Package 'nlmixr2est'

October 15, 2026

Type Package

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

Version 7.2.1

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

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)

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

Depends nlmixr2data, R (>= 4.1.0)

Imports backports, checkmate, cli, graphics, knitr, L0Learn, lbfgsb3c (>= 2024-3.6), lotri, methods, minqa, n1qn1 (>= 6.0.1-12), nlme, Rcpp, RcppTrust (>= 0.1.0), rex, rxode2 (>= 5.1.8), stats, symengine, utils

Suggests broom.mixed, callr, crayon, data.table, devtools, digest, dplyr (>= 1.1.0), generics, nloptr, sys, testthat, tibble, withr, xgxr, sfsmisc, minpack.lm, Matrix, remotes, fastGHQuad, rmarkdown, rlang, numDeriv, rstan, loo

LinkingTo BH, n1qn1 (>= 6.0.1-12), lbfgsb3c (>= 2024-3.6), lotri (>= 0.5.0), Rcpp, RcppArmadillo (>= 0.11.2.3.1), RcppEigen, RcppTrust (>= 0.1.0), rxode2 (>= 5.1.8)

Biarch true

Config/testthat/edition 3

Encoding UTF-8

Language en-US

NeedsCompilation yes

LazyData true

VignetteBuilder knitr

Config/roxygen2/version 8.1.0

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

Repository CRAN

Date/Publication 2026-10-14 22:12:07 UTC

Contents

.augPredIpredModel 5
.nlmFamilyFitGeneric 5
.nlmixr0preProcessCovariatesPresent 7
.nlmixr0preProcessNpMuExpand 7
.nlmixrPreprocessLiteralFix 8
.preProcessDataUi 9
.preProcessLinCmtOde 9
.preProcessVaeNonMuTheta 10
.preProcessZeroOmega 11
.preProcessZeroTheta 11
aaaCranNlmixrThreads 12
addCwres 13

addNpde . . . 14
addTable . . . 16
agqControl . . . 17
assertNlmixrFit . . . 20
assertNlmixrFitData . . . 21
bobyqaControl . . . 22
boxCox . . . 26
cholSE . . . 27
emviControl . . . 28
fbviControl . . . 35
foceControl . . . 36
foceiControl . . . 37
foceiLikIterPrintStart . . . 58
foceiLikLoad . . . 60
foceiLikRun . . . 62
foceiLikUnload . . . 65
focepControl . . . 65
foControl . . . 66
foiControl . . . 67
formatMinWidth . . . 68
getValidNlmixrCtl.agq . . . 69
iagqControl . . . 74
ifoceControl . . . 75
ifoceiControl . . . 76
ifocepControl . . . 78
ilaplaceControl . . . 79
impControl . . . 80
impCovControl . . . 81
impmapControl . . . 82
iterPrintControl . . . 92
laplaceControl . . . 93
lbfgsb3cControl . . . 95
magqControl . . . 99
mfoceControl . . . 100
mfoceiControl . . . 102
mfocepControl . . . 103
mlaplaceControl . . . 104
n1qn1Control . . . 105
newuoaControl . . . 109
nlmControl . . . 114
nlminbControl . . . 119
nlmixr2 . . . 124
nlmixr2AllEst . . . 137
nlmixr2AllEstType . . . 137
nlmixr2AugPredSolve . . . 138
nlmixr2CreateOutputFromUi . . . 139
nlmixr2Est.agq . . . 140
nlmixr2fix . . . 145

nlmixr2Gill83 . . . 146
nlmixr2Hess . . . 148
nlmixr2Keywords . . . 150
nlmixr2Logo . . . 150
nlmixr2NlmeControl . . . 151
nlmixr2Validate . . . 156
nlmixr2Version . . . 157
nlmixrAddObjectiveFunctionDataFrame . . . 157
nlmixrAddTiming . . . 158
nlmixrCbind . . . 159
nlmixrClone . . . 160
nlmixrWithTiming . . . 161
nlsControl . . . 162
nmNearPD . . . 168
nmObjGetControl.agq . . . 170
nmObjGetEstimationModel . . . 174
nmObjGetFoceiControl.agq . . . 174
nmObjGetIpredModel . . . 176
nmObjGetPredOnly . . . 177
nmObjHandleControlObject.agqControl . . . 178
nmObjHandleModelObject . . . 180
nmObjUiSetCompressed . . . 181
nmsimplex . . . 182
npagControl . . . 182
npbControl . . . 185
ofv . . . 187
optimControl . . . 188
posthocControl . . . 194
predict.nlmixr2FitCore . . . 195
print.saemFit . . . 196
qrpemControl . . . 197
residuals.nlmixr2FitData . . . 198
rmEta . . . 198
rsControl . . . 200
saControl . . . 201
saemControl . . . 202
setCov . . . 212
setCov<- . . . 214
setCovAllMethods . . . 215
setCovOptions . . . 216
setCovValue . . . 217
setOfv . . . 218
sqrtm . . . 218
summary.saemFit . . . 219
tableControl . . . 219
trustControl . . . 221
uobyqaControl . . . 226
vaeControl . . . 231

vaeCovariates 244
vpcSim 245

Index **247**

`.augPredIpredModel` *Augment Prediction for Ipred Model*

Description

This function augments the prediction for an individual prediction (Ipred) model. It retrieves the simulation model from the fit object and evaluates the model variables.

Usage

```
.augPredIpredModel(fit)
```

Arguments

`fit`	The fitted model object from which to retrieve the simulation model.

Details

The function performs the following steps:

- Retrieves the simulation model from the provided ‘fit‘ object using ‘.getSimModel‘ with ‘hideIpred‘ and ‘tad‘ set to ‘FALSE‘.

- Evaluates the model variables using ‘rxModelVars‘.

Value

The evaluated model variables for the Ipred model.

`.nlmFamilyFitGeneric` *Shared fit driver for the nlm-family estimation methods*

Description

Shared fit driver for the nlm-family estimation methods

Usage

```
.nlmFamilyFitGeneric(
  env,
  method,
  fitModel,
  getTheta,
  controlToFocei,
  returnFlag,
  objective = NULL,
  message = function(fit) fit$message,
  emitFitWarnings = TRUE,
  extra = "",
  adjustOutput = TRUE,
  postSetup = NULL
)
```

Arguments

env	dispatch environment (provides ‘ui‘, ‘control‘, ‘data‘, ‘table‘)
method	estimation-method string; also the slot the raw fit is stored under (e.g. ‘'"nlm"‘ -> ‘.ret[["nlm"]]‘)
fitModel	‘function(ui, dataSav)‘ running the optimizer
getTheta	‘function(fit, ui)‘ returning the full theta vector, or the name of the optimizer’s estimates in the fit (e.g. ‘'"par"‘), which ‘.nlmFamilyGetTheta()‘ completes with the fixed thetas
controlToFocei	‘function(env)‘ translating the control to a focei-style control for output assembly
returnFlag	rxode2 control flag name that short-circuits and returns the raw optimizer result (e.g. ‘'"returnNlm"‘)
objective	optional ‘function(fit)‘ returning the raw objective, or the name of the fit’s minimized -log-likelihood, which is doubled; when ‘NULL‘ the driver does not set ‘$objective‘ (a ‘postSetup‘ closure did)
message	‘function(fit)‘ returning the ‘$message‘ (default ‘fit$message‘)
emitFitWarnings	when TRUE (the default), re-emit the warnings collected from ‘fitModel‘ (the optimizer, the covariance step and ‘nlmWarnings()‘) via ‘warning()‘, so they reach the fit’s ‘$runInfo‘; ‘FALSE‘ drops them
extra	‘$extra‘ print string, or a ‘function(control)‘ returning it
adjustOutput	when TRUE, run ‘.nlmFamilyAdjustOutput()‘
postSetup	optional ‘function(ret, ui, fitList)‘ returning a modified ‘ret‘, run right after the raw fit is stored and before ‘.nlmFamilyAdjustOutput()‘ (for methods that set cov/covMethod/objective with custom values)

Value

the assembled nlmixr2 fit (or the raw optimizer result if ‘returnFlag‘)

Author(s)

Matthew L. Fidler

`.nlmixr0preProcessCovariatesPresent`

Preprocess Covariates needed (or other data items)

Description

Preprocess Covariates needed (or other data items)

Usage

```
.nlmixr0preProcessCovariatesPresent(ui, est, data, control)
```

Arguments

`ui`	rxode2 ui
`est`	estimation method (all methods are shown by 'nlmixr2AllEst()'). Methods can be added for other tools
`data`	nlmixr data
`control`	The estimation control object. These are expected to be different for each type of estimation method

Value

list with the ui (possibly modified)

Author(s)

Matthew L. Fidler

`.nlmixr0preProcessNpMuExpand`

Preprocessing hook: mu-expand non-mu structural fixed-effect thetas for the nonparametric engines, before the rest of the pipeline builds on the ui. Doing it here (rather than mutating the ui mid-setup) keeps the injected pseudo-etas consistent through covariate/mu processing and model compilation. Off when control$muExpand is FALSE.

Description

Preprocessing hook: mu-expand non-mu structural fixed-effect thetas for the nonparametric engines, before the rest of the pipeline builds on the ui. Doing it here (rather than mutating the ui mid-setup) keeps the injected pseudo-etas consistent through covariate/mu processing and model compilation. Off when control$muExpand is FALSE.

Usage

```
.nlmixr0preProcessNpMuExpand(ui, est, data, control)
```

Arguments

ui	rxode2 ui
est	estimation method (all methods are shown by 'nlmixr2AllEst()'). Methods can be added for other tools
data	nlmixr data
control	The estimation control object. These are expected to be different for each type of estimation method

Value

list(ui=) when the model was expanded, else NULL

Author(s)

Matthew L. Fidler

.nlmixrPreprocessLiteralFix

This literally fixes parameters in the model

Description

Whenever there is a fixed parameter in the model, the parameter is replaced with the literal value inside of the model and dropped from the 'ini' block. This only occurs when the 'control$literalFix=TRUE'.

Usage

```
.nlmixrPreprocessLiteralFix(ui, est, data, control)
```

Arguments

ui	model function/object
est	estimation method (all methods are shown by 'nlmixr2AllEst()'). Methods can be added for other tools
data	nlmixr data
control	The estimation control object. These are expected to be different for each type of estimation method

Value

list with possibly updated ui

Author(s)

Matthew L. Fidler

`.preProcessDataUi` *Preprocess the zero omegas*

Description

Preprocess the zero omegas

Usage

```
.preProcessDataUi(ui, est, data, control)
```

Arguments

`ui`	rxode2 ui
`est`	estimation method (all methods are shown by 'nlmixr2AllEst()'). Methods can be added for other tools
`data`	nlmixr data
`control`	The estimation control object. These are expected to be different for each type of estimation method

Value

list with the ui (possibly modified)

Author(s)

Matthew L. Fidler

`.preProcessLinCmtOde` *Translate a mixed 'linCmt()'/ODE model to all-ODEs*

Description

rxode2 requires the 'linCmt()' compartments to be the last states of the solve ('op$linOffset = neq - numLin - numLinSens'), so their compartment number is one past the ODE states. The FOCEi inner model adds an ODE state per eta, and the nlm family one per theta, which pushes depot/central past the compartment numbers the data was translated against ('.foceiPreProcessData()' uses the plain model). A dose then silently lands in a sensitivity state – every prediction comes back 0. Solving the linear part as ODEs removes the 'linCmt()' block, so the numbering agrees again. SAEM builds no sensitivity states and is left alone.

Usage

```
.preProcessLinCmtOde(ui, est, data, control)
```

Arguments

ui	rxode2 ui
est	estimation method (all methods are shown by 'nlmixr2AllEst()'). Methods can be added for other tools
data	nlmixr data
control	The estimation control object. These are expected to be different for each type of estimation method

Value

list with the translated ui, or 'NULL' when nothing to do

Author(s)

Matthew L. Fidler

.preProcessVaeNonMuTheta

VAE preprocessing hook: inject etas for non-mu-referenced thetas per vaeControl(nonMuTheta=).

Description

VAE preprocessing hook: inject etas for non-mu-referenced thetas per vaeControl(nonMuTheta=).

Usage

```
.preProcessVaeNonMuTheta(ui, est, data, control)
```

Arguments

ui	rxode2 ui
est	estimation method (all methods are shown by 'nlmixr2AllEst()'). Methods can be added for other tools
data	nlmixr data
control	The estimation control object. These are expected to be different for each type of estimation method

Value

list(ui=) possibly with injected etas

Author(s)

Matthew L. Fidler

`.preProcessZeroOmega` *Preprocess the zero omegas*

Description

Preprocess the zero omegas

Usage

```
.preProcessZeroOmega(ui, est, data, control)
```

Arguments

`ui`	rxode2 ui model
`est`	estimation method (all methods are shown by ‘nlmixr2AllEst()‘). Methods can be added for other tools
`data`	nlmixr data
`control`	The estimation control object. These are expected to be different for each type of estimation method

Value

list with the ui (possibly modified)

Author(s)

Matthew L. Fidler

`.preProcessZeroTheta` *Nudge exactly-zero theta initial estimates off zero (FOCEi family)*

Description

FOCEi scales a linear parameter by its native magnitude ‘|init|‘, which is ‘0‘ (no scale) when a population parameter is initialized at exactly ‘0‘ (the parameter then freezes). This hook moves every estimated, non-fixed ‘theta‘ whose initial estimate is exactly ‘0‘ to ‘+zeroTheta‘ when that is within its bounds, otherwise ‘-zeroTheta‘; if neither is within the bounds it errors. Only runs for the FOCEi family (a ‘foceiControl‘), and runs before ‘.preProcessBoundedTransform‘ so the nudged value is what gets transformed.

Usage

```
.preProcessZeroTheta(ui, est, data, control)
```

Arguments

ui	rxode2 ui model
est	estimation method (all methods are shown by 'nlmixr2AllEst()'). Methods can be added for other tools
data	nlmixr data
control	The estimation control object. These are expected to be different for each type of estimation method

Value

list with the ui (possibly modified)

Author(s)

Matthew L. Fidler

---

aaaCranNlmixrThreads *This function is to set the number of threads to 2*

Description

In general it is a CRAN requirement that packages not use more than 2 threads. This function is to set the number of threads to 2 for CRAN testing. It is not intended for general use.

Usage

```
aaaCranNlmixrThreads()
```

Details

When testing with devtools::test() or testthat::test_package(), the NOT_CRAN environment variable is set to "true", so the number of threads will not be limited to 2.

Value

nothing, called for side effect of setting the number of threads to 2 for CRAN testing

Author(s)

Matthew L. Fidler

Examples

```
# Set the number of threads to 2 for CRAN testing
aaaCranNlmixrThreads()
```

---

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

...	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

```
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)
```

---

agqControl *Control options for the agq estimation method*

Description

This is the control options for the adaptive Gauss-Hermite quadrature for the likelihood. Note that nAGQ=1 is the same as the Laplace method.

Usage

```
agqControl(
```

```
  sigdig = 3,
  nAGQ = 2,
  ...,
  interaction = TRUE,
  agqLow = -Inf,
  agqHi = Inf
)
```

Arguments

sigdig	Optimization significant digits. One value drives, with a single consistent formula, the inner/outer optimizer convergence tolerance (10^-sigdig), the boundary check tolerance (5*10^(-sigdig+1)), and the ODE solver tolerances: the rtol exponent IS sigdig and atol sits three orders below, so rtol = 10^-sigdig, atol = 10^(-sigdig-3) for every solver (stiff, non-stiff or auto-switching). The sensitivity (atolSens/rtolSens) tolerances match the main solve (the outer gradient and covariance are built from them); the steady-state (ssAtol/ssRtol) tolerances run one order looser. Keying the optimizer to the same 10^-sigdig means it converges to exactly the precision the solve supports. At the default sigdig = 3 this is atol = 1e-6, rtol = 1e-3.
nAGQ	Number of Gauss-Hermite adaptive quadrature points. ‘0‘ disables AGQ; ‘1‘ is equivalent to Laplace. Cost grows quickly with ETAs: once the EBE is found, expect ‘nAGQ^neta‘ (even ‘nAGQ‘) or ‘(nAGQ^neta)-1‘ (odd ‘nAGQ‘) additional evaluations per subject.
...	Parameters used in the default ‘foceiControl()‘
interaction	boolean, Interaction term for the model, in this case the default is ‘TRUE‘; For adaptive quadrature, with normal distribution the Hessian is calculated with the foce(i) approximation
agqLow	The lower bound for adaptive quadrature log-likelihood. By default this is -Inf; in the original nlmixr’s gnlmm it was -700.
agqHi	The upper bound for adaptive quadrature log-likelihood. By default this is Inf; in the original nlmixr’s gnlmm was 400.

Value

agqControl object

Author(s)

Matthew L. Fidler

Examples

```
agqControl()

# Use adaptive quadrature
```

```
# x =  Litter size after 21 days, and the modeled value

r <- rats
r$dv <- r$x

# Time is not used in this model, but it is required in nlmixr2
# currently, add a dummy value

r$time <- 0

f <- function() {
  ini({
    t1 <- 1
    t2 <- 1
    t3 <- 1
    eta1 ~ 1
  })
  model({
    lp <- t1 * x1 + t2 * x2 + (x1 + x2*t3) * eta1
    p <- pnorm(lp)
    m1 <- m # need to add outside of model specification
    x ~ dbinom(m1, p)
  })
}

fit <- nlmixr(f, r, est="agq")


p <- pump

p$dv <- p$y
p$time <- 0 # dummy time

f <- function() {
  ini({
    t1 <- 1
    t2 <- 1
    t3 <- 1
    t4 <- 1
    eta1 ~ 1
  })
  model({
    if (group == 1) {
       lp <- t1 + t2 * logtstd
    } else {
       lp <- t3 + t4 * logtstd
    }
    lp <- lp + eta1
    lam <- exp(lp)
    y ~ dpois(lam)
  })
}
```

```
fit <- nlmixr(f, p, est="agq", control=agqControl(nAGQ=5))

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="agq")
```

`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)
```

bobyqaControl *Control for bobyqa estimation method in nlmixr2*

Description

Control for bobyqa estimation method in nlmixr2

Usage

```
bobyqaControl(
  npt = NULL,
  rhobeg = NULL,
  rhoend = NULL,
  iprint = 0L,
  maxfun = 100000L,
  returnBobyqa = FALSE,
  stickyRecalcN = 4,
  maxOdeRecalc = 5,
  odeRecalcFactor = 10^(0.5),
  indTolRelax = TRUE,
  useColor = NULL,
  printNcol = NULL,
  print = 1L,
  normType = c("rescale2", "mean", "rescale", "std", "len", "constant"),
  scaleType = c("nlmixr2", "norm", "mult", "multAdd"),
  scaleCmax = 1e+05,
  scaleCmin = 1e-05,
  scaleC = NULL,
  scaleTo = 1,
  rxControl = NULL,
  optExpression = TRUE,
  sumProd = FALSE,
  literalFix = TRUE,
  literalFixRes = TRUE,
  addProp = c("combined2", "combined1"),
  calcTables = TRUE,
  compress = FALSE,
  covMethod = c("r", ""),
  adjObf = TRUE,
  ci = 0.95,
  sigdig = 3,
  sigdigTable = NULL,
  eventSens = c("jump", "fd"),
  ...
)
```

Arguments

npt	Number of points for the quadratic approximation to the objective; must be in '[n+2, (n+1)(n+2)/2]'. Defaults to 'min(n*2, n+2)'.
rhobeg	Initial trust region radius (with 'rhoend', must satisfy '0 < rhoend < rhobeg'). Defaults to 'min(0.95, 0.2*max(abs(par)))'; adjusted upward if smaller than 'abs(upper-lower)/2'.
rhoend	Final trust region radius. When 'NULL' (default) it is derived from 'sigdig' the way 'foceiControl()' does ('10^(-sigdig)'); otherwise the minqa '1e-6*rhobeg' default applies.
iprint	Controls amount of printing ('0'=none, '1'=start/end only, '2'=each new rho, '3'=every function evaluation, '>3'=every 'iprint' evaluations). Default '0'.
maxfun	The maximum allowed number of function evaluations. If this is exceeded, the method will terminate.
returnBobyqa	return the bobyqa output instead of the nlmixr2 fit
stickyRecalcN	The number of bad ODE solves before reducing the atol/rtol for the rest of the problem.
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
indTolRelax	When 'TRUE' (default), only subjects whose ODE solve produced NaN/Inf have their tolerances relaxed, and the relaxed tolerance persists across optimizer calls (sticky). When 'FALSE', all subjects have their tolerances relaxed on each retry and tolerances are reset afterward.
useColor	Logical (or 'NULL') emit ANSI bold/color escapes in the iteration print. 'NULL' (default) defers to [crayon::has_color()].
printNcol	Integer (or 'NULL') parameter columns per row before wrapping. 'NULL' (default) uses 'floor((getOption("width") - 23) / 12)'.
print	Either a scalar print-frequency ('0' = suppress, '1' (default) = every evaluation, 'N' = every Nth), OR a pre-built [iterPrintControl()] object. Equivalent to 'iterPrintControl(every = print, ncol = printNcol, useColor = useColor)'.
normType	Parameter normalization/scaling used to get scaled initial values for `scaleType`, of the form `Vscaled = (Vunscaled-C1)/C2` (see Feature Scaling; `rescale2` follows the OptdesX manual): `"rescale2"` scales all parameters to (-1, 1); `"rescale"` (min-max) scales to (0, 1); `"mean"` centers on the mean with range (0, 1); `"std"` standardizes by mean/sd; `"len"` scales to unit (Euclidean) length; `"constant"` performs no normalization (`C1=0`, `C2=1`).
scaleType	The scaling scheme for nlmixr2: `"nlmixr2"` (default) scales as `(current-init)*scaleC[i] + scaleTo`, with `scaleTo` from `normType` and scales from `scaleC`; `"norm"` uses the simple scaling from `normType`; `"mult"` scales multiplicatively as `current/init*scaleTo`; `"multAdd"` scales linearly (`(current-init)+scaleTo`) for parameters in an exponential block (e.g. `exp(theta)`) and multiplicatively otherwise.
scaleCmax	Maximum value of the scaleC to prevent overflow.

`scaleCmin` Minimum value of the scaleC to prevent underflow.

`scaleC` Scaling constant used with `scaleType="nlmixr2"`; when not specified, chosen by parameter type to keep gradient sizes similar on a log scale: ‘1‘ for exp()-transformed/power/boxCox/ yeoJohnson parameters, ‘0.5*abs(est)‘ for additive/proportional/ lognormal error parameters, ‘abs(1/digamma(est+1))‘ for factorials, and ‘log(abs(est))*abs(est)‘ for log-scale parameters. May be set explicitly per parameter if these defaults don’t apply well.

`scaleTo` Scale the initial parameter estimate to this value. By default this is 1. When zero or below, no scaling is performed.

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

`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`.

`literalFix` boolean, substitute fixed population values as literals and re-adjust ui and parameter estimates after optimization; Default is ‘TRUE‘.

`literalFixRes` boolean, substitute fixed population values as literals and re-adjust ui and parameter estimates after optimization; Default is ‘TRUE‘.

`addProp` Type of additive-plus-proportional error: ‘"combined1"‘, where standard deviations add:

$$y = f + (a + b \times f^c) \times \varepsilon$$

; or ‘"combined2"‘, where variances add:

$$y = f + \sqrt{a^2 + b^2 \times f^{2 \times c}} \times \varepsilon$$

. Here y = observed, f = predicted, a = additive sd, b = proportional/power sd, c = power exponent (1 in the proportional case).

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

`compress` Should the object have compressed items

`covMethod` Method for calculating the covariance. `"r"` (the default) uses nlmixr2’s `nlmixr2Hess()` Hessian; `""` skips the covariance step.

`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. One value drives, with a single consistent formula, the inner/outer optimizer convergence tolerance (`10^-sigdig`), the boundary check tolerance (`5*10^(-sigdig+1)`), and the ODE solver tolerances: the `rtol` exponent IS `sigdig` and `atol` sits three orders below, so `rtol = 10^-sigdig`, `atol = 10^(-sigdig-3)` for every solver (stiff, non-stiff or auto-switching). The sensitivity (`atolSens`/`rtolSens`) tolerances match the main solve (the outer gradient and covariance are built from them); the steady-state (`ssAtol`/`ssRtol`) tolerances run one order looser. Keying the optimizer to the same `10^-sigdig` means it converges to exactly the precision the solve supports. At the default `sigdig = 3` this is `atol = 1e-6`, `rtol = 1e-3`.

sigdigTable	Significant digits in the final output table. If not specified (‘NULL‘), it defaults to ‘sigdig‘.
eventSens	Controls how dosing/event-parameter (‘alag‘, ‘F‘, ‘rate‘, ‘dur‘) sensitivities are computed for THETA/ETA gradients: ‘"jump"‘ (default) uses rxode2’s analytic event sensitivities; ‘"fd"‘ uses the legacy finite-difference behavior. Also gates the analytic moving-boundary correction for a modeled ‘alag()‘/‘f()‘ on a ‘linCmt()‘ compartment; set ‘"fd"‘ if that model infuses a dose into the lagged/scaled compartment (rxode2/rxode2#1236), or if the regimen also doses an *unlagged/unscaled* compartment alongside the lagged/scaled one – a common design for estimating ‘f()‘ from paired IV+oral data (rxode2/rxode2#1237).
...	Ignored parameters

Value

bobqya control structure

Author(s)

Matthew L. Fidler

Examples

```
# A logit regression example with emax model

dsn <- data.frame(i=1:1000)
dsn$time <- exp(rnorm(1000))
dsn$DV=rbinom(1000,1,exp(-1+dsn$time)/(1+exp(-1+dsn$time)))

mod <- function() {
 ini({
   E0 <- 0.5
   Em <- 0.5
   E50 <- 2
   g <- fix(2)
 })
 model({
   v <- E0+Em*time^g/(E50^g+time^g)
   ll(bin) ~ DV * v - log(1 + exp(v))
 })
}

fit2 <- nlmixr(mod, dsn, est="bobyqa")

print(fit2)

# you can also get the bobyqa output with

fit2$bobyqa
```

boxCox *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

Robert B. Schnabel and Elizabeth Eskow. 1990. "A New Modified Cholesky Factorization," SIAM Journal of Scientific Statistical Computing, 11, 6: 1136-58.

Elizabeth Eskow and Robert B. Schnabel 1991. "Algorithm 695 - Software for a New Modified Cholesky Factorization," ACM Transactions on Mathematical Software, Vol 17, No 3: 306-312

`emviControl` *Control for the variational inference methods emvi and fbvi in nlmixr2*

Description

Variational-inference NLME estimation in the style of Kucukelbir et al. (2017): the latent variables are transformed to an unconstrained real coordinate space, a Gaussian variational family is posited there, and the ELBO is maximized by stochastic gradient ascent using the reparameterization trick.

Usage

```
emviControl(
  seed = 42L,
  iters = 300L,
  nMc = 1L,
  viFamily = c("fullRank", "meanField"),
  pointEstimate = NULL,
  optim = c("advi", "adam"),
  adaptEta = TRUE,
  perNoCor = 0.75,
  etaCandidates = c(0.01, 0.025, 0.05, 0.1, 0.25),
  tau = 1,
  alpha = 0.1,
  tol = NULL,
  evalElbo = 100L,
  klWarmup = 0L,
  temperInit = 10,
  likelihood = c("focei", "foce", "focep", "laplace"),
  returnVi = FALSE,
  resume = NULL,
  print = 1L,
  useColor = NULL,
  printNcol = NULL,
  covMethod = c("vi", "analytic", "r,s", "r", "s", ""),
 covFallback = list(`r,s` = c("r", "s", "vi"), r = c("s", "vi"), s = "vi", analytic =
    c("r,s", "r", "s", "vi")),
  optExpression = TRUE,
  sumProd = FALSE,
  literalFix = TRUE,
  literalFixRes = TRUE,
  addProp = c("combined2", "combined1"),
  calcTables = TRUE,
  compress = FALSE,
  adjObf = TRUE,
  ci = 0.95,
  sigdig = 3,
  sigdigTable = NULL,
```

```
  stickyRecalcN = 4,
  maxOdeRecalc = 5,
  odeRecalcFactor = 10^(0.5),
  indTolRelax = TRUE,
  eventSens = c("jump", "fd"),
  rxControl = NULL,
  ...
)
```

Arguments

seed	Random seed for the variational optimization (reparameterization sampling); default 42. The Monte-Carlo gradient is stochastic, so a fixed seed makes every fit reproducible. Reparameterization noise is drawn from a counter-based stream keyed by the global iteration index, so a shorter run is a bit-for-bit prefix of a longer one and results are independent of the number of cores.
iters	Total number of stochastic gradient ascent iterations.
nMc	Number of Monte-Carlo samples used to approximate the ELBO gradient at each iteration (the paper's 'M'; typically 1-10).
viFamily	Variational family in the unconstrained space. '"fullRank"' (default) uses a block full-rank Gaussian: a dense 'neta x neta' Cholesky factor per subject plus a dense block over the population vector (mean-field across blocks). '"meanField"' uses a fully factorized (diagonal) Gaussian. Mean-field is faster but is known to underestimate marginal variances.
pointEstimate	Which of the two methods to run, normally left at its 'NULL' default so it follows 'est': 'est="emvi"' implies 'TRUE' and 'est="fbvi"' implies 'FALSE'. 'est' wins over a contradicting value, and says so – it has to, because re-estimating a fit with the other method pipes the completed fit's control forward. 'TRUE' runs the variational-EM hybrid: the variational posterior covers the per-subject etas only, and the population parameters (thetas / omega / residual error) are point estimates maximized by the ELBO gradient; output semantics match FOCEi/SAEM. 'FALSE' runs full Bayes: the variational posterior also covers the unconstrained population vector, with flat priors. Two things about "flat" are worth being explicit about, because they define the prior rather than merely describe the implementation. (1) A BOUNDED theta is fitted on its unconstrained scale, and the log-determinant of that constraining transform IS added to the full-Bayes objective, so the flat prior is flat on the NATURAL parameter, as in Stan. It is deliberately NOT added when 'pointEstimate=TRUE': a maximum-likelihood estimate has to stay invariant to reparameterization, which is why Stan's own 'optimize' defaults to 'jacobian=0'. (2) The between-subject variances are carried as per-eta LOG-variances and no Jacobian is applied to them, so the prior is flat on 'log(omega)' – the conventional weakly-informative choice for a scale parameter, but a choice, not an accident: it is not flat on 'omega'. These point estimates maximize the ELBO, NOT the likelihood, and the difference is not merely cosmetic. Since 'ELBO = log p(y|theta) - KL(q || p(eta|y,theta))', any dependence of that KL on 'theta' displaces the maximizer from the MLE –

"variational maximum likelihood". For VARIANCE components the displacement has a known direction: a variational family that understates posterior spread makes the omega M-step, ‘Omega = mean_i(mu_i mu_i’ + Sigma_i)‘, inherit that understatement, so between-subject variability is biased DOWNWARD. The bias is worst for ‘viFamily="meanField"‘, which cannot represent within-subject posterior correlation at all; ‘"fullRank"‘ can, which is why it is the default. Structural (typical-value) parameters are far less affected. If the between-subject variances are themselves the quantity of interest, prefer ‘"fullRank"‘ and cross-check against ‘est="focei"‘ or ‘est="saem"‘.

`optim` Stochastic optimizer. ‘"advi"‘ (default) uses the adaptive step-size sequence from the ADVI paper (Eqs 10-11) – the value keeps that name because the step-size rule really is the published one, even though the surrounding method is not; ‘"adam"‘ uses Adam.

`adaptEta` When ‘TRUE‘ (default) adaptively choose the step-size scale ‘eta‘ by a short search over ‘etaCandidates‘ before the main loop; when ‘FALSE‘ use a fixed ‘eta‘ (the first ‘etaCandidates‘ entry).

`perNoCor` Fraction of the run over which a declared correlated ‘omega‘ block is held at zero correlation, letting the population variances settle before the correlations are estimated. This is `saemControl()`’s ‘perNoCor‘ rule (0.75 there as well); it has no effect on a model with no declared off-diagonals.

Held at ZERO, following saem, not at the ‘ini()‘ value: retaining an initial covariance while the variances shrink around it can leave the block non-positive-definite. A ‘fixed()‘ covariance is exempt – it is not being estimated, so it keeps its value through the hold and out the other side. When the correlations are released the off-diagonal gain restarts rather than continuing the decayed schedule, so they are still estimable at the point they are unfrozen. This is needed here and not in `vaeControl()`: there the fraction is of the EM phase, where the gain is still 1 at release, whereas this run has no such phase and the fraction lands in the decayed schedule.

A value greater than 1 is an ABSOLUTE iteration count rather than a fraction, and must be a whole number. The resolved absolute iteration is stored with the fit and reused by ‘resume=‘, so a resumed run releases the correlations at the same global iteration a single long run would – recomputing the fraction from the resumed call’s ‘iters‘ would re-apply a hold the original run had passed. Only ‘est="emvi"‘ estimates the block at all; ‘est="fbvi"‘ carries ‘omega‘ as per-eta log-variances and errors on a correlated model rather than dropping the off-diagonal.

`etaCandidates` Candidate step-size scales searched when ‘adaptEta‘ is ‘TRUE‘. The default is narrower and smaller-valued than the paper’s ‘c(0.01, 0.1, 1, 10, 100)‘ because these gradients come from FOCEi sensitivities on the model’s own scale rather than from AD through a Stan program, so the useful step sizes sit lower; the paper’s grid can be passed verbatim if wanted. Each candidate costs ‘min(iters, 75)‘ iterations (a diverging one aborts early and is cheap), so the search is a substantial share of a fit – widen it deliberately. When the search selects the largest or smallest candidate the grid may be the binding constraint, and the fit says so in ‘$runInfo‘; ‘$etaScores‘ reports the per-candidate scores behind the choice.

`tau` Stabilizing constant ‘tau > 0‘ in the step-size denominator (paper Eq 10); the step-size is insensitive to it.

`alpha` Weighting ‘alpha‘ in (0, 1) of new vs old gradient information in the step-size memory recursion (paper Eq 11).

`tol` Convergence tolerance on the relative change in the ELBO: the loop stops early once the change falls below this. ‘NULL‘ (default) derives it from ‘sigdig‘ as ‘10^(-sigdig)‘, the same rule ‘saemControl()‘ and ‘foceiControl()‘ use for their optimizer tolerances, so it tightens with ‘sigdig‘ instead of staying pinned. ‘0‘ disables early stopping (run all ‘iters‘). Because the per-iteration ELBO is an ‘nMc‘-sample Monte-Carlo estimate and therefore noisy, the test compares the MEAN over the last ‘evalElbo‘ iterations against the mean over the window before it, rather than consecutive iterations. The ‘adaptEta‘ step-size search never stops early on this criterion (its scorer reads a short run as divergence).

`evalElbo` Window length, in iterations, for the ‘tol‘ convergence test. Stan’s ADVI re-evaluates the ELBO every 100 iterations with fresh draws; averaging the draws already taken is the cheaper equivalent. Shrunk automatically on a short run so ‘iters‘ well below ‘evalElbo‘ can still trigger the check.

`klWarmup` Number of iterations of PRIOR TEMPERING (0, the default, disables it). During the warm-up the population prior is inflated by a factor ramping geometrically from ‘temperInit‘ down to 1, which down-weights the prior term of the ELBO and keeps the per-subject variational posterior from collapsing before it is informative. This is the variational analogue of `saemControl()`’s ‘perSa‘ simulated-annealing phase (true simulated annealing does not transfer: variational inference has no MCMC kernel to keep wide).

It CHANGES THE OBJECTIVE FUNCTION MID-RUN. Early iterations maximize a tempered surrogate rather than the ELBO, so the convergence theory for the ELBO does not cover the warm-up, the reported ELBO trace is not comparable across the boundary, and the ‘tol‘ early-stopping test is suppressed until tempering ends. The ‘adaptEta‘ step-size search also scores candidates on the untempered objective. Off by default for those reasons; turn it on for a model where the variational scale collapses early.

`temperInit` Initial prior inflation factor for ‘klWarmup‘ tempering (default 10); ignored when ‘klWarmup = 0‘.

`likelihood` Inner likelihood used for the per-subject objective and gradient, run through the FOCEi inner interface: ‘"focei"‘ (default), ‘"foce"‘, ‘"focep"‘, or ‘"laplace"‘.

`returnVi` When ‘TRUE‘ return the raw variational optimization object instead of the nlmixr2 fit.

`resume` Optional warm-resume state: a previous ‘emvi‘/‘fbvi‘ fit (or its ‘envviState‘). The optimization continues from that state for ‘iters‘ more iterations, bit-for-bit identical to a single fresh run of the combined length (the counter-based RNG is keyed by the global iteration index).

That equivalence requires every schedule point to be an ABSOLUTE iteration. A FRACTIONAL ‘perNoCor‘ cannot provide it, and not because of any bookkeeping that could be fixed: ‘perNoCor = 0.75‘ of one 120-iteration run releases the correlations at iteration 90, while 0.75 of a first 60-iteration leg releases at 45. Those are different schedules, and the resulting correlation estimates genuinely

differ. Pin the schedule (‘perNoCor = 90‘) whenever a fit may be resumed; the resolved value is then stored with the fit and reused.

`print` Either a scalar print-frequency (‘0‘ = suppress, ‘1‘ (default) = every evaluation, ‘N‘ = every Nth), OR a pre-built [iterPrintControl()] object. Equivalent to ‘iterPrintControl(every = print, ncol = printNcol, useColor = useColor)‘.

`useColor` Logical (or ‘NULL‘) emit ANSI bold/color escapes in the iteration print. ‘NULL‘ (default) defers to [crayon::has_color()].

`printNcol` Integer (or ‘NULL‘) parameter columns per row before wrapping. ‘NULL‘ (default) uses ‘floor((getOption("width") - 23) / 12)‘.

`covMethod` Covariance of the population estimates.

"‘vi‘" (default) For a full-Bayes fit (‘fbvi‘, ‘pointEstimate = FALSE‘) the population variational covariance; a point-estimate fit (‘emvi‘) has no population variational block and uses the FOCEI ‘"r,s"‘ covariance.

"‘analytic‘", "‘r,s‘", "‘r‘" and "‘s‘" run the FOCEI covariance step (see [foceiControl()]) at the estimates, starting the individual estimates at the variational posterior means. When it gives no covariance, a full-Bayes fit installs the population variational covariance instead, with a message. An integer ‘covMethod‘ is a ‘foceiControl()‘ slot: ‘0‘ is no covariance, ‘1‘, ‘2‘ and ‘3‘ are ‘"r,s"‘, ‘"r"‘ and ‘"s"‘.

"" Does not calculate the covariance step.

`covFallback` what each ‘covMethod‘ falls back to when it gives no usable covariance: a named list, one element per method, each the ordered methods to try instead – the FOCEI ‘"r,s"‘, ‘"r"‘ and ‘"s"‘ (see [foceiControl()]), and ‘"vi"‘, the population variational covariance of a full-Bayes fit. The default is the established behaviour: each FOCEI method keeps its FOCEI fallbacks and ends with ‘"vi"‘. A list you give replaces the default: a method it does not name has no fallback.

`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`.

`literalFix` boolean, substitute fixed population values as literals and re-adjust ui and parameter estimates after optimization; Default is ‘TRUE‘.

`literalFixRes` boolean, substitute fixed population values as literals and re-adjust ui and parameter estimates after optimization; Default is ‘TRUE‘.

`addProp` Type of additive-plus-proportional error: ‘"combined1"‘, where standard deviations add:

$$y = f + (a + b \times f^c) \times \varepsilon$$

; or ‘"combined2"‘, where variances add:

$$y = f + \sqrt{a^2 + b^2 \times f^{2 \times c}} \times \varepsilon$$

. Here y = observed, f = predicted, a = additive sd, b = proportional/power sd, c = power exponent (1 in the proportional case).

`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` Specifies the "significant digits" that the ODE solving requests. This is `NULL` by default, and while it is `NULL` it has no effect at all: `rxSolve()` uses the standard `atol`/`rtol` (and the standard sensitivity and steady-state tolerances). `sigdig` only changes a tolerance when you ask for it explicitly.

When it is supplied, the tolerances are derived with one solver-independent formula – the same for stiff, non-stiff and auto-switching solvers. The `rtol` exponent IS `sigdig` and `atol` sits three orders below it:

- `rtol = 10^(-sigdig)`, `atol = 10^(-sigdig-3)`
- the sensitivity tolerances match the main solve, so `rtolSens = rtol` and `atolSens = atol` (gradients and covariances are built from them)
- the steady-state tolerances run one order looser than the corresponding main tolerance, so `ssRtol = ssRtolSens = 10*rtol` and `ssAtol = ssAtolSens = 10*atol`

Each of these is set only when you did not pass that tolerance yourself; a tolerance you supply always wins. Because they are resolved independently, an explicit `atol`/`rtol` overrides the main solve but does *not* propagate to the sensitivity or steady-state tolerances – set those directly if you need them changed too.

This mapping matches how `nlmixr2est` derives solver tolerances from its optimization `sigdig`, so a `sigdig` used for estimation and the same `sigdig` used for a plain `rxSolve()` mean the same thing. Note it is keyed to `sigdig` as a request for that many significant digits, and is looser than the `atol`/`rtol` defaults for small `sigdig` – at `sigdig = 4` it gives `rtol = 1e-4` against a default `rtol = 1e-6`. Raise `sigdig`, or set `atol`/`rtol` directly, when you want a tighter solve.

`sigdigTable` Significant digits in the final output table. If not specified (‘NULL‘), it defaults to ‘sigdig‘.

`stickyRecalcN` The number of bad ODE solves before reducing the atol/rtol for the rest of the problem.

`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

`indTolRelax` When ‘TRUE‘ (default), only subjects whose ODE solve produced NaN/Inf have their tolerances relaxed, and the relaxed tolerance persists across optimizer calls (sticky). When ‘FALSE‘, all subjects have their tolerances relaxed on each retry and tolerances are reset afterward.

`eventSens` Controls how dosing/event-parameter (‘alag‘, ‘F‘, ‘rate‘, ‘dur‘) sensitivities are computed for THETA/ETA gradients: ‘"jump"‘ (default) uses rxode2’s analytic event sensitivities; ‘"fd"‘ uses the legacy finite-difference behavior. Also gates the analytic moving-boundary correction for a modeled ‘alag()‘/‘f()‘ on a ‘linCmt()‘ compartment; set ‘"fd"‘ if that model infuses a dose into the lagged/scaled compartment (rxode2/rxode2#1236), or if the regimen also doses an *unlagged/unscaled* compartment alongside the lagged/scaled one – a common design for estimating ‘f()‘ from paired IV+oral data (rxode2/rxode2#1237).

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

`...` Other arguments to control SAEM.

Details

One control serves both methods, which differ only in what the variational posterior covers:

- ‘est="emvi"‘ – variational EM. The variational family covers the per-subject etas only; the population parameters (thetas, omega, residual error) are point estimates updated by an M-step, so the output semantics match FOCEi/SAEM.
- ‘est="fbvi"‘ – full-Bayes variational inference. The variational posterior additionally covers the unconstrained population vector under flat priors, which is the closer analogue of the published algorithm.

‘pointEstimate‘ is the switch between them and defaults to whichever the chosen ‘est‘ implies. ‘est‘ decides: a control that contradicts it is overridden, with a message saying so.

Neither method is the published ADVI algorithm, which is why neither is named for it. Two deviations matter. First, even ‘fbvi‘ parameterizes the between-subject omega as per-eta log-variances rather than freely, so a full-rank family does not yield the joint posterior covariance over the whole parameter vector that published ADVI gives. Second, the gradient of the log-joint comes from the FOCEi forward sensitivities (inner per-subject eta gradient plus the outer population sensitivity contraction) rather than from automatic differentiation – there is no AD in this implementation, which is why "advi" would misname it. Results should therefore not be read as reproducing a Stan ‘vb()‘ fit except on models where the two objectives coincide. The whole optimization loop runs in C++.

Value

variational-inference control structure (class ‘emviControl‘)

Author(s)

Matthew L. Fidler

`fbviControl` *Control for the fbvi (full-Bayes variational inference) method*

Description

A convenience wrapper around [emviControl()] with ‘pointEstimate = FALSE‘, i.e. the variational posterior covers the unconstrained population vector (under flat priors) as well as the per-subject etas, rather than point-estimating the population parameters by an M-step. See [emviControl()] for the full parameter list.

Usage

```
fbviControl(..., pointEstimate = FALSE)
```

Arguments

`...`	Parameters passed to [emviControl()].
`pointEstimate`	Which of the two methods to run, normally left at its ‘NULL‘ default so it follows ‘est‘: ‘est="emvi"‘ implies ‘TRUE‘ and ‘est="fbvi"‘ implies ‘FALSE‘. ‘est‘ wins over a contradicting value, and says so – it has to, because re-estimating a fit with the other method pipes the completed fit’s control forward. ‘TRUE‘ runs the variational-EM hybrid: the variational posterior covers the per-subject etas only, and the population parameters (thetas / omega / residual error) are point estimates maximized by the ELBO gradient; output semantics match FOCEi/SAEM. ‘FALSE‘ runs full Bayes: the variational posterior also covers the unconstrained population vector, with flat priors. Two things about "flat" are worth being explicit about, because they define the prior rather than merely describe the implementation. (1) A BOUNDED theta is fitted on its unconstrained scale, and the log-determinant of that constraining transform IS added to the full-Bayes objective, so the flat prior is flat on the NATURAL parameter, as in Stan. It is deliberately NOT added when ‘pointEstimate=TRUE‘: a maximum-likelihood estimate has to stay invariant to reparameterization, which is why Stan’s own ‘optimize‘ defaults to ‘jacobian=0‘. (2) The between-subject variances are carried as per-eta LOG-variances and no Jacobian is applied to them, so the prior is flat on ‘log(omega)‘ – the conventional weakly-informative choice for a scale parameter, but a choice, not an accident: it is not flat on ‘omega‘. These point estimates maximize the ELBO, NOT the likelihood, and the difference is not merely cosmetic. Since ‘ELBO = log p(y|theta) - KL(q || p(etaly,theta))‘, any dependence of that KL on ‘theta‘ displaces the maximizer from the MLE – "variational maximum likelihood". For VARIANCE components the displacement has a known direction: a variational family that understates posterior spread makes the omega M-step, ‘Omega = mean_i(mu_i mu_i’ + Sigma_i)‘, inherit that understatement, so between-subject variability is biased DOWNWARD. The bias is worst for ‘viFamily="meanField"‘, which cannot represent within-subject posterior correlation at all; ‘"fullRank"‘ can, which is why it is

the default. Structural (typical-value) parameters are far less affected. If the between-subject variances are themselves the quantity of interest, prefer ‘"fullRank"‘ and cross-check against ‘est="focei"‘ or ‘est="saem"‘.

Value

An ‘emviControl‘ object with ‘pointEstimate = FALSE‘.

Author(s)

Matthew L. Fidler

Examples

```
fbviControl()
```

`foceControl` *Control options for the foce estimation method*

Description

This is the first order option without the interaction between residuals and etas.

Usage

```
foceControl(sigdig = 3, ..., interaction = FALSE)
```

Arguments

`sigdig`	Optimization significant digits. One value drives, with a single consistent formula, the inner/outer optimizer convergence tolerance (`10^-sigdig`), the boundary check tolerance (`5*10^(-sigdig+1)`), and the ODE solver tolerances: the `rtol` exponent IS `sigdig` and `atol` sits three orders below, so `rtol = 10^-sigdig`, `atol = 10^(-sigdig-3)` for every solver (stiff, non-stiff or auto-switching). The sensitivity (`atolSens`/`rtolSens`) tolerances match the main solve (the outer gradient and covariance are built from them); the steady-state (`ssAtol`/`ssRtol`) tolerances run one order looser. Keying the optimizer to the same `10^-sigdig` means it converges to exactly the precision the solve supports. At the default `sigdig = 3` this is `atol = 1e-6`, `rtol = 1e-3`.
`...`	Parameters used in the default ‘foceiControl()‘
`interaction`	Interaction term for the model, in this case the default is ‘FALSE‘; it cannot be changed, use ‘focei‘ instead

Value

foceControl object

Author(s)

Matthew L. Fidler

Examples

```
foceControl()
```

`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 = NULL,
  scaleTo = 1,
  scaleObjective = 0,
  normType = c("rescale2", "mean", "rescale", "std", "len", "constant"),
  scaleType = c("nlmixr2", "norm", "mult", "multAdd"),
  scaleCmax = 1e+05,
  scaleCmin = 1e-05,
  scaleCband = c(0.1, 10),
  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", "analytic", "r", "s", "sa", "imp", ""),
  covSolveTol = NULL,
 covFallback = list(`r,s` = c("r", "s"), r = "s", s = character(0), analytic = c("r,s",
    "r", "s")),
  covPrecursor = c("fd", "analytic"),
  covShortcut = FALSE,
  covFull = TRUE,
  fast = FALSE,
  priorMethod = c("auto", "general", "nwpri", "tnpri", "none"),
```

```
fdOutlierZ = 3.5,
fdOutlierScale = TRUE,
fdRefine = c("chartrand", "lanczos", "richardson"),
fdLanczosM = 2L,
fdRichardsonR = 2L,
fdRichardsonV = 2,
fdChartrandAll = FALSE,
fdOutlierAny = FALSE,
fdIndividualStep = TRUE,
fdChartrand = TRUE,
foceEbeTol = NULL,
hessEps = (.Machine$double.eps)^(1/3),
hessEpsLlik = (.Machine$double.eps)^(1/3),
optimHessType = c("central", "forward"),
optimHessCovType = c("central", "forward"),
hessEtaStepMin = 0.05,
censOption = c("gauss", "laplace"),
eventType = c("central", "forward"),
eventSens = c("jump", "fd"),
centralDerivEps = rep(20 * sqrt(.Machine$double.eps), 2),
lbfgsLmm = 7L,
lbfgsPgtol = 0,
lbfgsFactr = NULL,
eigen = TRUE,
diagXform = c("sqrt", "log", "identity"),
iovXform = c("sd", "var", "logsd", "logvar"),
iovMethod = c("auto", "theta", "omega"),
sumProd = FALSE,
optExpression = TRUE,
literalFix = TRUE,
literalFixRes = TRUE,
ci = 0.95,
useColor = NULL,
boundTol = NULL,
calcTables = TRUE,
noAbort = TRUE,
interaction = TRUE,
foce = c("nonmem", "foce+"),
cholSEtol = (.Machine$double.eps)^(1/3),
cholAccept = 0.001,
resetEtaP = 0.15,
resetThetaP = 0,
resetThetaFinalP = 0,
diagOmegaBoundUpper = 5,
diagOmegaBoundLower = 100,
cholSEOpt = FALSE,
cholSECov = FALSE,
fo = FALSE,
```

```
 covTryHarder = FALSE,
 outerOpt = c("bobyqa", "nlminb", "lbfgsb3c", "L-BFGS-B", "mma", "lbfgsbLG", "slsqp",
   "uobyqa", "newuoa", "trust"),
 innerOpt = c("auto", "trust", "n1qn1", "lbfgsb3c"),
 innerHessian = c("focei", "conditional"),
 detHessian = c("focei", "conditional"),
 hessianMethod = c("fd", "bfgs", "sr1", "bofill"),
 trustConf = 0.975,
 trustRinit = NULL,
 trustRmax = NULL,
 trustFterm = NULL,
 trustMterm = NULL,
 trustPolish = FALSE,
 bobyqaStationary = TRUE,
 innerLbfgsLmm = 5L,
 innerLbfgsFactr = NULL,
 innerLbfgsPgtol = NULL,
 innerLbfgsAbstol = NULL,
 innerLbfgsReltol = NULL,
 outerTrustHessian = c("auto", "analytic", "bfgs", "fd"),
 outerTrustRinit = NULL,
 outerTrustRmax = NULL,
 outerTrustFterm = NULL,
 outerTrustMterm = NULL,
 outerTrustRelStep = 0.001,
 outerTrustRestarts = 3L,
 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,
 muModel = c("none", "irls", "lin"),
 muRefCovAlg = TRUE,
 muModelTol = 1e-05,
 muModelMaxCycles = 20L,
 muModelClampRetries = 10L,
 stateTrim = Inf,
 shi21maxOuter = 0L,
 shi21maxInner = 20L,
 shi21maxInnerCov = 20L,
 shi21maxFD = 20L,
 shi21hMax = 2,
 shi21hMin = 1e-04,
```

```
gillK = 10L,
gillStep = 4,
gillFtol = 0,
gillRtol = sqrt(.Machine$double.eps),
gillKcov = 10L,
gillKcovLlik = 10L,
gillStepCovLlik = 4.5,
gillStepCov = 2,
gillFtolCov = 0,
gillFtolCovLlik = 0,
rmatNorm = TRUE,
rmatNormLlik = TRUE,
smatNorm = TRUE,
smatNormLlik = 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),
etaRestart = 4L,
nRetries = 3,
seed = 42,
resetThetaCheckPer = 0.1,
etaMat = NULL,
repeatGillMax = 1,
stickyRecalcN = 4,
outerMaxOdeRecalc = 5,
outerOdeRecalcFactor = 10^(0.5),
outerStickyRecalcN = 4,
indTolRelax = TRUE,
gradProgressOfvTime = 10,
addProp = c("combined2", "combined1"),
badSolveObjfAdj = 100,
compress = FALSE,
rxControl = NULL,
sigdigTable = NULL,
fallbackFD = FALSE,
smatPer = 0.6,
sdLowerFact = 0.001,
zeroGradFirstReset = TRUE,
zeroGradRunReset = TRUE,
zeroGradBobyqa = TRUE,
```

```
  mceta = -2L,
  warm = c("calc", "save", "none"),
  nAGQ = 0,
  agqLow = -Inf,
  agqHi = Inf,
  sensMethod = c("default", "forward"),
  linCmtSensCarry = c("auto", "none"),
  zeroTheta = 0.001,
  boundedTransform = TRUE
)
```

Arguments

sigdig	Optimization significant digits. One value drives, with a single consistent formula, the inner/outer optimizer convergence tolerance (10^-sigdig), the boundary check tolerance (5*10^(-sigdig+1)), and the ODE solver tolerances: the rtol exponent IS sigdig and atol sits three orders below, so rtol = 10^-sigdig, atol = 10^(-sigdig-3) for every solver (stiff, non-stiff or auto-switching). The sensitivity (atolSens/rtolSens) tolerances match the main solve (the outer gradient and covariance are built from them); the steady-state (ssAtol/ssRtol) tolerances run one order looser. Keying the optimizer to the same 10^-sigdig means it converges to exactly the precision the solve supports. At the default sigdig = 3 this is atol = 1e-6, rtol = 1e-3.
...	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	Either a scalar print-frequency (‘0‘ = suppress, ‘1‘ (default) = every evaluation, ‘N‘ = every Nth), OR a pre-built [iterPrintControl()] object. Equivalent to ‘iterPrintControl(every = print, ncol = printNcol, useColor = useColor)‘.
printNcol	Integer (or ‘NULL‘) parameter columns per row before wrapping. ‘NULL‘ (default) uses ‘floor((getOption("width") - 23) / 12)‘.
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	Parameter normalization/scaling used to get scaled initial values for scaleType, of the form Vscaled = (Vunscaled-C1)/C2 (see Feature Scaling; rescale2 follows the OptdesX manual): "rescale2" scales all parameters to (-1, 1); "rescale" (min-max) scales to (0, 1); "mean" centers on the mean with range (0, 1); "std" standardizes by mean/sd; "len" scales to unit (Euclidean) length; "constant" performs no normalization (C1=0, C2=1).

`scaleType` The scaling scheme for nlmixr2: `"nlmixr2"` (default) scales as `(current-init)*scaleC[i] + scaleTo`, with `scaleTo` from `normType` and scales from `scaleC`; `"norm"` uses the simple scaling from `normType`; `"mult"` scales multiplicatively as `current/init*scaleTo`; `"multAdd"` scales linearly (`(current-init)+scaleTo`) for parameters in an exponential block (e.g. `exp(theta)`) and multiplicatively otherwise.

`scaleCmax` Maximum value of the scaleC to prevent overflow.

`scaleCmin` Minimum value of the scaleC to prevent underflow.

`scaleCband` Length-2 increasing pair 'c(low, high)' (default 'c(0.1, 10)'). The derivative-based scaling constant of a linear 'theta' ('1/linit|'), or of a transformed one whose transform has no band of its own, is kept when it lands inside this band, and otherwise replaced by the parameter's native magnitude 'linit|', so a small initial estimate (a covariate coefficient, say) does not get a huge constant. Transformed thetas are guarded to bands of their own transform, which catch 'log()' at init '1', 'logit' at the interval midpoint and 'factorial'/'gamma' at a digamma zero. The constants of residual-error parameters and those given in 'scaleC' are used as they are.

`scaleC` Scaling constant used with `scaleType="nlmixr2"`; when not specified, chosen by parameter type to keep gradient sizes similar on a log scale: '1' for exp()-transformed/power/boxCox/ yeoJohnson parameters, '0.5*abs(est)' for additive/proportional/ lognormal error parameters, 'abs(1/digamma(est+1))' for factorials, and 'log(abs(est))*abs(est)' for log-scale parameters. May be set explicitly per parameter if these defaults don't apply well.

`scaleC0` Number to adjust the scaling factor by if the initial gradient is zero.

`derivEps` Forward difference tolerances (relative, absolute); step size `h = abs(x)*derivEps[1] + derivEps[2]`.

`derivMethod` Derivative method for the outer problem: "switch", "central", or "forward". "switch" starts forward and toggles to central 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 the covariance. `"r,s"` (the default) is the sandwich estimator (see below). `"analytic"` uses the exact analytic observed-information R-matrix (reported as R^{-1}) and additionally returns the residual and `Omega` standard errors; it covers FOCEI/FOCE fits with additive, proportional, or combined error, mu-referenced/covariate/other structural parameters (and non-mu-referenced etas), and SD-scale inter-occasion variability, and emits a message and falls back to the finite-difference Hessian for anything out of scope (FO, `nAGQ > 1`, censoring, DV-transformed error, bounded-parameter transforms, a structural theta shared by two etas, non-SD `iovXform`, or a pure-proportional variance that vanishes at a near-zero prediction). The finite-difference methods use R (the Hessian) and S (the sum of individual gradient cross-products at the empirical Bayes estimates): `"r,s"` sandwich (`solve(R)%*%S%*%solve(R)`), `"r"` Hessian-based (`solve(R)`), `"s"` cross-product-based (`solve(S)`), or `""` to skip the covariance step. `"sa"` (SAEM Louis stochastic-approximation FIM) and `"imp"` (importance-sampling Monte-Carlo observed information) are also

accepted for any method; they are computed post-fit at the converged estimates by the decoupled recompute engine.

`covSolveTol` absolute/relative ODE tolerance for the covariance solves – the perturbed solves behind the finite-difference methods (every probe and the stencil centre it is compared against) and the augmented-sensitivity solves behind `covMethod="analytic"`. `NULL` (default) derives them from the fit's own tolerances: the finite-difference solves use `atol` and `rtol` each times 1e-3, capped at 1e-7 (at the default `sigdig = 3`, `rtol = 1e-7` and `atol = 1e-9`), and the analytic augmented solves use `max(1e-12, min(1e-8, 10^-(sigdig + 6)))`. The inner problems of the finite-difference probes are tightened the same way, whichever `innerOpt` runs them: `trustFterm` and `trustMterm`, `epsilon` (n1qn1), and the `innerLbfgs*` tolerances (lbfgsb3c) each times 1e-3, capped at 1e-9. No derived tolerance goes below 1e-12 unless the fit's own already is. A number sets `atol = rtol = covSolveTol` for both kinds of solve. Estimation itself always runs at the fit's tolerances. When the finite-difference standard errors look noisy (they change with small changes to the steps or the estimates, or disagree with `covMethod="analytic"`), a tighter value such as `1e-10` or `1e-12` can help: every probe's solve error enters the differences.

`covFallback` what each `covMethod` falls back to when it gives no usable covariance: a named list, one element per method, each the ordered methods to try instead (the whole list, not followed further). The default is the established behaviour: `"r,s"` falls to `"r"` then `"s"` (whichever matrix is usable), `"r"` to `"s"`, `"analytic"` to the finite-difference `"r,s"`, `"r"` and `"s"`, and `"s"` to nothing. A list may also name `"sa"` and `"imp"`, which are computed after the fit when the finite-difference methods before them gave no covariance. A list you give replaces the default: a method it does not name has no fallback. The `"r,s"` check of a doubtful sandwich (see `covSmall`) only picks a listed method. The methods tried and why each was not used are in `fit$env$covTried`.

`covPrecursor` what a finite-difference covariance may start from, in order of preference: `"fd"`, a full-stage R this fit computed under other settings (`setCov()` with another `rsControl()`), and `"analytic"`, the analytic covariance the fit holds. The first one the fit holds seeds the full stage's step searches with its diagonal; every step still passes the same acceptance test. A fresh fit holds neither, so this matters for `setCov()`. `NULL` uses none. How it served is in `fit$env$covPrecursorUsed` and the fit print.

`covShortcut` with a precursor (see `covPrecursor`), predict the full R from the measured diagonal and the precursor's correlations, check the prediction against the objective along four fixed directions, and skip measuring the off-diagonals when it agrees to 1% in each (the record says "accepted"); otherwise they are measured as usual ("fell back"). `FALSE` (default) always measures them.

`covFull` shape of `fit$cov`. `TRUE` (default) installs the full theta + residual sigma + Omega covariance (assembled analytically for `covMethod="analytic"`, or by central finite differences over the same parameter set otherwise). For the finite-difference methods it follows `covMethod`: `"r,s"` is the full sandwich `solve(Rfull) %*% Sfull %*% solve(Rfull)`, `"s"` is `solve(Sfull)`, `"r"` is `solve(Rfull)`. `FALSE` installs only the structural-theta block (the historical shape). The installed shape is named by `fit$covMethod` – `"r,s (full)"` versus `"r,s"` – and

the other shape is cached, so setCov() swaps between them without recomputing either. Both shapes come from one finite-difference computation over the full parameter set (the structural-theta shape is its theta block), at the steps gillKcov, gillStepCov and gillFtolCov choose, so rmatNorm, smatNorm and covGillF do not apply; the theta-only stage runs separately (with them) for a mixture, a mu-referenced model, a generalized log-likelihood, gillKcov = 0 or covDerivMethod = "forward".

fast — When TRUE, compute the outer (population) gradient analytically from Almquist (2015) sensitivity equations instead of by finite differences, and use the Eq-48 random-effect extrapolation for the next inner-problem starting values. Requires an analytic-scope model. Conditionally Gaussian endpoints route through the general (f,R) assembler, which covers more than the plain add/prop case – multiple endpoints, combined and power error, both-sides transforms and a single estimated boxCox/yeoJohnson lambda. A single non-Gaussian (ll()/generalized) endpoint instead differentiates the log-density directly, giving an exact inner Hessian and analytic outer gradient. Out of scope are linCmt(), fo, IOV, more than one estimated lambda, a theta mu-referenced by several random effects, and (for the non-Gaussian path) multiple endpoints, censoring or nAGQ > 1; those fall back to the finite-difference gradient with a message (linCmt() and out-of-scope log-likelihood models downgrade to fast=FALSE up front). When unspecified, the outer optimizer defaults to "lbfgsb3c" (vs "bobyqa" for fast=FALSE); pairing fast=TRUE with a derivative-free outerOpt reverts to fast=FALSE. The *f methods (e.g. foceif) default this to TRUE.

priorMethod — Which of the shared prior kernel's three omega conventions (nlmixr2/rxode2#1270) to evaluate an ini({}) prior() under – "general" (textbook Bayesian), "nwpri" (NONMEM's own $PRIOR NWPRI), or "tnpri" (the Monolix/NONMEM-own-estimation joint-normal convention on omega). The default, "auto", reads the convention off what the model's own ini({}) actually wrote (an invWishart() degrees-of- freedom prior on an omega block means "nwpri"; a normal prior directly on an omega element means "tnpri"; anything else, including a dcauchy() prior, means "general") – deliberately never a fixed default, since the identical invWishart(nu) syntax means a different number under "general" and "nwpri". Setting this explicitly forces that convention regardless of what auto-detection would have picked, and errors before any estimation starts if the model's priors are not representable under it (e.g. priorMethod="tnpri" on a model with an invWishart() prior). Ignored when the model has no prior at all.

"none" builds no prior specification: the ini({}) priors stay on the model but are not evaluated, so the objective is the plain (unpenalized) likelihood. It exists for callers that handled the priors somewhere else and only finalize through this control – a sampler that already drew from the posterior, say – where building a specification would at best be unused and at worst refuse a prior this kernel cannot evaluate but the caller could.

fdOutlierZ — Cut of the Iglewicz-Hoaglin modified z-score that decides whether a finite-differenced subject's slope is an outlier against the exact analytic slopes, and so whether 'fdChartrand' refines it. The conventional 3.5; lower it to make the pass fire more readily (and to exercise it), raise it to suppress it without turning 'fdChartrand' off.

`fdOutlierScale` Test the outlier criterion on the **per-observation** slope (‘TRUE‘, the default) rather than the raw one. A per-subject slope scales with how much data that subject carries, so raw slopes from a 3-observation and a 20-observation subject are not draws from one distribution – pooling them makes a legitimately large slope look like an outlier on unbalanced or sparse data. Only the test is scaled; the gradient itself is untouched. On balanced data this changes nothing, since dividing every slope by the same count leaves the modified z-score unchanged.

`fdRefine` Estimator used to recompute a finite-differenced slope once the outlier pass fires. On noisy, hard-to-solve likelihood surfaces the ordering is roughly ‘"richardson"‘ < ‘"lanczos"‘ < ‘"chartrand"‘:

* ‘"richardson"‘ – Richardson extrapolation of central differences at ‘h, h/v, h/v^2, ...‘, cancelling the ‘h^2, h^4, ...‘ truncation terms in turn. Cheapest, and right when the surface is smooth and only truncation matters. It extrapolates toward ‘h -> 0‘, which is *into* the noise, so it is the wrong instrument when the noise floor is what limits the difference. * ‘"lanczos"‘ – the Lanczos generalized derivative, a least-squares slope through ‘2m+1‘ points. Same ‘O(h^2)‘ truncation as a central difference but lower variance, since independent evaluation noise averages down as points are added rather than being amplified. The middle rung. * ‘"chartrand"‘ (default) – total-variation regularized differentiation over a wide interval, which absorbs curvature through the regularized derivative itself instead of assuming a stencil. Most expensive and most robust to a genuinely rough surface.

All three apply identically: they are gated by the same outlier test, touch only the finite-differenced subjects, and never recompute a subject whose analytic gradient is available.

`fdLanczosM` Half-width ‘m‘ of the ‘"lanczos"‘ estimator (‘2m‘ evaluations).

`fdRichardsonR` Depth of the ‘"richardson"‘ extrapolation table.

`fdRichardsonV` Step-shrink ratio ‘v‘ of the ‘"richardson"‘ estimator.

`fdChartrandAll` When the outlier pass fires for a parameter, refine **every** finite-differenced subject with the Chartrand TV derivative rather than only the outlying ones. Subjects whose augmented solve succeeded keep their exact analytic gradient either way – only finite differences are ever recomputed. Default ‘FALSE‘ (refine the outliers only).

`fdOutlierAny` Let an outlier among the **exact analytic** slopes fire the outlier pass as well, not only an outlier among the finite differences. Still only the finite differences are recomputed. Default ‘FALSE‘.

`fdIndividualStep`
For the per-subject finite-difference fallback of the analytic outer gradient (‘fast=TRUE‘), search the shi step size separately for every flagged subject (‘TRUE‘, the default) rather than once on the summed objective over them. The subjects that reach this path are the badly conditioned ones, so one shared step cannot suit them all; a per-subject search also supplies the population of converged peers a clamped step is repaired from. ‘FALSE‘ restores the single shared step, which costs fewer evaluations.

`fdChartrand` Refine finite-difference slopes that the robust outlier test flags (default `TRUE`). When a subject’s per-parameter slope sits far outside the modified z-score interval of the others, its central difference is suspect; those slopes – and only those

– are recomputed with a total-variation regularized derivative (Chartrand) on a wide interval. Set `FALSE` to keep the plain central difference.

On by default because the outlier test is itself the gate: a well-behaved problem flags nothing and pays nothing, so the cost falls only on the complex fits where a slope really is an outlier – exactly where you would want the refinement, and where a user is least likely to know to ask for it. `fit$env$nFdOutlier` reports flagged parameters and refined slopes, so you can see whether it engaged for a given fit.

Worth knowing when judging it: the measurements that originally motivated this refinement were taken while the likelihood and the Shi step selection were both faulty, so they do not evidence its value on current code, and it has not been observed to trigger on ordinary fits.

`foceEbeTol` Convergence tolerance on the score of the FOCE frozen-variance EBE re-solve, which the analytic outer gradient (`fast=TRUE`, `interaction=FALSE`) needs because FOCE's mode is not the inner problem's mode. `NULL` (default) uses `1e-9`; the first iteration uses a looser `1e-3` so an already-stationary eta is returned untouched. Unlike the solver and optimizer tolerances this is not derived from `sigdig` – it is a convergence target on an inner Newton rather than a precision request. It also bounds the Newton decrement at which a stalled subject is accepted at the ODE solve's noise floor rather than finite-differenced on its own. Set it explicitly to test whether a fit's finite-difference fallbacks are tolerance-driven.

`hessEps` is a double value representing the epsilon for the Hessian calculation. This is used for the R matrix calculation.

`hessEpsLlik` is a double value representing the epsilon for the Hessian calculation when doing focei generalized log-likelihood estimation. This is used for the R matrix calculation.

`optimHessType` Hessian type for numeric-difference individual Hessians in generalized log-likelihood estimation: "central" (matches R's 'optimHess()', default) or "forward" (faster).

`optimHessCovType`

Hessian type for numeric-difference individual Hessians used for the covariance step/final likelihood: "central" (more accurate, used here) or "forward".

`hessEtaStepMin` Floor on the finite-difference step used for the individual (eta) Hessian in generalized log-likelihood estimation, expressed as a fraction of that random effect's own standard deviation (`sqrt(diag(Omega))`). Default `0.05`; `0` restores the plain absolute `shi21hMin` floor.

The Shi (2021) step search is told the function's noise floor is `rxControl(atolSens=)`, but the inner gradient it differences comes out of a sensitivity solve that `rtolSens` governs as well, so the noise is understated and the search shrinks the step until the difference is taken inside it. `n1qn1` absorbs that – it uses this Hessian only as a warm-start seed and then corrects it by its own quasi-Newton updates as it iterates – but `innerOpt="trust"` re-derives it as its trust-region model Hessian at every trial point, with nothing to correct it, and adds its log-determinant to the reported objective. Flooring the step relative to the eta scale stops the runaway without paying for a tighter solve. It applies to every inner optimizer: the step a finite difference needs is a property of the problem, not of who consumes the Hessian.

censOption — Treatment of the second derivative for censored (M2/M3/M4/BLQ) observations in the FOCEI family. "gauss" (the default) keeps the historic uncensored Gauss-Newton curvature, matching common PMx tools; "laplace" uses the exact censored second derivative of the objective (a proper Laplace inner Hessian and analytic covariance). Accepted by saemControl/nlmControl for a uniform interface but inert there – SAEM (stochastic EM) has no Laplace inner Hessian, and NLM uses a finite-difference Hessian that already reflects censoring exactly.

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

eventSens — Controls how dosing/event-parameter (‘alag‘, ‘F‘, ‘rate‘, ‘dur‘) sensitivities are computed for THETA/ETA gradients: ‘"jump"‘ (default) uses rxode2's analytic event sensitivities; ‘"fd"‘ uses the legacy finite-difference behavior. Also gates the analytic moving-boundary correction for a modeled ‘alag()‘/‘f()‘ on a ‘linCmt()‘ compartment; set ‘"fd"‘ if that model infuses a dose into the lagged/scaled compartment (rxode2/rxode2#1236), or if the regimen also doses an *unlagged/unscaled* compartment alongside the lagged/scaled one – a common design for estimating ‘f()‘ from paired IV+oral data (rxode2/rxode2#1237).

centralDerivEps — Central difference tolerances (relative, absolute); step size 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 — Projected-gradient convergence tolerance for "L-BFGS-B": iteration stops when max(| proj g_i |) <= lbfgsPgtol. Defaults to ‘0‘ (check suppressed).

lbfgsFactr — Convergence factor for "L-BFGS-B": converges when the objective reduction is within lbfgsFactr * .Machine$double.eps. Derived from sigdig as 10^(-sigdig-2) / .Machine$double.eps, two orders tighter than the other sigdig-derived tolerances. It tests the objective reduction of a SINGLE step rather than stationarity, so a target of 10^-sigdig stops as soon as one step is small; the extra two orders are what make the analytic-gradient (fast=TRUE) methods reach the same optimum as the derivative-free default.

eigen — A boolean indicating if eigenvectors are calculated to include a condition number calculation.

diagXform — Transformation used on the diagonal of chol(solve(omega)) (the FOCEi-estimated parameters): one of "sqrt" (default), "log", or "identity".

iovXform — Transformation used on the diagonal of the IOV: one of "sd", "var", "logsd", or "logvar". This parameterizes the magnitude theta of the "theta" iovMethod; it has no effect under "omega", where the magnitude is fixed at one and the variability is carried by the omega block itself.

iovMethod — How inter-occasion variability is expanded before estimation: one of "auto" (default), "theta" or "omega".

"theta" is the long-standing expansion: each occasion parameter becomes a magnitude theta, with one unit-variance eta per occasion fixed to it. That shape cannot represent a correlation between two occasion parameters.

"omega" instead fixes the magnitude theta at one and estimates the per-occasion eta blocks directly, occasion one being the block and the rest repeating it (NONMEM's $OMEGA BLOCK(n) SAME). The correlation then lives in the estimated block.

"auto" picks "omega" when the IOV block has any off-diagonal element, since "theta" provably cannot represent one, and "theta" otherwise. "omega" may be asked for on a diagonal block as well, so the two can be compared on the same model. The choice is made per LEVEL of variability, so a correlation on occ does not change how an unrelated diagonal occ2 is expanded.

"omega" is only available for estimation methods that honour the repeated block (the FOCEi family). saem and the variational, nonparametric and importance-sampling methods estimate omega elsewhere, so they continue to refuse a correlated occasion block rather than silently estimate each occasion on its own.

The two are the same statistical model – at an occasion variance of one, where the parameterizations coincide, they agree on the objective to machine precision, and evaluated at matched random effects they agree to ~2e-8 at any variance. They are not interchangeable in practice: "theta" hands FOCEi's inner optimizer unit-scale etas and so converges the inner problem better when the occasion variance is far from one. That is why "auto" keeps "theta" unless a correlation forces "omega".

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.

literalFix — boolean, substitute fixed population values as literals and re-adjust ui and parameter estimates after optimization; Default is 'TRUE'.

literalFixRes — boolean, substitute fixed population values as literals and re-adjust ui and parameter estimates after optimization; Default is 'TRUE'.

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

useColor — Logical (or 'NULL') emit ANSI bold/color escapes in the iteration print. 'NULL' (default) defers to [crayon::has_color()].

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)

foce — Controls how FOCE (interaction = FALSE) evaluates the residual variance R in the inner objective; ignored for FOCEi. Either "nonmem" (default) or "foce+":

- "nonmem" freezes R at the eta = 0 population prediction and holds it constant across the inner optimization. This follows NONMEM's residual-variance convention for FOCE without interaction.
- "foce+" evaluates R at the current conditional eta (the live variance), keeping the truncated FOCE inner score, which is refined before evaluating the marginal objective. It retains the live-variance convention used in **nlmixr2est** 6.0.1 and earlier. It differs from NONMEM's FOCE without interaction and omits the variance derivatives included in FOCEI.

`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` P-value for resetting an individual ETA to 0 during optimization, based on a z-test of `chol(omega^-1) %*% eta` or `eta/sd(allEtas)`. '0' = never reset, '1' = always reset.

`resetThetaP` P-value for resetting mu-referenced THETAs based on ETA drift, checked at the start and near a local minimum (see `resetThetaCheckPer`). '0' = never reset (the default); '1' is not allowed. Defaults to off: when the etas cannot re-center the reset repeats without progress and can error the fit out, and where it converges it reaches a worse optimum than leaving it off.

`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. '0' = never reset (the default); see `resetThetaP`.

`diagOmegaBoundUpper`
Upper bound of the diagonal omega matrix, as `diag(omega)*diagOmegaBoundUpper`. '1' = no upper bound.

`diagOmegaBoundLower`
Lower bound of the diagonal omega matrix, as `diag(omega)/diagOmegaBoundLower`. '1' = no lower bound.

`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 Fast `"nlminb"` fits automatically use the analytical outer Hessian for supported Gaussian FOCE/FOCE+/FOCEI/AGQ models, restarting with gradients only if curvature is unavailable. `"trust"` is a trust-region Newton method (**RcppTrust**) built on that same Hessian; see `outerTrustHessian`. It is unbounded, so a trial point outside the box is reported as an infinite objective and the region shrinks instead.

`innerOpt` optimization method for the inner (per-subject eta) problem: '"auto"' (default), '"trust"' (RcppTrust trust-region Newton, using an exact Gauss-Newton+Omega^-1 Hessian every iteration), '"n1qn1"' (quasi-Newton, gets a Hessian only once as a warm-start seed) or '"lbfgsb3c"' (thread-safe L-BFGS-B from 'lbfgsb3c', controlled by the 'innerLbfgs*' arguments; '"auto"' never picks it).

'"auto"' picks '"n1qn1"' for a generalized-likelihood endpoint ('dnorm()', 'll()', 'dpois()', ...) and '"trust"' for everything else. Such an endpoint has no Gauss-Newton shortcut, so the inner eta Hessian is finite-differenced; '"trust"' rebuilds it at every trial point, which costs 2*neta inner solves each time, while '"n1qn1"' builds it once as a warm-start seed and corrects it with its own quasi-Newton updates. On everything else '"trust"' is typically the faster of the two.

`innerHessian` Inner optimization curvature: ‘"focei"‘ (default) or ‘"conditional"‘. Full conditional curvature requires fast Gaussian FOCEI. Inner trust uses it at each trial; n1qn1 uses it with ‘warm="calc"‘. Value, gradient and full curvature share one sensitivity solve. The marginal objective’s FOCEI curvature is unchanged. It is not supported with registered external likelihood contributions.

`detHessian` Curvature entering the objective’s Laplace log-determinant: ‘"focei"‘ (default), the Gauss-Newton expected information, or ‘"conditional"‘, the full conditional Hessian (the observed information at the conditional mode, as NONMEM’s LAPLACE). ‘"conditional"‘ requires fast Gaussian FOCEI; the analytic outer gradient then carries the matching third-order terms, and the analytic covariance falls back to finite differences.

`hessianMethod` For a non-normal-endpoint model (any distribution other than `norm`), the per-subject inner Hessian has no Gaussian Gauss-Newton shortcut and falls back to a finite difference of the gradient every ‘innerOpt="trust"‘ Newton step (‘calcEtaHessian()‘, ‘src/inner.cpp‘). ‘"fd"‘ (default) recomputes this Hessian from scratch every Newton step via finite difference. ‘"bfgs"‘ is the damped BFGS update (Nocedal & Wright, *Numerical Optimization*, 2nd ed., 2006, Procedure 18.2), always positive definite; ‘"sr1"‘ is the Symmetric Rank-1 update (Nocedal & Wright; Murtagh & Sargent, *Comput. J.* 13, 1970), not forced positive definite; ‘"bofill"‘ is Bofill’s (1994, *J. Comput. Chem.* 15, 1-11) SR1/Powell-Symmetric-Broyden blend. All three are built from consecutive Newton steps’ already-computed gradients (no extra evaluations), seeded from one ‘"fd"‘-style Hessian on the first Newton step of each inner solve.

Unlike ‘trustControl(hessianMethod=)‘’s analogous OUTER-theta option (where ‘"sr1"‘ is the default), this inner Hessian is not just a step-direction aid: ‘LikInner2()‘ (‘src/inner.cpp‘) adds this Hessian’s log-determinant directly into the reported Laplace objective, which the OUTER optimizer then searches over. A quasi-Newton estimate built from the handful of Newton steps one subject’s inner solve takes is accurate enough to guide the step but not accurate enough to serve as that objective term – confirmed on a real one-compartment IV model fit as a general `ll()`/‘dnorm()‘ endpoint: ‘"bfgs"‘/‘"sr1"‘/‘"bofill"‘ all converged to the same wrong ‘Vc‘ (about 90 against a simulated 70 and a plain (non-‘ll()‘) ‘focei‘ fit’s 67, with a *worse* reported objective than ‘"fd"‘’s correct answer) – the biased log-determinant misleads the outer search into a worse point it reports as better. ‘"fd"‘’s fresh finite difference has no such bias, so it stays the default here even though ‘"bfgs"‘/‘"sr1"‘/‘"bofill"‘ remain available (and are genuinely faster, per this package’s own small benchmark, ‘inst/benchmarks/benchmark-focei-hessian-method.R‘) for a caller who has verified their model does not depend on this Hessian’s precision. Has no effect for normal-endpoint models (the Gauss-Newton inner Hessian is used unconditionally there). Only meaningful with ‘innerOpt="trust"‘, which the default ‘innerOpt="auto"‘ does not pick for these models – so setting this needs ‘innerOpt="trust"‘ pinned as well. Asking for ‘"bfgs"‘/‘"sr1"‘/‘"bofill"‘ under any other inner optimizer is an error rather than a silent no-op: ‘foceiControl()‘ refuses a pinned ‘innerOpt‘, and the fit refuses one ‘"auto"‘ resolves to ‘"n1qn1"‘.

`trustConf` confidence level defining the ‘innerOpt="trust"‘ trust-region radius: since eta ~ N(0, Omega), the radius (in sqrt(diag(Omega))-scaled units) is ‘sqrt(qchisq(trustConf,

df=neta))‘, the boundary of the ‘trustConf‘-level eta confidence region. Default 0.975.

`trustRinit` initial ‘innerOpt="trust"‘ trust-region radius. ‘NULL‘ (default) derives it from ‘trustConf‘.

`trustRmax` maximum ‘innerOpt="trust"‘ trust-region radius. ‘NULL‘ (default) derives it from ‘trustConf‘.

`trustFterm`, `trustMterm`

‘innerOpt="trust"‘’s own function-value and predicted-decrease convergence tolerances for the per-subject Newton solve. ‘NULL‘ (default) uses ‘10^(-sigdig-2)‘, two orders tighter than the plain ‘10^(-sigdig)‘ most tolerances here use, for the same reason ‘lbfgsFactr‘ is: the inner solve is the function the outer problem differentiates, so this tolerance sets the objective’s noise floor and a finite-difference outer gradient cannot resolve a step below it. Left at the plain ‘10^(-sigdig)‘, an ‘nAGQ=2‘ ‘theo_sd‘ fit stopped at an objective of 134.46 against 118.52, and took longer doing it – the noisy gradient misleads the outer search as well as lengthening it. Deliberately NOT derived from ‘epsilon‘ (‘"n1qn1"‘’s own, unrelated "precision of estimate" tolerance) – tying ‘"trust"‘’s stopping criterion to a value picked for a different optimizer is exactly the coupling these parameters exist to remove. Has no effect unless ‘innerOpt="trust"‘.

`trustPolish` logical; when ‘TRUE‘, each converged ‘innerOpt="trust"‘ solve takes up to 4 more Newton steps on the ETAs, down to ‘trustFterm‘. This makes the objective less dependent on the warm-start ETAs and can help a fit that stops short of its minimum (#1152). The ‘"bobyqa"‘ outer search then also restarts once from where it stops, which costs about twice the outer evaluations. ‘FALSE‘ (default) keeps the plain trust solve and a single outer search.

`bobyqaStationary`

logical; when ‘TRUE‘ (default), a normal exit of the ‘"bobyqa"‘ outer search is checked with a central-difference gradient and a downhill probe. If the probe lowers the objective by more than ‘10^(1-sigdig)‘, bobyqa restarts from there (up to 3 times, within ‘maxOuterIterations‘). This catches a search that stops in a narrow valley short of the minimum (#1196); ‘FALSE‘ skips the check.

`innerLbfgsLmm` number of BFGS updates retained by the ‘innerOpt="lbfgsb3c"‘ L-BFGS-B solve; a whole number of at least 1. Defaults to 5.

`innerLbfgsFactr`, `innerLbfgsPgtol`

‘innerOpt="lbfgsb3c"‘’s own L-BFGS-B objective-reduction factor and projected-gradient tolerance, separate from the outer ‘lbfgsFactr‘/‘lbfgsPgtol‘. ‘NULL‘ (default) derives them from ‘sigdig‘: ‘innerLbfgsFactr‘ is ‘10^(-sigdig-2) / .Machine$double.eps‘ (floored at 1) and ‘innerLbfgsPgtol‘ is ‘10^(-sigdig-2)‘, the same scale as ‘trustFterm‘.

`innerLbfgsAbstol`, `innerLbfgsReltol`

‘innerOpt="lbfgsb3c"‘’s absolute and relative eta-change tolerances, separate from the outer ‘abstol‘/‘reltol‘. ‘NULL‘ (default) uses ‘10^(-sigdig-2)‘. With ‘innerOpt="lbfgsb3c"‘, ‘maxInnerIterations‘ caps the function evaluations of each solve.

`outerTrustHessian`

Curvature source for `outerOpt="trust"`. `"auto"` (default) uses the analytical outer Hessian when `fast=TRUE` makes it available and the damped-BFGS

update otherwise, and falls back to that update if the Hessian is refused mid-fit. "analytic" requires fast=TRUE. "bfgs" is the damped BFGS update (Nocedal & Wright, Numerical Optimization 2nd ed., Procedure 18.2) built from consecutive gradients, so it adds no evaluations. "fd" differences the outer gradient, costing one extra population gradient per parameter per iteration.

outerTrustRinit, outerTrustRmax
: Initial and maximum trust-region radius for outerOpt="trust", in the scaled-parameter space the outer problem optimizes in. NULL (default) derives outerTrustRinit from minqa::bobyqa()'s own default-rhobeg formula and outerTrustRmax as 8 * outerTrustRinit.

outerTrustFterm, outerTrustMterm
: Function-value and predicted-decrease convergence tolerances for outerOpt="trust". NULL (default) uses 10^(-sigdig-2); outerTrustMterm defaults to outerTrustFterm.

outerTrustRelStep
: Relative step handed to the analytical outer Hessian, and used for the outerTrustHessian="fd" gradient difference.

outerTrustRestarts
: How many times outerOpt="trust" may re-enter the trust region from its own reported solution when the Newton decrement there says the point is not stationary. A collapsing trust region satisfies the solver's own convergence test at a point that is not a minimum; re-entering restores the initial radius. maxOuterIterations is the total across restarts, not per restart.

rhobeg
: Initial trust region radius for the bobyqa outer optimizer (with 'rhoend', must satisfy '0 < rhoend < rhobeg'). Default '0.2' (20 'abs(upper-lower)/2'. (bobyqa)

rhoend
: Final trust region radius. If not defined, '10^(-sigdig)' is used. (bobyqa)

npt
: Number of points for bobyqa's quadratic approximation to the objective; must be in '[n+2, (n+1)(n+2)/2]'. Defaults 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.

muModel
: Mu-referenced-FOCEI-family regression variant: "none" (default, ordinary FOCEI); "lin" (mfocei/mfoce/magq/mlaplace) profiles mu-referenced population thetas and covariate coefficients out of the outer optimizer via closed-form OLS regression of each subject's back-calculated value on the covariates (muModelTol/muModelMaxCycles); "irls" (ifocei/ifoce/iagq/ilaplace) reweights that by inner-optimization curvature. Bounded mu parameters are regression-updated with a clamped step (muModelClampRetries); a user-fixed (fix()) mu theta is never updated.

`muRefCovAlg` When 'TRUE' (default), algebraic expressions that can be mu-referenced are internally rewritten as mu-referenced covariates and restored after optimization. Mirrors `saemControl(muRefCovAlg=)`/`nlmeControl(muRefCovAlg=)`; for `foceiControl()` only takes effect when `muModel != "none"`.

`muModelTol` Convergence tolerance for the mu-referenced-FOCEI-family "re-optimize etas, then regress" cycle (`muModel != "none"`): repeats until the max mu-group theta change drops below this value or `muModelMaxCycles` is reached.

`muModelMaxCycles`
Maximum number of "re-optimize etas, regress" cycles per outer iteration (see `muModel`, `muModelTol`).

`muModelClampRetries`
Maximum number of active-set re-solve passes per group per regression update when a bounded mu-referenced parameter must be clamped to its bound (see `muModel`); on hitting the cap the current clamped-feasible solution is used.

`stateTrim` Trim state amounts/concentrations to this value.

`shi21maxOuter` The maximum number of steps for the optimization of the forward-difference step size. When not zero, use this instead of Gill differences.

`shi21maxInner` The maximum number of steps for the optimization of the individual Hessian matrices in the generalized likelihood problem. When 0, un-optimized finite differences are used.

`shi21maxInnerCov`
The maximum number of steps for the optimization of the individual Hessian matrices in the generalized likelihood problem for the covariance step. When 0, un-optimized finite differences are used.

`shi21maxFD` The maximum number of steps for the optimization of the forward difference step size when using dosing events (lag time, modeled duration/rate and bioavailability)

`shi21hMax` Upper bound on the adaptive shi21 finite-difference step size for FOCEi gradients (both the inner eta and outer theta/covariate finite differences). The step-size search never probes a parameter by more than this on its estimation scale; a larger value lets the gradient of a flat, small-magnitude parameter (e.g. a covariate coefficient near 0) clear the ODE-solver noise floor, at the cost of risking a degenerate solve at the probe.

`shi21hMin` Lower bound on the adaptive shi21 finite-difference step size for FOCEi gradients. The floor is limited by the ODE solver tolerance (atol/rtol), not machine precision; below it the finite difference is dominated by solver noise.

`gillK` Max steps to determine the optimal forward/central difference step size per parameter (Gill 1983). '0' = no 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` Max steps to determine the optimal forward/central difference step size per parameter (Gill 1983) during the covariance step. '0' = no optimal step size determined.

`gillKcovLlik` Same as `gillK` but for the generalized focei log-likelihood method (Gill 1986).

`gillStepCovLlik`
Same as above but during generalized focei log-likelihood

`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.

`gillFtolCovLlik`
Same as above but applied during generalized log-likelihood estimation.

`rmatNorm` A parameter to normalize gradient step size by the parameter value during the calculation of the R matrix of the theta-only stage (see `covFull`)

`rmatNormLlik` A parameter to normalize gradient step size by the parameter value during the calculation of the R matrix if you are using generalized log-likelihood Hessian matrix.

`smatNorm` A parameter to normalize gradient step size by the parameter value during the calculation of the S matrix of the theta-only stage (see `covFull`)

`smatNormLlik` A parameter to normalize gradient step size by the parameter value during the calculation of the S matrix if you are using the generalized log-likelihood.

`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 of the theta-only stage (see `covFull`).

`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` Small number used to compare covariance estimates (sandwich vs R/S matrix) before rejecting one as too small to be the final covariance estimate.

`adjLik` When 'TRUE', adjusts the likelihood by the 2*pi constant nlmixr2's objective function otherwise omits (to match NONMEM), more closely matching nlme/SAS likelihood approximations. The objective function itself always matches NONMEM regardless.

`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` When n1qn1 optimization of an ETA (starting at zero) misbehaves, reset the Hessian and nudge the ETA up by this value, then down if it still doesn't move. Defaults to 'qnorm(1-0.05/2)*1/sqrt(3)'. Falls back to `etaNudge2`, then to zero (stop optimizing) if unsuccessful.

`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

`etaRestart` Number of Omega draws the inner restart cascade tries once the 'etaNudge'/'etaNudge2' restarts are spent and the ETA solve is still not converged. Every nudge sets EVERY ETA to the same constant, which explores poorly when the inner problem has more than one basin; a draw from Omega is a starting point from the distribution the ETAs actually come from. The draws are taken once per fit, seeded from 'seed' (so the objective stays a function of theta alone and the fit stays reproducible), and are only read after an inner solve has already failed – a fit whose inner solves converge never pays for them. Use 0 to disable.

This applies to 'innerOpt="trust"' (the default for a normal endpoint), which reports a convergence verdict per solve. 'n1qn1' reports none, so its own restart cascade is unchanged.

`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` Integer seed (default '42') used to make a FOCEi fit reproducible and self-contained. The fit (including the 'mceta' Monte-Carlo initial-ETA draws, which pull from rxode2's threefry engine) runs inside [rxode2::rxWithSeed()], so it neither depends on the ambient RNG state nor advances/leaks it – repeated fits in the same session, and fits following other estimation methods, give identical results.

`resetThetaCheckPer`

represents objective function % percentage below which resetThetaP is checked.

`etaMat` Initial (or final) ETA estimates; can also be a prior fit, whose final ETAs are then used as initial values. By default, uses the last fit's ETAs if supplied, else all ETAs start at zero ('NULL'). 'NA' disables reuse from a prior fit.

`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.

`outerMaxOdeRecalc`

Maximum number of times to reduce the ODE tolerances for a single subject and retry when the analytic outer (augmented sensitivity) solve fails. Tracked separately from 'maxOdeRecalc', which governs the inner problem. A subject that solves after loosening still contributes an analytic gradient instead of dropping the whole gradient to finite differences.

`outerOdeRecalcFactor`

The factor the atol/rtol is loosened by on each analytic outer retry; the outer counterpart of 'odeRecalcFactor'.

`outerStickyRecalcN`

The number of bad analytic outer solves for a subject before its loosened tolerance is kept for the rest of the problem; the outer counterpart of 'stickyRecalcN'.

`indTolRelax` When ‘TRUE‘ (default), only subjects whose ODE solve produced NaN/Inf have their tolerances relaxed, and the relaxed tolerance persists across optimizer calls (sticky). When ‘FALSE‘, all subjects have their tolerances relaxed on each retry and tolerances are reset afterward.

`gradProgressOfvTime`
This is the time for a single objective function evaluation (in seconds) to start progress bars on gradient evaluations

`addProp` Type of additive-plus-proportional error: ‘"combined1"‘, where standard deviations add:

$$y = f + (a + b \times f^c) \times \varepsilon$$

; or ‘"combined2"‘, where variances add:

$$y = f + \sqrt{a^2 + b^2 \times f^{2 \times c}} \times \varepsilon$$

. Here y = observed, f = predicted, a = additive sd, b = proportional/power sd, c = power exponent (1 in the proportional case).

`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 (‘NULL‘), it defaults to ‘sigdig‘.

`fallbackFD` Fallback to the finite differences if the sensitivity equations do not solve.

`smatPer` Percentage of failed per-individual parameter gradients (replaced with the overall parameter gradient) out of the total (‘ntheta*nsub‘) above which the S matrix is considered bad.

`sdLowerFact` Factor multiplying the estimate when the lower bound is zero for a standard-deviation error parameter (add.sd, prop.sd, etc); e.g. estimate 0.15 with lower bound 0 assumes a lower bound of 0.00015. ‘0‘ disables this.

`zeroGradFirstReset`
When ‘TRUE‘ (default), reset a zero first gradient to ‘sqrt(.Machine$double.eps)‘ instead of erroring; ‘FALSE‘ errors; ‘NA‘ ignores it only on the last reset attempt.

`zeroGradRunReset`
When ‘TRUE‘ (default), reset a zero gradient encountered mid-run to ‘sqrt(.Machine$double.eps)‘ instead of erroring.

`zeroGradBobyqa` When ‘TRUE‘ (default), a zero-gradient reset switches to the gradient-free bobyqa method; ‘NA‘ only does so for the first zero gradient.

`mceta` Monte Carlo sampling for the best initial ETA estimate (based on ‘omega‘): ‘-2‘ (default) uses the Almquist (2015) Eq-48 extrapolation ‘eta^0 = eta* + (d eta*/d theta)(theta_new - theta_old)‘ when the analytic gradient supplies ‘d eta*/d theta‘ (‘fast = TRUE‘), accepting the extrapolated eta only when it is within the standardized-eta reset bound (else keeping the last eta, or resetting to 0 when that is also out of bound); ‘-1‘ jumps between the extrapolated eta

and eta=0, keeping the better; both ‘-2‘ and ‘-1‘ fall back to keeping the last eta when no analytic ‘d eta*/d theta‘ is available (‘fast = FALSE‘). ‘0‘ uses eta=0 for each inner optimization; for ‘n>0‘, eta=0 and n-1 etas sampled from omega are each evaluated and the best (by inner objective) starts the inner optimization. The carried last eta is deliberately not one of the ‘n>0‘ candidates – it is the previous iteration’s converged conditional mode, so it would win for every subject and ‘n>0‘ would reduce to the keep-last behavior of ‘-1‘/‘-2‘. When a sampled eta wins, the inner problem is also solved from eta=0 and the better converged result kept – compared on the marginal objective the fit reports, which carries the Laplace ‘log|H|‘ term, not on the inner objective the optimizer minimizes – so no inner solve ends above the one ‘0‘ would have reached. The search is skipped while the outer gradient is being differenced, where the eta is pinned to the central evaluation’s mode.

`warm` Seeding of the n1qn1 inner-optimization Hessian: ‘"calc"‘ (default) warm-starts each inner problem with the eta Hessian calculated at the starting eta and the current theta; since theta moves between outer evaluations it is always recalculated, never reused from an earlier round. ‘"save"‘ restarts from the curvature n1qn1 built during the subject’s previous inner solve. ‘"none"‘ lets n1qn1 initialize its own diagonal Hessian. Ignored by ‘innerOpt = "trust"‘, which always supplies its own exact Hessian.

`nAGQ` Number of Gauss-Hermite adaptive quadrature points. ‘0‘ disables AGQ; ‘1‘ is equivalent to Laplace. Cost grows quickly with ETAs: once the EBE is found, expect ‘nAGQ^neta‘ (even ‘nAGQ‘) or ‘(nAGQ^neta)-1‘ (odd ‘nAGQ‘) additional evaluations per subject.

`agqLow` The lower bound for adaptive quadrature log-likelihood. By default this is -Inf; in the original nlmixr’s gnlmm it was -700.

`agqHi` The upper bound for adaptive quadrature log-likelihood. By default this is Inf; in the original nlmixr’s gnlmm was 400.

`sensMethod` Method used to compute the ODE parameter sensitivities. ‘"forward"‘ uses the classic variational (forward) sensitivity ODEs; ‘"default"‘ is the same thing.

`linCmtSensCarry`
‘"auto"‘ (default) substitutes the exact sensitivity-carry gradient for a ‘linCmt()‘ parameter driven by both an eta and a time-varying covariate (needs an rxode2 with the carry sentinels; silently keeps the standard gradient otherwise); ‘"none"‘ always keeps the standard gradient.

`zeroTheta` Positive magnitude (default ‘0.001‘) used to nudge a population parameter (‘theta‘) whose initial estimate is exactly ‘0‘ off zero before estimation. FOCEi scales a linear parameter by its native magnitude ‘|init|‘, which is ‘0‘ (no scale) for a zero initial estimate, so the parameter is moved to ‘+zeroTheta‘ when it is within the parameter’s bounds, otherwise ‘-zeroTheta‘; if neither is within the bounds an error is raised. Fixed parameters (including those fixed at ‘0‘) are left untouched.

`boundedTransform`
When ‘TRUE‘ (default), bounded parameters are transformed for unbounded optimization methods and back-transformed for final estimates. ‘FALSE‘ optimizes on the original scale with bounds passed to the optimizer. ‘NA‘ transforms for optimization but skips the final back-transform.

Details

Custom outer optimizers receive `control$hessian(par, relStep=1e-3)`. It settles the requested point and assembles the reported objective's Hessian in the existing C++ sensitivity pool. Third-order terms are obtained by differencing second-order sensitivities in ETA directions. The result uses optimizer coordinates and objective scaling and retains negative curvature. This requires `fast=TRUE`. M2/M3/M4 censoring uses the analytical-SE `censOption="gauss"` convention. Censored `"laplace"` curvature, priors, clipped AGQ and estimated transformations are unsupported. Unsuccessful inner solves and an active variance floor also make curvature unavailable.

Uses R's L-BFGS-B (`optim`) for the outer problem and BFGS `n1qn1` (restoring the prior individual Hessian) for the inner problem, which is left unscaled since eta estimates start near zero. The covariance step is performed on the unscaled problem, so its condition number may differ from the scaled problem's.

Value

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

Author(s)

Matthew L. Fidler

References

Gill, P.E., Murray, W., Saunders, M.A., & Wright, M.H. (1983). Computing Forward-Difference Intervals for Numerical Optimization. Siam Journal on Scientific and Statistical Computing, 4, 310-321.

Shi, H.M., Xie, Y., Xuan, M.Q., & Nocedal, J. (2021). Adaptive Finite-Difference Interval Estimation for Noisy Derivative-Free Optimization.

See Also

`optim`

`n1qn1`

`rxSolve`

Other Estimation control: `nlmixr2NlmeControl()`, `saemControl()`

`foceiLikIterPrintStart`

Iteration printing / parameter history over a loaded likelihood

---

Description

The standard nlmixr2est iteration table (the shared ‘scale.h‘ machinery behind every estimation method’s printout) exposed for an EXTERNAL sampler driving [foceiLikLoad()]’s conditional likelihood – e.g. nlmixr2stan’s ‘est="stan"‘. ‘foceiLikIterPrintStart()‘ prints the header and arms the row entry (entry 8 of the FOCEi C API table, ‘nlmixr2FoceiIterPrintRow‘), which the sampler’s gradient evaluations call from C; every ‘every‘-th call prints a row AND records it, so the returned history holds exactly the printed rows. ‘foceiLikIterPrintEnd()‘ prints the closing line and returns the history as a ‘parHistData‘-style data frame.

Usage

```
foceiLikIterPrintStart(
  every,
  initPar,
  names,
  iterPrintControl = NULL,
  xform = NULL
)

foceiLikIterPrintEnd()
```

Arguments

every	print/record cadence in row-entry calls (0 arms nothing; the row entry then records nothing and returns immediately)
initPar	display vector at the starting point (defines the width)
names	column names, same length as ‘initPar‘
iterPrintControl	optional list applied via the shared ‘scaleApplyIterPrintControl‘ (e.g. ‘list(useColor=FALSE, printNcol=6L)‘)
xform	optional back-transform list (the ‘.iterPrintXParFromUi‘ shape) driving the ‘X‘ row

Details

The display vector is CALLER-defined, not the internal focei parameter vector: pass natural-scale thetas plus (optionally) the sampler’s current actual omega entries, named ‘om.<eta>‘ for variances and ‘cov.<eta1>.<eta2>‘ for covariances. (The internal vector’s omega tail is ‘chol(Omega^-1)‘ in the ‘diagXform‘ parameterization, which a Bayesian sampler neither uses nor understands, so it is NOT printed.)

Value

‘foceiLikIterPrintStart()‘: invisibly ‘NULL‘; ‘foceiLikIterPrintEnd()‘: the recorded history data frame (or ‘NULL‘ when printing was never started)

Author(s)

Matthew L. Fidler

`foceiLikLoad` *Load a general FOCE-family likelihood into memory*

Description

Compiles the inner (FOCEi sensitivity) model from an rxode2 UI model, preprocesses the data, and sets up the FOCEi inner problem in memory so that individual log-likelihoods can be evaluated repeatedly (in parallel per subject) at supplied etas without recompiling – the setup used internally by ‘est="emvi"‘/‘est="fbvi"‘, ‘est="vae"‘ and the f-SAEM fast kernel, exposed here for MCMC/SAMBA-style callers (issue #414).

Usage

```
foceiLikLoad(
  object,
  data,
  likelihood = c("focei", "focep", "foce"),
  rxControl = rxode2::rxControl(),
  scale = c("focei", "natural"),
  thetaSens = FALSE,
  combSens = FALSE,
  iovXform = c("sd", "var", "logsd", "logvar"),
  est = "focei",
  ...
)
```

Arguments

`object`	An ‘rxode2‘/‘nlmixr2‘ UI model (a model function or its compiled UI).
`data`	The estimation data (a data frame with the usual nlmixr2 columns).
`likelihood`	The individual likelihood type: ‘"focei"‘ (FOCE with interaction), ‘"focep"‘ (FOCE+, interaction off with the residual variance at the conditional eta) or ‘"foce"‘ (NONMEM-style FOCE, residual variance frozen at eta=0).
`rxControl`	An [rxode2::rxControl()] object for the ODE solving options.
`scale`	The parameter scale [foceiLikRun()]’s ‘theta‘ is on: ‘"focei"‘ (default) is the FOCEi estimation scale (the historical behavior); ‘"natural"‘ pins the scaling to the identity (‘scaleType="mult"‘, ‘scaleTo=0‘), so the population thetas in ‘theta‘ (and in the handle’s ‘initPar‘) are the natural-scale values directly comparable with ‘ui$iniDf$est‘ – no unscaling step for external callers (#939). With either choice the trailing omega-block entries of the parameter vector remain in the internal ‘diagXform‘ parameterization of ‘chol(Omega^-1)‘; ‘"natural"‘ leaves them unscaled but does not change that parameterization.

`thetaSens` When ‘TRUE‘, also build and wire the theta-sensitivity model (‘d(f)/d(theta)‘, ‘d(V)/d(theta)‘ forward sensitivities for the estimated non-mu structural and residual-error thetas), the model the imp/advi engines use for their outer population gradient. Off by default: building it costs compile and solve time a value-only caller should not pay (#939). The handle’s ‘thetaSensIdx‘ reports which ‘ntheta‘ indices carry sensitivities (‘integer(0)‘ when none do, e.g. when every theta is mu-referenced).

`combSens` When ‘TRUE‘, use the combined eta+theta sensitivity build (#958): the INNER model itself carries the theta-sensitivity columns (appended after the FOCEi block), so one ODE integration per subject serves the value, ‘d/d(eta)‘ AND ‘d/d(theta)‘ – the fused batch entry ‘nlmixr2FoceiCondBatchThetaGrad‘ reads all three from a single solve (~2x cheaper per evaluation than the two-model path). Implies a theta-sensitivity request; the separate theta-sensitivity model is neither built nor compiled.

`iovXform` Transformation carrying the IOV magnitude theta (the ‘.uiApplyIov‘ hook): ‘"sd"‘ (default), ‘"var"‘, ‘"logsd"‘, ‘"logvar"‘. The presence of this argument also marks the loaded nlmixr2est as carrying the exact IOV magnitude sensitivity (#952)

`est` Estimation-method name the standard pre-process hooks run against (default ‘"focei"‘). The hooks read the named method’s capability attributes (for example whether a bounded-transform reparameterization is wanted), so a caller preparing the problem for a different consumer can name it here and get that consumer’s preprocessing guarantees instead of focei’s (#939). The inner engine itself is unaffected.

`...` Additional solving/model options passed to ‘.foceiLikControl‘ (e.g. ‘optExpression‘, ‘addProp‘, ‘eventSens‘).

Details

Only one likelihood system may be loaded at a time; loading errors if one is already loaded. Use [foceiLikRun()] to evaluate and [foceiLikUnload()] to free.

Value

Invisibly, a handle list with the loaded system’s dimensions: ‘initPar‘ (the parameter vector at the model’s initial estimates on the requested ‘scale‘, a ready ‘theta‘ for [foceiLikRun()]), ‘npars‘, ‘ntheta‘, ‘neta‘, ‘nid‘, ‘thetaNames‘, ‘etaNames‘, ‘idLvl‘, ‘likelihood‘, ‘scale‘, ‘thetaSens‘ (theta sensitivities wired, by either build), ‘combSens‘ (the #958 combined build) and ‘thetaSensIdx‘.

Author(s)

Matthew L. Fidler

See Also

[foceiLikRun()], [foceiLikUnload()]

Examples

```
one.cmt <- function() {
  ini({
    tka <- 0.45
    tcl <- 1
    tv <- 3.45
    add.sd <- 0.7
    eta.ka ~ 0.6
    eta.cl ~ 0.3
    eta.v ~ 0.1
  })
  model({
    ka <- exp(tka + eta.ka)
    cl <- exp(tcl + eta.cl)
    v <- exp(tv + eta.v)
    linCmt() ~ add(add.sd)
  })
}

# Set the likelihood up in memory once; only one may be loaded at a time
h <- foceiLikLoad(one.cmt, theo_sd, "focei")

# The handle carries the dimensions and a ready starting parameter vector
h$nid
h$neta
h$initPar

# Individual joint log-likelihood at eta = 0, one value per subject
eta <- matrix(0, h$nid, h$neta)
foceiLikRun(h$initPar, eta)

# Free it when done (loading again before this errors)
foceiLikUnload()
```

`foceiLikRun` *Evaluate a loaded general FOCE-family likelihood at supplied etas*

Description

Writes the population parameter vector into the loaded system and returns the per-subject log-likelihood at the supplied etas, computed in parallel over subjects. Requires a system loaded by [foceiLikLoad()].

Usage

```
foceiLikRun(
```

```
  theta,
  eta,
  type = c("joint", "cond"),
  cores = rxode2::getRxThreads()
)
```

Arguments

theta	The parameter vector (length ‘handle$npars‘) on the scale the system was loaded with: population thetas followed by the estimated Omega elements. With ‘foceiLikLoad(scale="focei")‘ (the default) this is the FOCEi estimation scale; with ‘scale="natural"‘ the theta entries are the natural-scale values directly comparable with ‘ui$iniDf$est‘ (the omega entries stay in the internal ‘diagXform‘ parameterization either way). ‘handle$initPar‘ from [foceiLikLoad()] is a ready starting value.
eta	A ‘nid‘ by ‘neta‘ matrix of random effects (one row per subject, in the loaded system’s subject order).
type	‘"joint"‘ (default) returns the individual joint log density ‘log p(y_i, eta_i)‘; ‘"cond"‘ returns the conditional data log-likelihood ‘log p(y_i | eta_i)‘ alone. See Details.
cores	Number of threads for the parallel per-subject evaluation.

Details

Both types are evaluated at the etas you supply, so both use each subject’s individual predictions; neither is a population (eta = 0) quantity. They differ only by the random-effect prior term:

- ‘"cond"‘ is the conditional data log-likelihood ‘log p(y_i | eta_i)‘, the observation contribution alone. - ‘"joint"‘ is ‘log p(y_i, eta_i) = log p(y_i | eta_i) + log p(eta_i)‘, which adds the Gaussian random-effect prior ‘log p(eta_i) = -0.5 eta_i’ Omega^-1 eta_i + 0.5 log|Omega^-1| - neta/2 log(2 pi)‘.

So ‘"joint"‘ minus ‘"cond"‘ is exactly ‘log p(eta_i)‘. ‘"joint"‘ is the default because it is the usual target for MCMC/SAMBA-style samplers: as a function of ‘eta_i‘ it is the individual’s posterior kernel, and it is the quantity the FOCEi inner problem optimizes over the etas. Use ‘"cond"‘ when you supply the random-effect density yourself, or when you need the observation contribution separately.

The prior is built from the loaded system’s own ‘Omega^-1‘ and its log determinant – the same Omega the inner likelihood uses – so ‘"joint"‘ stays internally consistent with the engine rather than with the nominal ‘ini()‘ values (the two differ by a small amount through Omega’s internal ‘rxSymInv‘ representation).

For Gaussian endpoints the observation contribution follows nlmixr2’s internal residual-likelihood convention, ‘-0.5 err^2/r - 0.5 log(r)‘, which omits the additive ‘-0.5 log(2 pi)‘ per observation; general log-likelihood (‘ll()‘) endpoints contribute the user’s log density as written. The eta prior above is fully normalized. Both types are therefore proper log densities up to a fixed per-observation constant that does not depend on ‘theta‘ or ‘eta‘, so likelihood ratios, and any sampler that uses them, are unaffected.

Value

A named numeric vector (length ‘nid‘, named by subject id) of per-subject log-likelihoods.

Author(s)

Matthew L. Fidler

See Also

[foceiLikLoad()], [foceiLikUnload()]

Examples

```
one.cmt <- function() {
  ini({
    tka <- 0.45
    tcl <- 1
    tv <- 3.45
    add.sd <- 0.7
    eta.ka ~ 0.6
    eta.cl ~ 0.3
    eta.v ~ 0.1
  })
  model({
    ka <- exp(tka + eta.ka)
    cl <- exp(tcl + eta.cl)
    v <- exp(tv + eta.v)
    linCmt() ~ add(add.sd)
  })
}

h <- foceiLikLoad(one.cmt, theo_sd, "focei")

eta <- matrix(0, h$nid, h$neta)

# The individual joint log density log p(y_i, eta_i) (the default)
foceiLikRun(h$initPar, eta)

# The conditional data log-likelihood log p(y_i | eta_i) alone; the two
# differ by the Gaussian eta prior
foceiLikRun(h$initPar, eta, type = "cond")

# Non-zero etas
set.seed(42)
foceiLikRun(h$initPar, matrix(stats::rnorm(h$nid * h$neta, 0, 0.1), h$nid, h$neta))

# A new population parameter vector needs no reload
theta <- h$initPar
theta[1] <- theta[1] + 0.1
foceiLikRun(theta, eta)
```

```
foceiLikUnload()
```

`foceiLikUnload` *Unload the general FOCE-family likelihood from memory*

Description

Frees the FOCEi inner problem set up by [foceiLikLoad()]. A no-op (returns ‘FALSE‘) if nothing is loaded.

Usage

```
foceiLikUnload()
```

Value

Invisibly ‘TRUE‘ if a system was freed, ‘FALSE‘ if none was loaded.

Author(s)

Matthew L. Fidler

See Also

[foceiLikLoad()], [foceiLikRun()]

`focepControl` *Control options for the focep (FOCE+) estimation method*

Description

This is the first order conditional estimation without eta/residual interaction, but keeping the live conditional residual variance R (the ‘foce = "foce+"‘ option of [foceiControl()]). It is the ‘foce‘ method with ‘foce = "foce+"‘ forced; use ‘foce‘ (est = "foce") for the NONMEM-matching frozen-R behavior.

Usage

```
focepControl(sigdig = 3, ..., interaction = FALSE, foce = "foce+")
```

Arguments

sigdig	Optimization significant digits. One value drives, with a single consistent formula, the inner/outer optimizer convergence tolerance (10^-sigdig), the boundary check tolerance (5*10^(-sigdig+1)), and the ODE solver tolerances: the rtol exponent IS sigdig and atol sits three orders below, so rtol = 10^-sigdig, atol = 10^(-sigdig-3) for every solver (stiff, non-stiff or auto-switching). The sensitivity (atolSens/rtolSens) tolerances match the main solve (the outer gradient and covariance are built from them); the steady-state (ssAtol/ssRtol) tolerances run one order looser. Keying the optimizer to the same 10^-sigdig means it converges to exactly the precision the solve supports. At the default sigdig = 3 this is atol = 1e-6, rtol = 1e-3.
...	Parameters used in the default ‘foceiControl()‘
interaction	Interaction term for the model, in this case the default is ‘FALSE‘; it cannot be changed, use ‘focei‘ instead
foce	FOCE residual-variance mode; for ‘focepControl()‘ this is always ‘"foce+"‘ and cannot be changed – use ‘foceControl()‘ for ‘"nonmem"‘

Value

focepControl object

Author(s)

Matthew L. Fidler

Examples

```
focepControl()
```

---

foControl *Control options for the FO estimation method*

Description

This is related to the focei methods and uses most of their control options. Some are ignored, ‘posthoc‘ is an extra parameter

Usage

```
foControl(sigdig = 3, ..., posthoc = TRUE, interaction = NULL, fo = NULL)
```

Arguments

sigdig	Optimization significant digits. One value drives, with a single consistent formula, the inner/outer optimizer convergence tolerance (10^-sigdig), the boundary check tolerance (5*10^(-sigdig+1)), and the ODE solver tolerances: the rtol exponent IS sigdig and atol sits three orders below, so rtol = 10^-sigdig, atol = 10^(-sigdig-3) for every solver (stiff, non-stiff or auto-switching). The sensitivity (atolSens/rtolSens) tolerances match the main solve (the outer gradient and covariance are built from them); the steady-state (ssAtol/ssRtol) tolerances run one order looser. Keying the optimizer to the same 10^-sigdig means it converges to exactly the precision the solve supports. At the default sigdig = 3 this is atol = 1e-6, rtol = 1e-3.
...	Parameters used in the default 'foceiConrol()'
posthoc	Logical indicating if the estimation method should calculate 'foce' posthoc predicted parameters.
interaction	Interaction term for the model; ignored by fo
fo	Logical indicating if the estimation method is FO (first order), but this is controlled by the estimation method so this is ignored.

Value

foControl object

Author(s)

Matthew L. Fidler

Examples

```
foControl()
```

---

foiControl *Control options foir the FOI estimation method*

Description

This is related to the focei methods and uses most of their control options. Some are ignored, 'posthoc' is an extra parameter

Usage

```
foiControl(sigdig = 3, ..., posthoc = TRUE, interaction = NULL, fo = NULL)
```

Arguments

sigdig	Optimization significant digits. One value drives, with a single consistent formula, the inner/outer optimizer convergence tolerance (10^-sigdig), the boundary check tolerance (5*10^(-sigdig+1)), and the ODE solver tolerances: the rtol exponent IS sigdig and atol sits three orders below, so rtol = 10^-sigdig, atol = 10^(-sigdig-3) for every solver (stiff, non-stiff or auto-switching). The sensitivity (atolSens/rtolSens) tolerances match the main solve (the outer gradient and covariance are built from them); the steady-state (ssAtol/ssRtol) tolerances run one order looser. Keying the optimizer to the same 10^-sigdig means it converges to exactly the precision the solve supports. At the default sigdig = 3 this is atol = 1e-6, rtol = 1e-3.
...	Parameters used in the default ‘foceiConrol()‘
posthoc	Logical indicating if the estimation method should calculate ‘foce‘ posthoc predicted parameters.
interaction	Interaction term for the model; ignored by fo
fo	Logical indicating if the estimation method is FO (first order), but this is controlled by the estimation method so this is ignored.

Value

foiControl object

Author(s)

Matthew L. Fidler

Examples

```
foiControl()
```

---

formatMinWidth *Format numeric values to minimize the printing width*

Description

Special values (‘NaN‘, ‘Inf‘, ‘-Inf‘, ‘0‘) are shown as-is; ‘NA‘ uses ‘naValue‘.

Usage

```
formatMinWidth(x, digits = 3, naValue = "NA")
```

Arguments

x	The numeric vector to convert
digits	The number of significant digits to show
naValue	The value to return if ‘is.na(x)‘

Value

A character vector converting the numbers with minimum width

`getValidNlmixrCtl.agq` *Get valid nlmixr control object*

Description

Get valid nlmixr control object

Usage

```
## S3 method for class 'agq'
getValidNlmixrCtl(control)

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

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

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

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

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

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

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

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

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

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

## S3 method for class 'mfocepf'
```

```
getValidNlmixrCtl(control)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

## S3 method for class 'imp'
```

```
getValidNlmixrCtl(control)

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

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

## S3 method for class 'lbfgsb3c'
getValidNlmixrCtl(control)

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

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

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

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

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

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

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

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

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

## S3 method for class 'n1qn1'
getValidNlmixrCtl(control)

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

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

## S3 method for class 'nlminb'
```

```
getValidNlmixrCtl(control)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

## S3 method for class 'qrpem'
```

```
getValidNlmixrCtl(control)

getValidNlmixrControl(control, est)

getValidNlmixrCtl(control)

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

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

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

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

## Default S3 method:
getValidNlmixrCtl(control)

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

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

## S3 method for class 'vae'
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.

`iagqControl` *Control options for the iagq estimation method*

Description

Mu-referenced-FOCEI-family reweighted-regression (‘"irls"‘) variant of adaptive Gauss-Hermite quadrature; see ‘foceiControl(muModel=)‘.

Usage

```
iagqControl(
  sigdig = 3,
  nAGQ = 2,
  ...,
  interaction = TRUE,
  agqLow = -Inf,
  agqHi = Inf,
  muModel = c("irls", "lin", "none")
)
```

Arguments

`sigdig`	Optimization significant digits. One value drives, with a single consistent formula, the inner/outer optimizer convergence tolerance (`10^-sigdig`), the boundary check tolerance (`5*10^(-sigdig+1)`), and the ODE solver tolerances: the `rtol` exponent IS `sigdig` and `atol` sits three orders below, so `rtol = 10^-sigdig`, `atol = 10^(-sigdig-3)` for every solver (stiff, non-stiff or auto-switching). The sensitivity (`atolSens`/`rtolSens`) tolerances match the main solve (the outer gradient and covariance are built from them); the steady-state (`ssAtol`/`ssRtol`) tolerances run one order looser. Keying the optimizer to the same `10^-sigdig` means it converges to exactly the precision the solve supports. At the default `sigdig = 3` this is `atol = 1e-6`, `rtol = 1e-3`.
`nAGQ`	Number of Gauss-Hermite adaptive quadrature points. ‘0‘ disables AGQ; ‘1‘ is equivalent to Laplace. Cost grows quickly with ETAs: once the EBE is found, expect ‘nAGQ^neta‘ (even ‘nAGQ‘) or ‘(nAGQ^neta)-1‘ (odd ‘nAGQ‘) additional evaluations per subject.
`...`	Parameters used in the default ‘foceiControl()‘

interaction	boolean, Interaction term for the model, in this case the default is 'TRUE'; For adaptive quadrature, with normal distribution the Hessian is calculated with the foce(i) approximation
agqLow	The lower bound for adaptive quadrature log-likelihood. By default this is -Inf; in the original nlmixr's gnlmm it was -700.
agqHi	The upper bound for adaptive quadrature log-likelihood. By default this is Inf; in the original nlmixr's gnlmm was 400.
muModel	Selects the regression variant; for 'iagqControl()' this is always '"irls"' and cannot be changed – use 'magqControl()' for the closed-form OLS variant.

Value

iagqControl object

Author(s)

Matthew L. Fidler

Examples

```
iagqControl()
```

`ifoceControl` *Control options for the ifoce estimation method*

Description

Mu-referenced-FOCEI-family reweighted-regression ('"irls"') variant of FOCE (no interaction); see 'foceiControl(muModel=)'.

Usage

```
ifoceControl(
  sigdig = 3,
  ...,
  interaction = FALSE,
  muModel = c("irls", "lin", "none")
)
```

Arguments

sigdig	Optimization significant digits. One value drives, with a single consistent formula, the inner/outer optimizer convergence tolerance (`10^-sigdig`), the boundary check tolerance (`5*10^(-sigdig+1)`), and the ODE solver tolerances: the `rtol` exponent IS `sigdig` and `atol` sits three orders below, so `rtol = 10^-sigdig`, `atol = 10^(-sigdig-3)` for every solver (stiff, non-stiff or auto-switching). The sensitivity (`atolSens`/`rtolSens`) tolerances match the main solve (the outer

	gradient and covariance are built from them); the steady-state (`ssAtol`/`ssRtol`) tolerances run one order looser. Keying the optimizer to the same `10^-sigdig` means it converges to exactly the precision the solve supports. At the default `sigdig = 3` this is `atol = 1e-6`, `rtol = 1e-3`.
`...`	Parameters used in the default 'foceiControl()'
`interaction`	Interaction term for the model, in this case the default is 'FALSE'; it cannot be changed, use 'ifocei' instead
`muModel`	Selects the regression variant; for 'ifoceControl()' this is always '"irls"' and cannot be changed – use 'mfoceControl()' for the closed-form OLS variant.

Value

ifoceControl object

Difference from 'focei'

The 'mfocei'/'ifocei' (and related) methods apply the mu2+ covariate hooks, which expand algebraic mu-referenced covariate expressions (e.g. 'cl.wt*log(WT/70)') into estimable mu-referenced parameters and split covariates into non-time-varying (absorbed into the phi term) and time-varying (kept as 'beta' regressors). Calling 'focei' directly does NOT apply these hooks, so these methods can estimate more mu-referenced models than plain 'focei' – there is a genuine difference between calling e.g. 'est="mfocei"' and 'est="focei"'.

All mu-referenced population thetas – with or without covariates – are profiled out of the outer optimizer by the in-C++ regression (intercept-only for covariate-free pairs), so outer gradients are only calculated for the non-mu-referenced parameters (residual errors, omegas, non-mu thetas). Bounded mu-referenced parameters are regression-updated with the update clamped to the bounds (a clamp is reported once as a fit note); user-fixed ('fix()') mu thetas stay out of the regression.

Author(s)

Matthew L. Fidler

Examples

```
ifoceControl()
```

`ifoceiControl` *Control options for the ifocei estimation method*

Description

Mu-referenced-FOCEI-family reweighted-regression ('"irls"') variant of FOCEI; see 'foceiControl(muModel=)'.

Usage

```
ifoceiControl(sigdig = 3, ..., muModel = c("irls", "lin", "none"))
```

Arguments

sigdig	Optimization significant digits. One value drives, with a single consistent formula, the inner/outer optimizer convergence tolerance (10^-sigdig), the boundary check tolerance (5*10^(-sigdig+1)), and the ODE solver tolerances: the rtol exponent IS sigdig and atol sits three orders below, so rtol = 10^-sigdig, atol = 10^(-sigdig-3) for every solver (stiff, non-stiff or auto-switching). The sensitivity (atolSens/rtolSens) tolerances match the main solve (the outer gradient and covariance are built from them); the steady-state (ssAtol/ssRtol) tolerances run one order looser. Keying the optimizer to the same 10^-sigdig means it converges to exactly the precision the solve supports. At the default sigdig = 3 this is atol = 1e-6, rtol = 1e-3.
...	Parameters used in the default ‘foceiControl()‘
muModel	Selects the regression variant; for ‘ifoceiControl()‘ this is always ‘"irls"‘ and cannot be changed – use ‘mfoceiControl()‘ for the closed-form OLS variant.

Value

ifoceiControl object

Difference from ‘focei‘

The ‘mfocei‘/‘ifocei‘ (and related) methods apply the mu2+ covariate hooks, which expand algebraic mu-referenced covariate expressions (e.g. ‘cl.wt*log(WT/70)‘) into estimable mu-referenced parameters and split covariates into non-time-varying (absorbed into the phi term) and time-varying (kept as ‘beta‘ regressors). Calling ‘focei‘ directly does NOT apply these hooks, so these methods can estimate more mu-referenced models than plain ‘focei‘ – there is a genuine difference between calling e.g. ‘est="mfocei"‘ and ‘est="focei"‘.

All mu-referenced population thetas – with or without covariates – are profiled out of the outer optimizer by the in-C++ regression (intercept-only for covariate-free pairs), so outer gradients are only calculated for the non-mu-referenced parameters (residual errors, omegas, non-mu thetas). Bounded mu-referenced parameters are regression-updated with the update clamped to the bounds (a clamp is reported once as a fit note); user-fixed (‘fix()‘) mu thetas stay out of the regression.

Author(s)

Matthew L. Fidler

Examples

```
ifoceiControl()
```

`ifocepControl` *Control options for the ifocep (IRLS FOCE+) estimation method*

Description

Mu-referenced-FOCEI-family reweighted-regression (‘"irls"‘) variant of FOCE+ (no interaction, live conditional residual variance R); it is ‘ifoce‘ with ‘foce = "foce+"‘ forced (see ‘foceiControl(foce=)‘).

Usage

```
ifocepControl(
  sigdig = 3,
  ...,
  interaction = FALSE,
  muModel = c("irls", "lin", "none"),
  foce = "foce+"
)
```

Arguments

`sigdig`	Optimization significant digits. One value drives, with a single consistent formula, the inner/outer optimizer convergence tolerance (`10^-sigdig`), the boundary check tolerance (`5*10^(-sigdig+1)`), and the ODE solver tolerances: the `rtol` exponent IS `sigdig` and `atol` sits three orders below, so `rtol = 10^-sigdig`, `atol = 10^(-sigdig-3)` for every solver (stiff, non-stiff or auto-switching). The sensitivity (`atolSens`/`rtolSens`) tolerances match the main solve (the outer gradient and covariance are built from them); the steady-state (`ssAtol`/`ssRtol`) tolerances run one order looser. Keying the optimizer to the same `10^-sigdig` means it converges to exactly the precision the solve supports. At the default `sigdig = 3` this is `atol = 1e-6`, `rtol = 1e-3`.
`...`	Parameters used in the default ‘foceiControl()‘
`interaction`	Interaction term for the model, in this case the default is ‘FALSE‘; it cannot be changed, use ‘ifocei‘ instead
`muModel`	Selects the regression variant; for ‘ifocepControl()‘ this is always ‘"irls"‘ and cannot be changed – use ‘mfocepControl()‘ for the closed-form OLS variant.
`foce`	FOCE residual-variance mode; for ‘ifocepControl()‘ this is always ‘"foce+"‘ and cannot be changed – use ‘ifoceControl()‘ for ‘"nonmem"‘

Value

ifocepControl object

Difference from 'focei'

The 'mfocei'/'ifocei' (and related) methods apply the mu2+ covariate hooks, which expand algebraic mu-referenced covariate expressions (e.g. 'cl.wt*log(WT/70)') into estimable mu-referenced parameters and split covariates into non-time-varying (absorbed into the phi term) and time-varying (kept as 'beta' regressors). Calling 'focei' directly does NOT apply these hooks, so these methods can estimate more mu-referenced models than plain 'focei' – there is a genuine difference between calling e.g. 'est="mfocei"' and 'est="focei"'.

All mu-referenced population thetas – with or without covariates – are profiled out of the outer optimizer by the in-C++ regression (intercept-only for covariate-free pairs), so outer gradients are only calculated for the non-mu-referenced parameters (residual errors, omegas, non-mu thetas). Bounded mu-referenced parameters are regression-updated with the update clamped to the bounds (a clamp is reported once as a fit note); user-fixed ('fix()') mu thetas stay out of the regression.

Author(s)

Matthew L. Fidler

Examples

```
ifocepControl()
```

---

ilaplaceControl *Control options for the ilaplace estimation method*

Description

Mu-referenced-FOCEI-family reweighted-regression ('"irls"') variant of the Laplace method ('nAGQ=1'); see 'foceiControl(muModel=)'.

Usage

```
ilaplaceControl(sigdig = 3, ..., nAGQ = 1, muModel = c("irls", "lin", "none"))
```

Arguments

`sigdig`	Optimization significant digits. One value drives, with a single consistent formula, the inner/outer optimizer convergence tolerance (`10^-sigdig`), the boundary check tolerance (`5*10^(-sigdig+1)`), and the ODE solver tolerances: the `rtol` exponent IS `sigdig` and `atol` sits three orders below, so `rtol = 10^-sigdig`, `atol = 10^(-sigdig-3)` for every solver (stiff, non-stiff or auto-switching). The sensitivity (`atolSens`/`rtolSens`) tolerances match the main solve (the outer gradient and covariance are built from them); the steady-state (`ssAtol`/`ssRtol`) tolerances run one order looser. Keying the optimizer to the same `10^-sigdig` means it converges to exactly the precision the solve supports. At the default `sigdig = 3` this is `atol = 1e-6`, `rtol = 1e-3`.
`...`	Parameters used in the default 'foceiControl()'

nAGQ	Number of Gauss-Hermite adaptive quadrature points. ‘0‘ disables AGQ; ‘1‘ is equivalent to Laplace. Cost grows quickly with ETAs: once the EBE is found, expect ‘nAGQ^neta‘ (even ‘nAGQ‘) or ‘(nAGQ^neta)-1‘ (odd ‘nAGQ‘) additional evaluations per subject.
muModel	Selects the regression variant; for ‘ilaplaceControl()‘ this is always ‘"irls"‘ and cannot be changed – use ‘mlaplaceControl()‘ for the closed-form OLS variant.

Value

ilaplaceControl object

Author(s)

Matthew L. Fidler

Examples

```
ilaplaceControl()
```

---

impControl *Control for the imp (importance-sampling EM without MAP search) method*

Description

A convenience wrapper around [impmapControl()] with ‘mapIter = 0‘, i.e. the importance-sampling proposal is centered at the running conditional mean rather than re-optimized to the MAP mode each iteration (NONMEM METHOD=IMP). See [impmapControl()] for the full parameter list.

Usage

```
impControl(...)
```

Arguments

...	Parameters passed to [impmapControl()].

Value

An ‘impmapControl‘ object with ‘mapIter = 0‘.

Author(s)

Matthew L. Fidler

Examples

```
impControl()
```

`impCovControl` *Options for the importance-sampling covariance in setCov()*

Description

Used by `setCov(fit, "imp")`, which runs frozen importance-sampling EM iterations at the fit's estimates: E-steps only, so the parameters are never updated and the covariance is the one at those estimates.

Usage

```
impCovControl(nIter = 1L, isample = 300L, impSeed = 42L)
```

Arguments

`nIter`	frozen EM iterations: E-steps at the fit's estimates, with the proposal adapting between them (`0` and `1` are both a single E-step)
`isample`	Number of importance samples drawn per subject per iteration (NONMEM ISAMPLE). Either a single count used for every subject, or a vector of length 'nsub' giving a count **per subject**. Per-subject counts are the NM7 Technical Guide's own remedy for poor coverage (its derivation is Gaussian throughout and never mentions a t proposal): a subject whose weights are badly behaved can be given more samples without charging every other subject for them. Note this treats the symptom rather than the cause – more draws from a proposal whose tails are too light still gives weights with infinite variance, which 'fitenvimpPsisK' will show. See 'df' for the shape-based remedy.
`impSeed`	Base seed for the per-subject thread-safe (threefry) RNG streams; results are reproducible and independent of the thread count.

Value

`impCovControl` object

Author(s)

Matt Fidler

See Also

`setCov()`, `impmapControl()`

Examples

```
impCovControl(isample = 1000)
```

`impmapControl` *Control options for the impmap (importance-sampling EM) estimation method*

Description

A NONMEM-style Monte Carlo importance-sampling EM built on the mu-referenced FOCEI MAP. The proposal density for each subject is centered at the MAP mode (‘muModel="lin"‘); mu-referenced population parameters are updated by the EM gradient, while non-mu parameters (structural and residual error) are updated by a symbolic-sensitivity Newton step – the importance-sampling-weighted score and Gauss-Newton information built from the analytic ‘d(f)/d(theta)‘ and ‘d(V)/d(theta)‘ (exact censored partials for BLQ/M2/M3/M4 points).

Usage

```
impmapControl(
  sigdig = 3,
  ...,
  isample = 300L,
  nIter = 100L,
  mapIter = 1L,
  nBurn = 0L,
  burnFreezeOmega = FALSE,
  gamma = 1,
  gammaMethod = c("auto", "global", "individual"),
  gammaRule = c("target", "floor"),
  df = 0,
  proposal = c("auto", "normal", "t", "laplace", "mixture"),
  propMixScale = c(1, 9),
  propMixWeight = c(0.9, 0.1),
  auto = TRUE,
  autoNonmemSparse = FALSE,
  autoDfPatience = 2L,
  iscaleMin = 0.1,
  iscaleMax = 10,
  iaccept = 0.4,
  ctol = NULL,
  nConvWindow = 10L,
  impSeed = 42L,
  covMethod = c("imp", "analytic", "r,s", "r", "s", "sa", ""),
  qr = FALSE,
  qrShift = TRUE,
  qrRefresh = TRUE,
  qrScramble = c("none", "owen", "lms"),
  sir = FALSE,
  sirSample = NULL,
  muModel = c("lin", "none"),
```

```
  combSens = TRUE
)
```

Arguments

sigdig	Optimization significant digits. One value drives, with a single consistent formula, the inner/outer optimizer convergence tolerance (`10^-sigdig`), the boundary check tolerance (`5*10^(-sigdig+1)`), and the ODE solver tolerances: the `rtol` exponent IS `sigdig` and `atol` sits three orders below, so `rtol = 10^-sigdig`, `atol = 10^(-sigdig-3)` for every solver (stiff, non-stiff or auto-switching). The sensitivity (`atolSens`/`rtolSens`) tolerances match the main solve (the outer gradient and covariance are built from them); the steady-state (`ssAtol`/`ssRtol`) tolerances run one order looser. Keying the optimizer to the same `10^-sigdig` means it converges to exactly the precision the solve supports. At the default `sigdig = 3` this is `atol = 1e-6`, `rtol = 1e-3`.
...	Parameters used in the default ‘foceiControl()‘
isample	Number of importance samples drawn per subject per iteration (NONMEM ISAMPLE). Either a single count used for every subject, or a vector of length ‘nsub‘ giving a count **per subject**. Per-subject counts are the NM7 Technical Guide’s own remedy for poor coverage (its derivation is Gaussian throughout and never mentions a t proposal): a subject whose weights are badly behaved can be given more samples without charging every other subject for them. Note this treats the symptom rather than the cause – more draws from a proposal whose tails are too light still gives weights with infinite variance, which ‘fitenvimpPsisK‘ will show. See ‘df‘ for the shape-based remedy.
nIter	Maximum number of importance-sampling EM iterations. ‘0‘ evaluates at the supplied parameters: one E-step with no M-step (like NONMEM ‘EONLY=1‘); ‘nBurn‘ is ignored.
mapIter	MAP-assist period, in EM iterations. ‘1‘ (default) re-centers the proposal at each subject’s MAP mode every iteration; ‘k > 1‘ re-centers every ‘k‘th iteration; ‘0‘ never re-centers after the startup MAP pass. The MAP search is the mu-referenced FOCEI inner problem and is the dominant per-iteration cost of ‘est="impmap"‘, so raising ‘mapIter‘ trades proposal accuracy for speed. It is worth raising once the population parameters are moving slowly enough that the mode barely shifts between iterations, and not before – a proposal centered away from the mode costs effective sample size, which ‘fitenvimpNeffFrac‘ will show. A skipped iteration does not freeze the proposal center: the M-step reseeds every subject’s eta with its conditional mean, so the proposal still moves each iteration – it is the MAP re-optimization, not the center, that is skipped. The covariance is still built from the inner Hessian there, which is what distinguishes this from ‘est="imp"‘; ‘est="imp"‘ uses the running conditional variance instead and is not the ‘mapIter = 0‘ case. ‘mapIter‘ cannot change an ‘est="imp"‘ fit, which never re-centers at all. **Measured trade-off.** ‘mapIter = 3‘ against the default, same harness: theta RMSE 0.00102 vs 0.00157 (1 ETA), 0.00130 vs 0.00144 (3 ETAs), 0.00173 vs

0.00173 (general ‘ll()‘), with ‘Omega‘ RMSE, effective sample size and k-hat essentially unmoved. On those fixtures skipping two MAP searches in three is close to free, which is the point of the knob.

At 8 ETAs it becomes a genuine trade rather than a free saving, and in a direction worth knowing: theta RMSE 0.0205 against 0.0146 (worse), but Pareto k-hat +0.88 against +0.97 and the number of subjects above 0.7 down from 2.1 to **0.25** – by a wide margin the healthiest sampler of any setting measured on that fixture. Re-optimizing the mode every iteration is not automatically the safer choice at high eta dimension.

‘1‘ stays the default: it is the previous behaviour, and it never runs a stale proposal. Raise it when the per-iteration MAP cost matters (it is the inner FOCEI problem, so it scales with how hard that is), and watch ‘fitenvimpNeffFrac‘ – a proposal centred away from the mode costs effective sample size on a model whose mode moves quickly.

`nBurn` Number of burn-in EM iterations run before the ‘nIter‘ budget, to let the proposal-scale (‘gamma‘) and ‘auto‘ controllers settle before the estimates they influence are judged. ‘0‘ (default) is no burn-in.

These are EXTRA iterations, not carved out of ‘nIter‘, so raising ‘nBurn‘ never silently shortens the fit; a fit runs at most ‘nBurn + nIter‘ iterations. Everything runs normally during them – the E-step, both controllers, and the theta M-step – except that convergence cannot fire, and ‘Omega‘ is held when ‘burnFreezeOmega = TRUE‘.

Burn-in iterations are the first ‘nBurn‘ rows of ‘$parHist‘ and of ‘fit$env$impObjTrace‘; ‘fit$env$impNburn‘ reports how many there were.

Measured trade-off. ‘nBurn = 5, burnFreezeOmega = TRUE‘ against the default, same harness as ‘proposal‘: theta RMSE 0.00140 vs 0.00157 (1 ETA), 0.00185 vs 0.00144 (3 ETAs), 0.00104 vs 0.00173 (general ‘ll()‘); ‘Omega‘ RMSE worse on all three (0.00404 / 0.00261 / 0.00249 against 0.00344 / 0.00184 / 0.00170), for 2-5 extra iterations.

It can be much worse than that, so do not switch it on speculatively. On an 8-ETA warfarin PK/PD fit the same setting gave theta RMSE 0.149 against 0.0146 for the default – ten times worse – with k-hat and the failing-subject count both degraded (+1.30 / 3.1 against +0.97 / 2.1). Holding ‘Omega‘ while the thetas move is a coherent thing to want, but on a high-dimensional model the thetas can travel a long way from an ‘Omega‘ that is not allowed to follow, and the fit does not recover the difference in the iterations that remain.

Off by default, and worth trying only when a fit’s ‘Omega‘ visibly overshoots in the first rows of ‘$parHist‘ and then has to come back – which is the specific failure it addresses.

`burnFreezeOmega`

When ‘TRUE‘, hold ‘Omega‘ at its starting value for the ‘nBurn‘ burn-in iterations while the structural and residual-error thetas update normally. Ignored when ‘nBurn = 0‘.

The case for it: the first EM iterations run under a proposal whose scale the controller has not yet adapted, and ‘Omega‘’s update is the one that absorbs that – it is built from the conditional variances (‘Omega = mean(eta eta’ + condVar)‘), which an over- or under-dispersed proposal estimates badly. Freezing it lets

the thetas move the model toward the data while the proposal settles, instead of chasing an ‘Omega‘ excursion the fit then has to undo.

Convergence is not tested until the whole trailing ‘nConvWindow‘ lies past the burn-in, so a frozen ‘Omega‘ cannot be mistaken for a settled one.

`gamma` Initial proposal-variance inflation factor (NONMEM ISCALE); the proposal covariance is ‘gamma‘ times the inverse of the inner information matrix at the mode.

`gammaMethod` How the proposal scale ‘gamma‘ is adapted during the EM.

‘"auto"‘ (default) picks per model: ‘"individual"‘ when the model is not transformably normal – a general log-likelihood (‘ll()‘) endpoint, or a count/categorical/time-to-event distribution – and ‘"global"‘ otherwise. That is the split the hypothesis actually rests on: ‘gamma = 1‘ is already the efficient proposal when the individual posterior is close to Gaussian, so a normal model gains nothing from per-subject adaptation and would only pay for it in effective sample size, while a general-likelihood model is exactly where the posteriors go non-Gaussian and per-subject coverage starts to vary. The test is ‘all(ui$predDf$distribution == "norm")‘, the same line [rxode2::assertRxUiTransformNormal()] draws. The resolved value is reported in the fit’s ‘$runInfo‘.

‘"global"‘ keeps one scale shared by every subject, inflated (never relaxed) only when the *mean* Kish effective-sample fraction falls below ‘iaccept‘. It leaves ‘gamma‘ at its efficient starting value while coverage is healthy, which is the right behaviour when the individual posteriors are close to Gaussian.

‘"individual"‘ gives every subject its own ‘gamma_i‘ and adapts it two-sided toward a target on that subject’s ‘xi_i‘, clamped to ‘[iscaleMin, iscaleMax]‘. This follows NONMEM’s *objective* – the NM7 Technical Guide states that ‘gamma‘ is per-subject (eq. 1.90) and is "continually adjusted so that xi_i approximates IACCEPT" (note after eq. 1.76), bounded by ‘ISCALE_MIN‘/‘ISCALE_MAX‘ – but the guide publishes no update formula, so the functional form used here is nlmixr2’s own. Prefer this when the individual posteriors are heavy-tailed or the design is heterogeneous: a global scale is driven by the mean, so a handful of badly-covered subjects never trip it and their likelihood and ‘Omega‘ contributions end up carried by a few samples.

The two modes report *different* efficiency statistics – ‘"individual"‘ targets ‘xi‘ (the mean normalized importance weight, NONMEM’s ‘IACCEPT‘ quantity) while ‘"global"‘ targets the Kish effective-sample fraction. These are not comparable; the fit’s ‘$runInfo‘ says which is in force.

`gammaRule` How the SHARED (‘gammaMethod="global"‘) proposal scale is adapted. Ignored for ‘gammaMethod="individual"‘, which always follows NONMEM’s two-sided per-subject rule.

‘"floor"‘ treats ‘iaccept‘ as a one-sided FLOOR on the mean Kish effective-sample fraction: ‘gamma‘ stays at its efficient starting value while coverage is healthy and is inflated only when coverage drops below the floor. It never comes back down.

‘"target"‘ (default) follows the NM7 Technical Guide, which says ‘gamma‘ is "continually adjusted so that xi_i approximates IACCEPT" – adapted BOTH ways, on ‘xi‘ rather than the Kish fraction, using the same analytic inversion as

the per-subject controller (‘gamma * (xi/iaccept)^(2/neta)‘, capped 1.25x each way and clamped to ‘[iscaleMin, iscaleMax]‘).

The two rules settle at different operating points, so they are not interchangeable, and constants tuned against one do not carry over to the other.

Measured trade-off (theophylline, 6 seeds, ‘isample = 300‘, RMSE against an ‘isample = 6000‘ reference):

- 3 ETAs – ‘"floor"‘: theta RMSE 0.00185, ‘Omega‘ RMSE 0.00298, max Pareto k-hat 0.604, 0.33 subjects above 0.7, converged 100 ‘"target"‘: theta RMSE 0.00257, ‘Omega‘ RMSE 0.00378, max k-hat -0.491, 0 subjects above 0.7, converged 0
- 8 ETAs – ‘"floor"‘ never adapts at all (‘gamma‘ pinned at 1.0 with ‘xi‘ 1.35, i.e. a proposal far too narrow), because it watches the mean Kish fraction, which stays healthy while ‘xi‘ says the proposal is wrong. ‘"target"‘ moves ‘gamma‘ to 1.25 and puts ‘xi‘ on 0.41.

So ‘"target"‘ fixes the tail and is the only rule that reacts at high ETA dimension, but it pays for it: putting ‘xi‘ on ‘iaccept‘ deliberately widens the proposal (‘gamma‘ 1.79 on the 3-ETA fixture), which costs effective sample size (0.96 to 0.70) and adds Monte-Carlo noise to the objective – enough that the fit often does not meet ‘ctol‘ within ‘nIter‘.

‘"target"‘ is the default: tuned against tuned, it wins every column at 3 ETAs and is the only rule that adapts at all at 8 ETAs. Its cost is on the 1-ETA fixture, where it roughly doubles theta RMSE (0.00113 to 0.00224) and takes about twice as many iterations. That trade follows the same weighting ‘auto‘ uses: weights with infinite variance are a correctness problem whose error is unbounded, while the extra Monte-Carlo noise is bounded and measurable.

Choose ‘"floor"‘ for the previous behaviour – a proposal left at its efficient starting value while coverage is healthy. It is also what the tail-machinery tests pin, because ‘"target"‘ repairs the tail itself and leaves the ‘df‘ ladder nothing to fix.

The tuned constants travel with the rule. Selecting a rule also selects its tuned defaults (‘nConvWindow‘ 20 for ‘"target"‘, 10 for ‘"floor"‘), so switching to the NONMEM method does not silently run NONMEM’s law on the other rule’s tuning. An explicitly supplied value always wins.

`df` Degrees of freedom of the importance-sampling proposal (NONMEM ‘DF‘). ‘0‘ (default) uses a multivariate **normal** proposal; any value ‘> 0‘ uses a multivariate **t** with that many degrees of freedom.

This changes the proposal’s SHAPE rather than its width, and that is the distinction that matters. ‘gamma‘ can only make a Gaussian proposal wider; it cannot give it heavier tails. When the target posterior has heavier tails than the proposal, the importance weights have infinite variance – and neither ‘xi‘ nor the Kish effective sample size can detect it, because the offending mass lies where the proposal rarely lands. The Pareto k-hat diagnostic (‘fitenvimpPsisK‘) does detect it: ‘k > 0.7‘ means that subject’s weights are unreliable. A t proposal has polynomial tails that dominate a Gaussian target’s, which bounds the weights.

NONMEM’s guidance (Bauer, *NONMEM Tutorial Part II*) is to set a nonzero ‘DF‘ when there are fewer data points than etas, or for categorical data. Small values (3-8) are heavy; large values approach the Gaussian.

`proposal` Importance-sampling proposal family. ‘"auto"‘ (default) resolves to ‘"t"‘ when ‘df > 0‘ and ‘"normal"‘ otherwise, so the historical ‘df‘ behaviour is unchanged.

* ‘"normal"‘ / ‘"t"‘ – the multivariate normal and t proposals ‘df‘ already selected. See ‘df‘ for the shape-versus-width discussion these all turn on. * ‘"laplace"‘ – a spherical (Kotz-type) multivariate Laplace, ‘f(x) ~ exp(-sqrt(x’ S^-1 x))‘. Its exponential tail dominates the joint target’s, which for a bounded likelihood is at worst Gaussian, so the importance weights are BOUNDED by construction – without having to pick a ‘df‘, and without the low-‘df‘ t’s cost in effective sample size. Its scale is covariance-matched (‘S = Sigma/(p+1)‘), so ‘gamma‘, ‘iscaleMin‘ and ‘iscaleMax‘ mean what they mean for ‘"normal"‘. * ‘"mixture"‘ – a defensive scale mixture about the same mode, ‘sum_k w_k N(mode, c_k gamma Sigma)‘, with ‘propMixScale‘ and ‘propMixWeight‘. A broad component covers what a narrow one misses, so the mixture is deliberately over-dispersed relative to ‘gamma * Sigma‘. Note this does NOT bound the weights: its widest component is still Gaussian-tailed, so use ‘"laplace"‘ when that is what you need.

‘auto‘ adapts ‘df‘ only on the normal/t axis – its ladder encodes "Gaussian" as ‘df <= 0‘ and its constants were tuned against a Gaussian baseline, so a subject on ‘"laplace"‘ or ‘"mixture"‘ keeps its family. The ‘isample‘ budget reallocation still applies to every family.

‘fitenvimpProposal‘ reports the resolved family and ‘fitenvimpPropInd‘ the per-subject families actually used, so an ‘auto‘ escalation from normal to t is visible.

Measured trade-off. 8 seeds, ‘isample = 300‘, RMSE against an ‘isample = 8000‘ reference (‘design/qrpem/qrpem-options-bench.R‘), reported as theta RMSE / ‘Omega‘ RMSE / max Pareto k-hat / mean effective-sample fraction:

- 1 ETA – ‘"normal"‘ 0.00157 / 0.00344 / -1.18 / 0.624; ‘"laplace"‘ 0.00128 / 0.00367 / -0.86 / 0.753; ‘"mixture"‘ 0.00173 / 0.00510 / -1.33 / 0.592
- 3 ETAs – ‘"normal"‘ 0.00144 / 0.00184 / -0.22 / 0.724; ‘"laplace"‘ 0.00110 / 0.00229 / +0.28 / 0.817; ‘"mixture"‘ 0.00140 / 0.00296 / -0.33 / 0.668
- general ‘ll()‘ – ‘"normal"‘ 0.00173 / 0.00170 / -0.66 / 0.658; ‘"laplace"‘ 0.00214 / 0.00227 / -0.31 / 0.798; ‘"mixture"‘ 0.00144 / 0.00208 / -0.67 / 0.611
- 8 ETAs (warfarin PK/PD) – ‘"normal"‘ 0.01459 / 0.01941 / +0.97 / 0.551; ‘"laplace"‘ 0.01161 / 0.01430 / +1.07 / 0.576; ‘"mixture"‘ 0.01905 / 0.02003 / +1.19 / 0.527

‘"auto"‘ stays the default – no family wins everywhere – but the 8-ETA row is the informative one and it is worth reading carefully. It is the only fixture here whose sampler actually fails (2.1 subjects above k-hat 0.7 under the default), and there ‘"laplace"‘ gives the best theta RMSE of every setting measured, 20 effective sample size. On the three easy fixtures no subject exceeds 0.7 under any setting, so there is nothing for a bounded-weight proposal to repair and it merely costs accuracy. That is the rule of thumb: reach for ‘"laplace"‘ when ‘fitenvimpPsisK‘ shows a tail the ‘df‘ ladder is not fixing, not otherwise.

One honest caveat on the mechanism. ‘"laplace"‘ improved ACCURACY at 8 ETAs without improving k-hat itself (+1.07 against +0.97). Bounded weights

are an asymptotic guarantee, and 300 samples in 8 dimensions is not the asymptotic regime; the tail here is also not the non-identified kind the ‘autoNonmemSparse‘ note describes, since every subject has 13 observations for its 8 random effects. So take the accuracy gain as measured and the tail argument as theory that this fixture does not confirm.

`propMixScale` Component variance multipliers for ‘proposal="mixture"‘, length 2 or 3, starting at ‘1‘ and strictly increasing. Used as given: component 1 is the Laplace-approximation covariance itself and the rest are the wider defensive components, so the mixture’s covariance is ‘(sum_k w_k c_k) * gamma * Sigma‘ – deliberately wider than ‘gamma * Sigma‘, which is the whole mechanism. ‘gamma‘ still scales the entire mixture, so ‘iscaleMin‘/‘iscaleMax‘ still bound it.

`propMixWeight` Component weights for ‘proposal="mixture"‘, same length as ‘propMixScale‘, positive, normalized internally.

`auto` NONMEM ‘AUTO=1‘ equivalent: adapt the proposal degrees of freedom, the sample count and the acceptance target **per subject** rather than applying one global setting to everybody.

* **‘df‘** – any subject gets a heavy-tailed t proposal when the model is not transformably normal, and any subject whose Pareto k-hat reports tail failure gets one on that evidence. The tutorial’s other trigger, "fewer data points than there are ETAs", is **not** applied on its own – see ‘autoNonmemSparse‘. An escalation that fails to improve k-hat over two iterations is withdrawn, since a heavy tail the data creates is not repairable by proposal shape. * **‘isample‘** – the total sample budget (‘isample * nsub‘) is reallocated toward subjects whose effective-sample fraction is lowest, the tutorial’s "many ETAs or ... large stochastic fluctuations". It is load-balancing, not a cost increase. Note sample count is deliberately *not* driven by Pareto k-hat: a heavy tail is not repairable by more draws (see ‘df‘). * **‘iaccept‘** – lowered to 0.2 for the same sparse/categorical subjects, per the tutorial.

The concrete values (the ‘df‘ ladder, the k-hat thresholds, the budget reallocation rule) are **nlmixr2’s choices**. NONMEM does not publish what ‘AUTO=1‘ picks internally; only the triggers and ‘IACCEPT ~ 0.2‘ are documented. Unlike NONMEM’s AUTO, which its own tutorial warns "may result in lack of stochastic reproducibility", this remains seeded and thread-count independent.

Per-subject values are reported in ‘fitenvimpDfInd‘, ‘fitenvimpNsampleInd‘ and ‘fitenvimpIacceptInd‘.

Measured trade-off. ‘auto = TRUE‘ (the default) improves the *tail* behaviour of the importance weights and the accuracy of ‘Omega‘, at some cost in Monte-Carlo noise on the objective. On theophylline with three ETAs – a one-ETA model has no tail failure to fix, so it shows nothing either way – against a reference computed at ‘isample = 8000‘, over 8 seeds:

- ‘auto = FALSE‘: max Pareto k-hat 0.941, 2.38 subjects above 0.7; objective RMSE 0.0113, ‘Omega‘ RMSE 0.00406
- ‘auto = TRUE‘: max Pareto k-hat 0.571, 0.25 subjects above 0.7; objective RMSE 0.0165, ‘Omega‘ RMSE 0.00301

So ‘auto‘ removes the tail failure and estimates ‘Omega‘ about 26 accurately, for about 46 on by default because weights with infinite variance are a correctness

problem – their error is unbounded in the worst case – whereas the extra noise is a bounded, measurable cost. Setting ‘df‘ globally gives a better objective still on a model that is uniformly heavy-tailed, but costs 75 more objective RMSE on one that is not, which is why it is not the default.

Set ‘auto = FALSE‘ to recover the previous behaviour exactly (that path remains bit-identical to earlier versions), which is worth doing when you want the tightest possible objective on a model whose ‘fitenvimpPsisK‘ values are already comfortably below 0.7.

`autoNonmemSparse`

Apply the NONMEM tutorial’s ‘nobs < neta‘ trigger unconditionally, so a subject with fewer observations than random effects always gets a t proposal. ‘FALSE‘ (default) leaves such subjects to the Pareto k-hat evidence like any other, and withdraws an escalation that does not improve k-hat within two iterations.

The default diverges from the tutorial deliberately, on measurement. On a fixture built to the tutorial’s own definition of sparse (2 observations, 3 etas), 8 seeds against an ‘isample = 8000‘ reference, applying the rule made every number worse – objective RMSE 0.1517 -> 0.2059, ‘Omega‘ RMSE 0.02379 -> 0.03163, and *more* failing subjects (3.88 -> 4.88). With fewer observations than random effects the individual posterior is not identified, so the heavy tail is structural and no proposal shape repairs it; the reference itself still reads max k-hat 0.794.

Set ‘TRUE‘ to get the documented NONMEM rule anyway. NONMEM’s own testing is not published and was not done on these models, so someone who measures the opposite on their own problem should be able to have it.

`autoDfPatience` Number of consecutive iterations an escalated ‘df‘ may fail to improve Pareto k-hat before that escalation is withdrawn and the subject returned to the proposal it had without it. ‘0‘ never withdraws. Withdrawal is final, so escalation and withdrawal cannot oscillate, and it never goes below the ‘df‘ the model itself requires – a non-normal endpoint keeps its t proposal.

A rung is judged against what the subject manages WITHOUT any escalation, measured while it sits there, so deterioration that happens before any escalation is tracked and an escalation is never credited for it.

`iscaleMin`, `iscaleMax`

Lower/upper bounds for the adapted ‘gamma‘ (NONMEM ISCALE_MIN / ISCALE_MAX). Both bounds are reachable under ‘gammaRule="target"‘; under ‘"floor"‘ ‘gamma‘ only ever moves up, so only ‘iscaleMax‘ can bind.

`iaccept` Minimum importance-sampling effective-sample fraction (NONMEM IACCEPT). The proposal scale ‘gamma‘ is kept at its efficient starting value while the achieved fraction stays at or above ‘iaccept‘, and is inflated (toward ‘iscaleMax‘) only when it drops below this floor.

`ctol` Convergence tolerance on the windowed objective-function change; ‘NULL‘ derives it from ‘sigdig‘.

`nConvWindow` Length of the trailing iteration window used to average the objective-function change for convergence (NONMEM-style CTYPE).

`impSeed` Base seed for the per-subject thread-safe (threefry) RNG streams; results are reproducible and independent of the thread count.

`covMethod` Covariance method. ‘"imp"‘ (default) computes the Monte-Carlo importance-sampling observed-information covariance for the estimated thetas and Omega elements: a finite-difference Hessian of the importance-sampling objective over fixed common-random-number samples, taken in the parameterization the fit estimates Omega in (the entries of ‘chol(Omega^-1)‘) and mapped to the Omega variances and covariances by the delta method. It is stashed as ‘$impCov‘ / ‘$impSe‘ (‘$impCovInternal‘ in the estimation parameterization, ‘$impCovJacobian‘ the map) and installed as the fit covariance when it is positive definite. Otherwise the information is repaired as the FOCEI ‘"|r|"‘ covariance is, by ‘sqrtm(info with a warning; when that fails too, a warning says why and no covariance is installed. The theta standard errors match the Hessian-based FOCEI covariance, though the variance of a tightly-determined random effect (an Omega diagonal) can be over-estimated because the fixed samples barely span its prior variation. ‘"analytic"‘, ‘"r,s"‘, ‘"r"‘, ‘"s"‘ instead compute the FOCEI covariance post-fit at the converged estimates (see [foceiControl()]); ‘""‘ skips the covariance step.

`qr` When ‘TRUE‘, draw quasi-random (Sobol low-discrepancy) importance samples instead of pseudo-random Gaussian samples (QRPEM, Leary & Dunlavey PAGE 2012); the E-step integrals converge at O(1/N) instead of O(1/sqrt(N)).

`qrShift` Only used with ‘qr=TRUE‘. When ‘TRUE‘ each (iteration, subject) applies a random Cranley-Patterson shift to the Sobol points (seeded, thread-count independent); ‘FALSE‘ reuses one fixed Sobol point set everywhere (fully deterministic E-step, no RNG in the draw).

`qrRefresh` Only used with ‘qr=TRUE‘ and ‘qrShift=TRUE‘. When ‘TRUE‘ the shift is redrawn each iteration so residual quasi-random error averages out over the EM; ‘FALSE‘ draws one shift per subject at the fit start, making each EM iteration a deterministic map (smoothest objective trace).

`qrScramble` Only used with ‘qr=TRUE‘. Scrambling of the Sobol point set: ‘"none"‘ (default) uses the raw sequence, randomized only by the Cranley-Patterson shift; ‘"owen"‘ applies a hash-based nested uniform (Owen) scramble; ‘"lms"‘ applies a linear matrix scramble with a digital shift.

A Cranley-Patterson shift randomizes the point set but leaves the correlation structure between the sequence's dimensions intact; scrambling permutes the digits and breaks it.

Measured trade-off, and it is not the one the paragraph above suggests. The scramble's advantage over the shift is real but does NOT grow with the number of random effects: on a smooth test integrand at ‘isample = 300‘ it is 1.79x at one dimension and 1.53x at twelve. And it lands on the E-STEP INTEGRAL rather than on the estimates – single-iteration importance- sampling ‘-2LL‘ RMSE 0.056 under ‘"owen"‘ against 0.080 unscrambled (3 ETAs), while converged parameter accuracy is unmoved or worse. Against an ‘isample = 8000‘ reference over 8 seeds, theta RMSE:

- 1 ETA – shift 0.00022, ‘"owen"‘ 0.00018, ‘"lms"‘ 0.00022
- 3 ETAs – shift 0.00031, ‘"owen"‘ 0.00045, ‘"lms"‘ 0.00049
- general ‘ll()‘ – shift 0.00020, ‘"owen"‘ 0.00032, ‘"lms"‘ 0.00028
- 8 ETAs – shift 0.01493, ‘"owen"‘ 0.01787, ‘"lms"‘ 0.01924

So ‘"none"‘ stays the default: at the eta dimension where a scramble was expected to pay off most it is the worst of the three on theta, and it never improves Pareto k-hat. Set ‘"owen"‘ when the quantity you care about is the reported importance-sampling objective (‘fitenvimpObj‘, e.g. for comparing models by IS likelihood) rather than the parameter estimates – that is the integral it measurably improves.

Scrambling IS the randomization, so it REPLACES the shift rather than composing with it: ‘qrShift‘ is ignored when this is not ‘"none"‘, while ‘qrRefresh‘ still decides whether the randomization is redrawn each iteration (‘FALSE‘ pins one scramble per subject, making each EM iteration a deterministic map). The scramble key is derived arithmetically from ‘impSeed‘ and the (iteration, subject, dimension) indices rather than drawn from the RNG, so it consumes no draws and the fit stays reproducible and independent of the thread count.

‘"lms"‘ is named for the method (Matousek/Tezuka linear matrix scrambling) and is not a claim to reproduce any particular vendor's variant.

`sir` When ‘TRUE‘, accelerate the non-mu / residual-error M-step by SIR (sampling-importance-resampling): the theta-sensitivity Newton step uses ‘sirSample‘ equal-weight resampled points per subject instead of all ‘isample‘ weighted samples.

`sirSample` Number of SIR resampled points per subject; ‘NULL‘ uses ‘max(25, ceiling(isample/10))‘. Must be at most ‘isample‘.

`muModel` Mu-referencing variant for the MAP inner problem; for ‘impmapControl()‘ this is always ‘"lin"‘ and cannot be changed.

`combSens` When ‘TRUE‘ (the default) and there are non-mu (structural or residual-error) thetas to estimate, carry their sensitivity columns on the INNER model itself (the combined eta+theta sensitivity build, #958) instead of compiling and solving a separate, dedicated model for them. With ‘sir=FALSE‘ (the default) the E-step's own per-sample inner solve then supplies the M-step's Newton step directly, with no second solve – roughly halving the ODE solving the theta-sensitivity Newton step costs. Set ‘FALSE‘ to use the older two-model path instead. (Earlier versions shipped ‘FALSE‘ as the default: on a ‘linCmt()‘ model with a single non-mu structural theta, ‘combSens=TRUE‘/‘FALSE‘ measurably disagreed – not ‘combSens‘ moving the answer, but an unrelated pre-existing bug in how the shared solve pool swapped between peer models, since fixed. The two now agree on every fixture measured, so ‘TRUE‘ is the default.)

Value

impmapControl object

Author(s)

Matthew L. Fidler

Examples

```
impmapControl()
```

`iterPrintControl` *Control iteration-time print formatting*

Description

Bundles the options controlling the iteration progress output emitted by ‘nlmixr2‘ estimators. Pass as the ‘print‘ argument to any ‘*Control()‘ function; the scalar form (‘print = N‘) still works and is wrapped into an ‘iterPrintControl()‘ internally.

Usage

```
iterPrintControl(
  every = 1L,
  ncol = NULL,
  headerEvery = NULL,
  useColor = NULL,
  simple = FALSE
)
```

Arguments

`every`	Integer. Print one iteration row every ‘every‘ parameter evaluations; ‘0‘ suppresses output. Defaults to ‘1L‘.
`ncol`	Integer or ‘NULL‘. Parameter columns per row before wrapping. ‘NULL‘ (default) uses ‘floor((getOption("width") - 23) / 12)‘.
`headerEvery`	Integer or ‘NULL‘. Re-emit the column header every ‘headerEvery‘ parameter-print events; ‘0‘ prints it once at fit start. ‘NULL‘ (default) uses ‘10L‘.
`useColor`	Logical (or ‘NULL‘) emit ANSI bold/color escapes in the iteration print. ‘NULL‘ (default) defers to [crayon::has_color()].
`simple`	Logical. When ‘TRUE‘, print a single row per iteration, suppressing the unscaled (‘U‘) / back-transformed (‘X‘) rows. Defaults to ‘FALSE‘.

Value

A list with the validated, defaulted iteration-print options. Has class ‘"iterPrintControl"‘ so the outer ‘*Control()‘ functions can distinguish a pre-built object from a scalar ‘print = N‘.

Author(s)

Bill Denney, Matthew L. Fidler

Examples

```
iterPrintControl()
iterPrintControl(every = 5, headerEvery = 0)
```

`laplaceControl` *Control options for the Laplace estimation method*

Description

This is the control options for the adaptive Gauss-Hermite quadrature for the likelihood. Note that nAGQ=1 is the same as the Laplace method.

Usage

```
laplaceControl(sigdig = 3, ..., nAGQ = 1)
```

Arguments

`sigdig`	Optimization significant digits. One value drives, with a single consistent formula, the inner/outer optimizer convergence tolerance (`10^-sigdig`), the boundary check tolerance (`5*10^(-sigdig+1)`), and the ODE solver tolerances: the `rtol` exponent IS `sigdig` and `atol` sits three orders below, so `rtol = 10^-sigdig`, `atol = 10^(-sigdig-3)` for every solver (stiff, non-stiff or auto-switching). The sensitivity (`atolSens`/`rtolSens`) tolerances match the main solve (the outer gradient and covariance are built from them); the steady-state (`ssAtol`/`ssRtol`) tolerances run one order looser. Keying the optimizer to the same `10^-sigdig` means it converges to exactly the precision the solve supports. At the default `sigdig = 3` this is `atol = 1e-6`, `rtol = 1e-3`.
`...`	Parameters used in the default 'foceiControl()'
`nAGQ`	Number of Gauss-Hermite adaptive quadrature points. '0' disables AGQ; '1' is equivalent to Laplace. Cost grows quickly with ETAs: once the EBE is found, expect 'nAGQ^neta' (even 'nAGQ') or '(nAGQ^neta)-1' (odd 'nAGQ') additional evaluations per subject.

Details

This method can be made to more closely matches NONMEM-style Laplace estimation by requesting the log-likelihood from STAN as well as numerically calculated Hessian matrix. This is done with adding '+dnorm()' to the model for any normal end-points.

Value

laplaceControl object

Author(s)

Matthew L. Fidler

Examples

```
laplaceControl()

# Use adaptive quadrature

# x =  Litter size after 21 days, and the modeled value

r <- rats
r$dv <- r$x

# Time is not used in this model, but it is required in nlmixr2
# currently, add a dummy value

r$time <- 0

f <- function() {
  ini({
    t1 <- 1
    t2 <- 1
    t3 <- 1
    eta1 ~ 1
  })
  model({
    lp <- t1 * x1 + t2 * x2 + (x1 + x2*t3) * eta1
    p <- pnorm(lp)
    m1 <- m # need to add outside of model specification
    x ~ dbinom(m1, p)
  })
}

fit <- nlmixr(f, r, est="laplace")


p <- pump

p$dv <- p$y
p$time <- 0 # dummy time

f <- function() {
  ini({
    t1 <- 1
    t2 <- 1
    t3 <- 1
    t4 <- 1
    eta1 ~ 1
  })
  model({
    if (group == 1) {
       lp <- t1 + t2 * logtstd
    } else {
```

```
      lp <- t3 + t4 * logtstd
    }
    lp <- lp + eta1
    lam <- exp(lp)
    y ~ dpois(lam)
  })
}

fit <- nlmixr(f, p, est="laplace")
```

`lbfgsb3cControl` *Control for lbfgsb3c estimation method in nlmixr2*

Description

Control for lbfgsb3c estimation method in nlmixr2

Usage

```
lbfgsb3cControl(
  trace = 0,
  factr = NULL,
  pgtol = 0,
  abstol = 0,
  reltol = 0,
  lmm = 5L,
  maxit = 10000L,
  returnLbfgsb3c = FALSE,
  stickyRecalcN = 4,
  maxOdeRecalc = 5,
  odeRecalcFactor = 10^(0.5),
  indTolRelax = TRUE,
  useColor = NULL,
  printNcol = NULL,
  print = 1L,
  normType = c("rescale2", "mean", "rescale", "std", "len", "constant"),
  scaleType = c("nlmixr2", "norm", "mult", "multAdd"),
  scaleCmax = 1e+05,
  scaleCmin = 1e-05,
  scaleC = NULL,
  scaleTo = 1,
  gradTo = 1,
  rxControl = NULL,
  optExpression = TRUE,
  sumProd = FALSE,
```

```
  literalFix = TRUE,
  literalFixRes = TRUE,
  addProp = c("combined2", "combined1"),
  eventSens = c("jump", "fd"),
  sensMethod = c("default", "forward"),
  calcTables = TRUE,
  compress = FALSE,
  covMethod = c("r", ""),
  adjObf = TRUE,
  ci = 0.95,
  sigdig = 3,
  sigdigTable = NULL,
  ...
)
```

Arguments

trace	Ignored: the thread-safe C++ L-BFGS-B prints nothing; use ‘print‘ for the iteration log. Kept so existing calls still work.
factr	Convergence tolerance factor for "L-BFGS-B"; converges when the objective reduction is within this factor of machine tolerance. ‘NULL‘ (default) uses ‘10^(-sigdig-2) / .Machine$double.eps‘ (at least 10), two orders tighter than ‘sigdig‘, as ‘foceiControl(lbfgsFactr=)‘ does.
pgtol	Tolerance on the projected gradient for "L-BFGS-B"; 0 (default) suppresses the check.
abstol	Absolute x-value tolerance for "L-BFGS-B"; 0 (default) suppresses the check.
reltol	Relative x-value tolerance for "L-BFGS-B"; 0 (default) suppresses the check.
lmm	Number of BFGS updates retained in "L-BFGS-B" (default 5).
maxit	maximum number of objective function evaluations.
returnLbfgsb3c	return the lbfgsb3c output instead of the nlmixr2 fit
stickyRecalcN	The number of bad ODE solves before reducing the atol/rtol for the rest of the problem.
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
indTolRelax	When ‘TRUE‘ (default), only subjects whose ODE solve produced NaN/Inf have their tolerances relaxed, and the relaxed tolerance persists across optimizer calls (sticky). When ‘FALSE‘, all subjects have their tolerances relaxed on each retry and tolerances are reset afterward.
useColor	Logical (or ‘NULL‘) emit ANSI bold/color escapes in the iteration print. ‘NULL‘ (default) defers to [crayon::has_color()].
printNcol	Integer (or ‘NULL‘) parameter columns per row before wrapping. ‘NULL‘ (default) uses ‘floor((getOption("width") - 23) / 12)‘.

`print` Either a scalar print-frequency (‘0‘ = suppress, ‘1‘ (default) = every evaluation, ‘N‘ = every Nth), OR a pre-built [iterPrintControl()] object. Equivalent to ‘iterPrintControl(every = print, ncol = printNcol, useColor = useColor)‘.

`normType` Parameter normalization/scaling used to get scaled initial values for `scaleType`, of the form `Vscaled = (Vunscaled-C1)/C2` (see Feature Scaling; `rescale2` follows the OptdesX manual): `"rescale2"` scales all parameters to (-1, 1); `"rescale"` (min-max) scales to (0, 1); `"mean"` centers on the mean with range (0, 1); `"std"` standardizes by mean/sd; `"len"` scales to unit (Euclidean) length; `"constant"` performs no normalization (`C1=0`, `C2=1`).

`scaleType` The scaling scheme for nlmixr2: `"nlmixr2"` (default) scales as `(current-init)*scaleC[i] + scaleTo`, with `scaleTo` from `normType` and scales from `scaleC`; `"norm"` uses the simple scaling from `normType`; `"mult"` scales multiplicatively as `current/init*scaleTo`; `"multAdd"` scales linearly (`(current-init)+scaleTo`) for parameters in an exponential block (e.g. `exp(theta)`) and multiplicatively otherwise.

`scaleCmax` Maximum value of the scaleC to prevent overflow.

`scaleCmin` Minimum value of the scaleC to prevent underflow.

`scaleC` Scaling constant used with `scaleType="nlmixr2"`; when not specified, chosen by parameter type to keep gradient sizes similar on a log scale: ‘1‘ for exp()-transformed/power/boxCox/ yeoJohnson parameters, ‘0.5*abs(est)‘ for additive/proportional/ lognormal error parameters, ‘abs(1/digamma(est+1))‘ for factorials, and ‘log(abs(est))*abs(est)‘ for log-scale parameters. May be set explicitly per parameter if these defaults don’t apply well.

`scaleTo` Scale the initial parameter estimate to this value. By default this is 1. When zero or below, no scaling is performed.

`gradTo` this is the factor that the gradient is scaled to before optimizing. This only works with scaleType="nlmixr2".

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

`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`.

`literalFix` boolean, substitute fixed population values as literals and re-adjust ui and parameter estimates after optimization; Default is ‘TRUE‘.

`literalFixRes` boolean, substitute fixed population values as literals and re-adjust ui and parameter estimates after optimization; Default is ‘TRUE‘.

`addProp` Type of additive-plus-proportional error: ‘"combined1"‘, where standard deviations add:

$$y = f + (a + b \times f^c) \times \varepsilon$$

; or ‘"combined2"‘, where variances add:

$$y = f + \sqrt{a^2 + b^2 \times f^{2 \times c}} \times \varepsilon$$

. Here y = observed, f = predicted, a = additive sd, b = proportional/power sd, c = power exponent (1 in the proportional case).

eventSens	Controls how dosing/event-parameter (‘alag‘, ‘F‘, ‘rate‘, ‘dur‘) sensitivities are computed for THETA/ETA gradients: ‘"jump"‘ (default) uses rxode2’s analytic event sensitivities; ‘"fd"‘ uses the legacy finite-difference behavior. Also gates the analytic moving-boundary correction for a modeled ‘alag()‘/‘f()‘ on a ‘linCmt()‘ compartment; set ‘"fd"‘ if that model infuses a dose into the lagged/scaled compartment (rxode2/rxode2#1236), or if the regimen also doses an *unlagged/unscaled* compartment alongside the lagged/scaled one – a common design for estimating ‘f()‘ from paired IV+oral data (rxode2/rxode2#1237).
sensMethod	Method used to compute the ODE parameter sensitivities. ‘"forward"‘ uses the classic variational (forward) sensitivity ODEs; ‘"default"‘ is the same thing.
calcTables	This boolean is to determine if the foceiFit will calculate tables. By default this is TRUE
compress	Should the object have compressed items
covMethod	Method for calculating the covariance. "r" (the default) uses nlmixr2’s nlmixr2Hess() Hessian; "" skips the covariance step.
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. One value drives, with a single consistent formula, the inner/outer optimizer convergence tolerance (10^-sigdig), the boundary check tolerance (5*10^(-sigdig+1)), and the ODE solver tolerances: the rtol exponent IS sigdig and atol sits three orders below, so rtol = 10^-sigdig, atol = 10^(-sigdig-3) for every solver (stiff, non-stiff or auto-switching). The sensitivity (atolSens/rtolSens) tolerances match the main solve (the outer gradient and covariance are built from them); the steady-state (ssAtol/ssRtol) tolerances run one order looser. Keying the optimizer to the same 10^-sigdig means it converges to exactly the precision the solve supports. At the default sigdig = 3 this is atol = 1e-6, rtol = 1e-3.
sigdigTable	Significant digits in the final output table. If not specified (‘NULL‘), it defaults to ‘sigdig‘.
...	Ignored parameters

Value

bobqya control structure

Author(s)

Matthew L. Fidler

Examples

```
# A logit regression example with emax model

dsn <- data.frame(i=1:1000)
dsn$time <- exp(rnorm(1000))
```

```
dsn$DV=rbinom(1000,1,exp(-1+dsn$time)/(1+exp(-1+dsn$time)))

mod <- function() {
 ini({
   E0 <- 0.5
   Em <- 0.5
   E50 <- 2
   g <- fix(2)
 })
 model({
   v <- E0+Em*time^g/(E50^g+time^g)
   ll(bin) ~ DV * v - log(1 + exp(v))
 })
}

fit2 <- nlmixr(mod, dsn, est="lbfgsb3c")

print(fit2)

# you can also get the nlm output with fit2$lbfgsb3c

fit2$lbfgsb3c

# The nlm control has been modified slightly to include
# extra components and name the parameters
```

`magqControl` *Control options for the magq estimation method*

Description

Mu-referenced-FOCEI-family closed-form-regression (‘"lin"‘) variant of adaptive Gauss-Hermite quadrature; see ‘foceiControl(muModel=)‘.

Usage

```
magqControl(
  sigdig = 3,
  nAGQ = 2,
  ...,
  interaction = TRUE,
  agqLow = -Inf,
  agqHi = Inf,
  muModel = c("lin", "irls", "none")
)
```

Arguments

sigdig	Optimization significant digits. One value drives, with a single consistent formula, the inner/outer optimizer convergence tolerance (10^-sigdig), the boundary check tolerance (5*10^(-sigdig+1)), and the ODE solver tolerances: the rtol exponent IS sigdig and atol sits three orders below, so rtol = 10^-sigdig, atol = 10^(-sigdig-3) for every solver (stiff, non-stiff or auto-switching). The sensitivity (atolSens/rtolSens) tolerances match the main solve (the outer gradient and covariance are built from them); the steady-state (ssAtol/ssRtol) tolerances run one order looser. Keying the optimizer to the same 10^-sigdig means it converges to exactly the precision the solve supports. At the default sigdig = 3 this is atol = 1e-6, rtol = 1e-3.
nAGQ	Number of Gauss-Hermite adaptive quadrature points. ‘0‘ disables AGQ; ‘1‘ is equivalent to Laplace. Cost grows quickly with ETAs: once the EBE is found, expect ‘nAGQ^neta‘ (even ‘nAGQ‘) or ‘(nAGQ^neta)-1‘ (odd ‘nAGQ‘) additional evaluations per subject.
...	Parameters used in the default ‘foceiControl()‘
interaction	boolean, Interaction term for the model, in this case the default is ‘TRUE‘; For adaptive quadrature, with normal distribution the Hessian is calculated with the foce(i) approximation
agqLow	The lower bound for adaptive quadrature log-likelihood. By default this is -Inf; in the original nlmixr’s gnlmm it was -700.
agqHi	The upper bound for adaptive quadrature log-likelihood. By default this is Inf; in the original nlmixr’s gnlmm was 400.
muModel	Selects the regression variant; for ‘magqControl()‘ this is always ‘"lin"‘ and cannot be changed – use ‘iagqControl()‘ for the IRLS variant.

Value

magqControl object

Author(s)

Matthew L. Fidler

Examples

```
magqControl()
```

`mfoceControl` *Control options for the mfoce estimation method*

Description

Mu-referenced-FOCEI-family closed-form-regression (‘"lin"‘) variant of FOCE (no interaction); see ‘foceiControl(muModel=)‘.

Usage

```
mfoceControl(
  sigdig = 3,
  ...,
  interaction = FALSE,
  muModel = c("lin", "irls", "none")
)
```

Arguments

sigdig	Optimization significant digits. One value drives, with a single consistent formula, the inner/outer optimizer convergence tolerance (10^-sigdig), the boundary check tolerance (5*10^(-sigdig+1)), and the ODE solver tolerances: the rtol exponent IS sigdig and atol sits three orders below, so rtol = 10^-sigdig, atol = 10^(-sigdig-3) for every solver (stiff, non-stiff or auto-switching). The sensitivity (atolSens/rtolSens) tolerances match the main solve (the outer gradient and covariance are built from them); the steady-state (ssAtol/ssRtol) tolerances run one order looser. Keying the optimizer to the same 10^-sigdig means it converges to exactly the precision the solve supports. At the default sigdig = 3 this is atol = 1e-6, rtol = 1e-3.
...	Parameters used in the default ‘foceiControl()‘
interaction	Interaction term for the model, in this case the default is ‘FALSE‘; it cannot be changed, use ‘mfocei‘ instead
muModel	Selects the regression variant; for ‘mfoceControl()‘ this is always ‘"lin"‘ and cannot be changed – use ‘ifoceControl()‘ for the IRLS variant.

Value

mfoceControl object

Difference from ‘focei‘

The ‘mfocei‘/‘ifocei‘ (and related) methods apply the mu2+ covariate hooks, which expand algebraic mu-referenced covariate expressions (e.g. ‘cl.wt*log(WT/70)‘) into estimable mu-referenced parameters and split covariates into non-time-varying (absorbed into the phi term) and time-varying (kept as ‘beta‘ regressors). Calling ‘focei‘ directly does NOT apply these hooks, so these methods can estimate more mu-referenced models than plain ‘focei‘ – there is a genuine difference between calling e.g. ‘est="mfocei"‘ and ‘est="focei"‘.

All mu-referenced population thetas – with or without covariates – are profiled out of the outer optimizer by the in-C++ regression (intercept-only for covariate-free pairs), so outer gradients are only calculated for the non-mu-referenced parameters (residual errors, omegas, non-mu thetas). Bounded mu-referenced parameters are regression-updated with the update clamped to the bounds (a clamp is reported once as a fit note); user-fixed (‘fix()‘) mu thetas stay out of the regression.

Author(s)

Matthew L. Fidler

Examples

```
mfoceControl()
```

`mfoceiControl` *Control options for the mfocei estimation method*

Description

Mu-referenced-FOCEI-family closed-form-regression (‘"lin"‘) variant of FOCEI; see ‘foceiControl(muModel=)‘.

Usage

```
mfoceiControl(sigdig = 3, ..., muModel = c("lin", "irls", "none"))
```

Arguments

`sigdig`	Optimization significant digits. One value drives, with a single consistent formula, the inner/outer optimizer convergence tolerance (`10^-sigdig`), the boundary check tolerance (`5*10^(-sigdig+1)`), and the ODE solver tolerances: the `rtol` exponent IS `sigdig` and `atol` sits three orders below, so `rtol = 10^-sigdig`, `atol = 10^(-sigdig-3)` for every solver (stiff, non-stiff or auto-switching). The sensitivity (`atolSens`/`rtolSens`) tolerances match the main solve (the outer gradient and covariance are built from them); the steady-state (`ssAtol`/`ssRtol`) tolerances run one order looser. Keying the optimizer to the same `10^-sigdig` means it converges to exactly the precision the solve supports. At the default `sigdig = 3` this is `atol = 1e-6`, `rtol = 1e-3`.
`...`	Parameters used in the default ‘foceiControl()‘
`muModel`	Selects the regression variant; for ‘mfoceiControl()‘ this is always ‘"lin"‘ and cannot be changed – use ‘ifoceiControl()‘ for the IRLS variant.

Value

mfoceiControl object

Difference from ‘focei‘

The ‘mfocei‘/‘ifocei‘ (and related) methods apply the mu2+ covariate hooks, which expand algebraic mu-referenced covariate expressions (e.g. ‘cl.wt*log(WT/70)‘) into estimable mu-referenced parameters and split covariates into non-time-varying (absorbed into the phi term) and time-varying (kept as ‘beta‘ regressors). Calling ‘focei‘ directly does NOT apply these hooks, so these methods can estimate more mu-referenced models than plain ‘focei‘ – there is a genuine difference between calling e.g. ‘est="mfocei"‘ and ‘est="focei"‘.

All mu-referenced population thetas – with or without covariates – are profiled out of the outer optimizer by the in-C++ regression (intercept-only for covariate-free pairs), so outer gradients are only calculated for the non-mu-referenced parameters (residual errors, omegas, non-mu thetas). Bounded mu-referenced parameters are regression-updated with the update clamped to the bounds (a clamp is reported once as a fit note); user-fixed (‘fix()‘) mu thetas stay out of the regression.

Author(s)

Matthew L. Fidler

Examples

```
mfoceiControl()
```

`mfocepControl` *Control options for the mfocep (mu-referenced FOCE+) estimation method*

Description

Mu-referenced-FOCEI-family closed-form-regression (‘"lin"‘) variant of FOCE+ (no interaction, live conditional residual variance R); it is ‘mfoce‘ with ‘foce = "foce+"‘ forced (see ‘foceiControl(foce=)‘).

Usage

```
mfocepControl(
  sigdig = 3,
  ...,
  interaction = FALSE,
  muModel = c("lin", "irls", "none"),
  foce = "foce+"
)
```

Arguments

`sigdig`	Optimization significant digits. One value drives, with a single consistent formula, the inner/outer optimizer convergence tolerance (`10^-sigdig`), the boundary check tolerance (`5*10^(-sigdig+1)`), and the ODE solver tolerances: the `rtol` exponent IS `sigdig` and `atol` sits three orders below, so `rtol = 10^-sigdig`, `atol = 10^(-sigdig-3)` for every solver (stiff, non-stiff or auto-switching). The sensitivity (`atolSens`/`rtolSens`) tolerances match the main solve (the outer gradient and covariance are built from them); the steady-state (`ssAtol`/`ssRtol`) tolerances run one order looser. Keying the optimizer to the same `10^-sigdig` means it converges to exactly the precision the solve supports. At the default `sigdig = 3` this is `atol = 1e-6`, `rtol = 1e-3`.
`...`	Parameters used in the default ‘foceiControl()‘
`interaction`	Interaction term for the model, in this case the default is ‘FALSE‘; it cannot be changed, use ‘mfocei‘ instead
`muModel`	Selects the regression variant; for ‘mfocepControl()‘ this is always ‘"lin"‘ and cannot be changed – use ‘ifocepControl()‘ for the IRLS variant.
`foce`	FOCE residual-variance mode; for ‘mfocepControl()‘ this is always ‘"foce+"‘ and cannot be changed – use ‘mfoceControl()‘ for ‘"nonmem"‘

Value

mfocepControl object

Difference from ‘focei‘

The ‘mfocei‘/‘ifocei‘ (and related) methods apply the mu2+ covariate hooks, which expand algebraic mu-referenced covariate expressions (e.g. ‘cl.wt*log(WT/70)‘) into estimable mu-referenced parameters and split covariates into non-time-varying (absorbed into the phi term) and time-varying (kept as ‘beta‘ regressors). Calling ‘focei‘ directly does NOT apply these hooks, so these methods can estimate more mu-referenced models than plain ‘focei‘ – there is a genuine difference between calling e.g. ‘est="mfocei"‘ and ‘est="focei"‘.

All mu-referenced population thetas – with or without covariates – are profiled out of the outer optimizer by the in-C++ regression (intercept-only for covariate-free pairs), so outer gradients are only calculated for the non-mu-referenced parameters (residual errors, omegas, non-mu thetas). Bounded mu-referenced parameters are regression-updated with the update clamped to the bounds (a clamp is reported once as a fit note); user-fixed (‘fix()‘) mu thetas stay out of the regression.

Author(s)

Matthew L. Fidler

Examples

```
mfocepControl()
```

`mlaplaceControl` *Control options for the mlaplace estimation method*

Description

Mu-referenced-FOCEI-family closed-form-regression (‘"lin"‘) variant of the Laplace method (‘nAGQ=1‘); see ‘foceiControl(muModel=)‘.

Usage

```
mlaplaceControl(sigdig = 3, ..., nAGQ = 1, muModel = c("lin", "irls", "none"))
```

Arguments

`sigdig`	Optimization significant digits. One value drives, with a single consistent formula, the inner/outer optimizer convergence tolerance (`10^-sigdig`), the boundary check tolerance (`5*10^(-sigdig+1)`), and the ODE solver tolerances: the `rtol` exponent IS `sigdig` and `atol` sits three orders below, so `rtol = 10^-sigdig`, `atol = 10^(-sigdig-3)` for every solver (stiff, non-stiff or auto-switching). The sensitivity (`atolSens`/`rtolSens`) tolerances match the main solve (the outer gradient and covariance are built from them); the steady-state (`ssAtol`/`ssRtol`) tolerances run one order looser. Keying the optimizer to the same `10^-sigdig`

	means it converges to exactly the precision the solve supports. At the default `sigdig = 3` this is `atol = 1e-6`, `rtol = 1e-3`.
`...`	Parameters used in the default ‘foceiControl()‘
`nAGQ`	Number of Gauss-Hermite adaptive quadrature points. ‘0‘ disables AGQ; ‘1‘ is equivalent to Laplace. Cost grows quickly with ETAs: once the EBE is found, expect ‘nAGQ^neta‘ (even ‘nAGQ‘) or ‘(nAGQ^neta)-1‘ (odd ‘nAGQ‘) additional evaluations per subject.
`muModel`	Selects the regression variant; for ‘mlaplaceControl()‘ this is always ‘"lin"‘ and cannot be changed – use ‘ilaplaceControl()‘ for the IRLS variant.

Value

mlaplaceControl object

Author(s)

Matthew L. Fidler

Examples

```
mlaplaceControl()
```

`n1qn1Control` *Control for n1qn1 estimation method in nlmixr2*

Description

Control for n1qn1 estimation method in nlmixr2

Usage

```
n1qn1Control(
  epsilon = NULL,
  max_iterations = 10000,
  nsim = 10000,
  imp = 0,
  print.functions = FALSE,
  returnN1qn1 = FALSE,
  stickyRecalcN = 4,
  maxOdeRecalc = 5,
  odeRecalcFactor = 10^(0.5),
  indTolRelax = TRUE,
  useColor = NULL,
  printNcol = NULL,
  print = 1L,
  normType = c("rescale2", "mean", "rescale", "std", "len", "constant"),
  scaleType = c("nlmixr2", "norm", "mult", "multAdd"),
```

```
  scaleCmax = 1e+05,
  scaleCmin = 1e-05,
  scaleC = NULL,
  scaleTo = 1,
  gradTo = 1,
  rxControl = NULL,
  optExpression = TRUE,
  sumProd = FALSE,
  literalFix = TRUE,
  literalFixRes = TRUE,
  addProp = c("combined2", "combined1"),
  eventSens = c("jump", "fd"),
  sensMethod = c("default", "forward"),
  calcTables = TRUE,
  compress = FALSE,
  covMethod = c("r", "n1qn1", ""),
  adjObf = TRUE,
  ci = 0.95,
  sigdig = 3,
  sigdigTable = NULL,
  boundedTransform = TRUE,
  ...
)
```

Arguments

epsilon	Precision of estimate for n1qn1 optimization.
max_iterations	Number of iterations
nsim	Number of function evaluations
imp	Verbosity of messages.
print.functions	Boolean to control if the function value and parameter estimates are echoed every time a function is called.
returnN1qn1	return the n1qn1 output instead of the nlmixr2 fit
stickyRecalcN	The number of bad ODE solves before reducing the atol/rtol for the rest of the problem.
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
indTolRelax	When 'TRUE' (default), only subjects whose ODE solve produced NaN/Inf have their tolerances relaxed, and the relaxed tolerance persists across optimizer calls (sticky). When 'FALSE', all subjects have their tolerances relaxed on each retry and tolerances are reset afterward.

`useColor` Logical (or 'NULL') emit ANSI bold/color escapes in the iteration print. 'NULL' (default) defers to [crayon::has_color()].

`printNcol` Integer (or 'NULL') parameter columns per row before wrapping. 'NULL' (default) uses 'floor((getOption("width") - 23) / 12)'.

`print` Either a scalar print-frequency ('0' = suppress, '1' (default) = every evaluation, 'N' = every Nth), OR a pre-built [iterPrintControl()] object. Equivalent to 'iterPrintControl(every = print, ncol = printNcol, useColor = useColor)'.

`normType` Parameter normalization/scaling used to get scaled initial values for `scaleType`, of the form `Vscaled = (Vunscaled-C1)/C2` (see Feature Scaling; `rescale2` follows the OptdesX manual): `"rescale2"` scales all parameters to (-1, 1); `"rescale"` (min-max) scales to (0, 1); `"mean"` centers on the mean with range (0, 1); `"std"` standardizes by mean/sd; `"len"` scales to unit (Euclidean) length; `"constant"` performs no normalization (`C1=0`, `C2=1`).

`scaleType` The scaling scheme for nlmixr2: `"nlmixr2"` (default) scales as `(current-init)*scaleC[i] + scaleTo`, with `scaleTo` from `normType` and scales from `scaleC`; `"norm"` uses the simple scaling from `normType`; `"mult"` scales multiplicatively as `current/init*scaleTo`; `"multAdd"` scales linearly (`(current-init)+scaleTo`) for parameters in an exponential block (e.g. `exp(theta)`) and multiplicatively otherwise.

`scaleCmax` Maximum value of the scaleC to prevent overflow.

`scaleCmin` Minimum value of the scaleC to prevent underflow.

`scaleC` Scaling constant used with `scaleType="nlmixr2"`; when not specified, chosen by parameter type to keep gradient sizes similar on a log scale: '1' for exp()-transformed/power/boxCox/ yeoJohnson parameters, '0.5*abs(est)' for additive/proportional/ lognormal error parameters, 'abs(1/digamma(est+1))' for factorials, and 'log(abs(est))*abs(est)' for log-scale parameters. May be set explicitly per parameter if these defaults don't apply well.

`scaleTo` Scale the initial parameter estimate to this value. By default this is 1. When zero or below, no scaling is performed.

`gradTo` this is the factor that the gradient is scaled to before optimizing. This only works with scaleType="nlmixr2".

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

`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`.

`literalFix` boolean, substitute fixed population values as literals and re-adjust ui and parameter estimates after optimization; Default is 'TRUE'.

`literalFixRes` boolean, substitute fixed population values as literals and re-adjust ui and parameter estimates after optimization; Default is 'TRUE'.

`addProp` Type of additive-plus-proportional error: '"combined1"', where standard deviations add:

$$y = f + (a + b \times f^c) \times \varepsilon$$

; or '"combined2"', where variances add:

$$y = f + \sqrt{a^2 + b^2 \times f^{2\times c}} \times \varepsilon$$

. Here y = observed, f = predicted, a = additive sd, b = proportional/power sd, c = power exponent (1 in the proportional case).

`eventSens` Controls how dosing/event-parameter ('alag', 'F', 'rate', 'dur') sensitivities are computed for THETA/ETA gradients: '"jump"' (default) uses rxode2's analytic event sensitivities; '"fd"' uses the legacy finite-difference behavior. Also gates the analytic moving-boundary correction for a modeled 'alag()'/'f()' on a 'linCmt()' compartment; set '"fd"' if that model infuses a dose into the lagged/scaled compartment (rxode2/rxode2#1236), or if the regimen also doses an *unlagged/unscaled* compartment alongside the lagged/scaled one – a common design for estimating 'f()' from paired IV+oral data (rxode2/rxode2#1237).

`sensMethod` Method used to compute the ODE parameter sensitivities. '"forward"' uses the classic variational (forward) sensitivity ODEs; '"default"' is the same thing.

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

`compress` Should the object have compressed items

`covMethod` Method for calculating the covariance. `"r"` (the default) uses nlmixr2's `nlmixr2Hess()` Hessian; `"n1qn1"` uses the optimizer's own Hessian, the quasi-Newton approximation n1qn1 built along its path (`H`), not a finite-difference Hessian at the estimates; `""` skips the covariance step.

`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. One value drives, with a single consistent formula, the inner/outer optimizer convergence tolerance (`10^-sigdig`), the boundary check tolerance (`5*10^(-sigdig+1)`), and the ODE solver tolerances: the `rtol` exponent IS `sigdig` and `atol` sits three orders below, so `rtol = 10^-sigdig`, `atol = 10^(-sigdig-3)` for every solver (stiff, non-stiff or auto-switching). The sensitivity (`atolSens`/`rtolSens`) tolerances match the main solve (the outer gradient and covariance are built from them); the steady-state (`ssAtol`/`ssRtol`) tolerances run one order looser. Keying the optimizer to the same `10^-sigdig` means it converges to exactly the precision the solve supports. At the default `sigdig = 3` this is `atol = 1e-6`, `rtol = 1e-3`.

`sigdigTable` Significant digits in the final output table. If not specified ('NULL'), it defaults to 'sigdig'.

`boundedTransform`
When 'TRUE' (default), bounded parameters are transformed for unbounded optimization methods and back-transformed for final estimates. 'FALSE' optimizes on the original scale with bounds passed to the optimizer. 'NA' transforms for optimization but skips the final back-transform.

`...` Ignored parameters

Value

bobqya control structure

Author(s)

Matthew L. Fidler

Examples

```
# A logit regression example with emax model

dsn <- data.frame(i=1:1000)
dsn$time <- exp(rnorm(1000))
dsn$DV=rbinom(1000,1,exp(-1+dsn$time)/(1+exp(-1+dsn$time)))

mod <- function() {
 ini({
   E0 <- 0.5
   Em <- 0.5
   E50 <- 2
   g <- fix(2)
 })
 model({
   v <- E0+Em*time^g/(E50^g+time^g)
   ll(bin) ~ DV * v - log(1 + exp(v))
 })
}

fit2 <- nlmixr(mod, dsn, est="n1qn1")

print(fit2)

# you can also get the nlm output with fit2$n1qn1

fit2$n1qn1

# The nlm control has been modified slightly to include
# extra components and name the parameters
```

---

newuoaControl *Control for newuoa estimation method in nlmixr2*

Description

Control for newuoa estimation method in nlmixr2

Usage

```
newuoaControl(
  npt = NULL,
  rhobeg = NULL,
  rhoend = NULL,
  iprint = 0L,
  maxfun = 100000L,
  returnNewuoa = FALSE,
  stickyRecalcN = 4,
  maxOdeRecalc = 5,
  odeRecalcFactor = 10^(0.5),
  indTolRelax = TRUE,
  useColor = NULL,
  printNcol = NULL,
  print = 1L,
  normType = c("rescale2", "mean", "rescale", "std", "len", "constant"),
  scaleType = c("nlmixr2", "norm", "mult", "multAdd"),
  scaleCmax = 1e+05,
  scaleCmin = 1e-05,
  scaleC = NULL,
  scaleTo = 1,
  rxControl = NULL,
  optExpression = TRUE,
  sumProd = FALSE,
  literalFix = TRUE,
  literalFixRes = TRUE,
  addProp = c("combined2", "combined1"),
  calcTables = TRUE,
  compress = FALSE,
  covMethod = c("r", ""),
  adjObf = TRUE,
  ci = 0.95,
  sigdig = 3,
  sigdigTable = NULL,
  boundedTransform = TRUE,
  eventSens = c("jump", "fd"),
  ...
)
```

Arguments

npt	Number of points for bobyqa's quadratic approximation to the objective; must be in '[n+2, (n+1)(n+2)/2]'. Defaults to '2*n + 1'. (bobyqa)
rhobeg	Initial trust region radius for the bobyqa outer optimizer (with 'rhoend', must satisfy '0 < rhoend < rhobeg'). Default '0.2' (20 'abs(upper-lower)/2'. (bobyqa)
rhoend	Final trust region radius. If not defined, '10^(-sigdig)' is used. (bobyqa)
iprint	Controls amount of printing ('0'=none, '1'=start/end only, '2'=each new rho, '3'=every function evaluation, '>3'=every 'iprint' evaluations). Default '0'.

`maxfun` The maximum allowed number of function evaluations. If this is exceeded, the method will terminate.

`returnNewuoa` return the newuoa output instead of the nlmixr2 fit

`stickyRecalcN` The number of bad ODE solves before reducing the atol/rtol for the rest of the problem.

`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

`indTolRelax` When 'TRUE' (default), only subjects whose ODE solve produced NaN/Inf have their tolerances relaxed, and the relaxed tolerance persists across optimizer calls (sticky). When 'FALSE', all subjects have their tolerances relaxed on each retry and tolerances are reset afterward.

`useColor` Logical (or 'NULL') emit ANSI bold/color escapes in the iteration print. 'NULL' (default) defers to [crayon::has_color()].

`printNcol` Integer (or 'NULL') parameter columns per row before wrapping. 'NULL' (default) uses 'floor((getOption("width") - 23) / 12)'.

`print` Either a scalar print-frequency ('0' = suppress, '1' (default) = every evaluation, 'N' = every Nth), OR a pre-built [iterPrintControl()] object. Equivalent to 'iterPrintControl(every = print, ncol = printNcol, useColor = useColor)'.

`normType` Parameter normalization/scaling used to get scaled initial values for `scaleType`, of the form `Vscaled = (Vunscaled-C1)/C2` (see Feature Scaling; `rescale2` follows the OptdesX manual): `"rescale2"` scales all parameters to (-1, 1); `"rescale"` (min-max) scales to (0, 1); `"mean"` centers on the mean with range (0, 1); `"std"` standardizes by mean/sd; `"len"` scales to unit (Euclidean) length; `"constant"` performs no normalization (`C1=0`, `C2=1`).

`scaleType` The scaling scheme for nlmixr2: `"nlmixr2"` (default) scales as `(current-init)*scaleC[i] + scaleTo`, with `scaleTo` from `normType` and scales from `scaleC`; `"norm"` uses the simple scaling from `normType`; `"mult"` scales multiplicatively as `current/init*scaleTo`; `"multAdd"` scales linearly (`(current-init)+scaleTo`) for parameters in an exponential block (e.g. `exp(theta)`) and multiplicatively otherwise.

`scaleCmax` Maximum value of the scaleC to prevent overflow.

`scaleCmin` Minimum value of the scaleC to prevent underflow.

`scaleC` Scaling constant used with `scaleType="nlmixr2"`; when not specified, chosen by parameter type to keep gradient sizes similar on a log scale: '1' for exp()-transformed/power/boxCox/ yeoJohnson parameters, '0.5*abs(est)' for additive/proportional/lognormal error parameters, 'abs(1/digamma(est+1))' for factorials, and 'log(abs(est))*abs(est)' for log-scale parameters. May be set explicitly per parameter if these defaults don't apply well.

`scaleTo` Scale the initial parameter estimate to this value. By default this is 1. When zero or below, no scaling is performed.

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

`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`.

`literalFix` boolean, substitute fixed population values as literals and re-adjust ui and parameter estimates after optimization; Default is 'TRUE'.

`literalFixRes` boolean, substitute fixed population values as literals and re-adjust ui and parameter estimates after optimization; Default is 'TRUE'.

`addProp` Type of additive-plus-proportional error: '"combined1"', where standard deviations add:

$$y = f + (a + b \times f^c) \times \varepsilon$$

; or '"combined2"', where variances add:

$$y = f + \sqrt{a^2 + b^2 \times f^{2 \times c}} \times \varepsilon$$

. Here y = observed, f = predicted, a = additive sd, b = proportional/power sd, c = power exponent (1 in the proportional case).

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

`compress` Should the object have compressed items

`covMethod` Method for calculating the covariance. `"r"` (the default) uses nlmixr2's `nlmixr2Hess()` Hessian; `""` skips the covariance step.

`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. One value drives, with a single consistent formula, the inner/outer optimizer convergence tolerance (`10^-sigdig`), the boundary check tolerance (`5*10^(-sigdig+1)`), and the ODE solver tolerances: the `rtol` exponent IS `sigdig` and `atol` sits three orders below, so `rtol = 10^-sigdig`, `atol = 10^(-sigdig-3)` for every solver (stiff, non-stiff or auto-switching). The sensitivity (`atolSens`/`rtolSens`) tolerances match the main solve (the outer gradient and covariance are built from them); the steady-state (`ssAtol`/`ssRtol`) tolerances run one order looser. Keying the optimizer to the same `10^-sigdig` means it converges to exactly the precision the solve supports. At the default `sigdig = 3` this is `atol = 1e-6`, `rtol = 1e-3`.

`sigdigTable` Significant digits in the final output table. If not specified ('NULL'), it defaults to 'sigdig'.

`boundedTransform`

When 'TRUE' (default), bounded parameters are transformed for unbounded optimization methods and back-transformed for final estimates. 'FALSE' optimizes on the original scale with bounds passed to the optimizer. 'NA' transforms for optimization but skips the final back-transform.

eventSens	Controls how dosing/event-parameter (‘alag‘, ‘F‘, ‘rate‘, ‘dur‘) sensitivities are computed for THETA/ETA gradients: ‘"jump"‘ (default) uses rxode2’s analytic event sensitivities; ‘"fd"‘ uses the legacy finite-difference behavior. Also gates the analytic moving-boundary correction for a modeled ‘alag()‘/‘f()‘ on a ‘linCmt()‘ compartment; set ‘"fd"‘ if that model infuses a dose into the lagged/scaled compartment (rxode2/rxode2#1236), or if the regimen also doses an *unlagged/unscaled* compartment alongside the lagged/scaled one – a common design for estimating ‘f()‘ from paired IV+oral data (rxode2/rxode2#1237).
...	Ignored parameters

Value

newuoa control structure

Author(s)

Matthew L. Fidler

Examples

```
# A logit regression example with emax model

dsn <- data.frame(i=1:1000)
dsn$time <- exp(rnorm(1000))
dsn$DV=rbinom(1000,1,exp(-1+dsn$time)/(1+exp(-1+dsn$time)))

mod <- function() {
 ini({
   E0 <- 0.5
   Em <- 0.5
   E50 <- 2
   g <- fix(2)
 })
 model({
   v <- E0+Em*time^g/(E50^g+time^g)
   ll(bin) ~ DV * v - log(1 + exp(v))
 })
}

fit2 <- nlmixr(mod, dsn, est="newuoa")

print(fit2)

# you can also get the nlm output with

fit2$newuoa

# The nlm control has been modified slightly to include
# extra components and name the parameters
```

nlmControl *nlmixr2 defaults controls for nlm*

Description

nlmixr2 defaults controls for nlm

Usage

```
nlmControl(
  typsize = NULL,
  fscale = 1,
  print.level = 0,
  ndigit = NULL,
  gradtol = NULL,
  stepmax = NULL,
  steptol = NULL,
  iterlim = 10000,
  check.analyticals = FALSE,
  returnNlm = FALSE,
  solveType = c("hessian", "grad", "fun"),
  stickyRecalcN = 4,
  maxOdeRecalc = 5,
  odeRecalcFactor = 10^(0.5),
  indTolRelax = TRUE,
  eventType = c("central", "forward"),
  shiErr = (.Machine$double.eps)^(1/3),
  shi21maxFD = 20L,
  optimHessType = c("central", "forward"),
  hessErr = (.Machine$double.eps)^(1/3),
  shi21maxHess = 20L,
  censOption = c("gauss", "laplace"),
  eventSens = c("jump", "fd"),
  linCmtSensCarry = c("auto", "none"),
  sensMethod = c("default", "forward"),
  useColor = NULL,
  printNcol = NULL,
  print = 1L,
  normType = c("rescale2", "mean", "rescale", "std", "len", "constant"),
  scaleType = c("nlmixr2", "norm", "mult", "multAdd"),
  scaleCmax = 1e+05,
  scaleCmin = 1e-05,
  scaleC = NULL,
  scaleTo = 1,
  gradTo = 1,
  rxControl = NULL,
  optExpression = TRUE,
```

```
  sumProd = FALSE,
  literalFix = TRUE,
  literalFixRes = TRUE,
  addProp = c("combined2", "combined1"),
  calcTables = TRUE,
  compress = FALSE,
  covMethod = c("r", "nlm", ""),
  adjObf = TRUE,
  ci = 0.95,
  sigdig = 3,
  sigdigTable = NULL,
  boundedTransform = TRUE,
  ...
)
```

Arguments

typsize	an estimate of the size of each parameter at the minimum.
fscale	an estimate of the size of f at the minimum.
print.level	this argument determines the level of printing which is done during the minimization process. The default value of 0 means that no printing occurs, a value of 1 means that initial and final details are printed and a value of 2 means that full tracing information is printed.
ndigit	the number of significant digits in the function f.
gradtol	a positive scalar giving the tolerance at which the scaled gradient is considered close enough to zero to terminate the algorithm. The scaled gradient is a measure of the relative change in f in each direction p[i] divided by the relative change in p[i].
stepmax	a positive scalar which gives the maximum allowable scaled step length. stepmax is used to prevent steps which would cause the optimization function to overflow, to prevent the algorithm from leaving the area of interest in parameter space, or to detect divergence in the algorithm. stepmax would be chosen small enough to prevent the first two of these occurrences, but should be larger than any anticipated reasonable step.
steptol	A positive scalar providing the minimum allowable relative step length.
iterlim	a positive integer specifying the maximum number of iterations to be performed before the program is terminated.
check.analyticals	a logical scalar specifying whether the analytic gradients and Hessians, if they are supplied, should be checked against numerical derivatives at the initial parameter values. This can help detect incorrectly formulated gradients or Hessians.
returnNlm	is a logical that allows a return of the ‘nlm‘ object
solveType	controls whether ‘nlm‘ uses nlmixr2’s analytical gradients (event-related parameters like lag time/duration/rate/F use Shi2021 finite differences instead): ‘"hessian"‘ builds a Hessian from the analytical gradient via finite differences,

‘"gradient"‘ supplies the gradient and lets ‘nlm‘ compute the finite-difference Hessian, and ‘"fun"‘ lets ‘nlm‘ compute both by finite differences.

`stickyRecalcN` The number of bad ODE solves before reducing the atol/rtol for the rest of the problem.

`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

`indTolRelax` When ‘TRUE‘ (default), only subjects whose ODE solve produced NaN/Inf have their tolerances relaxed, and the relaxed tolerance persists across optimizer calls (sticky). When ‘FALSE‘, all subjects have their tolerances relaxed on each retry and tolerances are reset afterward.

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

`shiErr` This represents the epsilon when optimizing the ideal step size for numeric differentiation using the Shi2021 method

`shi21maxFD` The maximum number of steps for the optimization of the forward difference step size when using dosing events (lag time, modeled duration/rate and bioavailability)

`optimHessType` Hessian type for numeric-difference individual Hessians in generalized log-likelihood estimation: "central" (matches R’s ‘optimHess()‘, default) or "forward" (faster).

`hessErr` This represents the epsilon when optimizing the Hessian step size using the Shi2021 method.

`shi21maxHess` Maximum number of times to optimize the best step size for the hessian calculation

`censOption` Treatment of the second derivative for censored (M2/M3/M4/BLQ) observations in the FOCEI family. `"gauss"` (the default) keeps the historic uncensored Gauss-Newton curvature, matching common PMx tools; `"laplace"` uses the exact censored second derivative of the objective (a proper Laplace inner Hessian and analytic covariance). Accepted by `saemControl`/`nlmControl` for a uniform interface but inert there – SAEM (stochastic EM) has no Laplace inner Hessian, and NLM uses a finite-difference Hessian that already reflects censoring exactly.

`eventSens` Controls how dosing/event-parameter (‘alag‘, ‘F‘, ‘rate‘, ‘dur‘) sensitivities are computed for THETA/ETA gradients: ‘"jump"‘ (default) uses rxode2’s analytic event sensitivities; ‘"fd"‘ uses the legacy finite-difference behavior. Also gates the analytic moving-boundary correction for a modeled ‘alag()‘/‘f()‘ on a ‘linCmt()‘ compartment; set ‘"fd"‘ if that model infuses a dose into the lagged/scaled compartment (rxode2/rxode2#1236), or if the regimen also doses an *unlagged/unscaled* compartment alongside the lagged/scaled one – a common design for estimating ‘f()‘ from paired IV+oral data (rxode2/rxode2#1237).

`linCmtSensCarry`
use the linCmt() sensitivity carry for a theta on a covariate-driven linCmt() parameter ("auto", the default) or keep the standard gradient ("none"); see ‘foceiControl()‘’s argument of the same name.

`sensMethod` Method used to compute the ODE parameter sensitivities. ‘"forward"‘ uses the classic variational (forward) sensitivity ODEs; ‘"default"‘ is the same thing.

`useColor` Logical (or ‘NULL‘) emit ANSI bold/color escapes in the iteration print. ‘NULL‘ (default) defers to [crayon::has_color()].

`printNcol` Integer (or ‘NULL‘) parameter columns per row before wrapping. ‘NULL‘ (default) uses ‘floor((getOption("width") - 23) / 12)‘.

`print` Either a scalar print-frequency (‘0‘ = suppress, ‘1‘ (default) = every evaluation, ‘N‘ = every Nth), OR a pre-built [iterPrintControl()] object. Equivalent to ‘iterPrintControl(every = print, ncol = printNcol, useColor = useColor)‘.

`normType` Parameter normalization/scaling used to get scaled initial values for `scaleType`, of the form `Vscaled = (Vunscaled-C1)/C2` (see Feature Scaling; `rescale2` follows the OptdesX manual): `"rescale2"` scales all parameters to (-1, 1); `"rescale"` (min-max) scales to (0, 1); `"mean"` centers on the mean with range (0, 1); `"std"` standardizes by mean/sd; `"len"` scales to unit (Euclidean) length; `"constant"` performs no normalization (`C1=0`, `C2=1`).

`scaleType` The scaling scheme for nlmixr2: `"nlmixr2"` (default) scales as `(current-init)*scaleC[i] + scaleTo`, with `scaleTo` from `normType` and scales from `scaleC`; `"norm"` uses the simple scaling from `normType`; `"mult"` scales multiplicatively as `current/init*scaleTo`; `"multAdd"` scales linearly (`(current-init)+scaleTo`) for parameters in an exponential block (e.g. `exp(theta)`) and multiplicatively otherwise.

`scaleCmax` Maximum value of the scaleC to prevent overflow.

`scaleCmin` Minimum value of the scaleC to prevent underflow.

`scaleC` Scaling constant used with `scaleType="nlmixr2"`; when not specified, chosen by parameter type to keep gradient sizes similar on a log scale: ‘1‘ for exp()-transformed/power/boxCox/ yeoJohnson parameters, ‘0.5*abs(est)‘ for additive/proportional/ lognormal error parameters, ‘abs(1/digamma(est+1))‘ for factorials, and ‘log(abs(est))*abs(est)‘ for log-scale parameters. May be set explicitly per parameter if these defaults don’t apply well.

`scaleTo` Scale the initial parameter estimate to this value. By default this is 1. When zero or below, no scaling is performed.

`gradTo` this is the factor that the gradient is scaled to before optimizing. This only works with scaleType="nlmixr2".

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

`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`.

`literalFix` boolean, substitute fixed population values as literals and re-adjust ui and parameter estimates after optimization; Default is ‘TRUE‘.

`literalFixRes` boolean, substitute fixed population values as literals and re-adjust ui and parameter estimates after optimization; Default is ‘TRUE‘.

addProp — Type of additive-plus-proportional error: '"combined1"', where standard deviations add:

$$y = f + (a + b \times f^c) \times \varepsilon$$

; or '"combined2"', where variances add:

$$y = f + \sqrt{a^2 + b^2 \times f^{2 \times c}} \times \varepsilon$$

. Here y = observed, f = predicted, a = additive sd, b = proportional/power sd, c = power exponent (1 in the proportional case).

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

compress — Should the object have compressed items

covMethod — "r" uses nlmixr2's 'nlmixr2Hess()' for the hessian, or "nlm" uses a Hessian built from nlmixr2's analytical gradient (central differences over 'nlmixr2Gill83()' steps) with 'solveType' '"hessian"' or '"grad"', and 'stats::nlm(.., hessian=TRUE)''s with '"fun"'; defaults to "nlm" when using nlmixr2's hessian/gradient for solving.

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. One value drives, with a single consistent formula, the inner/outer optimizer convergence tolerance (10^-sigdig), the boundary check tolerance (5*10^(-sigdig+1)), and the ODE solver tolerances: the rtol exponent IS sigdig and atol sits three orders below, so rtol = 10^-sigdig, atol = 10^(-sigdig-3) for every solver (stiff, non-stiff or auto-switching). The sensitivity (atolSens/rtolSens) tolerances match the main solve (the outer gradient and covariance are built from them); the steady-state (ssAtol/ssRtol) tolerances run one order looser. Keying the optimizer to the same 10^-sigdig means it converges to exactly the precision the solve supports. At the default sigdig = 3 this is atol = 1e-6, rtol = 1e-3.

sigdigTable — Significant digits in the final output table. If not specified ('NULL'), it defaults to 'sigdig'.

boundedTransform — When 'TRUE' (default), bounded parameters are transformed for unbounded optimization methods and back-transformed for final estimates. 'FALSE' optimizes on the original scale with bounds passed to the optimizer. 'NA' transforms for optimization but skips the final back-transform.

... — additional arguments to be passed to f.

Value

nlm control object

Author(s)

Matthew L. Fidler

Examples

```
# A logit regression example with emax model

dsn <- data.frame(i = 1:1000)
dsn$time <- exp(rnorm(1000))
dsn$DV <- rbinom(1000, 1, exp(-1 + dsn$time) / (1 + exp(-1 + dsn$time)))

mod <- function() {
  ini({
    E0 <- 0.5
    Em <- 0.5
    E50 <- 2
    g <- fix(2)
  })
  model({
    v <- E0 + Em * time^g / (E50^g + time^g)
    ll(bin) ~ DV * v - log(1 + exp(v))
  })
}

fit2 <- nlmixr(mod, dsn, est = "nlm")

print(fit2)

# you can also get the nlm output with fit2$nlm

fit2$nlm

# The nlm control has been modified slightly to include
# extra components and name the parameters
```

---

nlminbControl *nlmixr2 nlminb defaults*

Description

nlmixr2 nlminb defaults

Usage

```
nlminbControl(
  eval.max = 200,
  iter.max = 150,
  trace = 0,
  abs.tol = 0,
  rel.tol = NULL,
  x.tol = NULL,
  xf.tol = 2.2e-14,
```

```
  step.min = 1,
  step.max = 1,
  sing.tol = rel.tol,
  scale = 1,
  scale.init = NULL,
  diff.g = NULL,
  rxControl = NULL,
  optExpression = TRUE,
  sumProd = FALSE,
  literalFix = TRUE,
  literalFixRes = TRUE,
  returnNlminb = FALSE,
  solveType = c("hessian", "grad", "fun"),
  stickyRecalcN = 4,
  maxOdeRecalc = 5,
  odeRecalcFactor = 10^(0.5),
  indTolRelax = TRUE,
  eventType = c("central", "forward"),
  shiErr = (.Machine$double.eps)^(1/3),
  shi21maxFD = 20L,
  optimHessType = c("central", "forward"),
  hessErr = (.Machine$double.eps)^(1/3),
  shi21maxHess = 20L,
  useColor = NULL,
  printNcol = NULL,
  print = 1L,
  normType = c("rescale2", "mean", "rescale", "std", "len", "constant"),
  scaleType = c("nlmixr2", "norm", "mult", "multAdd"),
  scaleCmax = 1e+05,
  scaleCmin = 1e-05,
  scaleC = NULL,
  scaleTo = 1,
  gradTo = 1,
  addProp = c("combined2", "combined1"),
  eventSens = c("jump", "fd"),
  sensMethod = c("default", "forward"),
  calcTables = TRUE,
  compress = TRUE,
  covMethod = c("r", "nlminb", ""),
  adjObf = TRUE,
  ci = 0.95,
  sigdig = 3,
  sigdigTable = NULL,
  ...
)
```

Arguments

eval.max	Maximum number of evaluations of the objective function allowed. Defaults to 200.
iter.max	Maximum number of iterations allowed. Defaults to 150.
trace	The value of the objective function and the parameters is printed every trace'th iteration. When 0 no trace information is to be printed
abs.tol	Absolute tolerance. Defaults to 0 so the absolute convergence test is not used. If the objective function is known to be non-negative, the previous default of '1e-20' would be more appropriate
rel.tol	Relative tolerance. When 'NULL' (default) it is derived from 'sigdig' the way 'foceiControl()' does ('10^(-sigdig)').
x.tol	X tolerance. When 'NULL' (default) it is derived from 'sigdig' ('10^(-sigdig)').
xf.tol	false convergence tolerance. Defaults to '2.2e-14'.
step.min	Minimum step size. Default to '1.'.
step.max	Maximum step size. Default to '1.'.
sing.tol	singular convergence tolerance; defaults to 'rel.tol;.
scale	See PORT documentation (or leave alone).
scale.init	... probably need to check PORT documentation
diff.g	an estimated bound on the relative error in the objective function value
rxControl	'rxode2' ODE solving options during fitting, created with 'rxControl()'
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.
literalFix	boolean, substitute fixed population values as literals and re-adjust ui and parameter estimates after optimization; Default is 'TRUE'.
literalFixRes	boolean, substitute fixed population values as literals and re-adjust ui and parameter estimates after optimization; Default is 'TRUE'.
returnNlminb	logical; when TRUE this will return the nlminb result instead of the nlmixr2 fit object
solveType	controls whether 'nlm' uses nlmixr2's analytical gradients (event-related parameters like lag time/duration/rate/F use Shi2021 finite differences instead): '"hessian"' builds a Hessian from the analytical gradient via finite differences, '"gradient"' supplies the gradient and lets 'nlm' compute the finite-difference Hessian, and '"fun"' lets 'nlm' compute both by finite differences.
stickyRecalcN	The number of bad ODE solves before reducing the atol/rtol for the rest of the problem.
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

`indTolRelax`
: When 'TRUE' (default), only subjects whose ODE solve produced NaN/Inf have their tolerances relaxed, and the relaxed tolerance persists across optimizer calls (sticky). When 'FALSE', all subjects have their tolerances relaxed on each retry and tolerances are reset afterward.

`eventType`
: Event gradient type for dosing events; Can be "central" or "forward"

`shiErr`
: This represents the epsilon when optimizing the ideal step size for numeric differentiation using the Shi2021 method

`shi21maxFD`
: The maximum number of steps for the optimization of the forward difference step size when using dosing events (lag time, modeled duration/rate and bioavailability)

`optimHessType`
: Hessian type for numeric-difference individual Hessians in generalized log-likelihood estimation: "central" (matches R's 'optimHess()', default) or "forward" (faster).

`hessErr`
: This represents the epsilon when optimizing the Hessian step size using the Shi2021 method.

`shi21maxHess`
: Maximum number of times to optimize the best step size for the hessian calculation

`useColor`
: Logical (or 'NULL') emit ANSI bold/color escapes in the iteration print. 'NULL' (default) defers to [crayon::has_color()].

`printNcol`
: Integer (or 'NULL') parameter columns per row before wrapping. 'NULL' (default) uses 'floor((getOption("width") - 23) / 12)'.

`print`
: Either a scalar print-frequency ('0' = suppress, '1' (default) = every evaluation, 'N' = every Nth), OR a pre-built [iterPrintControl()] object. Equivalent to 'iterPrintControl(every = print, ncol = printNcol, useColor = useColor)'.

`normType`
: Parameter normalization/scaling used to get scaled initial values for `scaleType`, of the form `Vscaled = (Vunscaled-C1)/C2` (see Feature Scaling; `rescale2` follows the OptdesX manual): `"rescale2"` scales all parameters to (-1, 1); `"rescale"` (min-max) scales to (0, 1); `"mean"` centers on the mean with range (0, 1); `"std"` standardizes by mean/sd; `"len"` scales to unit (Euclidean) length; `"constant"` performs no normalization (`C1=0`, `C2=1`).

`scaleType`
: The scaling scheme for nlmixr2: `"nlmixr2"` (default) scales as `(current-init)*scaleC[i] + scaleTo`, with `scaleTo` from `normType` and scales from `scaleC`; `"norm"` uses the simple scaling from `normType`; `"mult"` scales multiplicatively as `current/init*scaleTo`; `"multAdd"` scales linearly (`(current-init)+scaleTo`) for parameters in an exponential block (e.g. `exp(theta)`) and multiplicatively otherwise.

`scaleCmax`
: Maximum value of the scaleC to prevent overflow.

`scaleCmin`
: Minimum value of the scaleC to prevent underflow.

`scaleC`
: Scaling constant used with `scaleType="nlmixr2"`; when not specified, chosen by parameter type to keep gradient sizes similar on a log scale: '1' for exp()-transformed/power/boxCox/ yeoJohnson parameters, '0.5*abs(est)' for additive/proportional/ lognormal error parameters, 'abs(1/digamma(est+1))' for factorials, and 'log(abs(est))*abs(est)' for log-scale parameters. May be set explicitly per parameter if these defaults don't apply well.

`scaleTo` Scale the initial parameter estimate to this value. By default this is 1. When zero or below, no scaling is performed.

`gradTo` this is the factor that the gradient is scaled to before optimizing. This only works with scaleType="nlmixr2".

`addProp` Type of additive-plus-proportional error: ‘"combined1"‘, where standard deviations add:

$$y = f + (a + b \times f^c) \times \varepsilon$$

; or ‘"combined2"‘, where variances add:

$$y = f + \sqrt{a^2 + b^2 \times f^{2 \times c}} \times \varepsilon$$

. Here y = observed, f = predicted, a = additive sd, b = proportional/power sd, c = power exponent (1 in the proportional case).

`eventSens` Controls how dosing/event-parameter (‘alag‘, ‘F‘, ‘rate‘, ‘dur‘) sensitivities are computed for THETA/ETA gradients: ‘"jump"‘ (default) uses rxode2’s analytic event sensitivities; ‘"fd"‘ uses the legacy finite-difference behavior. Also gates the analytic moving-boundary correction for a modeled ‘alag()‘/‘f()‘ on a ‘linCmt()‘ compartment; set ‘"fd"‘ if that model infuses a dose into the lagged/scaled compartment (rxode2/rxode2#1236), or if the regimen also doses an *unlagged/unscaled* compartment alongside the lagged/scaled one – a common design for estimating ‘f()‘ from paired IV+oral data (rxode2/rxode2#1237).

`sensMethod` Method used to compute the ODE parameter sensitivities. ‘"forward"‘ uses the classic variational (forward) sensitivity ODEs; ‘"default"‘ is the same thing.

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

`compress` Should the object have compressed items

`covMethod` Method for calculating the covariance. `"r"` uses nlmixr2’s `nlmixr2Hess()` Hessian; `"nlminb"` uses central differences of the analytic gradient at the final estimates, as `nlmControl(covMethod = "nlm")` does; it needs `solveType = "hessian"` or `"grad"`. The default is `"nlminb"` with `solveType = "hessian"` and `"r"` otherwise. `""` skips the covariance step.

`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. One value drives, with a single consistent formula, the inner/outer optimizer convergence tolerance (`10^-sigdig`), the boundary check tolerance (`5*10^(-sigdig+1)`), and the ODE solver tolerances: the `rtol` exponent IS `sigdig` and `atol` sits three orders below, so `rtol = 10^-sigdig`, `atol = 10^(-sigdig-3)` for every solver (stiff, non-stiff or auto-switching). The sensitivity (`atolSens`/`rtolSens`) tolerances match the main solve (the outer gradient and covariance are built from them); the steady-state (`ssAtol`/`ssRtol`) tolerances run one order looser. Keying the optimizer to the same `10^-sigdig` means it converges to exactly the precision the solve supports. At the default `sigdig = 3` this is `atol = 1e-6`, `rtol = 1e-3`.

`sigdigTable` Significant digits in the final output table. If not specified (‘NULL‘), it defaults to ‘sigdig‘.

`...` Further arguments to be supplied to `objective`.

Author(s)

Matthew L. Fidler

Examples

```
# A logit regression example with emax model

dsn <- data.frame(i=1:1000)
dsn$time <- exp(rnorm(1000))
dsn$DV=rbinom(1000,1,exp(-1+dsn$time)/(1+exp(-1+dsn$time)))

mod <- function() {
 ini({
   E0 <- 0.5
   Em <- 0.5
   E50 <- 2
   g <- fix(2)
 })
 model({
   v <- E0+Em*time^g/(E50^g+time^g)
   ll(bin) ~ DV * v - log(1 + exp(v))
 })
}

fit2 <- nlmixr(mod, dsn, est="nlminb")

print(fit2)

# you can also get the nlm output with fit2$nlminb

fit2$nlminb
```

---

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()
)

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()
)

## S3 method for class 'nlmixr2FitCoreSilent'
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.

The nlmixr object has the following fields:

Field	Note	Description
censInfo		Gives the censorng information abot the fit (the type of censoring that was seend and handled in th
conditionNumber		Condition number, that is the highest divided by the lowest eigenvalue in the population covarianc
cor		Correlation matrix
cov		Variance-covariance matrix
covMethod		Method used to calculate covariance of the fixed effects
dataLloq		Gives the lloq from the dataset (average) when cesoring has occured; Requires the fit to have a tab
dataMergeFull		Full data merge with the fit output and the original dataset; Also includes nlmixrLlikObs which in
dataMergeInner		Inner data merge with the fit output and the original dataset; Also includes nlmixrLlikObs which i
dataMergeLeft		Left data merge with the fit output and the original dataset; Also includes nlmixrLlikObs which in
dataMergeRight		Right data merge with the fit output and the original dataset; Also includes nlmixrLlikObs which i
dataUloq		Gives the uloq from the dataset (average) when censoring has occured; requires the fit to have a ta
env		This is the environment where all the information for the fit is stored outside of the data-frame. It
runInfo		This returns a list of all the warnings or fit information
rxControl		Integration options used to control rxode2
scaleInfo		The scaling factors used for nlmixr2 estimation in focei; The can be changed by foceiControl(scal
seed		This is the initial seed used for saem
shrink		This is a table of shrinkages for all the individual ETAs as well as the variance shrinkage as well a
simInfo		This returns a list of all the fit information used for a traditional rxode2 simulation, which you can
table		These are the table options that were used when generating the table output (were CWRES includ
theta		Estimates for eta for each individual
time		Duration of different parts of the analysis (e.g. setup, optimization, calculation of covariance, etc.)
ui		Final estimates for the model
atol	n2r	Absolute tolerance that NONMEM specified; will be used when solving
dfObs	n2r	Degrees of freedom by observation
dfSub	n2r	Degrees of freedom by subject
etaData	n2r	Subject level IIV values
ipredAtol	n2r	Absolute tolerance difference between NONMEM and rxode2 individual predictions
ipredCompare	n2r	Data frame with ipred values
ipredRtol	n2r	Relative tolerance difference between NONMEM and rxode2 individual predictions
nonmemData	n2r	Original dataset used for NONMEM analysis
predAtol	n2r	Absolute tolerance difference between NONMEM and rxode2 population predictions
predCompare	n2r	Data frame with pred values
predRtol	n2r	Relative tolerance difference between NONMEM and rxode2 population predictions
rtol	n2r	Relative tolerance that NONMEM specified; will be used when solving
sigma	n2r	Error model matrix
ssAtol	n2r	Steady state absolute tolerance that NONMEM specified; will be used for solving.
ssRtol	n2r	Steady state relative tolerance that NONMEM specified will be used for solving
thetaMat	n2r	Covariance Matrix (matches rxSolve(thetaMat=)

n2r - These fields are added when a NONMEM model is imported using `nonmem2rx()`

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")
```

`nlmixr2AllEst` *Show all the current estimation methods*

Description

Show all the current estimation methods

Usage

```
nlmixr2AllEst()
```

Value

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

Examples

```
nlmixr2AllEst()
```

`nlmixr2AllEstType` *Tagged list of the available nlmixr2 estimation methods*

Description

Returns the built-in (and any attribute-tagged third-party) ‘est=‘ methods grouped by their estimation category, as used when an unsupported method is requested.

Usage

```
nlmixr2AllEstType()
```

Value

data.frame with columns ‘est‘, ‘type‘ and ‘description‘

Examples

```
nlmixr2AllEstType()
```

`nlmixr2AugPredSolve` *Augmented Prediction for nlmixr2 fit*

Description

Augmented Prediction for nlmixr2 fit

Usage

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

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

Arguments

`fit`	Nlmixr2 fit object
`covsInterpolation`	covariate interpolation; by default the one the fit was estimated with
`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

```
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 - ‘$covLvl‘ – Level information for items to convert to factor - ‘$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 - ‘$adjObf‘ Should the objective function value be adjusted - ‘$objective‘ objective function value - ‘$extra‘ Extra print information - ‘$method‘ Estimation method (for printing) - ‘$omega‘ Omega matrix (final estimate); with ‘$fullTheta‘ it is written into the reported ui, and the ui passed in is kept as ‘$iniDf0‘ unless an ‘nlmixr2()‘ run supplies the original model - ‘$theta‘ Is a theta data frame - ‘$model‘ a list of model information for table generation. Needs a ‘predOnly‘ model - ‘$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.agq` *Generic for nlmixr2 estimation methods*

Description

Generic for nlmixr2 estimation methods

Usage

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

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

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

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

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

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

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

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

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

## S3 method for class 'focef'
```

```
nlmixr2Est(env, ...)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

## S3 method for class 'ifagq'
```

```
nlmixr2Est(env, ...)

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

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

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

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

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

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

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

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

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

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

## S3 method for class 'lbfgsb3c'
nlmixr2Est(env, ...)

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

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

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

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

## S3 method for class 'mlaplace'
```

```
nlmixr2Est(env, ...)

## S3 method for class 'n1qn1'
nlmixr2Est(env, ...)

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

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

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

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

nlmixr2Est(env, ...)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

## S3 method for class 'vae'
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

A prior distribution given in the 'ini()' block is never silently ignored. Before dispatching, 'nlmixr2Est()' refuses any prior the method cannot use, so a method that does not handle priors gets that for free and a model carrying one fails with an explanation instead of being fit to something other than what it says.

A method says what it supports with an attribute on itself:

"' attr(nlmixr2Est.myMethod, "nlmixr2Priors") <- "general" "'

- '"none"', which is also what an absent attribute means – the method cannot use priors at all - '"theta"' – priors on population parameters only ('dnorm()', 'dcauchy()', 'stdNormal()' and the 'multiNormal()' family among thetas); anything referencing an omega element is refused. No FOCEi-family method uses this level – they declare '"general"'. - '"general"' – everything the shared kernel's '"general"' method covers: the above, plus a normal prior directly on an omega element and a textbook inverse-Wishart on an omega block. This is what 'est="focei"' and the rest of its family declare. - '"nwpri"' – normal priors and degrees of freedom on an omega block ('invWishart(4)'), evaluated the way a NONMEM NWPRI model works; a normal prior on the omega values themselves is refused - '"tnpri"' – normal priors including directly on omega elements (Monolix's/NONMEM's own-estimation joint-normal assumption); 'dcauchy()' and 'invWishart()' are refused - '"all"' – the method handles everything, so nothing is checked

Value

nlmixr2 fit object

Author(s)

Matthew Fidler

`nlmixr2fix` *Try to fix a nlmixr2 fit*

Description

Re-evaluates the model function against the current version of rxode2, for fits created with an older nlmixr2/rxode2 version.

Usage

```
nlmixr2fix(fit)
```

Arguments

fit	nlmixr2 fit object from a different version of nlmixr2.

Value

A nlmixr2 fit that has been (possibly) adjusted to work with the current version of nlmixr2.

Author(s)

Matthew L. Fidler

Examples

```
## Not run:
  # requires the qs package to read an older nlmixr2 v3 fit (qs is no
  # longer on CRAN); regenerates the rxode2 model so it works again
  # fit <- readRDS(system.file("testfit_nlmixr3.rds", package = "nlmixr2est"))
  # fit <- try(nlmixr2fix(fit))
  # if (!inherits(fit, "try-error")) rxSolve(fit)

## End(Not run)
```

---

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	Max steps to determine the optimal forward/central difference step size per parameter (Gill 1983). '0' = no 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.

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: the parameter is left out by which, or gillK = 0, which determines no interval and reports the one the search starts from as hf

- "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

```
## These are taken from the numDeriv's 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)
exact <- 10*cos(10*x) + exp(-x)
c(numd1$df, exact, (numd1$df - exact)/exact)

x <- c(1:10)
numd1 <- nlmixr2Gill83(func1, x)
exact <- 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)
exact <- 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 procedures 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
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.

Details

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

A parameter that 'which' (passed on to nlmixr2Gill83) leaves out of the step search is differenced with the interval the search would start from, the one 'gillK = 0' gives every parameter.

Value

Hessian matrix based on Gill83

Author(s)

Matthew Fidler

See Also

nlmixr2Gill83, optimHess

Examples

```
 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)
```

---

nlmixr2Keywords *A list and description of the fields in the nlmxir2 object*

Description

A list and description of the fields in the nlmxir2 object

Usage

```
nlmixr2Keywords
```

Format

A data frame with 2 columns and 40 or more rows

Field Name of the field in the nlmixr2 object

Note Note regarding the source of the field

Description Description of the information in the field

---

nlmixr2Logo *Messages the nlmixr2 logo...*

Description

Messages the nlmixr2 logo...

Usage

```
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

```
nlmixr2NlmeControl(
  maxIter = 100,
  pnlsMaxIter = 100,
  msMaxIter = 100,
  minScale = 0.001,
  tolerance = NULL,
  niterEM = 25,
  pnlsTol = NULL,
  msTol = NULL,
  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,
  literalFix = 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 = 3,
  sigdigTable = NULL,
  muRefCovAlg = TRUE,
  eventSens = c("jump", "fd"),
  print = NULL,
  covMethod = c("nlme", "analytic", "r,s", "r", "s", "sa", "imp", ""),
  ...
)

nlmeControl(
  maxIter = 100,
  pnlsMaxIter = 100,
  msMaxIter = 100,
  minScale = 0.001,
  tolerance = NULL,
  niterEM = 25,
  pnlsTol = NULL,
  msTol = NULL,
  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,
  literalFix = 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 = 3,
  sigdigTable = NULL,
  muRefCovAlg = TRUE,
  eventSens = c("jump", "fd"),
```

```
  print = NULL,
  covMethod = c("nlme", "analytic", "r,s", "r", "s", "sa", "imp", ""),
  ...
)
```

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.

literalFix — boolean, substitute fixed population values as literals and re-adjust ui and parameter estimates after optimization; Default 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.

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` Type of additive-plus-proportional error: ‘"combined1"‘, where standard deviations add:

$$y = f + (a + b \times f^c) \times \varepsilon$$

; or ‘"combined2"‘, where variances add:

$$y = f + \sqrt{a^2 + b^2 \times f^{2 \times c}} \times \varepsilon$$

. Here y = observed, f = predicted, a = additive sd, b = proportional/power sd, c = power exponent (1 in the proportional case).

`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. One value drives, with a single consistent formula, the inner/outer optimizer convergence tolerance (`10^-sigdig`), the boundary check tolerance (`5*10^(-sigdig+1)`), and the ODE solver tolerances: the `rtol` exponent IS `sigdig` and `atol` sits three orders below, so `rtol = 10^-sigdig`, `atol = 10^(-sigdig-3)` for every solver (stiff, non-stiff or auto-switching). The sensitivity (`atolSens`/`rtolSens`) tolerances match the main solve (the outer gradient and covariance are built from them); the steady-state (`ssAtol`/`ssRtol`) tolerances run one order looser. Keying the optimizer to the same `10^-sigdig` means it converges to exactly the precision the solve supports. At the default `sigdig = 3` this is `atol = 1e-6`, `rtol = 1e-3`.

`sigdigTable` Significant digits in the final output table. If not specified (‘NULL‘), it defaults to ‘sigdig‘.

`muRefCovAlg` When ‘TRUE‘ (default), algebraic expressions that can be mu-referenced are internally rewritten as mu-referenced covariates and restored after optimization. Mirrors `saemControl(muRefCovAlg=)`/`nlmeControl(muRefCovAlg=)`; for `foceiControl()` only takes effect when `muModel != "none"`.

`eventSens` Controls how dosing/event-parameter (‘alag‘, ‘F‘, ‘rate‘, ‘dur‘) sensitivities are computed for THETA/ETA gradients: ‘"jump"‘ (default) uses rxode2's analytic event sensitivities; ‘"fd"‘ uses the legacy finite-difference behavior. Also gates the analytic moving-boundary correction for a modeled ‘alag()‘/‘f()‘ on a ‘linCmt()‘ compartment; set ‘"fd"‘ if that model infuses a dose into the lagged/scaled compartment (rxode2/rxode2#1236), or if the regimen also doses an *unlagged/unscaled* compartment alongside the lagged/scaled one – a common design for estimating ‘f()‘ from paired IV+oral data (rxode2/rxode2#1237).

`print` Convenience alias for the shared nlmixr ‘print‘ control. ‘nlme‘ prints progress through its own ‘verbose‘ option, so ‘print‘ maps to it: ‘print=0‘ runs quietly (‘verbose=FALSE‘) and any positive value is verbose (‘verbose=TRUE‘). When ‘print‘ is not supplied an explicit ‘verbose‘ is used as given.

covMethod	Covariance method: ‘"analytic"‘ (default) computes the focei observed-information covariance at the converged nlme estimates post-fit (falling back to the finite-difference ‘"r,s"‘ -> ‘"r"‘/‘"s"‘ chain when out of analytic scope); ‘"r,s"‘, ‘"r"‘, ‘"s"‘ request the finite-difference forms directly; ‘"nlme"‘ and ‘""‘ skip the re-compute and keep nlme’s own standard errors. When the recompute fails the ‘"nlme"‘ covariance is kept.
...	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).

Value

a nlmixr-nlme list

See Also

Other Estimation control: foceiControl(), saemControl()

Examples

```
nlmeControl()
nlmixr2NlmeControl()
```

---

nlmixr2Validate *Validate nlmixr2*

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

nlmixrAddObjectiveFunctionDataFrame *Add objective function data frame to the current objective function*

Description

Add objective function data frame to the current objective function

Usage

```
nlmixrAddObjectiveFunctionDataFrame(fit, objDf, type, etaObf = NULL)
```

Arguments

fit	nlmixr fit object
objDf	nlmixr objective function data frame which has column names "OBJF", "AIC", "BIC", "Log-likelihood" and "Condition#(Cov)" "Condition#(Cor)"
type	Objective Function Type
etaObf	Eta objective function table to add (with focei) to give focei objective function

Value

Nothing, called for 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)
 })
```

```
}

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 *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

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

Arguments

`name`	Name of the timing to be integrated
`code`	Code to be evaluated and timed
`envir`	nlmixr2 fit data, fit environment, or NULL (timing is added when the fit is finalized); supply this if called after a fit already exists

Value

Result of code

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)
 })
}
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)
```

`nlsControl` *nlmixr2 defaults controls for nls*

Description

nlmixr2 defaults controls for nls

Usage

```
nlsControl(
  maxiter = 10000,
  tol = NULL,
  minFactor = 1/1024,
  printEval = FALSE,
  warnOnly = FALSE,
  scaleOffset = 0,
  nDcentral = FALSE,
  algorithm = c("LM", "default", "plinear", "port"),
  ftol = NULL,
  ptol = NULL,
  gtol = 0,
  diag = list(),
  epsfcn = 0,
  factor = 100,
  maxfev = integer(),
  nprint = 0,
```

```
  solveType = c("grad", "fun"),
  stickyRecalcN = 4,
  maxOdeRecalc = 5,
  odeRecalcFactor = 10^(0.5),
  indTolRelax = TRUE,
  eventType = c("central", "forward"),
  shiErr = (.Machine$double.eps)^(1/3),
  shi21maxFD = 20L,
  useColor = NULL,
  printNcol = NULL,
  print = 1L,
  normType = c("rescale2", "mean", "rescale", "std", "len", "constant"),
  scaleType = c("nlmixr2", "norm", "mult", "multAdd"),
  scaleCmax = 1e+05,
  scaleCmin = 1e-05,
  scaleC = NULL,
  scaleTo = 1,
  gradTo = 1,
  trace = FALSE,
  rxControl = NULL,
  optExpression = TRUE,
  sumProd = FALSE,
  literalFix = TRUE,
  returnNls = FALSE,
  addProp = c("combined2", "combined1"),
  eventSens = c("jump", "fd"),
  linCmtSensCarry = c("auto", "none"),
  calcTables = TRUE,
  compress = TRUE,
  adjObf = TRUE,
  ci = 0.95,
  sigdig = 4,
  sigdigTable = NULL,
  boundedTransform = TRUE,
  ...
)
```

Arguments

maxiter	A positive integer specifying the maximum number of iterations allowed.
tol	A positive numeric value specifying the tolerance level for the relative offset convergence criterion.
minFactor	A positive numeric value specifying the minimum step-size factor allowed on any step in the iteration. The increment is calculated with a Gauss-Newton algorithm and successively halved until the residual sum of squares has been decreased or until the step-size factor has been reduced below this limit.
printEval	a logical specifying whether the number of evaluations (steps in the gradient direction taken each iteration) is printed.

`warnOnly` a logical specifying whether `nls()` should return instead of signalling an error in the case of termination before convergence. Termination before convergence happens upon completion of `maxiter` iterations, in the case of a singular gradient, and in the case that the step-size factor is reduced below `minFactor`.

`scaleOffset` a constant to be added to the denominator of the relative offset convergence criterion calculation to avoid a zero divide in the case where the fit of a model to data is very close. The default value of `0` keeps the legacy behaviour of `nls()`. A value such as `1` seems to work for problems of reasonable scale with very small residuals.

`nDcentral` only when *numerical* derivatives are used: `logical` indicating if *central* differences should be employed, i.e., `numericDeriv(*, central=TRUE)` be used.

`algorithm` character string specifying the algorithm to use. The default algorithm is a Gauss-Newton algorithm. Other possible values are `"plinear"` for the Golub-Pereyra algorithm for partially linear least-squares models and `"port"` for the 'nl2sol' algorithm from the Port library – see the references. Can be abbreviated.

`ftol` non-negative numeric. Termination occurs when both the actual and predicted relative reductions in the sum of squares are at most `ftol`. Therefore, `ftol` measures the relative error desired in the sum of squares.

`ptol` non-negative numeric. Termination occurs when the relative error between two consecutive iterates is at most `ptol`. Therefore, `ptol` measures the relative error desired in the approximate solution.

`gtol` non-negative numeric. Termination occurs when the cosine of the angle between result of `fn` evaluation $fvec$ and any column of the Jacobian is at most `gtol` in absolute value. Therefore, `gtol` measures the orthogonality desired between the function vector and the columns of the Jacobian.

`diag` a list or numeric vector containing positive entries that serve as multiplicative scale factors for the parameters. Length of `diag` should be equal to that of `par`. If not, user-provided `diag` is ignored and `diag` is internally set.

`epsfcn` (used if `jac` is not provided) is a numeric used in determining a suitable step for the forward-difference approximation. This approximation assumes that the relative errors in the functions are of the order of `epsfcn`. If `epsfcn` is less than the machine precision, it is assumed that the relative errors in the functions are of the order of the machine precision.

`factor` positive numeric, used in determining the initial step bound. This bound is set to the product of `factor` and the $|\texttt{diag} * \texttt{par}|$ if nonzero, or else to `factor` itself. In most cases `factor` should lie in the interval (0.1,100). 100 is a generally recommended value.

`maxfev` integer; termination occurs when the number of calls to `fn` has reached `maxfev`. Note that `nls.lm` sets the value of `maxfev` to `100*(length(par) + 1)` if `maxfev = integer()`, where `par` is the list or vector of parameters to be optimized.

`nprint` is an integer; set `nprint` to be positive to enable printing of iterates

`solveType` controls whether 'nlm' uses nlmixr2's analytical gradients (event-related parameters like lag time/duration/rate/F use Shi2021 finite differences instead): '"hessian"' builds a Hessian from the analytical gradient via finite differences, '"gradient"' supplies the gradient and lets 'nlm' compute the finite-difference Hessian, and '"fun"' lets 'nlm' compute both by finite differences.

`stickyRecalcN` The number of bad ODE solves before reducing the atol/rtol for the rest of the problem.

`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

`indTolRelax` When 'TRUE' (default), only subjects whose ODE solve produced NaN/Inf have their tolerances relaxed, and the relaxed tolerance persists across optimizer calls (sticky). When 'FALSE', all subjects have their tolerances relaxed on each retry and tolerances are reset afterward.

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

`shiErr` This represents the epsilon when optimizing the ideal step size for numeric differentiation using the Shi2021 method

`shi21maxFD` The maximum number of steps for the optimization of the forward difference step size when using dosing events (lag time, modeled duration/rate and bioavailability)

`useColor` Logical (or 'NULL') emit ANSI bold/color escapes in the iteration print. 'NULL' (default) defers to [crayon::has_color()].

`printNcol` Integer (or 'NULL') parameter columns per row before wrapping. 'NULL' (default) uses 'floor((getOption("width") - 23) / 12)'.

`print` Either a scalar print-frequency ('0' = suppress, '1' (default) = every evaluation, 'N' = every Nth), OR a pre-built [iterPrintControl()] object. Equivalent to 'iterPrintControl(every = print, ncol = printNcol, useColor = useColor)'.

`normType` Parameter normalization/scaling used to get scaled initial values for `scaleType`, of the form `Vscaled = (Vunscaled-C1)/C2` (see Feature Scaling; `rescale2` follows the OptdesX manual): `"rescale2"` scales all parameters to (-1, 1); `"rescale"` (min-max) scales to (0, 1); `"mean"` centers on the mean with range (0, 1); `"std"` standardizes by mean/sd; `"len"` scales to unit (Euclidean) length; `"constant"` performs no normalization (`C1=0`, `C2=1`).

`scaleType` The scaling scheme for nlmixr2: `"nlmixr2"` (default) scales as `(current-init)*scaleC[i] + scaleTo`, with `scaleTo` from `normType` and scales from `scaleC`; `"norm"` uses the simple scaling from `normType`; `"mult"` scales multiplicatively as `current/init*scaleTo`; `"multAdd"` scales linearly (`(current-init)+scaleTo`) for parameters in an exponential block (e.g. `exp(theta)`) and multiplicatively otherwise.

`scaleCmax` Maximum value of the scaleC to prevent overflow.

`scaleCmin` Minimum value of the scaleC to prevent underflow.

`scaleC` Scaling constant used with `scaleType="nlmixr2"`; when not specified, chosen by parameter type to keep gradient sizes similar on a log scale: '1' for exp()-transformed/power/boxCox/ yeoJohnson parameters, '0.5*abs(est)' for additive/proportional/ lognormal error parameters, 'abs(1/digamma(est+1))' for factorials, and 'log(abs(est))*abs(est)' for log-scale parameters. May be set explicitly per parameter if these defaults don't apply well.

`scaleTo` Scale the initial parameter estimate to this value. By default this is 1. When zero or below, no scaling is performed.

`gradTo` this is the factor that the gradient is scaled to before optimizing. This only works with scaleType="nlmixr2".

`trace` logical value indicating if a trace of the iteration progress should be printed. Default is `FALSE`. If `TRUE` the residual (weighted) sum-of-squares, the convergence criterion and the parameter values are printed at the conclusion of each iteration. Note that `format()` is used, so these mostly depend on `getOption("digits")`. When the `"plinear"` algorithm is used, the conditional estimates of the linear parameters are printed after the nonlinear parameters. When the `"port"` algorithm is used the objective function value printed is half the residual (weighted) sum-of-squares.

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

`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`.

`literalFix` boolean, substitute fixed population values as literals and re-adjust ui and parameter estimates after optimization; Default is ‘TRUE‘.

`returnNls` logical; when TRUE, will return the nls object instead of the nlmixr object

`addProp` Type of additive-plus-proportional error: ‘"combined1"‘, where standard deviations add:

$$y = f + (a + b \times f^c) \times \varepsilon$$

; or ‘"combined2"‘, where variances add:

$$y = f + \sqrt{a^2 + b^2 \times f^{2 \times c}} \times \varepsilon$$

. Here y = observed, f = predicted, a = additive sd, b = proportional/power sd, c = power exponent (1 in the proportional case).

`eventSens` Controls how dosing/event-parameter (‘alag‘, ‘F‘, ‘rate‘, ‘dur‘) sensitivities are computed for THETA/ETA gradients: ‘"jump"‘ (default) uses rxode2’s analytic event sensitivities; ‘"fd"‘ uses the legacy finite-difference behavior. Also gates the analytic moving-boundary correction for a modeled ‘alag()‘/‘f()‘ on a ‘linCmt()‘ compartment; set ‘"fd"‘ if that model infuses a dose into the lagged/scaled compartment (rxode2/rxode2#1236), or if the regimen also doses an *unlagged/unscaled* compartment alongside the lagged/scaled one – a common design for estimating ‘f()‘ from paired IV+oral data (rxode2/rxode2#1237).

`linCmtSensCarry` ‘"auto"‘ (default) substitutes the exact sensitivity-carry gradient for a ‘linCmt()‘ parameter driven by both an eta and a time-varying covariate (needs an rxode2 with the carry sentinels; silently keeps the standard gradient otherwise); ‘"none"‘ always keeps the standard gradient.

`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. One value drives, with a single consistent formula, the inner/outer optimizer convergence tolerance (`10^-sigdig`), the boundary check tolerance (`5*10^(-sigdig+1)`), and the ODE solver tolerances: the `rtol` exponent IS `sigdig` and `atol` sits three orders below, so `rtol = 10^-sigdig`, `atol = 10^(-sigdig-3)` for every solver (stiff, non-stiff or auto-switching). The sensitivity (`atolSens`/`rtolSens`) tolerances match the main solve (the outer gradient and covariance are built from them); the steady-state (`ssAtol`/`ssRtol`) tolerances run one order looser. Keying the optimizer to the same `10^-sigdig` means it converges to exactly the precision the solve supports. At the default `sigdig = 3` this is `atol = 1e-6`, `rtol = 1e-3`.

`sigdigTable` Significant digits in the final output table. If not specified (‘NULL‘), it defaults to ‘sigdig‘.

`boundedTransform` When ‘TRUE‘ (default), bounded parameters are transformed for unbounded optimization methods and back-transformed for final estimates. ‘FALSE‘ optimizes on the original scale with bounds passed to the optimizer. ‘NA‘ transforms for optimization but skips the final back-transform.

`...` Additional optional arguments. None are used at present.

Value

nls control object

Pooled (population-only) estimation

‘est="nls"‘ is a *pooled* estimation method. It estimates population parameters only; there is no eta-conditional inner problem, and sensitivities are taken with respect to the population parameters alone.

A model carrying any random effect is therefore refused before fitting rather than being fit to something other than what it says – it stops with ‘can only have population estimates for the estimation routine ’nls’, try ’focei’‘.

A successful ‘nls‘ fit consequently has no ‘$omega‘, no empirical Bayes estimates and no shrinkage, and its objective function table carries a single ‘Pop‘ row. Use a mixed effects routine – ‘est="focei"‘, ‘est="saem"‘ and so on – for any model with between-subject variability.

Note that this help page inherits its parameter list from ‘foceiControl()‘, ‘saemControl()‘ and ‘nlmControl()‘, because ‘nls‘ shares many options with them. Inherited options that only have meaning for random effects do not apply to ‘nls‘.

Author(s)

Matthew L. Fidler

Examples

```
one.cmt <- function() {
  ini({
```

```
    tka <- 0.45
    tcl <- log(c(0, 2.7, 100))
    tv <- 3.45
    add.sd <- 0.7
  })
  model({
    ka <- exp(tka)
    cl <- exp(tcl)
    v <- exp(tv)
    linCmt() ~ add(add.sd)
  })
}

# Note that `one.cmt` declares no random effects: `nls` is a pooled
# method and refuses a model that has any.

# Uses nlsLM from minpack.lm if available

fit1 <- nlmixr(one.cmt, nlmixr2data::theo_sd, est = "nls", nlsControl(algorithm = "LM"))

# Uses port and respect parameter boundaries
fit2 <- nlmixr(one.cmt, nlmixr2data::theo_sd, est = "nls", nlsControl(algorithm = "port"))

# You can access the underlying nls object with `$nls`
fit2$nls
```

---

nmNearPD *C++ implementation of Matrix's nearPD*

Description

With 'ensureSymmetry' it makes sure it is symmetric by applying 0.5*(t(x) + x) before using nmNearPD

Usage

```
nmNearPD(
  x,
  keepDiag = FALSE,
  do2eigen = TRUE,
  doDykstra = TRUE,
  only.values = FALSE,
  ensureSymmetry = !isSymmetric(x),
  eig.tol = 1e-06,
  conv.tol = 1e-07,
  posd.tol = 1e-08,
  maxit = 100L,
  trace = FALSE
)
```

Arguments

x	numeric $n \times n$ approximately positive definite matrix, typically an approximation to a correlation or covariance matrix. If x is not symmetric (and ensureSymmetry is not false), symmpart(x) is used.
keepDiag	logical, generalizing corr: if TRUE, the resulting matrix should have the same diagonal (diag(x)) as the input matrix.
do2eigen	logical indicating if a posdefify() eigen step should be applied to the result of the Higham algorithm.
doDykstra	logical indicating if Dykstra's correction should be used; true by default. If false, the algorithm is basically the direct fixpoint iteration $Y_k = P_U(P_S(Y_{k-1}))$.
only.values	logical; if TRUE, the result is just the vector of eigenvalues of the approximating matrix.
ensureSymmetry	logical; symmetrizes 'x' via symmpart unless already symmetric. *Beware*: setting FALSE for asymmetric input is typically nonsense.
eig.tol	defines relative positiveness of eigenvalues compared to largest one, λ_1. Eigenvalues λ_k are treated as if zero when $\lambda_k/\lambda_1 \leq eig.tol$.
conv.tol	convergence tolerance for Higham algorithm.
posd.tol	tolerance for enforcing positive definiteness (in the final posdefify step when do2eigen is TRUE).
maxit	maximum number of iterations allowed.
trace	logical or integer specifying if convergence monitoring should be traced.

Details

This implements the algorithm of Higham (2002), and then (if do2eigen is true) forces positive definiteness using code from posdefify. The algorithm of Knol and ten Berge (1989) (not implemented here) is more general in that it allows constraints to (1) fix some rows (and columns) of the matrix and (2) force the smallest eigenvalue to have a certain value.

Note that setting corr = TRUE just sets diag(.) <- 1 within the algorithm.

Higham (2002) uses Dykstra's correction, but the version by Jens Oehlschlägel did not use it (accidentally), and still gave reasonable results; this simplification, now only used if doDykstra = FALSE, was active in nearPD() up to Matrix version 0.999375-40.

Value

unlike the matrix package, this simply returns the nearest positive definite matrix

Author(s)

Jens Oehlschlägel donated a first version. Subsequent changes by the Matrix package authors.

References

Cheng, Sheung Hun and Higham, Nick (1998) A Modified Cholesky Algorithm Based on a Symmetric Indefinite Factorization; *SIAM J. Matrix Anal.\ Appl.*, **19**, 1097–1110.

Knol DL, ten Berge JMF (1989) Least-squares approximation of an improper correlation matrix by a proper one. *Psychometrika* **54**, 53–61.

Higham, Nick (2002) Computing the nearest correlation matrix - a problem from finance; *IMA Journal of Numerical Analysis* **22**, 329–343.

See Also

A first version of this (with non-optional `corr=TRUE`) has been available as `nearcor()`; and more simple versions with a similar purpose `posdefify()`, both from package **sfsmisc**.

Examples

```
set.seed(27)
m <- matrix(round(rnorm(25),2), 5, 5)
m <- m + t(m)
diag(m) <- pmax(0, diag(m)) + 1
(m <- round(cov2cor(m), 2))

near.m <- nmNearPD(m)
round(near.m, 2)
norm(m - near.m) # 1.102 / 1.08

round(nmNearPD(m, only.values=TRUE), 9)

## A longer example, extended from Jens' original,
## showing the effects of some of the options:

pr <- matrix(c(1,     0.477, 0.644, 0.478, 0.651, 0.826,
               0.477, 1,     0.516, 0.233, 0.682, 0.75,
               0.644, 0.516, 1,     0.599, 0.581, 0.742,
               0.478, 0.233, 0.599, 1,     0.741, 0.8,
               0.651, 0.682, 0.581, 0.741, 1,     0.798,
               0.826, 0.75,  0.742, 0.8,   0.798, 1),
               nrow = 6, ncol = 6)

nc  <- nmNearPD(pr)
```

`nmObjGetControl.agq` *Get control object from fit*

Description

Get control object from fit

Usage

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

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

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

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

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

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

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

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

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

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

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

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

## S3 method for class 'lbfgsb3c'
nmObjGetControl(x, ...)

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

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

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

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

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

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

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

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

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

## S3 method for class 'n1qn1'
nmObjGetControl(x, ...)

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

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

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

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

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

nmObjGetControl(x, ...)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

## S3 method for class 'vae'
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

```
nmObjGetEstimationModel(x)
```

Arguments

x	nlmixr fit object

Value

returns the estimation '$model' for the estimation type

nmObjGetFoceiControl.agq

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 'agq'
nmObjGetFoceiControl(x, ...)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

nmObjGetFoceiControl(x, ...)

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

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

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

## S3 method for class 'mnpb'
```

```
nmObjGetFoceiControl(x, ...)

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

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

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

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

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

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

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

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

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

Arguments

x	nlmixr composed fit object
...	Other parameters

Value

foceiControl translated from current control

`nmObjGetIpredModel` *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:
nmObjGetIpredModel(x)

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

## Default S3 method:
nmObjGetEstimationModel(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'
nmObjGetPredOnly(x)

## Default S3 method:
nmObjGetPredOnly(x)
```

Arguments

x	nlmixr fit object

Value

rxode2 pred-only model

`nmObjHandleControlObject.agqControl` *Handle the control object*

Description

Handle the control object

Usage

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

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

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

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

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

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

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

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

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

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

## S3 method for class 'lbfgsb3cControl'
nmObjHandleControlObject(control, env)

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

## S3 method for class 'mfoceiControl'
```

```
nmObjHandleControlObject(control, env)

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

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

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

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

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

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

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

## S3 method for class 'n1qn1Control'
nmObjHandleControlObject(control, env)

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

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

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

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

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

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)

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

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

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

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

## S3 method for class 'vaeControl'
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

---

nmObjUiSetCompressed *Set if the nlmixr2 object will return a compressed ui*

Description

Set if the nlmixr2 object will return a compressed ui

Usage

```
nmObjUiSetCompressed(type)
```

Arguments

type	is a boolean indicating if the compressed ui will be returned ('TRUE') or not be returned ('FALSE')

Value

invisible logical type

Author(s)

Matthew L. Fidler

Examples

```
nmObjUiSetCompressed(FALSE) # now the $ui will return an environment
nmObjUiSetCompressed(TRUE) # now the $ui will return a compressed value
```

`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

Value

a list of ...

`npagControl` *Control for the npag (nonparametric adaptive grid) method*

Description

A wrapper around [impmapControl()] that reuses the shared FOCEI family plumbing for the nonparametric adaptive grid engine. The nonparametric support-point knobs are added in a later milestone.

Usage

```
npagControl(
  points = NULL,
  cycles = 100L,
  gammaOptimize = TRUE,
  residOptimize = c("alternate", "final", "none"),
  muExpand = FALSE,
  gridWidth = 4,
  gridBounds = c("auto", "ini", "both"),
  dfScan = -1L,
  cores = NULL,
  rhoend = 1e-04,
  gamma,
  df,
  ...
)
```

Arguments

points	Initial Sobol grid size (support points). 'NULL' (default) picks it automatically from the number of support-point dimensions (etas): 'max(2028, 512 * n_eta)' – a fixed grid (Pmetrics uses 2028) covers a low-dimensional model but grows sparse and can collapse in high dimensions, so the auto size floors at 2028 and scales up per added eta. Supply an integer to override.
cycles	Maximum adaptive-grid cycles.
gammaOptimize	Use a global assay-error multiplier as a per-cycle warm start for the overall residual magnitude, folded into the variance-scale coefficients ('add'/'prop'/'lnorm'). The per-endpoint values, the add/prop ratio, and the transform/autocorrelation parameters come from `residOptimize`. Only valid for normal endpoints; censoring and transform-both-sides are supported. Unrelated to [impmapControl()]'s 'gamma', which inflates an importance-sampling proposal's variance and has no meaning here.
residOptimize	How to estimate the residual-error thetas (every endpoint's 'add'/'prop'/'lnorm', each transform 'lambda', each 'ar') with the support points and weights held fixed, using bounded `minqa::bobyqa` on the EXTENDED LEAST SQUARES objective 'sum_obs((f-dv)^2/r + log(r))' at the posterior-mean etas. The 'log(r)' term keeps the residual from drifting to zero on a flexible support (which the marginal likelihood would reward), giving the saem/focei residual; each variance scale is warm-started from the per-endpoint moment (additive SD from 'sqrt(mean(err^2))', proportional from 'sqrt(mean((err/f)^2))', on the transform-both-sides scale). `"alternate"` (default) optimizes every cycle (block- coordinate ascent); `"final"` optimizes once at the converged support; `"none"` holds them at their initial values. Fixed residual parameters are always held. After the residual thetas converge, a final adaptive-grid pass re-optimizes the support with them held constant so the support remains the nonparametric MLE (D(F) ~ 0) for the fitted residual.

`muExpand` how to estimate non-mu structural fixed-effect parameters (a theta with no eta, e.g. ‘ke <- exp(tke)‘; npag’s grid otherwise covers only mu-referenced and residual/likelihood parameters). ‘FALSE‘ (default) optimizes them directly as "regressors" in the residual step – ‘bobyqa‘ moves them alongside the residual parameters, re-deriving the posterior-mean etas each candidate (so the eta grid cannot stale-absorb the structural shift) – which identifies them well (e.g. recovering a clearance from a poor start). Not available for mix() models (the ELS step is not mixture-aware; component parameters are held). ‘TRUE‘ instead uses the saem-style mu-expansion: inject a pseudo-eta (‘ke <- exp(tke + eta.tke)‘), grid-estimate, and recover it as a fixed effect at finalization (support-mean folded into the theta, injected random effect collapsed). The regressor default usually identifies these parameters more sharply than the grid.

`gridWidth` support-point box half-width, in initial-eta SDs, for the ‘gridBounds="auto"‘ grid (default 4). A narrower box focuses the initial Sobol grid on the plausible region – useful for high-dimensional models where a wide box wastes points on near-zero-density support (which can collapse the fit).

`gridBounds` how to set the initial support-point box: ‘"auto"‘ (default) uses ‘+/- gridWidth * initial eta SD‘; ‘"ini"‘ uses each mu-referenced parameter’s ini-block lower/upper bounds where finite (else auto); ‘"both"‘ uses the ini bounds when present and auto otherwise. For a high-dimensional model, bounded ini estimates + ‘"ini"‘ keep the grid in range.

`dfScan` Size of the Sobol scan used for the D(F) global-optimality certificate: ‘-1‘ (default) auto-sizes it to ‘max(2048, 2 * points)‘, ‘0‘ skips the certificate (‘npagDF‘ is ‘NA‘), and a positive value sets an explicit scan size. The scan does not affect the fit, only the reported certificate; a smaller scan is faster.

`cores` Number of threads used for the parallel per-subject conditional- likelihood solves. ‘NULL‘ (default) uses the current ‘rxode2‘ thread count (‘rxode2::getRxThreads()‘); an integer sets the thread count for the fit (restored afterwards). Results are independent of the thread count.

`rhoend` Final trust-region radius (‘rhoend‘) of the inner bounded ‘bobyqa‘ that fits the residual-error thetas each cycle. A fixed default of ‘1e-4‘, matching the optimizer convergence tolerance ‘10^(-sigdig)‘ at ‘sigdig = 4‘ (npag has no ‘sigdig‘, so this is not derived from it).

`gamma`, `df` Declared only so they are REJECTED rather than partially matched. ‘gamma‘ is a prefix of ‘gammaOptimize‘, so without an explicit formal R bound ‘gamma = 2‘ to it and silently turned the assay-error optimisation off; ‘df‘ is an importance-sampling proposal control a nonparametric engine never builds. Passing either is an error.

`...` Parameters passed to [impmapControl()], for the shared FOCEI-family scaffolding only (the inner MAP problem, mu-referencing, the residual error model, threads). The importance-sampling controls are **rejected** rather than accepted: ‘isample‘, ‘df‘, ‘auto‘, ‘iaccept‘, ‘gamma‘, ‘qr‘, ‘sir‘ and the rest configure a proposal density that a nonparametric engine never builds, so passing one is an error rather than a silent no-op. Where an np control does the job the message names it (‘nIter‘ -> ‘cycles‘, ‘ctol‘ -> ‘rhoend‘).

Note ‘gamma‘ is NOT ‘gammaOptimize‘: ‘gamma‘ is impmap’s proposal-variance inflation (NONMEM ‘ISCALE‘), while ‘gammaOptimize‘ is a global assay-

error multiplier on the residual magnitude. They are unrelated, and ‘gamma‘ is a prefix of ‘gammaOptimize‘, so it is rejected explicitly to stop R’s partial matching from silently binding one to the other.

Details

Note: the npag objective is the nonparametric marginal log-likelihood and uses a different constant convention than NONMEM/FOCEI, so its ‘-2LL‘ is NOT comparable to nlmixr2’s FOCEI/SAEM/FOCE ‘-2LL‘. Compare npag runs to each other or to Pmetrics NPAG.

Note on residual error with a flexible support distribution: the residual parameters are estimated against the nonparametric objective (see `residOptimize`) with the support-point distribution held fixed. Because that distribution is flexible, it can absorb variability a parametric model (FOCEI/SAEM) would attribute to residual error – especially the additive term of a combined additive+proportional model at low concentrations. As a result the additive coefficient of a combined error model may be estimated smaller (sometimes toward zero) than the corresponding parametric fit, while the proportional term and per-endpoint magnitudes are recovered well. This is an expected property of nonparametric estimation, not a convergence failure; use `residOptimize = "none"` to hold the residual parameters at their initial values if a fixed error model is desired.

Value

An ‘impmapControl‘ object tagged for the npag engine.

Author(s)

Matthew L. Fidler

Examples

```
npagControl()
```

`npbControl` *Control for the npb (nonparametric Bayes) method*

Description

A wrapper around [impmapControl()] that reuses the shared FOCEI family plumbing for the nonparametric Bayes engine. The stick-breaking sampler knobs are added in a later milestone.

Usage

```
npbControl(
  points = 50L,
  alpha = 1,
  burnin = 500L,
  nsamp = 500L,
  nchains = 1L,
  propSd = 0.2,
```

```
  seed = 42L,
  residOptimize = c("alternate", "final", "none"),
  cycles = 100L,
  gammaOptimize = FALSE,
  muExpand = FALSE,
  cores = NULL,
  rhoend = 1e-04,
  gamma,
  df,
  ...
)
```

Arguments

points	Stick-breaking truncation level K (number of support points).
alpha	Dirichlet-process concentration parameter.
burnin	Number of burn-in Gibbs sweeps.
nsamp	Number of post-burn-in Gibbs samples collected.
nchains	Number of independent chains (Gelman-Rubin R-hat convergence is reported when nchains > 1).
propSd	Standard deviation of the Gaussian random-walk MH proposal for the support-point locations (eta space).
seed	Random seed for the sampler.
residOptimize	How to estimate the residual-error thetas (every endpoint's 'add'/'prop'/'lnorm', each transform 'lambda', each 'ar') and any non-mu structural "regressor" theta, with the sampled mixing distribution held fixed, using the bounded 'bobyqa' on the EXTENDED LEAST SQUARES objective (see [npagControl()]; the 'log(r)' term keeps the residual from collapsing to zero and the moment warm-start gives the saem-style SD). "alternate" (default) re-fits them during burn-in and then holds them fixed for the sampling phase (so every collected draw shares the converged residual scale); "final" holds them at their initial values through sampling and fits once at the converged draw; "none" holds them at their initial values throughout. Fixed residual parameters are always held. Unlike npag, npb does not optimize the assay-error multiplier (gamma); the residual thetas are fit directly.
cycles	Unused for npb (kept for control compatibility).
gammaOptimize	Unused for npb (kept for control compatibility).
muExpand	When 'TRUE', mu-expand non-mu structural fixed-effect thetas (a theta with no eta) into grid-estimable pseudo-etas before the fit; 'FALSE' (default) leaves them to the residual step.
cores	Number of threads used for the parallel per-subject conditional- likelihood solves in the Gibbs sweeps. 'NULL' (default) uses the current 'rxode2' thread count ('rxode2::getRxThreads()'); an integer sets the thread count for the fit (restored afterwards). With a fixed 'seed' the fit is bit-for-bit identical regardless of the thread count.

rhoend	Final trust-region radius (‘rhoend‘) of the inner bounded ‘bobyqa‘ that fits the residual-error thetas. A fixed default of ‘1e-4‘, matching the optimizer convergence tolerance ‘10^(-sigdig)‘ at ‘sigdig = 4‘ (npb has no ‘sigdig‘, so this is not derived from it).
gamma, df	Declared only so they are REJECTED rather than partially matched. ‘gamma‘ is a prefix of ‘gammaOptimize‘, so without an explicit formal R bound ‘gamma = 2‘ to it and silently turned the assay-error optimisation off; ‘df‘ is an importance-sampling proposal control a nonparametric engine never builds. Passing either is an error.
...	Parameters passed to [impmapControl()], for the shared FOCEI-family scaffolding only. The importance-sampling controls are **rejected** rather than accepted – see [npagControl()] – as are ‘cycles‘ and ‘gammaOptimize‘, which this signature carries but ‘npb‘ does not use. ‘impSeed‘ is rejected pointing at ‘seed‘.

Details

Note: the npb objective is the nonparametric marginal log-likelihood and uses a different constant convention than NONMEM/FOCEI, so its ‘-2LL‘ is NOT comparable to nlmixr2’s FOCEI/SAEM/FOCE ‘-2LL‘. Compare npb runs to each other or to Pmetrics NPAG.

Value

An ‘impmapControl‘ object tagged for the npb engine.

Author(s)

Matthew L. Fidler

Examples

```
npbControl()
```

---

ofv *Return the objective function*

Description

Return the objective function

Usage

```
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

---

optimControl *nlmixr2 optim defaults*

Description

nlmixr2 optim defaults

Usage

```
optimControl(
  method = c("Nelder-Mead", "BFGS", "CG", "L-BFGS-B", "SANN", "Brent"),
  trace = 0,
  fnscale = 1,
  parscale = 1,
  ndeps = 0.001,
  maxit = 10000,
  abstol = NULL,
  reltol = NULL,
  alpha = 1,
```

```
  beta = 0.5,
  gamma = 2,
  REPORT = NULL,
  warn.1d.NelderMead = TRUE,
  type = NULL,
  lmm = 5,
  factr = NULL,
  pgtol = 0,
  temp = 10,
  tmax = 10,
  stickyRecalcN = 4,
  maxOdeRecalc = 5,
  odeRecalcFactor = 10^(0.5),
  indTolRelax = TRUE,
  eventType = c("central", "forward"),
  shiErr = (.Machine$double.eps)^(1/3),
  shi21maxFD = 20L,
  solveType = c("grad", "fun"),
  useColor = NULL,
  printNcol = NULL,
  print = 1L,
  normType = c("rescale2", "mean", "rescale", "std", "len", "constant"),
  scaleType = c("nlmixr2", "norm", "mult", "multAdd"),
  scaleCmax = 1e+05,
  scaleCmin = 1e-05,
  scaleC = NULL,
  scaleTo = 1,
  gradTo = 1,
  rxControl = NULL,
  optExpression = TRUE,
  sumProd = FALSE,
  literalFix = TRUE,
  literalFixRes = TRUE,
  returnOptim = FALSE,
  addProp = c("combined2", "combined1"),
  eventSens = c("jump", "fd"),
  sensMethod = c("default", "forward"),
  calcTables = TRUE,
  compress = FALSE,
  covMethod = c("r", "optim", ""),
  adjObf = TRUE,
  ci = 0.95,
  sigdig = 3,
  sigdigTable = NULL,
  boundedTransform = TRUE,
  ...
)
```

Arguments

method	The method to be used. See 'Details'. Can be abbreviated.
trace	Non-negative integer. If positive, tracing information on the progress of the optimization is produced. Higher values may produce more tracing information: for method '"L-BFGS-B"', there are six levels of tracing. See 'optim()' for more information
fnscale	An overall scaling to be applied to the value of 'fn' and 'gr' during optimization. If negative, turns the problem into a maximization problem. Optimization is performed on 'fn(par)/fnscale'
parscale	A vector of scaling values for the parameters. Optimization is performed on 'par/parscale' and these should be comparable in the sense that a unit change in any element produces about a unit change in the scaled value. Not used (nor needed) for 'method = "Brent"'
ndeps	A vector of step sizes for the finite-difference approximation to the gradient, on 'par/parscale' scale. Defaults to '1e-3'
maxit	The maximum number of iterations. Defaults to '100' for the derivative-based methods, and '500' for '"Nelder-Mead"'.
abstol	The absolute convergence tolerance. Only useful for non-negative functions, as a tolerance for reaching zero.
reltol	Relative convergence tolerance. The algorithm stops if it is unable to reduce the value by a factor of 'reltol * (abs(val) + reltol)' at a step
alpha	Reflection factor for the '"Nelder-Mead"' method.
beta	Contraction factor for the '"Nelder-Mead"' method
gamma	Expansion factor for the '"Nelder-Mead"' method
REPORT	The frequency of reports for the '"BFGS"', '"L-BFGS-B"' and '"SANN"' methods if 'control$trace' is positive. Defaults to every 10 iterations for '"BFGS"' and '"L-BFGS-B"', or every 100 temperatures for '"SANN"'
warn.1d.NelderMead	a logical indicating if the (default) '"Nelder-Mead"' method should signal a warning when used for one-dimensional minimization. As the warning is sometimes inappropriate, you can suppress it by setting this option to 'FALSE'
type	for the conjugate-gradients method. Takes value '1' for the Fletcher-Reeves update, '2' for Polak-Ribiere and '3' for Beale-Sorenson.
lmm	is an integer giving the number of BFGS updates retained in the '"L-BFGS-B"' method, It defaults to '5'
factr	controls the convergence of the '"L-BFGS-B"' method. Convergence occurs when the reduction in the objective is within this factor of the machine tolerance. 'NULL' (default) uses '10^(-sigdig-2) / .Machine$double.eps' (at least 1), two orders tighter than 'sigdig', as 'foceiControl(lbfgsFactr=)' does.
pgtol	helps control the convergence of the '"L-BFGS-B"' method. It is a tolerance on the projected gradient in the current search direction. This defaults to zero, when the check is suppressed

`temp` controls the '"SANN"' method. It is the starting temperature for the cooling schedule. Defaults to '10'.

`tmax` is the number of function evaluations at each temperature for the '"SANN"' method. Defaults to '10'.

`stickyRecalcN` The number of bad ODE solves before reducing the atol/rtol for the rest of the problem.

`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

`indTolRelax` When 'TRUE' (default), only subjects whose ODE solve produced NaN/Inf have their tolerances relaxed, and the relaxed tolerance persists across optimizer calls (sticky). When 'FALSE', all subjects have their tolerances relaxed on each retry and tolerances are reset afterward.

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

`shiErr` This represents the epsilon when optimizing the ideal step size for numeric differentiation using the Shi2021 method

`shi21maxFD` The maximum number of steps for the optimization of the forward difference step size when using dosing events (lag time, modeled duration/rate and bioavailability)

`solveType` controls whether 'optim' uses nlmixr2's analytical gradients (event-related parameters like lag time/duration/rate/F use Shi2021 finite differences instead). '"gradient"' supplies the gradient and lets 'optim' compute the finite-difference Hessian; '"fun"' lets 'optim' compute both by finite differences. Only applies to the gradient-based methods: "BFGS", "CG", "L-BFGS-B".

`useColor` Logical (or 'NULL') emit ANSI bold/color escapes in the iteration print. 'NULL' (default) defers to [crayon::has_color()].

`printNcol` Integer (or 'NULL') parameter columns per row before wrapping. 'NULL' (default) uses 'floor((getOption("width") - 23) / 12)'.

`print` Either a scalar print-frequency ('0' = suppress, '1' (default) = every evaluation, 'N' = every Nth), OR a pre-built [iterPrintControl()] object. Equivalent to 'iterPrintControl(every = print, ncol = printNcol, useColor = useColor)'.

`normType` Parameter normalization/scaling used to get scaled initial values for `scaleType`, of the form `Vscaled = (Vunscaled-C1)/C2` (see Feature Scaling; `rescale2` follows the OptdesX manual): `"rescale2"` scales all parameters to (-1, 1); `"rescale"` (min-max) scales to (0, 1); `"mean"` centers on the mean with range (0, 1); `"std"` standardizes by mean/sd; `"len"` scales to unit (Euclidean) length; `"constant"` performs no normalization (`C1=0`, `C2=1`).

`scaleType` The scaling scheme for nlmixr2: `"nlmixr2"` (default) scales as `(current-init)*scaleC[i] + scaleTo`, with `scaleTo` from `normType` and scales from `scaleC`; `"norm"` uses the simple scaling from `normType`; `"mult"` scales multiplicatively as `current/init*scaleTo`; `"multAdd"` scales linearly (`(current-init)+scaleTo`) for parameters in an exponential block (e.g. `exp(theta)`) and multiplicatively otherwise.

`scaleCmax` Maximum value of the scaleC to prevent overflow.

`scaleCmin` Minimum value of the scaleC to prevent underflow.

`scaleC` Scaling constant used with `scaleType="nlmixr2"`; when not specified, chosen by parameter type to keep gradient sizes similar on a log scale: ‘1‘ for exp()-transformed/power/boxCox/ yeoJohnson parameters, ‘0.5*abs(est)‘ for additive/proportional/ lognormal error parameters, ‘abs(1/digamma(est+1))‘ for factorials, and ‘log(abs(est))*abs(est)‘ for log-scale parameters. May be set explicitly per parameter if these defaults don’t apply well.

`scaleTo` Scale the initial parameter estimate to this value. By default this is 1. When zero or below, no scaling is performed.

`gradTo` this is the factor that the gradient is scaled to before optimizing. This only works with scaleType="nlmixr2".

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

`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`.

`literalFix` boolean, substitute fixed population values as literals and re-adjust ui and parameter estimates after optimization; Default is ‘TRUE‘.

`literalFixRes` boolean, substitute fixed population values as literals and re-adjust ui and parameter estimates after optimization; Default is ‘TRUE‘.

`returnOptim` logical; when TRUE this will return the optim list instead of the nlmixr2 fit object

`addProp` Type of additive-plus-proportional error: ‘"combined1"‘, where standard deviations add:

$$y = f + (a + b \times f^c) \times \varepsilon$$

; or ‘"combined2"‘, where variances add:

$$y = f + \sqrt{a^2 + b^2 \times f^{2 \times c}} \times \varepsilon$$

. Here y = observed, f = predicted, a = additive sd, b = proportional/power sd, c = power exponent (1 in the proportional case).

`eventSens` Controls how dosing/event-parameter (‘alag‘, ‘F‘, ‘rate‘, ‘dur‘) sensitivities are computed for THETA/ETA gradients: ‘"jump"‘ (default) uses rxode2’s analytic event sensitivities; ‘"fd"‘ uses the legacy finite-difference behavior. Also gates the analytic moving-boundary correction for a modeled ‘alag()‘/‘f()‘ on a ‘linCmt()‘ compartment; set ‘"fd"‘ if that model infuses a dose into the lagged/scaled compartment (rxode2/rxode2#1236), or if the regimen also doses an *unlagged/unscaled* compartment alongside the lagged/scaled one – a common design for estimating ‘f()‘ from paired IV+oral data (rxode2/rxode2#1237).

`sensMethod` Method used to compute the ODE parameter sensitivities. ‘"forward"‘ uses the classic variational (forward) sensitivity ODEs; ‘"default"‘ is the same thing.

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

compress	Should the object have compressed items
covMethod	allows selection of "r", which uses nlmixr2's 'nlmixr2Hess()' for the hessian calculation or "optim" which uses the hessian from 'stats::optim(.., hessian=TRUE)'
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. One value drives, with a single consistent formula, the inner/outer optimizer convergence tolerance (10^-sigdig), the boundary check tolerance (5*10^(-sigdig+1)), and the ODE solver tolerances: the rtol exponent IS sigdig and atol sits three orders below, so rtol = 10^-sigdig, atol = 10^(-sigdig-3) for every solver (stiff, non-stiff or auto-switching). The sensitivity (atolSens/rtolSens) tolerances match the main solve (the outer gradient and covariance are built from them); the steady-state (ssAtol/ssRtol) tolerances run one order looser. Keying the optimizer to the same 10^-sigdig means it converges to exactly the precision the solve supports. At the default sigdig = 3 this is atol = 1e-6, rtol = 1e-3.
sigdigTable	Significant digits in the final output table. If not specified ('NULL'), it defaults to 'sigdig'.
boundedTransform	When 'TRUE' (default), bounded parameters are transformed for unbounded optimization methods and back-transformed for final estimates. 'FALSE' optimizes on the original scale with bounds passed to the optimizer. 'NA' transforms for optimization but skips the final back-transform.
...	Further arguments to be passed to fn and gr.

Value

optimControl object for nlmixr2

Author(s)

Matthew L. Fidler

Examples

```
# A logit regression example with emax model

dsn <- data.frame(i=1:1000)
dsn$time <- exp(rnorm(1000))
dsn$DV=rbinom(1000,1,exp(-1+dsn$time)/(1+exp(-1+dsn$time)))

mod <- function() {
 ini({
   E0 <- 0.5
   Em <- 0.5
   E50 <- 2
   g <- fix(2)
 })
```

```
 model({
   v <- E0+Em*time^g/(E50^g+time^g)
   ll(bin) ~ DV * v - log(1 + exp(v))
 })
}

fit2 <- nlmixr(mod, dsn, est="optim", optimControl(method="BFGS"))
fit2
```

---

posthocControl *Control options for the posthoc estimation method*

Description

This option is for simply getting the maximum a-prior (MAP) also called the posthoc estimates

Usage

```
posthocControl(sigdig = 3, ..., interaction = FALSE, maxOuterIterations = NULL)
```

Arguments

sigdig	Optimization significant digits. One value drives, with a single consistent formula, the inner/outer optimizer convergence tolerance (`10^-sigdig`), the boundary check tolerance (`5*10^(-sigdig+1)`), and the ODE solver tolerances: the `rtol` exponent IS `sigdig` and `atol` sits three orders below, so `rtol = 10^-sigdig`, `atol = 10^(-sigdig-3)` for every solver (stiff, non-stiff or auto-switching). The sensitivity (`atolSens`/`rtolSens`) tolerances match the main solve (the outer gradient and covariance are built from them); the steady-state (`ssAtol`/`ssRtol`) tolerances run one order looser. Keying the optimizer to the same `10^-sigdig` means it converges to exactly the precision the solve supports. At the default `sigdig = 3` this is `atol = 1e-6`, `rtol = 1e-3`.
...	Parameters used in the default 'foceiConrol()'
interaction	Interaction term for the model, in this case the default is 'FALSE', though you can set it to be 'TRUE' as well.
maxOuterIterations	ignored, posthoc always sets this to 0.

Value

posthocControl object

Author(s)

Matthew L. Fidler

Examples

```
posthocControl()
```

`predict.nlmixr2FitCore` *Predict method for nlmixr2 fit core objects*

Description

Generates population- or individual-level predictions from an ‘nlmixr2FitCore‘ object.

Usage

```
## S3 method for class 'nlmixr2FitCore'
predict(object, ..., level = c("population", "individual"))
```

Arguments

`object`	nlmixr2 fit core object to predict
`...`	additional arguments passed to rxode2::rxSolve or nlmixr2; matching other ‘predict‘ methods, these can include ‘newdata‘ and ‘rxControl‘ settings
`level`	the prediction level; one of ‘"population"‘ (default) or ‘"individual"‘; numeric values ‘0‘ and ‘1‘ are also accepted

Value

A data frame with predictions

Examples

```
one.compartment <- function() {
  ini({
    tka <- log(1)
    tcl <- log(10)
    tv <- log(35)
    eta.ka ~ 0.1
    eta.cl ~ 0.1
    eta.v ~ 0.1
    add.sd <- 0.1
  })
  model({
    ka <- exp(tka + eta.ka)
    cl <- exp(tcl + eta.cl)
    v <- exp(tv + eta.v)
    d/dt(depot) <- -ka * depot
    d/dt(center) <- ka * depot - cl / v * center
    cp <- center / v
```

```
    cp ~ add(add.sd)
  })
}

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

# Population predictions
ppred <- predict(fit, theo_sd, level = "population")

# Individual predictions
ipred <- predict(fit, theo_sd, level = "individual")
```

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

Description

Print an SAEM model fit summary

Usage

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

Arguments

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

Value

a list

`qrpemControl` *Control for the qrpem (quasi-random parametric EM) estimation method*

Description

A convenience wrapper around [impmapControl()] defaulting ‘qr=TRUE‘ (Sobol quasi-random importance samples) and ‘sir=TRUE‘ (SIR-accelerated non-mu / residual-error M-step); explicitly supplied arguments win. See [impmapControl()] for the full parameter list.

Usage

```
qrpemControl(..., qr = TRUE, sir = TRUE)
```

Arguments

`...`	Parameters passed to [impmapControl()].
`qr`	When ‘TRUE‘, draw quasi-random (Sobol low-discrepancy) importance samples instead of pseudo-random Gaussian samples (QRPEM, Leary & Dunlavey PAGE 2012); the E-step integrals converge at O(1/N) instead of O(1/sqrt(N)).
`sir`	When ‘TRUE‘, accelerate the non-mu / residual-error M-step by SIR (sampling-importance-resampling): the theta-sensitivity Newton step uses ‘sirSample‘ equal-weight resampled points per subject instead of all ‘isample‘ weighted samples.

Details

Note this is not know to be the same as the QRPEM implementation in Phoenix NLME since the details of their method are not public. However, this matches the QRPEM method of using quasi-random parametric EM and SIR accelerated parameter convergence described in Leary & Dunlavey (2012) PAGE 2012, 19(1): 1-6.

Value

An ‘impmapControl‘ object with the QRPEM defaults.

Author(s)

Matthew L. Fidler

Examples

```
qrpemControl()
```

`residuals.nlmixr2FitData` *Extract residuals from the FOCEI fit*

Description

Extract residuals from the FOCEI fit

Usage

```
## 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

`rmEta` *Remove an eta from the model*

Description

Remove an eta from the model

Usage

```
rmEta(ui, eta)
```

Arguments

`ui`	rxode2 user interface
`eta`	eta to remove

Value

ui model with eta removed

Author(s)

Matthew L. Fidler

Examples

```
mod <- function ()  {
 description <- "One compartment PK model with linear clearance"
 ini({
   lka <- 0.45
   lcl <- 1
   lvc <- 3.45
    propSd <- c(0, 0.5)
    etaKa ~ 0.1
  })
 model({
   ka <- exp(lka + etaKa)
   cl <- exp(lcl)
   vc <- exp(lvc)
   Cc <- linCmt()
   Cc ~ prop(propSd)
 })
}

mod |> rmEta("etaKa")

# This can also remove more than one eta

mod <- function ()  {
 description <- "One compartment PK model with linear clearance"
 ini({
   lka <- 0.45
   lcl <- 1
   lvc <- 3.45
   propSd <- c(0, 0.5)
   etaKa ~ 0.1
   etaCl ~ 0.2
   etaVc ~ 0.3
  })
 model({
   ka <- exp(lka + etaKa)
   cl <- exp(lcl + etaCl)
   vc <- exp(lvc + etaVc)
   Cc <- linCmt()
   Cc ~ prop(propSd)
 })
}

mod |> rmEta(c("etaKa", "etaCl"))
```

rsControl *Options for the finite-difference covariance in setCov()*

Description

Used by `setCov(fit, "r,s")`, `"r"` and `"s"`. Each option left `NULL` keeps the value the fit was estimated with, except `covFallback`.

Usage

```
rsControl(
  hessEps = NULL,
  gillKcov = NULL,
  gillStepCov = NULL,
  gillFtolCov = NULL,
  covGillF = NULL,
  covSmall = NULL,
  rmatNorm = NULL,
  smatNorm = NULL,
  covFallback = NULL,
  covPrecursor,
  covShortcut = NULL
)
```

Arguments

`hessEps`	is a double value representing the epsilon for the Hessian calculation. This is used for the R matrix calculation.
`gillKcov`	Max steps to determine the optimal forward/central difference step size per parameter (Gill 1983) during the covariance step. '0' = no 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.
`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 of the theta-only stage (see `covFull`).
`covSmall`	Small number used to compare covariance estimates (sandwich vs R/S matrix) before rejecting one as too small to be the final covariance estimate.
`rmatNorm`	A parameter to normalize gradient step size by the parameter value during the calculation of the R matrix of the theta-only stage (see `covFull`)

smatNorm	A parameter to normalize gradient step size by the parameter value during the calculation of the S matrix of the theta-only stage (see covFull)
covFallback	what the requested method may fall back to, as in foceiControl(covFallback=). NULL (default) is none: setCov() installs the method asked for or stops with an error. With a list, a covariance from a method it lists for the request is installed, with a warning that names it.
covPrecursor	what the covariance may start from, as in foceiControl(covPrecursor=). Left out, the fit's value is kept; NULL uses none.
covShortcut	as in foceiControl(covShortcut=); NULL (default) keeps the fit's value.

Value

rsControl object

Author(s)

Matt Fidler

See Also

setCov()

Examples

```
rsControl(hessEps = 1e-4)
```

---

saControl *Options for the SAEM stochastic-approximation covariance in setCov()*

Description

Used by setCov(fit, "sa"), which runs the SAEM covariance phase at the fit's estimates, after warm-up iterations unless it continues a SAEM fit's own chains. Every population parameter is held at the fit's estimates throughout (mixture proportions excepted), so the covariance is the one at those estimates.

Usage

```
saControl(
  nBurn = 100L,
  nEm = 100L,
  nSaCov = 500L,
  seed = 99L,
  warmStart = TRUE
)
```

Arguments

nBurn, nEm	warm-up iterations that equilibrate the MCMC chains before the covariance phase, for a fit without SAEM chain history (or with warmStart = FALSE)
nSaCov	iterations in the covariance phase; more gives a less noisy covariance
seed	random seed
warmStart	for a SAEM fit, continue the fit's own MCMC chains: the covariance phase starts from the fit's last iteration with no new warm-up iterations. This changes the Monte Carlo path, so the covariance differs from a cold start by Monte Carlo noise. Other fits, and SAEM fits that kept no chain history, always run nBurn/nEm.

Value

saControl object

Author(s)

Matt Fidler

See Also

setCov(), saemControl()

Examples

```
saControl(nSaCov = 1000)
```

---

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 = 1L,
  trace = 0,
  covMethod = c("sa", "analytic", "linFim", "fim", "r,s", "r", "s", "imp", ""),
  covMethodDeferred = NA_character_,
```

```
covFallback = list(sa = c("linFim", "Ha"), fim = c("linFim", "Ha"), analytic =
  c("linFim", "Ha"), linFim = "Ha"),
covFull = TRUE,
nSaCov = 500L,
calcTables = TRUE,
logLik = FALSE,
nnodesGq = 3,
nsdGq = 1.6,
optExpression = TRUE,
literalFix = FALSE,
adjObf = TRUE,
sumProd = FALSE,
addProp = c("combined2", "combined1"),
tol = NULL,
itmax = 30,
type = c("newuoa", "nelder-mead"),
powRange = 10,
lambdaRange = 3,
odeRecalcFactor = 10^(0.5),
maxOdeRecalc = 5L,
indTolRelax = TRUE,
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,
muRefCovAlg = TRUE,
handleUninformativeEtas = TRUE,
revisitUninformativeEtas = FALSE,
iovXform = c("sd", "var", "logsd", "logvar"),
iovMethod = c("twoLevel", "collapsed", "theta"),
boundedTransform = TRUE,
eventSens = c("jump", "fd"),
mixProbMethod = c("regress", "regularized", "annealed"),
mixProbStepExp = 1,
mixProbPriorN = 20,
mixSampleMethod = c("parallel", "msaem"),
nonMuTheta = c("regress", "eta"),
nonMuThetaOpt = c("newuoa", "optimize", "nelderMead"),
nonMuThetaSweeps = 2L,
nonMuThetaMaxEval = 25L,
nonMuThetaTol = .Machine$double.eps^0.25,
nonMuThetaEvery = 1L,
```

```
  phi1ThetaMaxEval = 50L,
  phi1ThetaEvery = 1L,
  phi1Hessian = FALSE,
  residWarmStart = TRUE,
  censOption = c("gauss", "laplace"),
  ...
)
```

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). When unset, a residual error model that depends on an eta uses c(4,4,4), since its chains mix more slowly. 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	Either a scalar print-frequency (‘0‘ = suppress, ‘1‘ (default) = every evaluation, ‘N‘ = every Nth), OR a pre-built [iterPrintControl()] object. Equivalent to ‘iterPrintControl(every = print, ncol = printNcol, useColor = useColor)‘.
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). "sa" (default) Use the stochastic-approximation Fisher Information Matrix. After estimation, a dedicated covariance phase (nSaCov iterations) holds the parameters at the converged estimate and keeps resimulating the individual parameters, Monte-Carlo averaging the Louis observed-information integrand into a converged FIM decoupled from the cooling schedule (the approach used by Monolix; Kuhn & Lavielle 2005). Always includes every estimated population parameter (theta, the Omega diagonal variances, and residual).

"analytic" Compute the FOCEI analytic observed-information covariance at the converged SAEM estimates. When the model is out of analytic-covariance scope (e.g. linCmt(), a non-normal likelihood, or a non-SD IOV parameterization) or the result is not positive definite, it falls back to the linearized Fisher information (linFim) with a message.

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

"fim" Use the Fisher Information Matrix accumulated during SAEM estimation to calculate the covariance. Like sa it inverts the observed information to a full theta + Omega diagonal + residual covariance, but uses the (noisier) estimation-phase matrix rather than a dedicated cov phase.

For both fim and sa the simulation-based Fisher information covers the structural theta, the Omega diagonal variances, and additive residual error. Off-diagonal Omega covariances and proportional/combined residual error are not estimated reliably by the simulation FIM (the complete-data correction is unstable when between-subject variability dominates the residual), so those variance-block standard errors are spliced in from the linearized FIM (linFim).

Neither is used for a general log-likelihood endpoint (ll()), which has no residual error to anchor the complete-data correction; such a model goes straight to linFim with a message. This includes a residual error with modeled components (a <- add.sd*exp(eta.sd); cp ~ add(a)), which saem fits as cp ~ add(a) + dnorm().

"r,s", "r" and "s" compute no R or S matrix in saem: all three invert the structural-theta block of the estimation-phase information matrix (fit$saem$Ha, the matrix fim inverts in full). The result covers the estimated mu-referenced thetas only, conditional on the Omega and residual parameters; a theta without a random effect has no row in it and gets no standard error. It is installed as fit$covMethod == "Ha", or "|Ha|" when the block was not positive definite and was replaced by sqrtm(Ha %*% Ha). For the finite-difference R and S covariances at the SAEM estimates use setCov(fit, "r,s"). An integer covMethod is a foceiControl() slot: 0 is no covariance, 1, 2 and 3 are "r,s", "r" and "s".

"" Does not calculate the covariance step.

covMethodDeferred

Internal. When a foreign covariance ("sa"/"imp") is requested, it is stashed here and computed post-fit at the converged estimates by the decoupled recompute engine (setCov() uses the same path); NA otherwise.

covFallback what each covMethod falls back to when it gives no usable covariance: a named list, one element per method, each the ordered methods to try instead ("sa", "fim", "linFim" or "Ha", the inverse of the information matrix's theta block). The default is the established behaviour: "sa", "fim" and "analytic" fall to "linFim" then "Ha", and "linFim" to "Ha". A list you give replaces the default: a method it does not name has no fallback.

covFull Boolean (default TRUE) indicating the covariance should include every estimated population parameter – the structural and residual thetas plus the Omega variance/covariance elements – named om.<eta> / cov.<eta>.<eta>. When FALSE

the legacy structural-theta-only covariance is reported. Ignored by `covMethod="sa"`, which is always full.

`nSaCov` Number of iterations in the dedicated stochastic-approximation covariance phase used by `covMethod="sa"`, a whole number of at least 1 (default `500`). These iterations run at the converged estimate (parameters frozen) and only resimulate the individual parameters to build the observed Fisher information; a larger value gives a less noisy covariance. Ignored by other covariance methods.

`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 3; 1 is the Laplacian likelihood; at most 25)

`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.

`literalFix` boolean, substitute fixed population values as literals and re-adjust ui and parameter estimates after optimization; Default is 'TRUE'.

`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` Type of additive-plus-proportional error: '"combined1"', where standard deviations add:

$$y = f + (a + b \times f^c) \times \varepsilon$$

; or '"combined2"', where variances add:

$$y = f + \sqrt{a^2 + b^2 \times f^{2\times c}} \times \varepsilon$$

. Here y = observed, f = predicted, a = additive sd, b = proportional/power sd, c = power exponent (1 in the proportional case).

`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.

`indTolRelax` When 'TRUE' (default), only subjects whose ODE solve produced NaN/Inf have their tolerances relaxed, and the relaxed tolerance persists across optimizer calls (sticky). When 'FALSE', all subjects have their tolerances relaxed on each retry and tolerances are reset afterward.

`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. This is `NULL` by default, and while it is `NULL` it has no effect at all: `rxSolve()` uses the standard `atol`/`rtol` (and the standard sensitivity and steady-state tolerances). `sigdig` only changes a tolerance when you ask for it explicitly.

When it is supplied, the tolerances are derived with one solver-independent formula – the same for stiff, non-stiff and auto-switching solvers. The `rtol` exponent IS `sigdig` and `atol` sits three orders below it:

- `rtol` = `10^(-sigdig)`, `atol` = `10^(-sigdig-3)`
- the sensitivity tolerances match the main solve, so `rtolSens` = `rtol` and `atolSens` = `atol` (gradients and covariances are built from them)
- the steady-state tolerances run one order looser than the corresponding main tolerance, so `ssRtol` = `ssRtolSens` = `10*rtol` and `ssAtol` = `ssAtolSens` = `10*atol`

Each of these is set only when you did not pass that tolerance yourself; a tolerance you supply always wins. Because they are resolved independently, an explicit `atol`/`rtol` overrides the main solve but does *not* propagate to the sensitivity or steady-state tolerances – set those directly if you need them changed too.

This mapping matches how `nlmixr2est` derives solver tolerances from its optimization `sigdig`, so a `sigdig` used for estimation and the same `sigdig` used for a plain `rxSolve()` mean the same thing. Note it is keyed to `sigdig` as a request for that many significant digits, and is looser than the `atol`/`rtol` defaults

for small `sigdig` – at `sigdig = 4` it gives `rtol = 1e-4` against a default `rtol = 1e-6`. Raise `sigdig`, or set `atol`/`rtol` directly, when you want a tighter solve.

`sigdigTable` Significant digits in the final output table. If not specified ('NULL'), it defaults to 'sigdig'.

`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.

`muRefCovAlg` This controls if algebraic expressions that can be mu-referenced are treated as mu-referenced covariates by:

1. Creating a internal data-variable 'nlmixrMuDerCov#' for each algebraic mu-referenced expression
2. Change the algebraic expression to 'nlmixrMuDerCov# * mu_cov_theta'
3. Use the internal mu-referenced covariate for saem
4. After optimization is completed, replace 'model()' with old 'model()' expression
5. Remove 'nlmixrMuDerCov#' from nlmix2 output

In general, these covariates should be more accurate since it changes the system to a linear compartment model. Therefore, by default this is 'TRUE'.

`handleUninformativeEtas` boolean that tells nlmixr2's saem to calculate uninformative etas and handle them specially (default is 'TRUE').

`revisitUninformativeEtas` boolean (default 'FALSE'); when 'TRUE' the uninformative-eta test is run a second time at the end of burn-in and that verdict replaces the first. The test asks whether perturbing an eta moves the subject's prediction, and it is otherwise only run at the *initial* estimates – so those decide, for the whole fit, which etas 'saem' may sample. Re-running it once 'theta' has moved decides on the estimates the data supports instead. It reuses the fit's own model evaluation, so it costs a few extra solves at a single iteration and leaves the random number stream unchanged; where it changes no verdict the fit is identical. It is off by default because the two verdicts only disagree when 'theta' moved a long way during burn-in, which usually means it has not settled – and the second verdict can freeze an eta for the rest of the fit. Skipped for mixture models and when 'handleUninformativeEtas=FALSE'.

`iovXform` Transformation used on the diagonal of the IOV: one of `"sd"`, `"var"`, `"logsd"`, or `"logvar"`. This parameterizes the magnitude theta of the `"theta"` `iovMethod`; it has no effect under `"omega"`, where the magnitude is fixed at one and the variability is carried by the omega block itself.

`iovMethod` How inter-occasion variability ('iov.x ~ v | occ') is estimated. `"theta"` uses the shared pre-processing rewrite, which carries the magnitude as a population parameter multiplying per-occasion unit-variance etas (the form every other estimation method uses). `"twoLevel"` (the default) keeps the occasion term as a second variance component and estimates it with the closed-form updates of

Panhard and Samson (2009), which is what the SAEM kernel is built around for every other variance. `"collapsed"` additionally uses that paper's own sampler: one parameter per occasion carrying `mu + b_i + c_ik` together, with a compound-symmetric prior, instead of sampling `b_i` and `c_ik` separately. It targets the same estimates and differs only in how well the chain mixes. A model the newer handling does not cover falls back to `"theta"` on its own.

`boundedTransform`

When 'TRUE' (default), bounded parameters are transformed for unbounded optimization methods and back-transformed for final estimates. 'FALSE' optimizes on the original scale with bounds passed to the optimizer. 'NA' transforms for optimization but skips the final back-transform.

`eventSens` Controls how dosing/event-parameter ('alag', 'F', 'rate', 'dur') sensitivities are computed for THETA/ETA gradients: '"jump"' (default) uses rxode2's analytic event sensitivities; '"fd"' uses the legacy finite-difference behavior. Also gates the analytic moving-boundary correction for a modeled 'alag()'/'f()' on a 'linCmt()' compartment; set '"fd"' if that model infuses a dose into the lagged/scaled compartment (rxode2/rxode2#1236), or if the regimen also doses an *unlagged/unscaled* compartment alongside the lagged/scaled one – a common design for estimating 'f()' from paired IV+oral data (rxode2/rxode2#1237).

`mixProbMethod` For mixture models ('mix()', more than one component), stabilizes the mixing-probability estimate against collapsing onto a single component (the responsibility used to update it is itself weighted by the current mixing probability, which can create a runaway feedback loop). Three options:

* '"regress"' (default): treat per-subject mixture membership as a fixed regressor. Each subject is hard-classified to a component up front, held fixed, and fed into the solve (via the existing mixture-index regressor), skipping the per-iteration soft-EM responsibility step entirely. Avoids the responsibility feedback loop / collapse by construction and is lower-bias; on heavily overlapping components it is higher-variance (an early misclassification is not revisited), so prefer '"regularized"' when membership is genuinely uncertain.

* '"regularized"': blend 'mixProbPriorN' pseudo-subjects, distributed per the initial mixing probability, into the responsibility average each iteration (Dirichlet/MAP-EM-style). Prevents collapse even in difficult cases, at the cost of some bias toward the initial guess; may need larger 'nBurn'/'nEm'.

* '"annealed"': give the mixing-probability update its own decaying step-size schedule ('mixProbStepExp') instead of the full-replacement step used during 'nBurn'. Lower bias, but does not by itself fix a systematic (non-noise-driven) collapse.

`mixProbStepExp` Only used when 'mixProbMethod="annealed"'. Decay exponent for the mixing-probability step size ('1/iteration^mixProbStepExp'), applied from iteration 1. Default 1; smaller values decay more slowly.

`mixProbPriorN` Only used when 'mixProbMethod="regularized"'. Number of pseudo-subjects blended into the responsibility average each iteration. Larger values are more robust to collapse but bias the estimate more and need more 'nBurn'/'nEm'. Default 20.

`mixSampleMethod`

For mixture models with per-component etas (split-ETA, e.g. 'cl <- mix(tcl1

+ eta.cl1, p1, tcl2 + eta.cl2)‘), controls the MCMC/sufficient-statistic architecture for the individual random effects, independent of ‘mixProbMethod‘. BSV (‘$omega‘) for split components is unreliable under ‘"parallel"‘ regardless of ‘mixProbMethod‘.

* ‘"parallel"‘ (default): one full MCMC chain per component per subject per iteration, blended post hoc by responsibility. Mirrors NONMEM’s ‘$MIX‘ and correctly estimates BSV shared across components, but cannot cleanly separate per-component BSV for split-ETA models (each "wrong-hypothesis" chain still explores its non-owned column(s) as unconstrained prior noise).

* ‘"msaem"‘ (experimental): the MSAEM algorithm (Lavielle & Mbogning 2014), as used by Monolix. Simulates one random-effects trajectory per subject per iteration (label marginalized out via a closed-form responsibility) instead of parallel per-component chains, so no post-hoc blending is needed. Not compute-matched to ‘"parallel"‘ at equal ‘nmc‘ – set ‘nmc‘ to roughly ‘nMix‘ times its default for a fair comparison. Uses a model-aware stratified initialization for split-ETA components that reliably achieves full theta/fixed-effect separation. Split-ETA BSV recovery is improved (two numerical bugs fixed: an ‘IGamma2_phi1‘ blowup that locked variance to exactly zero, and an inverted responsibility sign) but still not reliable – it often settles at a safety-floor value rather than the true variance. Prefer ‘"parallel"‘ unless specifically evaluating this method.

`nonMuTheta` Controls how a population ‘theta‘ that is not mu-referenced (does not appear linearly with an eta – the SAEM ‘phi0‘ fixed effects) is estimated.

* ‘"regress"‘ (default): keep the parameter as a plain directly-estimated ‘theta‘ regressor. Each iteration ‘phi0‘ is estimated by a bounded direct optimization of the observation likelihood (robust coordinate descent within a local trust region, honoring the ‘ini‘ bounds), held fixed rather than drawn stochastically with a shrinking variance. This recovers population parameters that have no associated random effect more accurately, at some extra runtime (the objective re-solves the ODE).

* ‘"eta"‘: the historic SAEM treatment (the parameter is carried through the stochastic ‘phi0‘ block).

`nonMuThetaOpt` Optimizer for the ‘nonMuTheta="regress"‘ refinement. ‘"newuoa"‘ (default) and ‘"nelderMead"‘ each run one clamped multivariate optimization over all free ‘phi0‘ coordinates under the ‘nonMuThetaMaxEval‘ evaluation budget; ‘"optimize"‘ is the historic coordinate descent with R’s golden-section ‘optimize()‘. When the non-mu thetas drive the ODE, every objective evaluation is a full re-solve, so this refinement can dominate the run time; the multivariate options spend a much smaller fixed budget and account for the coupling between coordinates that coordinate descent cannot see.

`nonMuThetaSweeps`

Number of coordinate-descent sweeps per refinement for ‘nonMuThetaOpt="optimize"‘ (default 2).

`nonMuThetaMaxEval`

Objective-evaluation budget of one refinement for ‘nonMuThetaOpt="newuoa"‘ or ‘"nelderMead"‘ (default 25); 0 means ten evaluations per free non-mu theta instead. ‘"newuoa"‘ needs ‘2n+3‘ evaluations to build its first quadratic model, and is raised to that when the budget is smaller.

`nonMuThetaTol` Convergence tolerance of the ‘nonMuTheta="regress"‘ refinement (‘newuoa‘’s ‘rhoend‘, the nelder-mead relative objective tolerance, or the ‘optimize()‘ ‘tol‘).

`nonMuThetaEvery`
Run the ‘nonMuTheta="regress"‘ refinement every ‘nonMuThetaEvery‘ iterations instead of every iteration (default 1). In between, ‘phi0‘ keeps its last refined value.

`phi1ThetaMaxEval`
Objective-evaluation budget (bobyqa’s ‘maxfun‘) of one general-likelihood phi1 (mu-referenced theta) refinement (default 50). Applies only when the fit has a general-likelihood (‘ll()‘/‘t()‘/ ‘cauchy()‘) endpoint; ignored otherwise. Each evaluation solves the FOCEi-style phi1-inner sensitivity model across every chain-replicated row, so it is the dominant per-iteration cost while this refinement is active – larger than ‘nonMuThetaMaxEval‘’s default because the objective is also more expensive per call.

`phi1ThetaEvery` Run the general-likelihood phi1 refinement every ‘phi1ThetaEvery‘ iterations instead of every iteration (default 1). In between, phi1’s mu keeps its last refined value (mirrors ‘nonMuThetaEvery‘).

`phi1Hessian` Boolean (default ‘FALSE‘); include the Laplace ‘log|H|‘ eta-Hessian correction in the general-likelihood phi1 refinement’s objective (‘-2*loglik + log|H|‘ instead of plain ‘-2*loglik‘). Opt-in: an ablation check found the correction was not what fixed a diverging Gaussian general-likelihood twin – the direct bobyqa optimization of the exact joint log-likelihood was already sufficient on its own, replacing the historic noisy Robbins-Monro stochastic-approximation recursion over MCMC-sampled phi. For a heavy-tailed endpoint (‘t()‘/‘cauchy()‘), ‘log|H|‘ can instead dominate the objective and cause divergence (its own curvature decays far from the data, unlike a normal’s), so it defaults off. Set ‘TRUE‘ only to experiment with the Hessian-corrected form; when it builds, the exact analytic eta-Hessian is used, else a finite-difference one (e.g. for ‘linCmt()‘).

`residWarmStart` Boolean (default ‘TRUE‘); warm-start the residual-error parameters from the observed per-endpoint moments at the initial predictions (additive SD from ‘sqrt(mean(err^2))‘, proportional SD from ‘sqrt(mean((err/f)^2))‘), the same moment estimate ‘est="npag"‘ uses. Gives the stochastic step a better starting residual scale. Set ‘FALSE‘ to start from the ‘ini‘-block residual values instead.

`censOption` Treatment of the second derivative for censored (M2/M3/M4/BLQ) observations in the FOCEI family. `"gauss"` (the default) keeps the historic uncensored Gauss-Newton curvature, matching common PMx tools; `"laplace"` uses the exact censored second derivative of the objective (a proper Laplace inner Hessian and analytic covariance). Accepted by `saemControl`/`nlmControl` for a uniform interface but inert there – SAEM (stochastic EM) has no Laplace inner Hessian, and NLM uses a finite-difference Hessian that already reflects censoring exactly.

`...` Other arguments to control SAEM.

Value

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

Author(s)

Wenping Wang & Matthew L. Fidler

References

Kuhn E, Lavielle M (2005). "Maximum likelihood estimation in nonlinear mixed effects models." Computational Statistics & Data Analysis, 49(4), 1020-1038. doi:10.1016/j.csda.2004.07.002

Jiang L, Roy A, Balasubramanian K, Davis D, Drusvyatskiy D, Na S (2025). "Online Covariance Estimation in Nonsmooth Stochastic Approximation." arXiv:2502.05305. doi:10.48550/arXiv.2502.05305

See Also

Other Estimation control: `foceiControl()`, `nlmixr2NlmeControl()`

`setCov` *Set the covariance type based on prior calculated covariances*

Description

Switches a completed fit's covariance to `method`. A previously computed covariance is re-installed from the cache; otherwise it is recomputed at the converged estimates: `"r,s"`/`"r"`/`"s"` by the finite-difference covariance step of the FOCEI family, `"analytic"` as the analytic observed information at the fit's estimates and ETAs, and `"sa"` (SAEM Louis FIM) / `"imp"` (importance-sampling Monte-Carlo) via the decoupled recompute engine (the latter two require a mixed-effects fit). When a covariance cannot be computed it is left unchanged (it is never silently downgraded to `"r,s"`).

Usage

```
setCov(fit, method, ...)

## Default S3 method:
setCov(fit, method, ...)

## S3 method for class 'analytic'
setCov(fit, method, ...)

## S3 method for class '`r,s`'
setCov(fit, method, control = rsControl(), ...)

## S3 method for class 'r'
setCov(fit, method, control = rsControl(), ...)

## S3 method for class 's'
setCov(fit, method, control = rsControl(), ...)

## S3 method for class 'sa'
```

```
setCov(fit, method, control = saControl(), ...)

## S3 method for class 'imp'
setCov(fit, method, control = impCovControl(), ...)
```

Arguments

fit	nlmixr2 fit
method	covariance method (see the covMethod argument of the control options for the choices)
...	arguments passed to the covariance method
control	options for the covariance method itself, only needed to change its defaults: rsControl() for "r,s", "r" and "s", saControl() for "sa" and impCovControl() for "imp"

Details

The finite-difference covariances differentiate the fit's marginal likelihood, in the fit's own approximation (its interaction): at every finite-difference step the ETAs are optimized again, starting from the fit's ETAs. So setCov(fit, "r,s") reproduces the "r,s" covariance a FOCEI-family fit computes during estimation, and on a fit of another method (saem, nlme, vae, emvi, fbvi) it is that covariance at the method's estimates.

Every focei covariance comes in two shapes (see covFull in foceiControl()), and both are named: "r,s", "r", "s" and "analytic" are the structural-theta block, while "r,s (full)", "r (full)", "s (full)" and "analytic (full)" are the full theta + residual sigma + Omega matrix. A focei fit computes both and caches the one it does not install, so swapping between them costs nothing. The shapes are not submatrices of one another on the finite-difference path – "s" inverts the theta block of the cross-product while "s (full)" takes the theta block of the full inverse, which also carries the Omega estimation uncertainty – so the standard errors differ. On the analytic path the assembly is always full and "analytic" is a submatrix of "analytic (full)", so the theta standard errors agree.

fit$covMethod names the installed covariance and names(fit$covList) the cached alternatives (the fit print shows both).

Value

Fit object with covariance updated

Adding a covariance method

setCov() is an S3 generic dispatched on method (without any " (full)" suffix), so another package (for example SIR or a bootstrap) can add a covariance by registering a method:

```
# registered with S3method(nlmixr2est::setCov, sir) in NAMESPACE
setCov.sir <- function(fit, method, control = sirControl(), ...) {
  # compute on the estimation scale, named like fit$cov
  mySirCovariance(fit, control)
}
```

A method that has options declares its own `control` holding only those options, so `setCov(fit, "sir")` uses the defaults and `setCov(fit, "sir", control = sirControl(...))` changes them. The method receives the fit and `method` (carrying dispatch classes; use `unclass(method)` for the plain name) and either returns a named covariance matrix, which `setCov()` checks for positive definiteness and installs as `method` (updating the standard errors and keeping the prior covariance in `fit$covList`), or installs the covariance itself and returns `NULL`. A mixture fit's matrix is rotated onto the probability scale unless it carries `attr(, "mixRotated")` set to `TRUE`. A method that cannot compute the covariance should `stop()`. `setCovAllMethods()` lists the available methods.

A method's options are recorded under the key `setCovOptions()` gives its control, by default the control itself. A covariance that depends on other state on the fit – one seeded from another covariance, say – adds a `setCovOptions()` method for its control class that puts that state in the key. A package can also let an already-computed result be installed with `setCov(fit) <- result` by adding a `setCovValue()` method for the result's class.

Each covariance remembers the options it was computed with (in `fit$env$covOptions`). A covariance already on the fit is reinstalled from `fit$covList` only when the requested options – the supplied `control`, or the method's default one – are the same; otherwise it is recomputed. A covariance computed during estimation used the fit's own settings.

Author(s)

Matt Fidler

See Also

`foceiControl()`, `saemControl()`

`setCov<-` *Install an already-computed covariance result on a fit*

Description

`setCov(fit) <- value` installs the covariance carried by `value` as the fit's covariance, the way `setCov(fit, method)` installs one it computes: the matrix is checked for positive definiteness, the standard errors are refreshed, and the prior covariance is kept in `fit$covList`. The options that produced `value` are recorded too, so a later `setCov(fit, method, control = ...)` reuses it only when the options it asks for are the same.

Usage

```
setCov(fit, method = NULL, ...) <- value
```

Arguments

`fit`	nlmixr2 fit
`method`	covariance-method name to install `value` as; `NULL` lets `setCovValue()` choose
`...`	passed to `setCovValue()`
`value`	a covariance matrix named like `fit$cov`, or a result with a `setCovValue()` method

Details

value is dispatched on through setCovValue(): a covariance matrix is installed as method (default "user") with no options, and other packages add methods for their own results (for example a SIR run), returning the matrix, the method name and the options.

Value

the fit, with its covariance updated

Author(s)

Matt Fidler

See Also

setCov(), setCovOptions()

Examples

```
## Not run:
setCov(fit) <- fit$cov * 2
setCov(fit, "doubled") <- fit$cov * 2

## End(Not run)
```

---

setCovAllMethods *List the covariance methods setCov() can compute*

Description

List the covariance methods setCov() can compute

Usage

```
setCovAllMethods()
```

Value

character vector of methods, including those registered by other packages

Examples

```
setCovAllMethods()
```

`setCovOptions` *The cache key of a covariance method's options*

Description

`setCov()` records the result of this generic for every covariance it computes, and reuses a cached covariance only when the key it asks for is `identical()` to the recorded one. The default key is the control object itself, as a plain list.

Usage

```
setCovOptions(control, fit, ...)

## Default S3 method:
setCovOptions(control, fit, ...)

## S3 method for class 'rsControl'
setCovOptions(control, fit, ...)
```

Arguments

`control`	covariance control object (for example `rsControl()`)
`fit`	nlmixr2 fit, or its environment when the key of a built-in method's defaults is taken
`...`	ignored

Details

A package whose covariance depends on more than its control – a covariance seeded from another covariance on the fit, say – adds a method for its control class that puts that state in the key, so a change in it recomputes the covariance instead of reinstalling a stale one. Options that do not change the result (parallel workers, say) can be left out.

Value

named list, compared with `identical()`

Author(s)

Matt Fidler

See Also

`setCov()`

Examples

```
setCovOptions(saControl(), NULL)
```

`setCovValue` *The covariance, method name and options a result installs*

Description

The generic behind `setCov(fit) <- value`. A method returns `list(cov =, method =, options =)`: the covariance named like `fit$cov`, the covariance-method name to install it as, and the options recorded for it (the same key `setCovOptions()` gives the control that would compute it, so `setCov()` can reuse it). It may also return `extra`, a named list of objects stored in the fit environment (so `fit$<name>` returns them) once the covariance is installed – the full result the covariance came from, say.

Usage

```
setCovValue(value, fit, method = NULL, ...)

## Default S3 method:
setCovValue(value, fit, method = NULL, ...)

## S3 method for class 'matrix'
setCovValue(value, fit, method = NULL, ...)
```

Arguments

`value`	the result to install
`fit`	nlmixr2 fit
`method`	requested covariance-method name, or `NULL`
`...`	ignored by the built-in methods

Value

`list(cov, method, options)`, optionally with `extra`

Author(s)

Matt Fidler

See Also

`setCov<-`

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. ‘"imp"‘ and ‘"impmap"‘ add an importance-sampling objective from an E-step-only run (‘nIter=0‘) at the fit’s estimates.

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. A matrix that is not finite is an error.

summary.saemFit *Print an SAEM model fit summary*

Description

Print an SAEM model fit summary

Usage

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

Arguments

object	a saemFit object
...	others

Value

a list

tableControl *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

Use addCwres to add CWRES/FOCEi objective function, or addNpde to add NPDE/EPRED columns.

Value

A list of table options for nlmixr2

Author(s)

Matthew L. Fidler

`trustControl` *Control for the trust estimation method in nlmixr2*

Description

‘est="trust"‘ is a trust-region Newton method for the population theta vector, backed by the ‘RcppTrust‘ package’s thread-safe ‘trust_solve_c()‘. Unlike every other nlm-family method (‘nlm‘/‘nlminb‘/‘bobyqa‘/‘newuoa‘/‘uobyqa‘/‘n1qn1‘/‘lbfgsb3c‘/‘optim‘), whose optimization loop lives in R (one ‘.Call()‘ per iteration), ‘trust‘’s entire loop runs inside a single ‘.Call()‘ – ‘RcppTrust‘ needs no R API, so there is no per-iteration R round-trip. It also supplies a full analytic-or-Shi21-finite-difference gradient AND a full finite-difference-of-the-gradient Hessian every iteration (there is no analytic outer-theta Hessian in this package), so each outer iteration costs roughly ‘ntheta‘ extra full population-gradient solves on top of the gradient solve itself – state this cost plainly, it is the price of true Newton-trust behavior rather than a derivative-free or quasi-Newton method. ‘trust‘ is unbounded, like ‘n1qn1‘/‘nlm‘: box constraints are handled upstream by ‘preProcessBoundedTransform.R‘, not by the optimizer itself.

Usage

```
trustControl(
  rinit = NULL,
  rmax = NULL,
  iterlim = 1000L,
  fterm = NULL,
  mterm = NULL,
  optimHessType = 1L,
  shi21maxHess = 20L,
  hessErr = NULL,
  hessianMethod = c("sr1", "fd", "bfgs", "bofill"),
  returnTrust = FALSE,
  stickyRecalcN = 4,
  maxOdeRecalc = 5,
  odeRecalcFactor = 10^(0.5),
  indTolRelax = TRUE,
  useColor = NULL,
  printNcol = NULL,
  print = 1L,
```

```
  normType = c("rescale2", "mean", "rescale", "std", "len", "constant"),
  scaleType = c("nlmixr2", "norm", "mult", "multAdd"),
  scaleCmax = 1e+05,
  scaleCmin = 1e-05,
  scaleC = NULL,
  scaleTo = 1,
  gradTo = 1,
  rxControl = NULL,
  optExpression = TRUE,
  sumProd = FALSE,
  literalFix = TRUE,
  literalFixRes = TRUE,
  addProp = c("combined2", "combined1"),
  eventSens = c("jump", "fd"),
  calcTables = TRUE,
  compress = FALSE,
  covMethod = c("r", "trust", ""),
  adjObf = TRUE,
  ci = 0.95,
  sigdig = 3,
  sigdigTable = NULL,
  boundedTransform = TRUE,
  ...
)
```

Arguments

`rinit`	Initial trust-region radius, in the SAME scaled-parameter space every nlm-family method (bobyqa included) optimizes in. ‘NULL‘ (default) derives it from the scaled starting vector using ‘minqa::bobyqa()‘’s own default-‘rhobeg‘ formula, ‘min(0.95, 0.2*max(abs(par.ini)))‘, so swapping ‘est="bobyqa"‘ for ‘est="trust"‘ on the same model starts from a comparable trust-region size.
`rmax`	Maximum trust-region radius, same scaled-parameter space. ‘NULL‘ (default) derives it as ‘8 * rinit‘ – the same growth-ceiling multiplier already used for ‘foceiControl(innerOpt="trust")‘’s per- subject eta trust region.
`iterlim`	Maximum number of ‘trust_solve_c()‘ iterations.
`fterm, mterm`	‘trust_solve_c()‘’s function-value and predicted- decrease convergence tolerances. ‘NULL‘ (default) uses ‘10^(-sigdig-2)‘, two orders tighter than ‘bobyqaControl()‘’s ‘rhoend‘ and ‘foceiControl()‘’s inner ‘epsilon‘ (both the plain ‘10^(-sigdig)‘), matching ‘foceiControl(trustFterm=, trustMterm=)‘ instead – the analogous tolerance for the other ‘RcppTrust‘-backed solve in this package. ‘mterm‘ defaults to ‘fterm‘ when not given separately.
`optimHessType`	Finite-difference type for the per-iteration outer Hessian: ‘1L‘ forward (default), ‘2L‘ central. Exposed explicitly (unlike bobyqa/n1qn1, where this is only ever set implicitly) because ‘trust‘ recomputes this Hessian every outer iteration, not once post-fit, making its accuracy/cost tradeoff far more consequential; a stiff or noisy model may benefit from ‘optimHessType=2‘.

`shi21maxHess` Maximum Shi (2021) adaptive-step-size iterations for the per-iteration outer Hessian.

`hessErr` Target relative error for the per-iteration outer Hessian's Shi (2021) step-size search. 'NULL' (default) uses '(.Machine$double.eps)^(1/3)', the same fallback '.nlmSetupEnv()' itself applies for every nlm-family method.

`hessianMethod` How the per-iteration outer Hessian is built. '"sr1"' (default) is the Symmetric Rank-1 quasi-Newton update (Nocedal & Wright, *Numerical Optimization*, 2nd ed., 2006, Eq. 6.24; Murtagh & Sargent, *Comput. J.* 13, 1970), built from consecutive outer iterations' gradients (already computed regardless of 'hessianMethod', so this adds no extra evaluations) – Nocedal & Wright's own recommendation for trust-region methods specifically, since (unlike BFGS) it is not forced positive definite, so it can represent indefinite curvature. '"fd"' instead recomputes the Hessian from scratch every outer iteration via 'nlmCalcHessian()''s Shi (2021) finite-difference-of-the-gradient (the 'optimHessType'/'shi21maxHess'/'hessErr' parameters above only apply to this method). '"bfgs"' is the damped BFGS update (Nocedal & Wright Procedure 18.2), always positive definite. '"bofill"' is Bofill's SR1/PSB blend (*J. Comput. Chem.* 15, 1-11, 1994). Every method seeds from one '"fd"'-style Hessian on the first outer iteration.

'"sr1"' was originally made default from a benchmark ('inst/benchmarks/benchmark-trust-outer.R') showing it ran faster with the same or slightly better accuracy than '"fd"'; that benchmark predated fixes for two real correctness bugs (issues #994 and #996) that independently distorted several of its models' results for EVERY 'hessianMethod' value alike, both upstream of Hessian construction (a 'scaleC' blowup for a near-zero starting gradient; 'est="trust"' missing the 'linCmt()'-to-ODE translation another nlm-family method needs) – so it was briefly reverted to '"fd"' pending confirmation. Re-run after both fixes, '"sr1"'/'"bofill"' track '"fd"' closely (median |objective diff| vs 'bobyqa' 1.53/1.55 vs '"fd"''s 1.55 across the corpus) and '"bfgs"' if anything tracks it slightly better (0.43) – confirming the earlier small '"sr1"'-vs-'"fd"' accuracy gap was, at least in part, noise from those two bugs, not a genuine difference between Hessian constructions for this OUTER problem, so '"sr1"' is restored as the default (faster, with no demonstrated accuracy cost for this problem).

This OUTER problem does not have the failure mode that keeps the analogous inner-problem option ('foceiControl(hessianMethod=)') defaulted to '"fd"': that inner Hessian's log-determinant feeds directly into the reported per-subject objective (not just the step), and a quasi-Newton estimate was shown to bias it on a real PK model. 'nlmTrustObjfun()''s reported value ('src/nlm.cpp') here is instead the plain log-likelihood, set before the Hessian is even touched, so a less-accurate 'hessianMethod' can only cost step quality/convergence speed, not silently bias the reported number – which is what the re-benchmark above confirms in practice.

`returnTrust` return the raw 'nlmTrustFit()' output list instead of the nlmixr2 fit.

`stickyRecalcN` The number of bad ODE solves before reducing the atol/rtol for the rest of the problem.

`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

`indTolRelax`
: When 'TRUE' (default), only subjects whose ODE solve produced NaN/Inf have their tolerances relaxed, and the relaxed tolerance persists across optimizer calls (sticky). When 'FALSE', all subjects have their tolerances relaxed on each retry and tolerances are reset afterward.

`useColor`
: Logical (or 'NULL') emit ANSI bold/color escapes in the iteration print. 'NULL' (default) defers to [crayon::has_color()].

`printNcol`
: Integer (or 'NULL') parameter columns per row before wrapping. 'NULL' (default) uses 'floor((getOption("width") - 23) / 12)'.

`print`
: Either a scalar print-frequency ('0' = suppress, '1' (default) = every evaluation, 'N' = every Nth), OR a pre-built [iterPrintControl()] object. Equivalent to 'iterPrintControl(every = print, ncol = printNcol, useColor = useColor)'.

`normType`
: Parameter normalization/scaling used to get scaled initial values for `scaleType`, of the form `Vscaled = (Vunscaled-C1)/C2` (see Feature Scaling; `rescale2` follows the OptdesX manual): `"rescale2"` scales all parameters to (-1, 1); `"rescale"` (min-max) scales to (0, 1); `"mean"` centers on the mean with range (0, 1); `"std"` standardizes by mean/sd; `"len"` scales to unit (Euclidean) length; `"constant"` performs no normalization (`C1=0`, `C2=1`).

`scaleType`
: The scaling scheme for nlmixr2: `"nlmixr2"` (default) scales as `(current-init)*scaleC[i] + scaleTo`, with `scaleTo` from `normType` and scales from `scaleC`; `"norm"` uses the simple scaling from `normType`; `"mult"` scales multiplicatively as `current/init*scaleTo`; `"multAdd"` scales linearly (`(current-init)+scaleTo`) for parameters in an exponential block (e.g. `exp(theta)`) and multiplicatively otherwise.

`scaleCmax`
: Maximum value of the scaleC to prevent overflow.

`scaleCmin`
: Minimum value of the scaleC to prevent underflow.

`scaleC`
: Scaling constant used with `scaleType="nlmixr2"`; when not specified, chosen by parameter type to keep gradient sizes similar on a log scale: '1' for exp()-transformed/power/boxCox/ yeoJohnson parameters, '0.5*abs(est)' for additive/proportional/ lognormal error parameters, 'abs(1/digamma(est+1))' for factorials, and 'log(abs(est))*abs(est)' for log-scale parameters. May be set explicitly per parameter if these defaults don't apply well.

`scaleTo`
: Scale the initial parameter estimate to this value. By default this is 1. When zero or below, no scaling is performed.

`gradTo`
: this is the factor that the gradient is scaled to before optimizing. This only works with scaleType="nlmixr2".

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

`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`.

`literalFix`
: boolean, substitute fixed population values as literals and re-adjust ui and parameter estimates after optimization; Default is 'TRUE'.

`literalFixRes` boolean, substitute fixed population values as literals and re-adjust ui and parameter estimates after optimization; Default is 'TRUE'.

`addProp` Type of additive-plus-proportional error: '"combined1"', where standard deviations add:

$$y = f + (a + b \times f^c) \times \varepsilon$$

; or '"combined2"', where variances add:

$$y = f + \sqrt{a^2 + b^2 \times f^{2 \times c}} \times \varepsilon$$

. Here y = observed, f = predicted, a = additive sd, b = proportional/power sd, c = power exponent (1 in the proportional case).

`eventSens` Controls how dosing/event-parameter ('alag', 'F', 'rate', 'dur') sensitivities are computed for THETA/ETA gradients: '"jump"' (default) uses rxode2's analytic event sensitivities; '"fd"' uses the legacy finite-difference behavior. Also gates the analytic moving-boundary correction for a modeled 'alag()'/'f()' on a 'linCmt()' compartment; set '"fd"' if that model infuses a dose into the lagged/scaled compartment (rxode2/rxode2#1236), or if the regimen also doses an *unlagged/unscaled* compartment alongside the lagged/scaled one – a common design for estimating 'f()' from paired IV+oral data (rxode2/rxode2#1237).

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

`compress` Should the object have compressed items

`covMethod` Method for calculating the covariance. '"r"' (the default) uses a finite-difference Hessian at the estimates: with 'hessianMethod = "fd"', central differences of the analytical gradient over 'nlmixr2Gill83()' steps, otherwise 'nlmixr2Hess()''s. '"trust"' uses the last outer iteration's Hessian whatever 'hessianMethod' built it (a quasi-Newton one by default); '""' skips the covariance step.

`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. One value drives, with a single consistent formula, the inner/outer optimizer convergence tolerance (`10^-sigdig`), the boundary check tolerance (`5*10^(-sigdig+1)`), and the ODE solver tolerances: the `rtol` exponent IS `sigdig` and `atol` sits three orders below, so `rtol = 10^-sigdig`, `atol = 10^(-sigdig-3)` for every solver (stiff, non-stiff or auto-switching). The sensitivity (`atolSens`/`rtolSens`) tolerances match the main solve (the outer gradient and covariance are built from them); the steady-state (`ssAtol`/`ssRtol`) tolerances run one order looser. Keying the optimizer to the same `10^-sigdig` means it converges to exactly the precision the solve supports. At the default `sigdig = 3` this is `atol = 1e-6`, `rtol = 1e-3`.

`sigdigTable` Significant digits in the final output table. If not specified ('NULL'), it defaults to 'sigdig'.

`boundedTransform`
When 'TRUE' (default), bounded parameters are transformed for unbounded optimization methods and back-transformed for final estimates. 'FALSE' optimizes on the original scale with bounds passed to the optimizer. 'NA' transforms for optimization but skips the final back-transform.

`...` Ignored parameters

Value

trust control structure

Author(s)

Matthew L. Fidler

Examples

```
# A logit regression example with emax model

dsn <- data.frame(i=1:1000)
dsn$time <- exp(rnorm(1000))
dsn$DV=rbinom(1000,1,exp(-1+dsn$time)/(1+exp(-1+dsn$time)))

mod <- function() {
 ini({
   E0 <- 0.5
   Em <- 0.5
   E50 <- 2
   g <- fix(2)
 })
 model({
   v <- E0+Em*time^g/(E50^g+time^g)
   ll(bin) ~ DV * v - log(1 + exp(v))
 })
}

fit <- nlmixr(mod, dsn, est="trust")

print(fit)

# you can also get the raw trust output with fit$trust

fit$trust
```

---

uobyqaControl *Control for uobyqa estimation method in nlmixr2*

Description

Control for uobyqa estimation method in nlmixr2

Usage

```
uobyqaControl(
  npt = NULL,
  rhobeg = NULL,
  rhoend = NULL,
  iprint = 0L,
  maxfun = 100000L,
  returnUobyqa = FALSE,
  stickyRecalcN = 4,
  maxOdeRecalc = 5,
  odeRecalcFactor = 10^(0.5),
  indTolRelax = TRUE,
  useColor = NULL,
  printNcol = NULL,
  print = 1L,
  normType = c("rescale2", "mean", "rescale", "std", "len", "constant"),
  scaleType = c("nlmixr2", "norm", "mult", "multAdd"),
  scaleCmax = 1e+05,
  scaleCmin = 1e-05,
  scaleC = NULL,
  scaleTo = 1,
  rxControl = NULL,
  optExpression = TRUE,
  sumProd = FALSE,
  literalFix = TRUE,
  literalFixRes = TRUE,
  addProp = c("combined2", "combined1"),
  calcTables = TRUE,
  compress = FALSE,
  covMethod = c("r", ""),
  adjObf = TRUE,
  ci = 0.95,
  sigdig = 3,
  sigdigTable = NULL,
  boundedTransform = TRUE,
  eventSens = c("jump", "fd"),
  ...
)
```

Arguments

npt	Number of points for bobyqa's quadratic approximation to the objective; must be in '[n+2, (n+1)(n+2)/2]'. Defaults to '2*n + 1'. (bobyqa)
rhobeg	Initial trust region radius for the bobyqa outer optimizer (with 'rhoend', must satisfy '0 < rhoend < rhobeg'). Default '0.2' (20 'abs(upper-lower)/2'. (bobyqa)
rhoend	Final trust region radius. If not defined, '10^(-sigdig)' is used. (bobyqa)
iprint	Controls amount of printing ('0'=none, '1'=start/end only, '2'=each new rho, '3'=every function evaluation, '>3'=every 'iprint' evaluations). Default '0'.

`maxfun` The maximum allowed number of function evaluations. If this is exceeded, the method will terminate.

`returnUobyqa` return the uobyqa output instead of the nlmixr2 fit

`stickyRecalcN` The number of bad ODE solves before reducing the atol/rtol for the rest of the problem.

`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

`indTolRelax` When 'TRUE' (default), only subjects whose ODE solve produced NaN/Inf have their tolerances relaxed, and the relaxed tolerance persists across optimizer calls (sticky). When 'FALSE', all subjects have their tolerances relaxed on each retry and tolerances are reset afterward.

`useColor` Logical (or 'NULL') emit ANSI bold/color escapes in the iteration print. 'NULL' (default) defers to [crayon::has_color()].

`printNcol` Integer (or 'NULL') parameter columns per row before wrapping. 'NULL' (default) uses 'floor((getOption("width") - 23) / 12)'.

`print` Either a scalar print-frequency ('0' = suppress, '1' (default) = every evaluation, 'N' = every Nth), OR a pre-built [iterPrintControl()] object. Equivalent to 'iterPrintControl(every = print, ncol = printNcol, useColor = useColor)'.

`normType` Parameter normalization/scaling used to get scaled initial values for `scaleType`, of the form `Vscaled = (Vunscaled-C1)/C2` (see Feature Scaling; `rescale2` follows the OptdesX manual): `"rescale2"` scales all parameters to (-1, 1); `"rescale"` (min-max) scales to (0, 1); `"mean"` centers on the mean with range (0, 1); `"std"` standardizes by mean/sd; `"len"` scales to unit (Euclidean) length; `"constant"` performs no normalization (`C1=0`, `C2=1`).

`scaleType` The scaling scheme for nlmixr2: `"nlmixr2"` (default) scales as `(current-init)*scaleC[i] + scaleTo`, with `scaleTo` from `normType` and scales from `scaleC`; `"norm"` uses the simple scaling from `normType`; `"mult"` scales multiplicatively as `current/init*scaleTo`; `"multAdd"` scales linearly (`(current-init)+scaleTo`) for parameters in an exponential block (e.g. `exp(theta)`) and multiplicatively otherwise.

`scaleCmax` Maximum value of the scaleC to prevent overflow.

`scaleCmin` Minimum value of the scaleC to prevent underflow.

`scaleC` Scaling constant used with `scaleType="nlmixr2"`; when not specified, chosen by parameter type to keep gradient sizes similar on a log scale: '1' for exp()-transformed/power/boxCox/ yeoJohnson parameters, '0.5*abs(est)' for additive/proportional/lognormal error parameters, 'abs(1/digamma(est+1))' for factorials, and 'log(abs(est))*abs(est)' for log-scale parameters. May be set explicitly per parameter if these defaults don't apply well.

`scaleTo` Scale the initial parameter estimate to this value. By default this is 1. When zero or below, no scaling is performed.

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

`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`.

`literalFix` boolean, substitute fixed population values as literals and re-adjust ui and parameter estimates after optimization; Default is 'TRUE'.

`literalFixRes` boolean, substitute fixed population values as literals and re-adjust ui and parameter estimates after optimization; Default is 'TRUE'.

`addProp` Type of additive-plus-proportional error: '"combined1"', where standard deviations add:

$$y = f + (a + b \times f^c) \times \varepsilon$$

; or '"combined2"', where variances add:

$$y = f + \sqrt{a^2 + b^2 \times f^{2 \times c}} \times \varepsilon$$

. Here y = observed, f = predicted, a = additive sd, b = proportional/power sd, c = power exponent (1 in the proportional case).

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

`compress` Should the object have compressed items

`covMethod` Method for calculating the covariance. `"r"` (the default) uses nlmixr2's `nlmixr2Hess()` Hessian; `""` skips the covariance step.

`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. One value drives, with a single consistent formula, the inner/outer optimizer convergence tolerance (`10^-sigdig`), the boundary check tolerance (`5*10^(-sigdig+1)`), and the ODE solver tolerances: the `rtol` exponent IS `sigdig` and `atol` sits three orders below, so `rtol = 10^-sigdig`, `atol = 10^(-sigdig-3)` for every solver (stiff, non-stiff or auto-switching). The sensitivity (`atolSens`/`rtolSens`) tolerances match the main solve (the outer gradient and covariance are built from them); the steady-state (`ssAtol`/`ssRtol`) tolerances run one order looser. Keying the optimizer to the same `10^-sigdig` means it converges to exactly the precision the solve supports. At the default `sigdig = 3` this is `atol = 1e-6`, `rtol = 1e-3`.

`sigdigTable` Significant digits in the final output table. If not specified ('NULL'), it defaults to 'sigdig'.

`boundedTransform`

When 'TRUE' (default), bounded parameters are transformed for unbounded optimization methods and back-transformed for final estimates. 'FALSE' optimizes on the original scale with bounds passed to the optimizer. 'NA' transforms for optimization but skips the final back-transform.

eventSens	Controls how dosing/event-parameter (‘alag‘, ‘F‘, ‘rate‘, ‘dur‘) sensitivities are computed for THETA/ETA gradients: ‘"jump"‘ (default) uses rxode2’s analytic event sensitivities; ‘"fd"‘ uses the legacy finite-difference behavior. Also gates the analytic moving-boundary correction for a modeled ‘alag()‘/‘f()‘ on a ‘linCmt()‘ compartment; set ‘"fd"‘ if that model infuses a dose into the lagged/scaled compartment (rxode2/rxode2#1236), or if the regimen also doses an *unlagged/unscaled* compartment alongside the lagged/scaled one – a common design for estimating ‘f()‘ from paired IV+oral data (rxode2/rxode2#1237).
...	Ignored parameters

Value

uobyqa control structure

Author(s)

Matthew L. Fidler

Examples

```
# A logit regression example with emax model

dsn <- data.frame(i=1:1000)
dsn$time <- exp(rnorm(1000))
dsn$DV=rbinom(1000,1,exp(-1+dsn$time)/(1+exp(-1+dsn$time)))

mod <- function() {
 ini({
   E0 <- 0.5
   Em <- 0.5
   E50 <- 2
   g <- fix(2)
 })
 model({
   v <- E0+Em*time^g/(E50^g+time^g)
   ll(bin) ~ DV * v - log(1 + exp(v))
 })
}

fit2 <- nlmixr(mod, dsn, est="uobyqa")

print(fit2)

# you can also get the nlm output with fit2$nlm

fit2$uobyqa

# The nlm control has been modified slightly to include
# extra components and name the parameters
```

vaeControl *Control for vae (variational autoencoder) estimation method in nlmixr2*

Description

Variational-autoencoder NLME estimation (Rohleff et al., CPT:PSP 2025): an LSTM encoder learns the individual posterior q(etaly) and an rxode2 decoder reconstructs the observations, trained on an ELBO / BICc-ELBO objective for simultaneous population-parameter estimation and covariate selection.

Usage

```
vaeControl(
  seed = 42L,
  itersBurnIn = 100L,
  klWarmup = 50L,
  gammaIter = 250L,
  iters = 300L,
  nGradStep = 5L,
  hiddenDim = 25L,
  learningRate = 0.005,
  burnInLearningRate = 0.008,
  sigma0 = NULL,
  covariateSelection = TRUE,
  pinCovariates = TRUE,
  muRefCovAlg = TRUE,
  shapes = c("power", "lin", "log", "identity", "center", "hockey"),
  covCenterType = c("median", "mean"),
  covCenter = NULL,
  catCutoff = 0.05,
  covSelectAlpha = 2,
  covSelectSmooth = TRUE,
  gammaSeries = c("reference", "saem"),
  sigma0Interp = c("sd", "reference"),
  residOptimize = c("twoStage", "moment", "optimize"),
  residRhoend = NULL,
  omegaUpdate = c("suffStat", "blend"),
  perNoCor = 0.75,
  inputScale = c("reference", "observed"),
  covSelectMethod = c("auto", "bnb", "l0learn"),
  covSelectMaxExact = 17L,
  covSelectColinearCut = .vaeColinearCut,
  covSelectPhiCor = c("suffStat", "mu", "resid"),
  covSelectPhiJoin = 0.9,
  covSelectPhiLeave = 0.8,
  covSelectPhiMaxDim = 4L,
```

```
  bnbStrategy = c("lifo", "fifo", "lc"),
  parEncoderBackward = !isTRUE(getOption("nlmixr2.identical", FALSE)),
  nonMuTheta = c("regress", "grad", "eta", "fix", "none"),
  nonMuEtaOmega = 0.01,
  mStepObjective = c("outer", "elbo"),
  likelihood = c("focei", "foce", "focep", "laplace"),
  objf = c("importanceSampling", "linear"),
  nIsSample = 3000L,
  returnVae = FALSE,
  print = 1L,
  useColor = NULL,
  printNcol = NULL,
  covMethod = c("r,s", "analytic", "r", "s", "sa", "imp", ""),
  optExpression = TRUE,
  sumProd = FALSE,
  literalFix = TRUE,
  literalFixRes = TRUE,
  addProp = c("combined2", "combined1"),
  calcTables = TRUE,
  compress = FALSE,
  adjObf = TRUE,
  ci = 0.95,
  sigdig = 3,
  sigdigTable = NULL,
  rhoend = NULL,
  stickyRecalcN = 4,
  maxOdeRecalc = 5,
  odeRecalcFactor = 10^(0.5),
  outerStickyRecalcN = 4,
  fdIndividualStep = TRUE,
  fdOutlierZ = 3.5,
  fdOutlierScale = TRUE,
  fdRefine = c("chartrand", "lanczos", "richardson"),
  fdChartrandAll = FALSE,
  fdOutlierAny = FALSE,
  outerMaxOdeRecalc = 5,
  outerOdeRecalcFactor = 10^(0.5),
  indTolRelax = TRUE,
  eventSens = c("jump", "fd"),
  rxControl = NULL,
  ...
)
```

Arguments

seed	Random seed for the VAE training (encoder init, Adam, reparameterization sampling); default 42. Training is stochastic, so a fixed seed makes every fit reproducible.

`itersBurnIn` Number of burn-in iterations (encoder-only, tiny KL weight) before the main EM phase.

`klWarmup` Number of KL-annealing iterations over which the KL weight is ramped from a small value to 1 (prevents posterior collapse).

`gammaIter` Number of main iterations before the EMA-smoothing phase of the population-parameter update begins.

`iters` Total number of main-loop iterations (after burn-in).

`nGradStep` Number of Adam gradient steps per EM outer iteration (the reference ‘L_iter‘).

`hiddenDim` LSTM hidden dimension (the reference ‘h_dim‘).

`learningRate` Adam learning rate used in the main training phase.

`burnInLearningRate`
Adam learning rate used during burn-in.

`sigma0` Encoder prior standard deviation(s) at initialization (a small value giving a sharp initial posterior). ‘NULL‘ uses a small default per individual parameter. This is distinct from the ‘ini()‘ omega.

`covariateSelection`
When ‘TRUE‘ (default) perform automated BICc-ELBO covariate selection during training; when ‘FALSE‘ fit only the covariate structure written in the model. In the ‘FALSE‘ case the model-declared covariate coefficients (both linear ‘beta*WT‘ effects and transformed ones such as ‘beta*log(WT/70)‘) are estimated in place by the regress M-step regardless of ‘nonMuTheta‘; a ‘ini(... ~ fix())‘ coefficient stays fixed.

`pinCovariates` When ‘TRUE‘ (default) and the model already declares covariate effects, restrict the automatic covariate selection to only the covariate/parameter pairs written in the model – the branch-and-bound search may still drop a declared covariate, but can never add one the model did not specify. A declared covariate that is not a valid search candidate (time-varying, or a raw-linear form that does not match the ‘log‘/centered encoding) is estimated in place by the regress M-step instead, with a note in ‘$runInfo‘. When the model declares no covariates there is nothing to pin and the full search runs. Has no effect when ‘covariateSelection‘ is ‘FALSE‘.

`muRefCovAlg` When ‘TRUE‘ (default) an algebraic/centered covariate effect written in the model (e.g. ‘wt.cl*(WT/70)‘ or ‘wt.cl*log(WT/70)‘) is handled as a mu2/mu3 reference: the covariate expression – including its centering – is evaluated into an internal ‘nlmixrMuDerCov#‘ data column and the model uses the linear ‘wt.cl*nlmixrMuDerCov#‘ form during fitting, so the VAE covariate search never re-centers it. The original expression is restored in the reported model.

`shapes` Which parameterizations ("shapes") of a continuous covariate the automatic search may consider, using the same vocabulary as ‘nlmixr2scm::runSCM()‘: ‘"power"‘ (‘beta*log(COV/ctr)‘), ‘"lin"‘ (‘beta*(COV - ctr)‘), ‘"log"‘ (‘beta*log(COV)‘), ‘"identity"‘ (‘beta*COV‘) and ‘"center"‘ (‘beta*(COV/ctr)‘). ‘"hockey"‘ is a two-armed piecewise linear relationship knotted at the centering value, written as ‘beta.low*(COV < ctr)*(COV - ctr) + beta.hi*(COV >= ctr)*(COV - ctr)‘; both arms enter or neither does, and it is skipped for a covariate with fewer than ‘catCutoff‘ of the subjects on one side of the knot. At most one shape of

a covariate may enter a given parameter. Because the selection objective is an ordinary least squares fit with a free intercept, ‘"power"‘/‘"log"‘ span the same model, as do ‘"lin"‘/‘"identity"‘/‘"center"‘; the shape therefore decides how an accepted relationship is written back, and when several eligible shapes span the same model the one listed first wins. ‘"hockey"‘ spans a strictly larger model than the linear shapes and costs two coefficients rather than one.

May also be a **list**, whose elements are dispatched individually so the two forms mix freely: an element named by covariate (‘WT = "power"‘) is shorthand for the covariate-wide rule ‘list(covar = "WT", shapes = "power")‘, and a ‘list(var=, covar=, shapes=)‘ element restricts one parameter/covariate pair. The most specific rule wins – ‘var‘+‘covar‘ beats ‘covar‘, which beats ‘var‘, which beats a rule naming neither – and ties go to the rule listed last.

In the list form, **naming a covariate also puts it in the search**. ‘fixCov = TRUE‘ (the default, given as an element of the list) fixes the searched set to exactly the covariates named, so ‘shapes = list(WT = "power")‘ searches ‘WT‘ and nothing else. Add ‘fixCov = FALSE‘ to restrict parameterizations without restricting the search, which is what the list form meant previously. A shape value of ‘TRUE‘ means "eligible, default shapes", and is how a categorical covariate is named (‘list(WT = "power", SEX = TRUE)‘) since a categorical takes no parameterization. A ‘var‘-only rule makes every covariate eligible on that parameter alone; a rule naming neither ‘var‘ nor ‘covar‘ contradicts ‘fixCov = TRUE‘ and is an error. A character vector names no covariate, so ‘fixCov‘ does not apply and every covariate stays searchable.

‘fixCov‘ is ignored when the model itself declares covariate effects: that already restricts the search (see ‘pinCovariates‘) and the declaration is the more specific statement. The disagreement is reported in ‘$runInfo‘, as is every covariate ‘fixCov‘ excludes. Categorical covariates always enter as indicators and take no shape, but ‘fixCov‘ still governs whether they are searched at all.

`covCenterType` Statistic used to center a continuous covariate, ‘"median"‘ (default) or ‘"mean"‘, computed over subjects rather than rows.

`covCenter` Named numeric vector of centering values overriding ‘covCenterType‘ for those covariates, e.g. ‘c(WT = 70)‘. Names are matched case-insensitively.

`catCutoff` Minimum proportion of subjects a non-reference level must hold to get its own indicator. Rarer levels are lumped with the reference. Default ‘0.05‘; ‘0‘ tests every level.

`covSelectAlpha` Starting multiplier for the covariate-selection L0 penalty, ramped linearly from ‘covSelectAlpha‘ down to ‘1‘ over the ‘klWarmup‘ warmup iterations and held at ‘1‘ afterward (matching the reference implementation’s ‘linspace(alpha, 1, kl_iter)‘). Values ‘> 1‘ penalize covariate entry more heavily early in training; ‘1‘ disables the ramp.

`covSelectSmooth`

When ‘TRUE‘ (default) the covariate selection regresses the SAEM sufficient statistic – an exponential moving average of the posterior means, updated with the same gain as the M-step – rather than the current posterior means. This matches the reference implementation (Rohleff et al. 2025), which is the reason for the default. In practice it changes little: ‘gamma‘ is exactly 1 until ‘gammaIter‘, so the statistic equals the posterior mean for most of a run and

is averaged only over the closing tail. ‘FALSE‘ regresses the current posterior means.

`gammaSeries` Decaying step-size series used once the smoothing phase starts (after ‘gammaIter‘); the gain is 1 throughout the EM phase either way.

* ‘"reference"‘ (default): ‘1/(iter - gammaIter)‘, the textbook Kuhn-Lavielle series the reference implementation uses. The first smoothing step is still a full replacement, and the decay follows. * ‘"saem"‘: ‘1/(1 + iter - gammaIter)‘, the CONTINUATION form `saemControl()` uses – nlmixr2est’s SAEM builds its series so it continues rather than repeating a gain of 1, so the decay begins at ‘1/2‘. Select this to match the step-size convention of the other nlmixr2 estimation methods rather than the reference.

`sigma0Interp` How ‘sigma0‘ is turned into the encoder’s initial posterior spread. The encoder head emits ‘logSigma‘ and forms ‘diag(L) = exp(logSigma)‘, so ‘diag(L)‘ is the posterior standard deviation.

* ‘"sd"‘ (default): the bias is ‘log(sigma0)‘, so the initial posterior SD is ‘sigma0‘ – what the argument says it is. * ‘"reference"‘: the bias is ‘log(sigma0^2)‘, matching the reference implementation, whose initial posterior SD is therefore ‘sigma0‘ SQUARED (‘1e-6‘ rather than ‘1e-3‘ for the first neonatal dimension). The reference documents ‘sigma0‘ as a standard deviation, so this appears to be unintended there; it is offered only to reproduce its published behavior.

`residOptimize` How the residual-error parameters are estimated.

Residual forms the optimizer estimates: ‘add‘, ‘prop‘, ‘add + prop‘, ‘pow‘, ‘lnorm‘, and a ‘boxCox‘ or ‘yeoJohnson‘ lambda (bounded to ‘(-2, 2)‘). For a transform-both-sides model the objective transforms ‘dv‘ only and carries the log-Jacobian, since ‘f‘ leaves the solve already on the transformed scale.

‘nonMuTheta = "grad"‘ bypasses this entirely: the analytic outer gradient already carries a residual sigma and a transform lambda as its own directions, so those parameters are stepped by the gradient through Adam and the two-stage path never runs. Which converges better is model-dependent.

* ‘"moment"‘: the closed-form moment estimator. For a model with a single additive error this is exactly the optimum (‘sqrt(SSE/n)‘); for any other error model it is either a different estimator or, for the forms with no closed form (‘pow‘, Box-Cox, Yeo-Johnson), no estimator at all – the parameter stays at its ‘ini()‘ value. There is no moment estimator for a log-likelihood (‘ll()‘) parameter either, so those also stay at ‘ini()‘; use ‘"twoStage"‘ for such a model. * ‘"twoStage"‘ (default): block coordinate descent, as ‘npag‘’s ‘residOptimize = "alternate"‘ does. Stage one optimizes the non-mu-referenced structural thetas with the residual parameters held, so it is driven by ‘(dv - f)‘; stage two then holds those and optimizes the residual parameters alone against the extended least-squares objective ‘sum[(y-f)^2/r + log r]‘ over the CACHED ‘(y, f)‘ pairs. Because ‘f‘ is fixed by stage one, stage two needs no ODE re-solve – the same structure SAEM uses. On ‘theo_sd‘ this beats the moment estimator on both a pure-additive model (objective 131.79 vs 131.81) and a combined one (121.03 vs 122.47).

Which parameters stage two owns is decided per parameter: an error parameter, or one that no ‘d/dt()‘ right-hand side, initial condition or dosing modifier

can reach. The second case is what a log-likelihood (‘ll()‘) or generalized endpoint needs – its residual-like parameters are plain thetas with no error row, and on the error-only rule stage two was empty for such a model, silently making ‘"twoStage"‘ behave like ‘"optimize"‘. When no regressed theta qualifies (every one feeds the solve) stage two has nothing to do and ‘residOptimize‘ has no effect. * ‘"optimize"‘ (EXPERIMENTAL, diagnostic): a single JOINT solve over the structural and residual parameters together, against the full outer objective. Fine with one free residual parameter, but with ‘add‘ and ‘prop‘ both free it diverges – they are near-collinear, and routing the residual through the full outer objective lets the Laplace terms move with it at frozen etas (objective 320.7 against the moment estimator’s 122.5). Retained for comparison; prefer ‘"twoStage"‘.

`residRhoend` Final trust-region radius (‘rhoend‘) of the bounded ‘bobyqa‘ that estimates the residual parameters – its convergence tolerance. ‘NULL‘ (default) derives it from ‘sigdig‘ (‘10^(-sigdig)‘), the same way every other optimizer tolerance in the package is derived, so ‘sigdig‘ stays the single knob that moves them together. Set it explicitly when the residual step should converge tighter than the rest: it runs with the ODE frozen, so tightening it is far cheaper than tightening ‘rhoend‘, which also tightens the structural regression that re-solves per candidate.

`omegaUpdate` How the population variances are updated in the covariate M-step. ‘"suffStat"‘ (default) follows the reference: ‘omega‘ is formed from the EMA sufficient statistics and ASSIGNED outright. ‘"blend"‘ is the historic behavior, blending the freshly computed ‘omega‘ with the previous value at the M-step gain (so it is smoothed twice). Applies to ‘omega‘ only.

Note this option reaches only ONE of the two omega M-steps. Which one runs is decided by ‘covariateSelection‘: with ‘TRUE‘ the covariate M-step runs and honors ‘omegaUpdate‘; with ‘FALSE‘ the plain closed-form M-step runs, whose variances are always raw posterior moments blended at the gain. A declared correlated block’s OFF-diagonals always follow whichever estimator that branch’s diagonal used – estimating the two halves of one block by different estimators need not even give a positive-definite result.

The two settings are the SAME update while the gain is 1, which it is throughout burn-in and the EM phase (assigning a value and blending it in with weight 1 are the same operation); they differ only once ‘gammaIter‘ decays the gain. A short run at default settings will show no difference.

‘mStepObjective‘ does not enter the omega update at all – it scores the non-mu theta M-step. Omega has a closed-form EM update from the variational posterior either way.

The residual error estimate is still EMA-smoothed on the standard-deviation scale, where the reference smooths the residual sum of squares and takes the root afterwards – a known remaining difference. Matching it would need per-endpoint sufficient statistics plus an optimizer branch for the error models with no closed form (‘add + prop‘, ‘add + pow‘, Box-Cox / Yeo-Johnson), as `saemControl()` does.

`perNoCor` Fraction of the EM phase (‘gammaIter‘ iterations) over which a declared correlated ‘omega‘ block is held at zero correlation, letting the variances settle before

the correlations are estimated. This is `saemControl()`'s 'perNoCor' rule (0.75 there as well); it has no effect on a model with no declared off-diagonals.

Held at ZERO, following saem, not at the 'ini()' value: retaining an initial covariance while the variances shrink around it can leave the block non-positive-definite. A 'fixed()' covariance is exempt – it is not being estimated, so it keeps its value through the hold and out the other side.

The fraction is of the EM phase, 'min(gammaIter, iters)', not of the whole run. That matters: the gain is 1 for 'it <= gammaIter', so the release point falls while the gain is still 1 and the correlations are estimable the moment they are unfrozen. (This is why no gain restart is needed here, whereas `emviControl()` – whose run has no separate unit-gain phase – has to restart the off-diagonal gain at release.)

A value greater than 1 is an ABSOLUTE iteration count rather than a fraction, and must be a whole number. Prefer the absolute form whenever a run may be resumed or reproduced at a different length: a fraction of a shorter run is a different schedule, not the same one truncated.

`inputScale` Which observations the encoder-input centering and scaling are computed over. '"reference"' (default) matches the reference implementation, which takes the mean and SD across the whole padded observation matrix, so the zero padding of subjects with fewer observations enters both statistics. On a ragged dataset that is a materially different scale from '"observed"', which uses only the observed values (on the neonatal case study the SD is 1582 against 506). Affects only the encoder's inputs, never the likelihood.

`covSelectMethod`

How the covariate M-step searches subsets. '"bnb"' is the exact branch-and-bound; it becomes impractical past a few dozen candidate covariates. '"l0learn"' has the 'L0Learn' package propose supports, which are then scored and polished with the same exact objective – so the search is approximate but the scoring is not. '"auto"' (default) uses '"l0learn"' for a latent dimension with at least 'covSelectMaxExact' candidate covariates and '"bnb"' otherwise. Set 'covSelectMaxExact = Inf' to force the exact search everywhere.

`covSelectMaxExact`

Search size at or above which 'covSelectMethod = "auto"' switches a latent dimension to 'L0Learn' (default '17', just above the measured wall-clock crossover of roughly 16 bits – see 'tools/benchVaeCovSelect.R', which finds the same crossover in bits whether a covariate carries one shape or two). Measured in bits of feasible-support space – 'sum over covariates of log2(1 + shapes tried)' – after 'pinCovariates' trimming, so it is the size of the search actually run. One shape per covariate costs exactly 1 bit, so with 'shapes' set to a single shape this is a plain candidate count; two shape families of one covariate cost 'log2(3)', keeping the exact search's worst-case node budget the same either way. 'Inf' forces the exact branch-and-bound everywhere.

`covSelectColinearCut`

'abs(cor)' at or above which two covariates are treated as near-interchangeable and put in one colinearity cluster (default '0.9'). A cluster never restricts what may be selected. It does two things: the covariate M-step keeps the previous iteration's choice unless a cluster mate beats it by a full covariate's L0 cost, which

stops the selection chattering between columns the design cannot tell apart; and the mates that came within that margin are reported in '$covNearTie'. Clusters are built at the mutual-exclusion group level, so two shapes of one covariate never cluster together.

`covSelectPhiCor`

Which quantity the correlation between latent dimensions is computed from when grouping them for the cross-parameter covariate refinement: '"suffStat"' (default, the smoothed EMA sufficient statistic the selection itself regresses), '"mu"' (the raw posterior means) or '"resid"' (the posterior means less the fitted covariate centers). '"resid"' runs systematically higher than the other two – measured one bracket higher on the same fit – so raise 'covSelectPhiJoin' if you select it.

`covSelectPhiJoin`, `covSelectPhiLeave`

'abs(cor)' at which two latent dimensions join a correlated group, and the lower value at which one leaves again (defaults '0.9' and '0.8'). Membership is re-evaluated every iteration; the gap between the two stops a pair whose correlation wanders around the threshold from joining and leaving repeatedly. 'covSelectPhiLeave' must not exceed 'covSelectPhiJoin'. The defaults are measured rather than conventional: on an ordinary, well-identified one-compartment model the clearance and volume dimensions correlate between '0.75' and '0.80' under the default 'covSelectPhiCor="suffStat"' (and between '0.80' and '0.85' under '"resid"'), so a join threshold at '0.8' would group a fit that has nothing wrong with it – see 'tools/vaeColinearPreflight.R', which measures this through the shipped gate.

`covSelectPhiMaxDim`

Largest correlated group the cross-parameter refinement will attempt (default '4'). A larger group is skipped rather than split: a latent space that entangled is not something per-covariate moves should be arbitrating, and splitting it would need an arbitrary tie-break on a near-tied graph.

`bnbStrategy` Frontier discipline for the exact branch-and-bound covariate selection: '"lifo"' (default, last-in-first-out depth-first search), '"fifo"' (first-in-first-out) or '"lc"' (least cost / best-first). The solver is exact, so the selected covariates are identical for every strategy; only the search order (and thus efficiency) differs.

`parEncoderBackward`

Parallelize the encoder backward (gradient) pass over subjects. Defaults to 'TRUE' unless 'options(nlmixr2.identical = TRUE)' is set (which flips the default to 'FALSE'); an explicit value here always wins. The encoder forward pass and the covariate branch-and-bound already run multi-threaded and are bit-identical to the serial run. The backward gradient is a continuous cross-subject sum, so parallelizing it (per-thread partials reduced in thread order) makes the result deterministic for a fixed number of 'cores' but no longer bit-identical to the serial path: the per-step gradient differs at ~1e-12, which compounds through the iterative SGD/EM training to a small final difference (well below any estimation tolerance), and results may differ across different 'cores'. When it is active (and 'cores > 1') a note is added to the fit's '$runInfo'. Set this to 'FALSE' – or globally 'options(nlmixr2.identical = TRUE)' – for bit-identical, fully reproducible results.

`nonMuTheta` How to treat a structural population ‘theta‘ that has no random effect (is not mu-referenced) so it can still be estimated by the VAE (which only estimates parameters that occupy the latent space). For the eta-injection modes a small eta is injected so the parameter enters the latent space, and the reported fixed effect is ‘theta + mean(eta)‘ with the temporary eta dropped from the output model.

* ‘"regress"‘ (default, matching ‘saemControl(nonMuTheta=)‘): no eta is injected; instead each such theta is estimated directly, re-optimized every M-step by a bounded ‘bobyqa‘ regression against the full FOCEi outer objective (bounds from the ‘ini()‘ lower/upper), blended with the M-step gain. ‘mStepObjective‘ selects which objective that regression targets. This recovers a no-random-effect population parameter without adding a spurious random effect. ‘nonMuEtaOmega‘ is unused in this mode. * ‘"grad"‘: same target as ‘"regress"‘ but stepped with the EXACT analytic outer gradient (Almquist sensitivity equations, the machinery behind ‘foceiControl(fast=TRUE)‘) instead of a derivative-free search: one augmented sensitivity solve per M-step replaces the bobyqa sweep. Both modes optimize the same full outer objective (with every mu-referenced theta held at its current M-step value), so this changes the optimizer, not the target. It is also the more natural fit for the method: the gradient is handed to the SAME Adam machinery that moves the encoder weights, so the parameter is learned alongside the rest of the model on a shared schedule (same gain, same KL warmup gate), whereas ‘"regress"‘ pauses each M-step to run a separate derivative-free optimizer to convergence and adopts its answer. This is NOT a speed option – it is measurably SLOWER than ‘"regress"‘ (on ‘theo_sd‘, 1.47x with one non-mu theta and 1.13x with three; the gap narrows as the number grows, since bobyqa’s cost scales in it and a single solve does not, but it does not close). Choose it for accuracy: the exact gradient lands closer to the maximum-likelihood value than the derivative-free search (‘theo_sd‘ non-mu ‘tv‘: 3.4294 vs 3.4324, against a FOCEi MLE of 3.4293). Applies to a conditionally Gaussian model and to a single non-Gaussian (‘ll()‘/generalized) endpoint, which differentiates the log-density directly. Falls back to ‘"regress"‘ when the model is out of analytic scope (‘linCmt()‘, IOV, ‘fo‘, a multi-endpoint or censored ‘ll()‘ model, ...); ‘nonMuEtaOmega‘ is unused. * ‘"eta"‘: inject the eta with an ESTIMATED omega (starting at ‘nonMuEtaOmega‘); the typical value is estimated and appears in the iteration table. * ‘"fix"‘: inject the eta with omega held FIXED at ‘nonMuEtaOmega‘ AND hold the typical-value theta fixed at its ‘ini()‘ value. Nothing about the parameter is estimated, so it is not shown in the iteration table (it is reported at its ‘ini()‘ value, marked fixed, with the injected eta dropped). * ‘"none"‘: leave non-mu-referenced thetas frozen at their ‘ini()‘ value (the historic behavior).

`nonMuEtaOmega` Variance of the eta injected for a non-mu-referenced theta (starting value for ‘nonMuTheta="eta"‘, fixed value for ‘nonMuTheta="fix"‘; unused for ‘"regress"‘).

`mStepObjective` Objective the non-mu-referenced theta M-step (‘nonMuTheta = "regress"‘ or ‘"grad"‘) is optimized against. It has no effect when there is no non-mu-referenced structural theta, and it never changes the encoder/ELBO training step or the covariate branch-and-bound criterion, both of which always follow the reference.

* ‘"outer"‘ (default): the full FOCEi outer objective – the frozen-eta joint likelihood PLUS the Laplace determinant, ‘0.5*log|Omega^-1|‘ and the DV-transform Jacobian. This is a deliberate deviation from Rohleff et al. (2025): it keeps the

quantity being optimized equal to the objective the fit reports, and it is the functional the analytic outer gradient differentiates, so ‘nonMuTheta = "grad"‘ optimizes one target rather than stepping one and scoring another. * ‘"elbo"‘: the reference behavior – the plain variational bound (frozen-eta joint likelihood, no Laplace term), matching the M-step in Rohleff et al. (2025). Use it to reproduce the reference implementation. The analytic outer gradient does not apply to this objective, so ‘nonMuTheta = "grad"‘ is downgraded to ‘"regress"‘ with a note in ‘$runInfo‘.

The two objectives differ by terms that depend on the non-mu thetas through the eta Hessian, so they can land on different estimates, and – because those estimates feed the latent means the covariate search regresses on – on different covariate sets.

`likelihood` Inner likelihood used for the objective, EBEs, and gradients, all run through the same FOCEi inner interface: ‘"focei"‘ (default, with eta-epsilon interaction), ‘"foce"‘ (no interaction, NONMEM FOCE with R frozen at the population prediction), ‘"focep"‘ (FOCE+, no interaction but R evaluated at the live conditional eta), or ‘"laplace"‘.

`objf` Which objective-function value is active for AIC/BIC/BICc. Both the linearization and importance-sampling -2LL are always computed and stored; this selects the default active one.

`nIsSample` Number of importance-sampling draws for the IS -2LL.

`returnVae` When ‘TRUE‘ return the raw VAE training object instead of the nlmixr2 fit.

`print` Either a scalar print-frequency (‘0‘ = suppress, ‘1‘ (default) = every evaluation, ‘N‘ = every Nth), OR a pre-built [iterPrintControl()] object. Equivalent to ‘iterPrintControl(every = print, ncol = printNcol, useColor = useColor)‘.

`useColor` Logical (or ‘NULL‘) emit ANSI bold/color escapes in the iteration print. ‘NULL‘ (default) defers to [crayon::has_color()].

`printNcol` Integer (or ‘NULL‘) parameter columns per row before wrapping. ‘NULL‘ (default) uses ‘floor((getOption("width") - 23) / 12)‘.

`covMethod` Method for calculating the covariance at the VAE estimates, run through the FOCEi covariance step with the interaction of `likelihood`; its finite-difference legs optimize the ETAs again, starting from the encoder's. The same choices as `foceiControl()`: `"analytic"` (default), `"r,s"`, `"r"`, `"s"`, or `""` to skip.

`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`.

`literalFix` boolean, substitute fixed population values as literals and re-adjust ui and parameter estimates after optimization; Default is ‘TRUE‘.

`literalFixRes` boolean, substitute fixed population values as literals and re-adjust ui and parameter estimates after optimization; Default is ‘TRUE‘.

`addProp` Type of additive-plus-proportional error: ‘"combined1"‘, where standard deviations add:

$$y = f + (a + b \times f^c) \times \varepsilon$$

; or '"combined2"', where variances add:

$$y = f + \sqrt{a^2 + b^2 \times f^{2 \times c}} \times \varepsilon$$

. Here y = observed, f = predicted, a = additive sd, b = proportional/power sd, c = power exponent (1 in the proportional case).

`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` Specifies the "significant digits" that the ODE solving requests. This is `NULL` by default, and while it is `NULL` it has no effect at all: `rxSolve()` uses the standard `atol`/`rtol` (and the standard sensitivity and steady-state tolerances). `sigdig` only changes a tolerance when you ask for it explicitly.

When it is supplied, the tolerances are derived with one solver-independent formula – the same for stiff, non-stiff and auto-switching solvers. The `rtol` exponent IS `sigdig` and `atol` sits three orders below it:

- `rtol` = `10^(-sigdig)`, `atol` = `10^(-sigdig-3)`
- the sensitivity tolerances match the main solve, so `rtolSens` = `rtol` and `atolSens` = `atol` (gradients and covariances are built from them)
- the steady-state tolerances run one order looser than the corresponding main tolerance, so `ssRtol` = `ssRtolSens` = `10*rtol` and `ssAtol` = `ssAtolSens` = `10*atol`

Each of these is set only when you did not pass that tolerance yourself; a tolerance you supply always wins. Because they are resolved independently, an explicit `atol`/`rtol` overrides the main solve but does *not* propagate to the sensitivity or steady-state tolerances – set those directly if you need them changed too.

This mapping matches how `nlmixr2est` derives solver tolerances from its optimization `sigdig`, so a `sigdig` used for estimation and the same `sigdig` used for a plain `rxSolve()` mean the same thing. Note it is keyed to `sigdig` as a request for that many significant digits, and is looser than the `atol`/`rtol` defaults for small `sigdig` – at `sigdig` = 4 it gives `rtol` = `1e-4` against a default `rtol` = `1e-6`. Raise `sigdig`, or set `atol`/`rtol` directly, when you want a tighter solve.

`sigdigTable` Significant digits in the final output table. If not specified ('NULL'), it defaults to 'sigdig'.

`rhoend` Final trust-region radius ('rhoend') of the inner bounded 'bobyqa' used by the non-mu / covariate regress M-step. 'NULL' (default) derives it from 'sigdig' ('10^(-sigdig)', matching the optimizer convergence tolerance), or '1e-4' when 'sigdig' is 'NULL'.

`stickyRecalcN` The number of bad ODE solves before reducing the atol/rtol for the rest of the problem.

`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

`outerStickyRecalcN`
: The number of bad analytic outer solves for a subject before its loosened tolerance is kept for the rest of the problem; the outer counterpart of ‘stickyRecalcN‘.

`fdIndividualStep`
: For the per-subject finite-difference fallback of the analytic outer gradient (‘fast=TRUE‘), search the shi step size separately for every flagged subject (‘TRUE‘, the default) rather than once on the summed objective over them. The subjects that reach this path are the badly conditioned ones, so one shared step cannot suit them all; a per-subject search also supplies the population of converged peers a clamped step is repaired from. ‘FALSE‘ restores the single shared step, which costs fewer evaluations.

`fdOutlierZ`
: Cut of the Iglewicz-Hoaglin modified z-score that decides whether a finite-differenced subject’s slope is an outlier against the exact analytic slopes, and so whether ‘fdChartrand‘ refines it. The conventional 3.5; lower it to make the pass fire more readily (and to exercise it), raise it to suppress it without turning ‘fdChartrand‘ off.

`fdOutlierScale`
: Test the outlier criterion on the **per-observation** slope (‘TRUE‘, the default) rather than the raw one. A per-subject slope scales with how much data that subject carries, so raw slopes from a 3-observation and a 20-observation subject are not draws from one distribution – pooling them makes a legitimately large slope look like an outlier on unbalanced or sparse data. Only the test is scaled; the gradient itself is untouched. On balanced data this changes nothing, since dividing every slope by the same count leaves the modified z-score unchanged.

`fdRefine`
: Estimator used to recompute a finite-differenced slope once the outlier pass fires. On noisy, hard-to-solve likelihood surfaces the ordering is roughly ‘"richardson"‘ < ‘"lanczos"‘ < ‘"chartrand"‘:

  * ‘"richardson"‘ – Richardson extrapolation of central differences at ‘h, h/v, h/v^2, ...‘, cancelling the ‘h^2, h^4, ...‘ truncation terms in turn. Cheapest, and right when the surface is smooth and only truncation matters. It extrapolates toward ‘h -> 0‘, which is *into* the noise, so it is the wrong instrument when the noise floor is what limits the difference. * ‘"lanczos"‘ – the Lanczos generalized derivative, a least-squares slope through ‘2m+1‘ points. Same ‘O(h^2)‘ truncation as a central difference but lower variance, since independent evaluation noise averages down as points are added rather than being amplified. The middle rung. * ‘"chartrand"‘ (default) – total-variation regularized differentiation over a wide interval, which absorbs curvature through the regularized derivative itself instead of assuming a stencil. Most expensive and most robust to a genuinely rough surface.

  All three apply identically: they are gated by the same outlier test, touch only the finite-differenced subjects, and never recompute a subject whose analytic gradient is available.

`fdChartrandAll`
: When the outlier pass fires for a parameter, refine **every** finite-differenced subject with the Chartrand TV derivative rather than only the outlying ones. Subjects whose augmented solve succeeded keep their exact analytic gradient

either way – only finite differences are ever recomputed. Default ‘FALSE‘ (refine the outliers only).

`fdOutlierAny` Let an outlier among the **exact analytic** slopes fire the outlier pass as well, not only an outlier among the finite differences. Still only the finite differences are recomputed. Default ‘FALSE‘.

`outerMaxOdeRecalc`
Maximum number of times to reduce the ODE tolerances for a single subject and retry when the analytic outer (augmented sensitivity) solve fails. Tracked separately from ‘maxOdeRecalc‘, which governs the inner problem. A subject that solves after loosening still contributes an analytic gradient instead of dropping the whole gradient to finite differences.

`outerOdeRecalcFactor`
The factor the atol/rtol is loosened by on each analytic outer retry; the outer counterpart of ‘odeRecalcFactor‘.

`indTolRelax` When ‘TRUE‘ (default), only subjects whose ODE solve produced NaN/Inf have their tolerances relaxed, and the relaxed tolerance persists across optimizer calls (sticky). When ‘FALSE‘, all subjects have their tolerances relaxed on each retry and tolerances are reset afterward.

`eventSens` Controls how dosing/event-parameter (‘alag‘, ‘F‘, ‘rate‘, ‘dur‘) sensitivities are computed for THETA/ETA gradients: ‘"jump"‘ (default) uses rxode2’s analytic event sensitivities; ‘"fd"‘ uses the legacy finite-difference behavior. Also gates the analytic moving-boundary correction for a modeled ‘alag()‘/‘f()‘ on a ‘linCmt()‘ compartment; set ‘"fd"‘ if that model infuses a dose into the lagged/scaled compartment (rxode2/rxode2#1236), or if the regimen also doses an *unlagged/unscaled* compartment alongside the lagged/scaled one – a common design for estimating ‘f()‘ from paired IV+oral data (rxode2/rxode2#1237).

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

`...` Other arguments to control SAEM.

Details

Covariate selection – MIQP vs. branch-and-bound. Per latent parameter the selection step minimizes the same L0/BIC objective ‘RSS_S/omega + log(N)*|S|‘ over subsets ‘S‘ of the candidate covariates (‘RSS_S‘ is the residual sum of squares of the ordinary-least-squares fit on the intercept plus ‘S‘). The reference implementation (Rohleff et al.) writes this as a Mixed-Integer Quadratic Program (MIQP) – binary include/exclude indicators with big-M constraints – and solves it with the commercial Gurobi solver through ‘cvxpy‘. No MIQP-capable solver is freely available in R: Gurobi is commercial/licensed, and the open QP solvers on CRAN (e.g. ‘osqp‘) are continuous-only and cannot represent the binary selection. A continuous convex relaxation (L1 / lasso) would be solvable but only approximates best subset.

This package instead solves the identical L0/BIC objective EXACTLY with a self-contained branch-and-bound: each candidate support’s coefficients are the closed-form OLS fit and branches are pruned by a valid lower bound (the RSS of the OLS fit using all still-free covariates). It therefore returns the same optimum the MIQP would – no commercial dependency and no relaxation/accuracy loss – and scales to a few dozen covariates. The search is worst-case exponential in the number of covariates, but the pruning makes the practical (sparse) case fast (e.g. 32 candidate covariates in a fraction of a second).

Value

vae control structure (class ‘vaeControl‘)

Author(s)

Matthew L. Fidler

`vaeCovariates` *Covariates explored by the VAE covariate search*

Description

Returns the candidate columns that ‘nlmixr2(..., est = "vae")‘ would explore during automated covariate selection, using the same discovery rules as the fit: every non-reserved data column that is constant within each subject is a candidate. A numeric candidate with more than two unique values is continuous and contributes one column per eligible shape; anything else is categorical and contributes an indicator per testable level. Columns sharing a ‘group‘ are alternate shapes of one covariate, so at most one of them can enter a given parameter. Time-varying columns cannot be searched and are excluded with a warning.

Usage

```
vaeCovariates(
  data,
  warn = TRUE,
  shapes = c("power", "lin", "log", "identity", "center", "hockey"),
  covCenterType = c("median", "mean"),
  covCenter = NULL,
  catCutoff = 0.05,
  colinearCut = .vaeColinearCut
)
```

Arguments

`data`	estimation dataset containing at least an ‘ID‘ column; column names are matched case-insensitively, as in the VAE fit
`warn`	when ‘TRUE‘ (default) warn about time-varying columns excluded from the search; when ‘FALSE‘ exclude them silently
`shapes`, `covCenterType`, `covCenter`, `catCutoff`	as in [vaeControl()]; control which shapes are explored and how covariates are centered
`colinearCut`	as ‘covSelectColinearCut‘ in [vaeControl()]; ‘abs(cor)‘ at or above which two covariates form a colinearity cluster

Value

a data frame with one row per candidate search column and columns ‘covariate‘ (the column name), ‘raw‘ (upper-cased data column it comes from), ‘shape‘, ‘level‘ (for categorical indicators), ‘group‘ (mutual exclusion group), ‘block‘ (columns selected all-or-none, i.e. the two arms of a ‘"hockey"‘ relationship), ‘cluster‘ (near-interchangeable covariates; always a coarsening of ‘group‘, so two shapes of one covariate never cluster together), ‘type‘ and ‘center‘; zero rows when nothing qualifies

Author(s)

Matthew L. Fidler

Examples

```
d <- data.frame(id = rep(1:3, each = 2), time = rep(0:1, 3), dv = rnorm(6),
                wt = rep(c(70, 80, 60), each = 2),
                sex = rep(c(0, 1, 0), each = 2))
vaeCovariates(d)

# restrict the explored shapes
vaeCovariates(d, shapes = "power")
```

---

vpcSim *VPC simulation*

Description

VPC simulation

Usage

```
vpcSim(
  object,
  ...,
  keep = NULL,
  n = 300,
  pred = FALSE,
  seed = 1009,
  nretry = 50,
  minN = 10,
  normRelated = TRUE
)
```

Arguments

object	This is the nlmixr2 fit object
...	Other arguments sent to ‘rxSolve()‘
keep	Column names to keep in the output simulated dataset

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
minN	With retries, the minimum number of studies to restimulate (by default 10)
normRelated	should the VPC style simulation be for normal related variables only

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

∗ **Estimation control**
  foceiControl, 37
  nlmixr2NlmeControl, 151
  saemControl, 202
∗ **datasets**
  nlmixr2Keywords, 150
.augPredIpredModel, 5
.nlmFamilyFitGeneric, 5
.nlmixr0preProcessCovariatesPresent, 7
.nlmixr0preProcessNpMuExpand, 7
.nlmixrPreprocessLiteralFix, 8
.preProcessDataUi, 9
.preProcessLinCmtOde, 9
.preProcessVaeNonMuTheta, 10
.preProcessZeroOmega, 11
.preProcessZeroTheta, 11

aaaCranNlmixrThreads, 12
addCwres, 13, *221*
addNpde, 14, *221*
addTable, 16
agqControl, 17
AIC, *135*
assertNlmixrFit, 20
assertNlmixrFitData, 21
augPred.nlmixr2FitData *(*nlmixr2AugPredSolve*)*, 138

BIC, *135*
bobyqaControl, 22
boxCox, 26

cholSE, 27

diag, *169*

emviControl, 28, *237*

fbviControl, 35
fixed.effects, *136*
foceControl, 36
foceiControl, 37, *213*, *214*, *240*
foceiControl(), *156*, *212*
foceiLikIterPrintEnd *(*foceiLikIterPrintStart*)*, 58
foceiLikIterPrintStart, 58
foceiLikLoad, 60
foceiLikRun, 62
foceiLikUnload, 65
focepControl, 65
foControl, 66
foiControl, 67
format, *166*
formatMinWidth, 68

getOfvType *(*setOfv*)*, 218
getOption, *166*
getValidNlmixrControl *(*getValidNlmixrCtl.agq*)*, 69
getValidNlmixrCtl *(*getValidNlmixrCtl.agq*)*, 69
getValidNlmixrCtl.agq, 69

iagqControl, 74
iBoxCox *(*boxCox*)*, 26
ifoceControl, 75
ifoceiControl, 76
ifocepControl, 78
ilaplaceControl, 79
impControl, 80
impCovControl, 81, *213*
impmapControl, *81*, 82
iterPrintControl, 92
iYeoJohnson *(*boxCox*)*, 26

laplaceControl, 93
lbfgsb3cControl, 95
logical, *164*
logLik, *135*

magqControl, 99

mfoceControl, 100
mfoceiControl, 102
mfocepControl, 103
mlaplaceControl, 104

n1qn1, *58*
n1qn1Control, 105
nearcor, *170*
newuoaControl, 109
nlm, *153*
nlmControl, 114
nlme, *128*, *135*
nlmeControl *(*nlmixr2NlmeControl*)*, 151
nlminb, *153*, *156*
nlminbControl, 119
nlmixr *(*nlmixr2*)*, 124
nlmixr2, 124, *211*
nlmixr2AllEst, 137
nlmixr2AllEstType, 137
nlmixr2AugPredSolve, 138
nlmixr2CreateOutputFromUi, 139
nlmixr2Est *(*nlmixr2Est.agq*)*, 140
nlmixr2Est.agq, 140
nlmixr2fix, 145
nlmixr2Gill83, 146, *148*, *149*
nlmixr2Hess, 148
nlmixr2Keywords, 150
nlmixr2Logo, 150
nlmixr2NlmeControl, 151
nlmixr2NlmeControl(), *58*, *212*
nlmixr2Validate, 156
nlmixr2Version, 157
nlmixrAddObjectiveFunctionDataFrame, 157
nlmixrAddTiming, 158
nlmixrCbind, 159
nlmixrClone, 160
nlmixrWithTiming, 161
nls, *164*
nlsControl, 162
nmNearPD, 168
nmObjGetControl *(*nmObjGetControl.agq*)*, 170
nmObjGetControl.agq, 170
nmObjGetEstimationModel, 174
nmObjGetEstimationModel.default *(*nmObjGetIpredModel*)*, 176
nmObjGetEstimationModel.saem *(*nmObjGetIpredModel*)*, 176
nmObjGetFoceiControl *(*nmObjGetFoceiControl.agq*)*, 174
nmObjGetFoceiControl.agq, 174
nmObjGetIpredModel, 176
nmObjGetPredOnly, 177
nmObjHandleControlObject *(*nmObjHandleControlObject.agqControl*)*, 178
nmObjHandleControlObject.agqControl, 178
nmObjHandleModelObject, 180
nmObjUiSetCompressed, 181
nmsimplex, 182
nmTest *(*nlmixr2Validate*)*, 156
npagControl, 182
npbControl, 185
numericDeriv, *164*

ofv, 187
optim, *58*
optimControl, 188
optimHess, *149*

pdClasses, *154*
posdefify, *169*, *170*
posthocControl, 194
predict.nlmixr2FitCore, 195
print.saemFit, 196

qrpemControl, 197

random.effects, *136*
residuals.nlmixr2FitData, 198
rmEta, 198
rsControl, 200, *213*
rxode2, *129*
rxSolve, *58*

saControl, 201, *213*
saemControl, *30*, *31*, *134*, *202*, 202, *214*, *235–237*
saemControl(), *58*, *156*
setCov, *44*, *81*, *201*, *202*, 212, *215*, *216*
setCov<-, 214
setCovAllMethods, 215
setCovOptions, *214*, *215*, 216
setCovValue, *214*, 217
setOfv, 218
setRxThreads(), *220*

sqrtm, 218
summary.saemFit, 219
symmpart, *169*

tableControl, 219
trustControl, 221

uobyqaControl, 226

vaeControl, *30*, 231
vaeCovariates, 244
varClasses, *154*
vpcSim, 245

warning, *153*

yeoJohnson *(*boxCox*)*, 26