log Cl
    ## This works with interactive models
    ## You may also label the preceding line with label("label text")
    tv <- 3.45; label("log V")
    ## the label("Label name") works with all models
    eta.ka ~ 0.6
    eta.cl ~ 0.3
  )
}
```
etav ~ 0.1
add.sd <- 0.7
}
model(
ka <- exp(tka + eta.ka)
cl <- exp(tcl + eta.cl)
v <- exp(tv + eta.v)
linCmt() ~ add(add.sd)
}
}
fit <- nlmixr(one.cmt, theo_sd, est="saem")

nlmixrWithTiming("time1", {
 Sys.sleep(1)
 # note this can be nested, time1 will exclude the timing from time2
 nlmixrWithTiming("time2", {
 Sys.sleep(1)
 }, envir=fit)
}, envir=fit)

print(fit)

nmObjGetControl.nlme

Get control object from fit

Description
Get control object from fit

Usage
## S3 method for class 'nlme'
nmObjGetControl(x, ...)

nmObjGetControl(x, ...)

## S3 method for class 'focei'
nmObjGetControl(x, ...)

## S3 method for class 'foce'
nmObjGetControl(x, ...)

## S3 method for class 'foi'
nmObjGetControl(x, ...)

## S3 method for class 'fo'

nmObjGetControl(x, ...)

## S3 method for class 'posthoc'
nmObjGetControl(x, ...)

## S3 method for class 'saem'
nmObjGetControl(x, ...)

## Default S3 method:
nmObjGetControl(x, ...)

**Arguments**

- `x` nlmixr fit object
- `...` Other parameters

**Value**

Control object of estimation method

**Author(s)**

Matthew L. Fidler

---

**nmObjGetEstimationModel**

*Get the estimation model for a fit object depending on the object type*

**Description**

By default it gets the focei models if available.

**Usage**

```r
nmObjGetEstimationModel(x)
```

**Arguments**

- `x` nlmixr fit object

**Value**

returns the estimation ‘$model‘ for the estimation type
Method for getting focei compatible control object from nlmixr object

Description
Method for getting focei compatible control object from nlmixr object

Usage

## S3 method for class 'nlme'
nmObjGetFoceiControl(x, ...)

## Default S3 method:
nmObjGetFoceiControl(x, ...)

## S3 method for class 'saem'
nmObjGetFoceiControl(x, ...)

Arguments

x nlmixr composed fit object
...
Other parameters

Value
foceiControl translated from current control

Get the ipred model for a fit object depending on the object type

Description
By default it gets the focei models if available.

Usage

nmObjGetIpredModel(x)

## S3 method for class 'saem'
nmObjGetIpredModel(x)

## Default S3 method:
nmObjGetPredOnly(x)

## S3 method for class 'saem'
mObjGetEstimationModel(x)

## Default S3 method:
mObjGetEstimationModel(x)

Arguments

x  nlmixr fit object

Value

ipred 'rxode2' model

nmObjGetPredOnly  Get the pred-only model for a fit depending on the object type

Description

By default it gets the focei models if available

Usage

nmObjGetPredOnly(x)

## S3 method for class 'saem'
mObjGetPredOnly(x)

## Default S3 method:
mObjGetPredOnly(x)

Arguments

x  nlmixr fit object

Value

rxode2 pred-only model
nmObjHandleControlObject.nlmeControl

Handle the control object

Description

Handle the control object

Usage

```r
## S3 method for class 'nlmeControl'
nmObjHandleControlObject(control, env)
```

## S3 method for class 'foceiControl'
```
nmObjHandleControlObject(control, env)
```

## S3 method for class 'saemControl'
```
nmObjHandleControlObject(control, env)
```

## Default S3 method:
```
nmObjHandleControlObject(control, env)
```

Arguments

- `control` : Control object
- `env` : fit environment

Value

Nothing, called for side effects

Author(s)

Matthew L. Fidler

nmObjHandleModelObject

Handle Model Object

Description

Handle Model Object
Usage

nmObjHandleModelObject(model, env)

## S3 method for class 'saemModellist'
nmObjHandleModelObject(model, env)

## S3 method for class 'foceiModellist'
nmObjHandleModelObject(model, env)

## Default S3 method:
nmObjHandleModelObject(model, env)

Arguments

model model list should have at least:
- `predOnly` – this is the prediction model with all the left handed equations added so they will be added the table. The model should have `rx_pred_`, the model based prediction, as the first defined lhs component. The second component should be `rx_r_`, the variance of the prediction. These variables may change based on distribution type. In additional all interesting calculated variables should be included.
- `predNoLhs` – This is the prediction model. It only has the prediction and no left handed equations.

env Environment for the fit information

Value

This returns the `$model` object for a fit. It is a s3 method because it may be different between different model types

nmsimplex

Nelder-Mead simplex search

Description

Nelder-Mead simplex search

Usage

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

Arguments

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

**Value**

a list of ...

---

**ofv**

*Return the objective function*

---

**Description**

Return the objective function

**Usage**

```r
ofv(x, type, ...)  
```  

**Arguments**

- **x**  
  object to return objective function value

- **type**  
  Objective function type value to retrieve or add.

  - focei For most models you can specify "focei" and it will add the focei objective function.
  
  - nlme This switches/chooses the nlme objective function if applicable. This objective function cannot be added if it isn’t present.
  
  - fo FO objective function value. Cannot be generated
  
  - foce FOCE object function value. Cannot be generated
  
  - laplace# This adds/retrieves the Laplace objective function value. The # represents the number of standard deviations requested when expanding the Gaussian Quadrature. This can currently only be used with saem fits.
  
  - gauss#.# This adds/retrieves the Gaussian Quadrature approximation of the objective function. The first number is the number of nodes to use in the approximation. The second number is the number of standard deviations to expand upon.

  ... Other arguments sent to ofv for other methods.

**Value**

Objective function value

**Author(s)**

Matthew Fidler
print.saemFit  

*Print an SAEM model fit summary*

**Description**

Print an SAEM model fit summary

**Usage**

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

**Arguments**

- `x`  
  a saemFit object

- `...`  
  others

**Value**

a list

---

residuals.nlmixr2FitData  

*Extract residuals from the FOCEI fit*

**Description**

Extract residuals from the FOCEI fit

**Usage**

```r
## S3 method for class 'nlmixr2FitData'
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

---

Control Options for SAEM

Description
Control Options for SAEM

Usage

```r
saemControl(
  seed = 99,
  nBurn = 200,
  nEm = 300,
  nmc = 3,
  nu = c(2, 2, 2),
  print = 1,
  trace = 0,
  covMethod = c("linFin", "fim", "r,s", "r", "s", ""),
  calcTables = TRUE,
  logLik = FALSE,
  nnodesGq = 3,
  nsdGq = 1.6,
  optExpression = TRUE,
  adjObf = TRUE,
  sumProd = FALSE,
  addProp = c("combined2", "combined1"),
  tol = 1e-06,
  itmax = 30,
  type = c("nelder-mead", "newuoa"),
  powRange = 10,
  lambdaRange = 3,
  odeRecalcFactor = 10^0.5,
  maxOdeRecalc = 5L,
  perSa = 0.75,
  perNoCor = 0.75,
  perFixOmega = 0.1,
  perFixResid = 0.1,
  compress = TRUE,
  rxControl = NULL,
  sigdig = NULL,
)```
Arguments

seed
Random Seed for SAEM step. (Needs to be set for reproducibility.) By default this is 99.

nBurn
Number of iterations in the first phase, i.e., the MCMC/Stochastic Approximation steps. 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 Hastings-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.

print
The number it 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).

"linFim" Use the Linearized Fisher Information Matrix to calculate the covariance.

"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 cross-product matrix to calculate the covariance as \(4 \times S^{-1}\)

"" Does not calculate the covariance step.

calcTables
This boolean is to determine if the foceliFit will calculate tables. By default this is TRUE.
logLik

LogLik is a boolean indicating that log-likelihood should be calculated by Gaussian quadrature.

nnodesGq

NnodesGq is the number of nodes to use for the Gaussian quadrature when computing the likelihood with this method (defaults to 1, equivalent to the Laplacian likelihood).

nsdGq

NsdGq is the span (in SD) over which to integrate when computing the likelihood by Gaussian quadrature. Defaults to 3 (eg 3 times the SD).

optExpression

OptoptExpression is a boolean indicating that the rxode2 expression should be optimized to speed up calculation. By default, this is turned on.

adjObf

AdjObf is a boolean to indicate if the objective function should be adjusted to be closer to NONMEM’s default objective function. By default, this is TRUE.

sumProd

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.

addProp

AddProp specifies the type of additive plus proportional errors, the one where standard deviations add (combined1) or the type where the variances add (combined2). The combined1 error type can be described by the following equation:

\[ y = f + (a + b*f^c)*err \]

The combined2 error model can be described by the following equation:

\[ y = f + \sqrt{a^2 + b^2*(f^c)^2} \cdot err \]

Where:
- \( y \) represents the observed value
- \( f \) represents the predicted value
- \( a \) is the additive standard deviation
- \( b \) is the proportional/power standard deviation
- \( c \) is the power exponent (in the proportional case \( c = 1 \))

tol

Tol is the tolerance for the regression models used for complex residual errors (ie add+prop etc).

itmax

Itmax is the maximum number of iterations for the regression models used for complex residual errors. The number of iterations is itmax*number of parameters.

type

Type indicates the type of optimization for the residuals; Can be one of c("nelder-mead", "newuoa")

powRange

PowRange This indicates the range that powers can take for residual errors; By default this is 10 indicating the range is c(-10, 10)

lambdaRange

LambdaRange This indicates the range that Box-Cox and Yeo-Johnson parameters are constrained to be; The default is 3 indicating the range c(-3,3)

odeRecalcFactor

OdeRecalcFactor The ODE recalculation factor when ODE solving goes bad, this is the factor the rtol/atol is reduced.

maxOdeRecalc

MaxOdeRecalc Maximum number of times to reduce the ODE tolerances and try to resolve the system if there was a bad ODE solve.

perSa

PerSa This is the percent of the time the ‘nBurn’ iterations in phase runs runs a simulated annealing.
This is the percentage of the MCMC phase of the SAEM algorithm where the variance/covariance matrix has no correlations. By default this is 0.75 or 75 Monte-carlo iteration.

This is the percentage of the ‘nBurn’ phase where the omega values are unfixed to allow better exploration of the likelihood surface. After this time, the omegas are fixed during optimization.

This is the percentage of the ‘nBurn’ phase where the residual components are unfixed to allow better exploration of the likelihood surface.

Should the object have compressed items

‘rxode2’ ODE solving options during fitting, created with ‘rxControl()’

Specifies the "significant digits" that the ode solving requests. When specified this controls the relative and absolute tolerances of the ODE solvers. By default the tolerance is \( 0.5 \times 10^{(-\text{sigdig}-2)} \) for regular ODEs. For the sensitivity equations and steady-state solutions the default is \( 0.5 \times 10^{(-\text{sigdig}-1.5)} \) (sensitivity changes only applicable for liblsoda). By default this is unspecified (NULL) and uses the standard atol/rtol.

Significant digits in the final output table. If not specified, then it matches the significant digits in the 'sigdig' optimization algorithm. If 'sigdig' is NULL, use 3.

Confidence level for some tables. By default this is 0.95 or 95% confidence.

This controls if mu-referenced covariates in ‘saem’ are handled differently than non mu-referenced covariates. When ‘TRUE’, mu-referenced covariates have special handling. When ‘FALSE’ mu-referenced covariates are treated the same as any other input parameter.

Other arguments to control SAEM.

List of options to be used in nlmixr2 fit for SAEM.

Wenping Wang & Matthew L. Fidler

**setValue**

Set the covariance type based on prior calculated covariances

Set the covariance type based on prior calculated covariances

```
setCov(fit, method)
```
Arguments

- **fit**: nlmixr2 fit
- **method**: covariance method

Value

Fit object with covariance updated

Author(s)

Matt Fidler

---

**setOfv**  
*Set/get Objective function type for a nlmixr2 object*

Description

Set/get Objective function type for a nlmixr2 object

Usage

- `setOfv(x, type)`
- `getOfvType(x)`

Arguments

- **x**: nlmixr2 fit object
- **type**: Type of objective function to use for AIC, BIC, and $objective

Value

Nothing

Author(s)

Matthew L. Fidler
**sqrtm**

*Return the square root of general square matrix A*

**Description**

Return the square root of general square matrix A

**Usage**

`sqrtm(m)`

**Arguments**

- `m` Matrix to take the square root of.

**Value**

A square root general square matrix of m

---

**summary.saemFit**

*Print an SAEM model fit summary*

**Description**

Print an SAEM model fit summary

**Usage**

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

**Arguments**

- `object` a saemFit object

- `...` others

**Value**

a list
tableControl

Output table/data.frame options

Description

Output table/data.frame options

Usage

tableControl(
    npde = NULL,
    cwres = NULL,
    nsim = 300,
    ties = TRUE,
    censMethod = c("truncated-normal", "cdf", "ipred", "pred", "epred", "omit"),
    seed = 1009,
    cholSEtol = (.Machine$double.eps)^(1/3),
    state = TRUE,
    lhs = TRUE,
    eta = TRUE,
    covariates = TRUE,
    addDosing = FALSE,
    subsetNonmem = TRUE,
    cores = NULL,
    keep = NULL,
    drop = NULL
)

Arguments

npde When TRUE, request npde regardless of the algorithm used.
cwres When TRUE, request CWRES and FOCEi likelihood regardless of the algorithm used.
nsim represents the number of simulations. For rxode2, if you supply single subject event tables (created with \[eventTable()\])
ties When ‘TRUE‘ jitter prediction-discrepancy points to discourage ties in cdf.
censMethod Handle censoring method:
  - “truncated-normal”: Simulates from a truncated normal distribution under the assumption of the model and censoring.
  - “cdf”: Use the cdf-method for censoring with npde and use this for any other residuals (‘cwres’ etc)
  - “omit” omit the residuals for censoring
seed an object specifying if and how the random number generator should be initialized
cholSEtol The tolerance for the ‘rxode2::choleSE’ function
state is a Boolean indicating if ‘state’ values will be included (default ‘TRUE’)

lhs is a Boolean indicating if remaining ‘lhs’ values will be included (default ‘TRUE’)

eta is a Boolean indicating if ‘eta’ values will be included (default ‘TRUE’)

covariates is a Boolean indicating if covariates will be included (default ‘TRUE’)

addDosing Boolean indicating if the solve should add rxode2 EVID and related columns. This will also include dosing information and estimates at the doses. Be default, rxode2 only includes estimates at the observations. (default FALSE). When addDosing is NULL, only include EVID=0 on solve and exclude any model-times or EVID=2. If addDosing is NA the classic rxode2 EVID events are returned. When addDosing is TRUE add the event information in NONMEM-style format; If subsetNonmem=FALSE rxode2 will also include extra event types (EVID) for ending infusion and modeled times:

- EVID=-1 when the modeled rate infusions are turned off (matches rate=-1)
- EVID=-2 When the modeled duration infusions are turned off (matches rate=-2)
- EVID=-10 When the specified rate infusions are turned off (matches rate>0)
- EVID=-20 When the specified dur infusions are turned off (matches dur>0)
- EVID=101,102,103,... Modeled time where 101 is the first model time, 102 is the second etc.

subsetNonmem subset to NONMEM compatible EVIDs only. By default TRUE.

cores Number of cores used in parallel ODE solving. This is equivalent to calling setRxThreads()

keep is the keep sent to the table

drop is the dropped variables sent to the table

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 nlmixr2

Author(s)

Matthew L. Fidler
vpcSim  

**VPC simulation**

**Description**

VPC simulation

**Usage**

```r
code
vpcSim(
  object,
  ...,  
  keep = NULL,
  n = 300,
  pred = FALSE,
  seed = 1009,
  nretry = 50
)
```

**Arguments**

- `object`: This is the nlmixr2 fit object
- `...`: Other arguments sent to `rxSolve()`
- `keep`: Keep character vector
- `n`: Number of simulations
- `pred`: Should predictions be added to the simulation
- `seed`: Seed to set for the VPC simulation
- `nretry`: Number of times to retry the simulation if there is NA values in the simulation

**Value**

data frame of the VPC simulation

**Author(s)**

Matthew L. Fidler

**Examples**

```r
code
one.cmt <- function() {
  ini({
    ## You may label each parameter with a comment
    tka <- 0.45 # Log Ka
    tcl <- log(c(0, 2.7, 100)) # Log Cl
  })
}
```
## This works with interactive models
## You may also label the preceding line with label("label text")

tv <- 3.45; label("log V")
## the label("Label name") works with all models
eta.ka ~ 0.6
eta.cl ~ 0.3
eta.v ~ 0.1
add.sd <- 0.7

})
model({
  ka <- exp(tka + eta.ka)
  cl <- exp(tcl + eta.cl)
  v <- exp(tv + eta.v)
  linCmt() ~ add(add.sd)
})

fit <- nlmixr(one.cmt, theo_sd, est="focei")

head(vpcSim(fit, pred=TRUE))
Index

addCwres, 3, 68
addNpde, 5, 68
addTable, 6
AIC, 32
assertNlmixrFit, 8
assertNlmixrFitData, 8
augPred.nlmixr2FitData
(nlmixr2AugPredSolve), 34
BIC, 32
boxCox, 9
cholSE, 10
fixed.effects, 33
foceiControl, 11
getOfvType (setOfv), 65
getValidNlmixrControl, 21
getValidNlmixrCtl
(ValidNlmixrControl), 21
iBoxCox (boxCox), 9
iYeoJohnson (boxCox), 9
logLik, 32
nlqn1, 20, 21
nlm, 44, 45
nlme, 24, 32
nlmeControl (nlmixr2NlmeControl), 42
nlminb, 44–46
nlmixr (nlmixr2), 22
nlmixr2, 22, 64
nlmixr2AllEst, 34
nlmixr2AugPredSolve, 34
nlmixr2CreateOutputFromUi, 35
nlmixr2Est (nlmixr2Est.focei), 36
nlmixr2Est.focei, 36
nlmixr2Gill183, 38, 40, 41
nlmixr2Hess, 40
nlmixr2Logo, 42
nlmixr2NlmeControl, 42
nlmixr2Validate, 47
nlmixr2Version, 47
nlmixrAddObjectiveFunctionDataFrame
(nlmixrAddObjectiveFunctionDataFrame), 48
nlmixrAddTiming, 49
nlmixrCbind, 50
nlmixrClone, 51
nlmixrWithTiming, 52
nmObjGetControl
(nmObjGetControl.nlme), 53
nmObjGetControl.nlme, 53
nmObjGetEstimationModel, 54
nmObjGetEstimationModel.default
(nmObjGetIpredModel), 55
nmObjGetEstimationModel.saem
(nmObjGetIpredModel), 55
nmObjGetFoceiControl
(nmObjGetFoceiControl.nlme), 55
nmObjGetFoceiControl.nlme, 55
nmObjGetIpredModel, 55
nmObjGetPredOnly, 56
nmObjHandleControlObject
(nmObjHandleControlObject.nlmeControl), 57
nmObjHandleControlObject.nlmeControl, 57
nmObjHandleModelObject, 57
nmsimplex, 58
nmTest (nlmixr2Validate), 47
ofv, 59
optim, 20, 21
optimHess, 41
print.saemFit, 60
random.effects, 33
residuals.nlmixr2FitData, 60
rvode2, 25
rvxSolve, 21
saemControl, 31, 61
setCov, 64
setOfv, 65
setRxThreads(), 68
sqrtm, 66
summary.saemFit, 66
tableControl, 67
vpcSim, 69
warning, 44
yeoJohnson (boxCox), 9