Package 'nlmixr2'

June 28, 2022

Title Nonlinear Mixed Effects Models in Population PK/PD

Version 2.0.7

Description Fit and compare nonlinear mixed-effects models in differential equations with flexible dosing information commonly seen in pharmacokinetics and pharmacodynamics (Almquist, Leander, and Jirstrand 2015 <doi:10.1007/s10928-015-9409-1>). Differential equation solving is by compiled C code provided in the 'rxode2' package (Wang, Hallow, and James 2015 <doi:10.1002/psp4.12052>).

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.2.0

Imports nlmixr2est, nlmixr2extra, rxode2, nlmixr2plot, magrittr

Depends nlmixr2data

Suggests rmarkdown, knitr, devtools, ggplot2, testthat, n1qn1, withr

VignetteBuilder knitr

BugReports https://github.com/nlmixr2/nlmixr2/issues/

URL https://nlmixr2.org/, https://github.com/nlmixr2/nlmixr2/

NeedsCompilation no

Author Matthew Fidler [aut, cre] (<https://orcid.org/0000-0001-8538-6691>),
Yuan Xiong [ctb],
Rik Schoemaker [ctb] (<https://orcid.org/0000-0002-7538-3005>),
Justin Wilkins [ctb] (<https://orcid.org/0000-0002-7099-9396>),
Wenping Wang [ctb],
Mirjam Trame [ctb],
Huijuan Xu [ctb],
John Harrold [ctb],
Bill Denney [ctb] (<https://orcid.org/0000-0002-5759-428X>),
Theodoros Papathanasiou [ctb],
Teun Post [ctb],
Richard Hooijmaijers [ctb]

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

Repository CRAN

Date/Publication 2022-06-27 22:20:02 UTC

R topics documented:

`addCwres` *Add CWRES*

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 foce 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

```
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
```

addNpde *NPDE calculation for nlmixr2*

Description

NPDE calculation for nlmixr2

Usage

```
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
...	Additional arguments passed to nlmixr2est::addNpde().
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

```
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

```
addTable(
  object,
  updateObject = FALSE,
  data = object$dataSav,
  thetaEtaParameters = object$foceiThetaEtaParameters,
  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

table	a ‘tableControl()‘ list of options
keep	Character Vector of items to keep
drop	Character Vector of items to drop or NULL
envir	Environment to search for updating

Value

Fit with table information attached

Author(s)

Matthew Fidler

Examples

```
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)
```

bootplot *Produce delta objective function for boostrap*

Description

Produce delta objective function for boostrap

Usage

```
bootplot(x, ...)
```

Arguments

x	fit object
...	Additional arguments passed to `nlmixr2extra::bootplot()`.

Value

Fit traceplot or nothing.

Author(s)

Vipul Mann, Matthew L. Fidler

References

R Niebecker, MO Karlsson. (2013) *Are datasets for NLME models large enough for a bootstrap to provide reliable parameter uncertainty distributions?* PAGE 2013. https://www.page-meeting.org/?abstract=2899

bootstrapFit *Bootstrap nlmixr2 fit*

Description

Bootstrap input dataset and rerun the model to get confidence bounds and aggregated parameters

Usage

```
bootstrapFit(
  fit,
  nboot = 200,
  nSampIndiv,
  stratVar,
  stdErrType = c("perc", "se"),
  ci = 0.95,
  pvalues = NULL,
  restart = FALSE,
  plotHist = FALSE,
  fitName = as.character(substitute(fit))
)
```

Arguments

fit	the nlmixr2 fit object
nboot	an integer giving the number of bootstrapped models to be fit; default value is 200
nSampIndiv	an integer specifying the number of samples in each bootstrapped sample; default is the number of unique subjects in the original dataset
stratVar	Variable in the original dataset to stratify on; This is useful to distinguish between sparse and full sampling and other features you may wish to keep distinct in your bootstrap
stdErrType	This gives the standard error type for the updated standard errors; The current possibilities are: "perc" which gives the standard errors by percentiles (default) or "se" which gives the standard errors by the traditional formula.
ci	Confidence interval level to calculate. Default is 0.95 for a 95 percent confidence interval
pvalues	a vector of pvalues indicating the probability of each subject to get selected; default value is NULL implying that probability of each subject is the same
restart	A boolean to try to restart an interrupted or incomplete boostrap. By default this is FALSE
plotHist	A boolean indicating if a histogram plot to assess how well the bootstrap is doing. By default this is turned off (FALSE)
fitName	is the fit name that is used for the name of the boostrap files. By default it is the fit provided though it could be something else.

Value

Nothing, called for the side effects; The original fit is updated with the bootstrap confidence bands

Author(s)

Vipul Mann, Matthew Fidler

Examples

```
one.cmt <- function() {
  ini({
    ## You may label each parameter with a comment
    tka <- 0.45 # Log Ka
    tcl <- 1 # 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 <- nlmixr2(one.cmt, nlmixr2data::theo_sd, "focei")

withr::with_tempdir({ # Run example in temp dir

bootstrapFit(fit, nboot = 5, restart = TRUE) # overwrites any of the existing data or model files
bootstrapFit(fit, nboot = 7) # resumes fitting using the stored data and model files

# Note this resumes because the total number of bootstrap samples is not 50

bootstrapFit(fit, nboot=50)

# Note the boostrap standard error and variance/covariance matrix is retained.
# If you wish to switch back you can change the covariance matrix by

nlmixr2est::setCov(fit,"r,s")

# And change it back again

nlmixr2est::setCov(fit,"boot50")

# This change will affect any simulations with uncertainty in their parameters

# You may also do a chi-square diagnostic plot check for the bootstrap with

bootplot(fit)
```

```
})
```

`covarSearchAuto` *Stepwise Covariate Model-selection (SCM) method*

Description

Stepwise Covariate Model-selection (SCM) method

Usage

```
covarSearchAuto(
  fit,
  varsVec,
  covarsVec,
  pVal = list(fwd = 0.05, bck = 0.01),
  covInformation = NULL,
  catCovariates = NULL,
  searchType = c("scm", "forward", "backward"),
  restart = FALSE
)
```

Arguments

`fit`	an nlmixr2 'fit' object
`varsVec`	a list of candidate variables to which the covariates could be added
`covarsVec`	a list of candidate covariates that need to be tested
`pVal`	a named list with names 'fwd' and 'bck' for specifying the p-values for the forward and backward searches, respectively
`covInformation`	a list containing additional information on the variables-covariates pairs that should be passed on to addCovMultiple function
`catCovariates`	a list of covariates that should be treated as categorical
`searchType`	one of 'scm', 'forward' and 'backward' to specify the covariate search method; default is 'scm'
`restart`	a boolean that controls if the search should be restarted; default is FALSE

Value

A list summarizing the covariate selection steps and output; This list has the "summaryTable" for the overall summary of the covariate selection as well as "resFwd" for the forward selection method and "resBck" for the backward selection method.

Author(s)

Vipul Mann, Matthew Fidler

Examples

```
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 <- nlmixr2(one.cmt, nlmixr2data::theo_sd,"focei")
rxode2::.rxWithWd(tempdir(), {# with temporary directory

auto1 <- covarSearchAuto(fit, varsVec = c("ka", "cl"),
    covarsVec = c("WT", "SEX"), catCovariates = c("SEX"))

})

## Note that this didn't include sex, add it to dataset and restart model


d <- nlmixr2data::theo_sd
d$SEX <-0
d$SEX[d$ID<=6] <-1

fit <- nlmixr2(one.cmt, d, "focei")

# This would restart if for some reason the search crashed:

rxode2::.rxWithWd(tempdir(), {# with temporary directory

auto2 <- covarSearchAuto(fit, varsVec = c("ka", "cl"), covarsVec = c("WT", "SEX"),
                catCovariates = c("SEX"), restart = TRUE)
```

```
auto3 <- covarSearchAuto(fit, varsVec = c("ka", "cl"), covarsVec = c("WT", "SEX"),
                catCovariates = c("SEX"), restart = TRUE,
                searchType = "forward")
})
```

`foceiControl` *Control Options for FOCEi*

Description

Control Options for FOCEi

Usage

```
foceiControl(
  sigdig = 3,
  ...,
  epsilon = NULL,
  maxInnerIterations = 1000,
  maxOuterIterations = 5000,
  n1qn1nsim = 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$double.eps)^(1/3),
  cholAccept = 0.001,
  resetEtaP = 0.15,
  resetThetaP = 0.05,
  resetThetaFinalP = 0.15,
  diagOmegaBoundUpper = 5,
  diagOmegaBoundLower = 100,
  cholSEOpt = FALSE,
  cholSECov = FALSE,
  fo = FALSE,
  covTryHarder = FALSE,
 outerOpt = c("nlminb", "bobyqa", "lbfgsb3c", "L-BFGS-B", "mma", "lbfgsbLG", "slsqp",
    "Rvmmin"),
  innerOpt = c("n1qn1", "BFGS"),
  rhobeg = 0.2,
  rhoend = NULL,
  npt = NULL,
  rel.tol = NULL,
  x.tol = NULL,
  eval.max = 4000,
  iter.max = 2000,
  abstol = NULL,
  reltol = NULL,
  resetHessianAndEta = FALSE,
  stateTrim = Inf,
  gillK = 10L,
  gillStep = 4,
  gillFtol = 0,
  gillRtol = sqrt(.Machine$double.eps),
  gillKcov = 10L,
  gillStepCov = 2,
  gillFtolCov = 0,
  rmatNorm = TRUE,
  smatNorm = TRUE,
  covGillF = TRUE,
  optGillF = TRUE,
  covSmall = 1e-05,
  adjLik = TRUE,
```

```
  gradTrim = Inf,
  maxOdeRecalc = 5,
  odeRecalcFactor = 10^(0.5),
  gradCalcCentralSmall = 1e-04,
  gradCalcCentralLarge = 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,
  etaMat = NULL,
  repeatGillMax = 3,
  stickyRecalcN = 5,
  gradProgressOfvTime = 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^-sigdig` • The tolerance of the ODE solvers is `0.5*10^(-sigdig-2)`; For the sensitivity equations and steady-state solutions the default is `0.5*10^(-sigdig-1.5)` (sensitivity changes only applicable for liblsoda) • The tolerance of the boundary check is `5 * 10 ^ (-sigdig + 1)`
`...`	Additional arguments passed to `nlmixr2est::foceiControl()`.
`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)

`normType` 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` The rescaling is the same type described in the OptdesX software manual.

In general, all all scaling formula can be described by:

v_scaled = (v_unscaled-C_1)/C_2

Where

The other data normalization approaches follow the following formula

v_scaled = (v_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 = (max(all unscaled values)+min(all unscaled values))/2
  C_2 = (max(all unscaled values) - min(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 = min(all unscaled values)
  C_2 = max(all unscaled values) - min(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 = mean(all unscaled values)
  C_2 = max(all unscaled values) - min(all unscaled values)
- `std` or standardization. This standardizes by the mean and standard deviation. In this approach:
  C_1 = mean(all unscaled values)
  C_2 = sd(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 + ... + v_n^2)
- `constant` which does not perform data normalization. That is
  C_1 = 0
  C_2 = 1

`scaleType` The scaling scheme for nlmixr2. The supported types are:

- `nlmixr2` In this approach the scaling is performed by the following equation:
  v_scaled = (v_current - v_init)/scaleC[i] + 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_scaled = v_current/v_init*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_scaled = (v_current-v_init) + scaleTo
  Otherwise the parameter is scaled multiplicatively.
  v_scaled = v_current/v_init*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 ini_value (to scale by 1/value; ie d/dt(log(ini_value)) = 1/ini_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*abs(initial_estimate)
- Factorials are scaled by abs(1/digamma(initial_estimate+1))
- parameters in a log scale (ie log(theta)) are transformed by log(abs(initial_estimate))*abs(initial_estim

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 = abs(x)*derivEps[1] + 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 abs(delta(OFV)) <= derivSwitchTol and switch back to forward differences when abs(delta(OFV)) > derivSwitchTol.

`derivSwitchTol` The tolerance to switch forward to central differences.

`covDerivMethod` indicates the method for calculating the derivatives while calculating the covariance components (Hessian and S).

`covMethod` Method for calculating covariance. In this discussion, R is the Hessian matrix of the objective function. The S matrix is the sum of individual gradient cross-product (evaluated at the individual empirical Bayes estimates).

- `"r,s"` Uses the sandwich matrix to calculate the covariance, that is: `solve(R) %*% S %*% solve(R)`

- "r" Uses the Hessian matrix to calculate the covariance as 2 %*% solve(R)
- "s" Uses the cross-product matrix to calculate the covariance as 4 %*% 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 = abs(x)*derivEps[1] + 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(\| proj g_i \| i = 1, ..., n) <= lbfgsPgtol

where 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 chol(solve(omega)). This matrix and values are the parameters estimated in FOCEi. The possibilities are:

- sqrt Estimates the sqrt of the diagonal elements of chol(solve(omega)). This is the default method.
- log Estimates the log of the diagonal elements of chol(solve(omega))
- 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 foceiFit will calculate tables. By default this is `TRUE`

`noAbort` Boolean to indicate if you should abort the FOCEi evaluation if it runs into troubles. (default TRUE)

`interaction` Boolean indicate FOCEi should be used (TRUE) instead of FOCE (FALSE)

`cholSEtol` tolerance for Generalized Cholesky Decomposition. Defaults to suggested (.Machine$double.eps)^(1/3)

`cholAccept` Tolerance to accept a Generalized Cholesky Decomposition for a R or S matrix.

`resetEtaP` represents the p-value for reseting 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 reseting 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 reseting 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.

`cholSEOpt` Boolean indicating if the generalized Cholesky should be used while optimizing.

`cholSECov` Boolean indicating if the generalized Cholesky should be used while calculating the Covariance Matrix.

`fo` is a boolean indicating if this is a FO approximation routine.

`covTryHarder` If the R matrix is non-positive definite and cannot be corrected to be non-positive definite try estimating the Hessian on the unscaled parameter space.

`outerOpt` optimization method for the outer problem

`innerOpt` optimization method for the inner problem (not implemented yet.)

`rhobeg` Beginning change in parameters for bobyqa algorithm (trust region). By default this is 0.2 or 20 parameters when the parameters are scaled to 1. rhobeg and rhoend must be set to the initial and final values of a trust region radius, so both must be positive with 0 < 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).

`x.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 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 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 n1qn1 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)*err

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

y = f + sqrt(a^2 + b^2*(f^c)^2)*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)

`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

See Also

optim

n1qn1

rxSolve

---

nlmeControl *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

```
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,
  gradHess = TRUE,
  apVar = TRUE,
  .relStep = .Machine$double.eps^(1/3),
  minAbsParApVar = 0.05,
  opt = c("nlminb", "nlm"),
  natural = TRUE,
  sigma = NULL,
  optExpression = TRUE,
  sumProd = FALSE,
  rxControl = 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,
  ci = 0.95,
  sigdig = 4,
  sigdigTable = NULL,
  ...
)
```

Arguments

maxIter	maximum number of iterations for the nlme optimization algorithm. Default is 50.
pnlsMaxIter	maximum number of iterations for the PNLS optimization step inside the nlme optimization. Default is 7.
msMaxIter	maximum number of iterations for nlminb (iter.max) or the nlm (iterlim, from the 10-th step) optimization step inside the 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 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 nlm, passed as the gradtol argument to the function (see documentation on 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 nlminb(.., control= list(trace = *, ..)) or as argument print.level to nlm(). Default is FALSE.
msWarnNoConv	logical indicating if a warning should be signalled whenever the minimization by (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 nlm optimization. This option is only available when the correlation structure (corStruct) and the variance function structure (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	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 r1+...+rn~x1+...+xm | g1/.../gQ, with r1,...,rn naming parameters included on the right hand side of model, x1+...+xm specifying the random-effects model for these parameters and g1/.../gQ 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 r1+...+rn~x1+..+xm, a list of two-sided formulas of the form r1~x1+...+xm, 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 f1+...+fn~x1+...+xm, or a list of two-sided formulas of the form f1~x1+...+xm, with possibly different models for different parameters. The f1,...,fn are the names of parameters included on the right hand side of model and the x1+...+xm 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)*err

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

y = f + sqrt(a^2 + b^2*(f^c)^2)*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 foceiFit 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^-sigdig`
- The tolerance of the ODE solvers is `0.5*10^(-sigdig-2)`; For the sensitivity equations and steady-state solutions the default is `0.5*10^(-sigdig-1.5)` (sensitivity changes only applicable for liblsoda)
- The tolerance of the boundary check is `5 * 10 ^ (-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.

`...` Additional arguments passed to `nlmixr2est::nlmeControl()`.

Value

a nlmixr-nlme list

Examples

```
nlmixr2est::nlmeControl()
nlmixr2NlmeControl()
```

`nlmixr2` *nlmixr2 fits population PK and PKPD non-linear mixed effects models.*

Description

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

Usage

```
nlmixr2(
  object,
  data,
  est = NULL,
  control = list(),
  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()')
`...`	Additional arguments passed to `nlmixr2est::nlmixr2()`.
`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 a R function with two blocks: `ini` and `model`. This R function can be named anything, but is not meant to be called directly from R. In fact if you try you will likely get an error such as `Error: could not find function "ini"`.

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

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

```
f <- function(){ ## Note the arguments to the function are currently
                 ## ignored by nlmixr
    ini({
        ## Initial conditions for population parameters (sometimes
        ## called theta parameters) are defined by either `<-` or '='
        lCl <- 1.6      #log Cl (L/hr)
        ## Note that simple expressions that evaluate to a number are
        ## OK for defining initial conditions (like in R)
        lVc = log(90)  #log V (L)
        ## Also a comment on a parameter is captured as a parameter label
        lKa <- 1 #log Ka (1/hr)
        ## Bounds may be specified by c(lower, est, upper), like NONMEM:
        ## Residuals errors are assumed to be population parameters
        prop.err <- c(0, 0.2, 1)
    })
    ## The model block will be discussed later
    model({})
}
```

As shown in the above examples:

- Simple parameter values are specified as a R-compatible assignment
- Boundaries my be specified by `c(lower, est, upper)`.

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

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

- Residual error estimates should be coded as population estimates (i.e. using an '=' or '<-' statement, not a '~').
- Naming variables that start with "_" are not supported. Note that R does not allow variable starting with "_" to be assigned without quoting them.
- Naming variables that start with "`rx_`" or "`nlmixr_`" is not supported since 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

```
f <- function(){
    ini({
        lCl <- 1.6      #log Cl (L/hr)
        lVc = log(90)  #log V (L)
        lKa <- 1 #log Ka (1/hr)
        prop.err <- c(0, 0.2, 1)
        ## Initial estimate for ka IIV variance
        ## Labels work for single parameters
        eta.ka ~ 0.1 # BSV Ka

        ## For correlated parameters, you specify the names of each
        ## correlated parameter separated by a addition operator `+`
        ## and the left handed side specifies the lower triangular
        ## matrix initial of the covariance matrix.
        eta.cl + eta.vc ~ c(0.1,
                            0.005, 0.1)
        ## Note that labels do not currently work for correlated
        ## parameters.  Also do not put comments inside the lower
        ## triangular matrix as this will currently break the model.
    })
```

```
    ## The model block will be discussed later
    model({})
}
```

As shown in the above examples:

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

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

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

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

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

Continuing describing the syntax with an annotated example:

```
f <- function(){
    ini({
        lCl <- 1.6      #log Cl (L/hr)
        lVc <- log(90)   #log Vc (L)
        lKA <- 0.1      #log Ka (1/hr)
        prop.err <- c(0, 0.2, 1)
        eta.Cl ~ 0.1 ## BSV Cl
        eta.Vc ~ 0.1 ## BSV Vc
        eta.KA ~ 0.1 ## BSV Ka
    })
    model({
        ## First parameters are defined in terms of the initial estimates
        ## parameter names.
        Cl <- exp(lCl + eta.Cl)
        Vc = exp(lVc + eta.Vc)
        KA <- exp(lKA + eta.KA)
        ## After the differential equations are defined
        kel <- Cl / Vc;
        d/dt(depot)    = -KA*depot;
        d/dt(centr)  =  KA*depot-kel*centr;
        ## And the concentration is then calculated
        cp = centr / Vc;
        ## Last, nlmixr is told that the plasma concentration follows
        ## a proportional error (estimated by the parameter prop.err)
        cp ~ prop(prop.err)
    })
}
```

A few points to note:

- Parameters are 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:

```
##'

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:

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

Omega ($omega):
     [,1] [,2] [,3]
[1,]  0.1  0.0  0.0
[2,]  0.0  0.1  0.0
[3,]  0.0  0.0  0.1

## Model:
################################################################################
Cl <- exp(lCl + eta.Cl)
Vc = exp(lVc + eta.Vc)
KA <- exp(lKA + eta.KA)
## Instead of specifying the ODEs, you can use
## the linCmt() function to use the solved system.
##
## This function determines the type of PK solved system
## to use by the parameters that are defined.  In this case
## it knows that this is a one-compartment model with first-order
## absorption.
linCmt() ~ prop(prop.err)
```

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

Using the model syntax for estimating a model

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

This dataset has to have RxODE compatible events IDs. Both Monolix and NONMEM use a a very similar standard to what nlmixr can support.

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

The method to estimate the model is:

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

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

```
> f <- function(){
    ini({
        lCl <- 1.6      #log Cl (L/hr)
        lVc <- log(90)   #log Vc (L)
        lKA <- 0.1      #log Ka (1/hr)
        prop.err <- c(0, 0.2, 1)
        eta.Cl ~ 0.1 ## BSV Cl
        eta.Vc ~ 0.1 ## BSV Vc
        eta.KA ~ 0.1 ## BSV Ka
    })
    model({
        ## First parameters are defined in terms of the initial estimates
        ## parameter names.
        Cl <- exp(lCl + eta.Cl)
        Vc = exp(lVc + eta.Vc)
        KA <- exp(lKA + eta.KA)
        ## After the differential equations are defined
        kel <- Cl / Vc;
        d/dt(depot)    = -KA*depot;
        d/dt(centr)  =  KA*depot-kel*centr;
        ## And the concentration is then calculated
        cp = centr / Vc;
        ## Last, nlmixr is told that the plasma concentration follows
        ## a proportional error (estimated by the parameter prop.err)
        cp ~ prop(prop.err)
    })
}
> fit.s <- nlmixr(f,d,est="saem",control=saemControl(n.burn=50,n.em=100,print=50));
Compiling RxODE differential equations...done.
c:/Rtools/mingw_64/bin/g++ -I"c:/R/R-34~1.1/include" -DNDEBUG   -I"d:/Compiler/gcc-4.9.3/local330/i
In file included from c:/R/R-34~1.1/library/RCPPAR~1/include/armadillo:52:0,
            from c:/R/R-34~1.1/library/RCPPAR~1/include/RcppArmadilloForward.h:46,
                 from c:/R/R-34~1.1/library/RCPPAR~1/include/RcppArmadillo.h:31,
                  from saem3090757b4bd1x64.cpp:1:
c:/R/R-34~1.1/library/RCPPAR~1/include/armadillo_bits/compiler_setup.hpp:474:96: note: #pragma messa
  #pragma message ("WARNING: use of OpenMP disabled; this compiler doesn't support OpenMP 3.0+")
                                                                                                ^
```

```
c:/Rtools/mingw_64/bin/g++ -shared -s -static-libgcc -o saem3090757b4bd1x64.dll tmp.def saem3090757b4b
done.
1:    1.8174   4.6328   0.0553   0.0950   0.0950   0.0950   0.6357
50:    1.3900   4.2039   0.0001   0.0679   0.0784   0.1082   0.1992
100:    1.3894   4.2054   0.0107   0.0686   0.0777   0.1111   0.1981
150:    1.3885   4.2041   0.0089   0.0683   0.0778   0.1117   0.1980
Using sympy via SnakeCharmR
## Calculate ETA-based prediction and error derivatives:
Calculate Jacobian...................done.
Calculate sensitivities.......
done.
## Calculate d(f)/d(eta)
## ...
## done
## ...
## done
The model-based sensitivities have been calculated
Calculating Table Variables...
done
```

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

nlmixr output objects

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

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

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

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

 -- Parameters (fit.s$par.fixed): -----------------------------------------------
              Parameter Estimate     SE
 lCl      log Cl (L/hr)     1.39 0.0240  1.73       4.01 (3.83, 4.20)    26.6
 lVc         log Vc (L)     4.20 0.0256 0.608       67.0 (63.7, 70.4)    28.5
 lKA      log Ka (1/hr)  0.00924 0.0323  349.      1.01 (0.947, 1.08)    34.3
 prop.err      prop.err    0.198                            19.8
          Shrink(SD)
 lCl          0.248
 lVc           1.09
```

```
lKA           4.19
prop.err      1.81

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

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

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

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

  - Special access to fit information (like the `$omega`) needs nlmixr to extract the information.
  - If you use the `$` to access information, the order of precedence is:
    * Fit data from the overall data.frame
    * Information about the parsed nlmixr model (via `$uif`)
    * Parameter history if available (via `$par.hist` and `$par.hist.stacked`)
    * Fixed effects table (via `$par.fixed`)
    * Individual differences from the typical population parameters (via `$eta`)
    * Fit information from the list of information generated during the post-hoc residual calculation.
    * Fit information from the environment where the post-hoc residual were calculated
    * Fit information about how the data and options interacted with the specified model (such as estimation options or if the solved system is for an infusion or an IV bolus).
  - While the printout may displays the data as a `data.table` object or `tbl` object, the data is NOT any of these objects, but rather a derived data frame.
  - Since the object *is* a data.frame, you can treat it like one.

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

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

Author(s)

Matthew L. Fidler

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
   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")
```

`preconditionFit` *Linearly re-parameterize the model to be less sensitive to rounding errors*

Description

Linearly re-parameterize the model to be less sensitive to rounding errors

Usage

```
preconditionFit(fit, estType = c("full", "posthoc", "none"), ntry = 10L)
```

Arguments

`fit`	A nlmixr2 fit to be preconditioned
`estType`	Once the fit has been linearly reparameterized, should a "full" estimation, "posthoc" estimation or simply a estimation of the covariance matrix "none" before the fit is updated
`ntry`	number of tries before giving up on a pre-conditioned covariance estimate

Value

A nlmixr2 fit object that was preconditioned to stabilize the variance/covariance calculation

References

Aoki Y, Nordgren R, Hooker AC. Preconditioning of Nonlinear Mixed Effects Models for Stabilisation of Variance-Covariance Matrix Computations. AAPS J. 2016;18(2):505-518. doi:10.1208/s12248-016-9866-5

`saemControl` *Control Options for SAEM*

Description

Control Options for SAEM

Usage

```
saemControl(
  seed = 99,
  nBurn = 200,
  nEm = 300,
  nmc = 3,
  nu = c(2, 2, 2),
  print = 1,
  trace = 0,
  covMethod = c("linFim", "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,
  sigdigTable = NULL,
  ci = 0.95,
  muRefCov = TRUE,
  ...
)
```

Arguments

seed	Random Seed for SAEM step. (Needs to be set for reproducibility.) By default this is 99.
nBurn	Number of iterations in the first phase, ie 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 Hasting-Metropolis algorithm. The default value is c(2,2,2), representing 40 for each transition initially (each value is multiplied by 20). The first value represents the initial number of multi-variate Gibbs samples are taken from a normal distribution. The second value represents the number of uni-variate, or multi- dimensional random walk Gibbs samples are taken. The third value represents the number of bootstrap/reshuffling or uni-dimensional random samples are taken.
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 crossproduct matrix to calculate the covariance as $4 \times S^-1$ "" Does not calculate the covariance step.
calcTables	This boolean is to determine if the foceiFit will calculate tables. By default this is TRUE
logLik	boolean indicating that log-likelihood should be calculate by Gaussian quadrature.
nnodesGq	number of nodes to use for the Gaussian quadrature when computing the likelihood with this method (defaults to 1, equivalent to the Laplaclian likelihood)
nsdGq	span (in SD) over which to integrate when computing the likelihood by Gaussian quadrature. Defaults to 3 (eg 3 times the SD)

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

`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` 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` 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)*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` This is the tolerance for the regression models used for complex residual errors (ie add+prop etc)

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

`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` 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`

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

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

`perSa` This is the percent of the time the 'nBurn' iterations in phase runs runs a simulated annealing.

`perNoCor` 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.

`perFixOmega` 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.

perFixResid	This is the percentage of the ‘nBurn‘ phase where the residual components are unfixed to allow better exploration of the likelihood surface.
compress	Should the object have compressed items
rxControl	‘rxode2‘ ODE solving options during fitting, created with ‘rxControl()‘
sigdig	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*10^(-sigdig-2)` for regular ODEs. For the sensitivity equations and steady-state solutions the default is `0.5*10^(-sigdig-1.5)` (sensitivity changes only applicable for liblsoda). By default this is unspecified (`NULL`) and uses the standard `atol`/`rtol`.
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.
ci	Confidence level for some tables. By default this is 0.95 or 95% confidence.
muRefCov	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.
...	Additional arguments passed to `nlmixr2est::saemControl()`.

Value

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

Author(s)

Wenping Wang & Matthew L. 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)
```

Arguments

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

Value

Nothing

Author(s)

Matthew L. Fidler

`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

traceplot *Produce trace-plot for fit if applicable*

Description

Produce trace-plot for fit if applicable

Usage

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

Arguments

x	fit object
...	Additional arguments passed to nlmixr2plot::traceplot().

Value

Fit traceplot or nothing.

Author(s)

Rik Schoemaker, Wenping Wang & Matthew L. Fidler

Examples

```
library(nlmixr2est)
## The basic model consiss of an ini block that has initial estimates
one.compartment <- function() {
  ini({
    tka <- 0.45 # Log Ka
    tcl <- 1 # Log Cl
    tv <- 3.45    # Log V
    eta.ka ~ 0.6
    eta.cl ~ 0.3
    eta.v ~ 0.1
    add.sd <- 0.7
  })
  # and a model block with the error sppecification and model specification
  model({
    ka <- exp(tka + eta.ka)
    cl <- exp(tcl + eta.cl)
    v <- exp(tv + eta.v)
    d/dt(depot) = -ka * depot
    d/dt(center) = ka * depot - cl / v * center
    cp = center / v
    cp ~ add(add.sd)
```

```
  })
}

## The fit is performed by the function nlmixr/nlmix2 specifying the model, data and estimate
fit <- nlmixr2(one.compartment, theo_sd,  est="saem", saemControl(print=0))

# This shows the traceplot of the fit (useful for saem)
traceplot(fit)
```

---

vpcPlot *VPC based on ui model*

Description

VPC based on ui model

Usage

```
vpcPlot(
  fit,
  data = NULL,
  n = 300,
  bins = "jenks",
  n_bins = "auto",
  bin_mid = "mean",
  show = NULL,
  stratify = NULL,
  pred_corr = FALSE,
  pred_corr_lower_bnd = 0,
  pi = c(0.05, 0.95),
  ci = c(0.05, 0.95),
  uloq = NULL,
  lloq = NULL,
  log_y = FALSE,
  log_y_min = 0.001,
  xlab = NULL,
  ylab = NULL,
  title = NULL,
  smooth = TRUE,
  vpc_theme = NULL,
  facet = "wrap",
  scales = "fixed",
  labeller = NULL,
  vpcdb = FALSE,
  verbose = FALSE,
  ...,
```

```
  seed = 1009
)
```

Arguments

fit	nlmixr2 fit object
data	this is the data to use to augment the VPC fit. By default is the fitted data, (can be retrieved by getData), but it can be changed by specifying this argument.
n	Number of VPC simulations. By default 100
bins	either "density", "time", or "data", "none", or one of the approaches available in classInterval() such as "jenks" (default) or "pretty", or a numeric vector specifying the bin separators.
n_bins	when using the "auto" binning method, what number of bins to aim for
bin_mid	either "mean" for the mean of all timepoints (default) or "middle" to use the average of the bin boundaries.
show	what to show in VPC (obs_dv, obs_ci, pi, pi_as_area, pi_ci, obs_median, sim_median, sim_median_ci)
stratify	character vector of stratification variables. Only 1 or 2 stratification variables can be supplied.
pred_corr	perform prediction-correction?
pred_corr_lower_bnd	lower bound for the prediction-correction
pi	simulated prediction interval to plot. Default is c(0.05, 0.95),
ci	confidence interval to plot. Default is (0.05, 0.95)
uloq	Number or NULL indicating upper limit of quantification. Default is NULL.
lloq	Number or NULL indicating lower limit of quantification. Default is NULL.
log_y	Boolean indicting whether y-axis should be shown as logarithmic. Default is FALSE.
log_y_min	minimal value when using log_y argument. Default is 1e-3.
xlab	label for x axis
ylab	label for y axis
title	title
smooth	"smooth" the VPC (connect bin midpoints) or show bins as rectangular boxes. Default is TRUE.
vpc_theme	theme to be used in VPC. Expects list of class vpc_theme created with function vpc_theme()
facet	either "wrap", "columns", or "rows"
scales	either "fixed" (default), "free_y", "free_x" or "free"
labeller	ggplot2 labeller function to be passed to underlying ggplot object
vpcdb	Boolean whether to return the underlying vpcdb rather than the plot
verbose	show debugging information (TRUE or FALSE)
...	Additional arguments passed to nlmixr2plot::vpcPlot().
seed	an object specifying if and how the random number generator should be initialized

Value

Simulated dataset (invisibly)

Author(s)

Matthew L. Fidler

Examples

```
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 <- nlmixr2est::nlmixr(one.cmt, nlmixr2data::theo_sd, est="focei")

vpcPlot(fit)
```

---

vpcSim *VPC simulation*

Description

VPC simulation

Usage

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

Arguments

object	This is the nlmixr2 fit object
...	Additional arguments passed to nlmixr2est::vpcSim().
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

```
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