Package 'babelmixr2'

October 16, 2026

Type Package

Title Use 'nlmixr2' to Interact with Open Source and Commercial Software

Version 0.1.12

Description Run other estimation and simulation software via the 'nlmixr2' (Fidler et al (2019) <doi:10.1002/psp4.12445>) interface including 'PKNCA', 'NONMEM' and 'Monolix'. It also converts 'nlmixr2'/'rxode2' models to 'PopED' optimal design databases (Fidler et al (2024) <doi:10.70534/XUMG6226>). While not required, you can get/install the 'lixoftConnectors' package in the 'Monolix' installation, as described at the following url <https://monolixsuite.slp-software.com/r-functions/2024R1/installation-and-initialization>. When 'lixoftConnectors' is available, 'Monolix' can be run directly instead of setting up command line usage.

License GPL (>= 3)

URL https://nlmixr2.github.io/babelmixr2/, https://github.com/nlmixr2/babelmixr2/

NeedsCompilation yes

Encoding UTF-8

Suggests qs2, testthat, withr, lixoftConnectors, PKNCA (>= 0.10.0), rmarkdown, spelling, PopED, units (>= 0.8-6), nlme, dplyr, devtools, memoise, FME, coda, crayon, saemix, lme4, nlmixr2lib

Depends R (>= 3.5)

Imports checkmate, cli, digest, lotri, nlmixr2data, nlmixr2extra, nlmixr2plot, magrittr, nlmixr2est (>= 6.2.0), nonmem2rx (>= 0.1.5), monolix2rx (>= 0.0.3), methods, rex, rxode2 (>= 5.1.8)

Config/testthat/edition 3

LinkingTo Rcpp, rxode2, RcppArmadillo, RcppEigen

Language en-US

Config/roxygen2/version 8.1.0

Author Matthew Fidler [aut, cre] (ORCID:
<https://orcid.org/0000-0001-8538-6691>),
Bill Denney [aut] (ORCID: <https://orcid.org/0000-0002-5759-428X>),
Theodoros Papathanasiou [ctb],
Nook Fulloption [ctb] (goldfish art)

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

Repository CRAN

Date/Publication 2026-10-16 17:40:07 UTC

Contents

.nlmerNlmerFun . . . 3
.setupPopEDdatabase . . . 3
as.nlmixr2 . . . 4
babel.poped.database . . . 6
babelBpopIdx . . . 7
bblDatToMonolix . . . 8
fmeMcmcControl . . . 11
getStandardColNames . . . 16
modelUnitConversion . . . 16
monolixControl . . . 17
nlmixr2Est.nlmer . . . 20
nlmixr2Est.pknca . . . 21
nlmixr2NlmerControl . . . 22
nonmemControl . . . 25
pkncaControl . . . 29
popedControl . . . 30
popedGetMultipleEndpointModelingTimes . . . 41
popedMultipleEndpointResetTimeIndex . . . 42
pseudoOptimControl . . . 43
rxToMonolix . . . 47
rxToNonmem . . . 48
rxUiGet.loadPruneNlmer . . . 48
rxUiGet.loadPruneNlmerSens . . . 49
rxUiGet.nlmerEnv . . . 49
rxUiGet.nlmerFormula . . . 50
rxUiGet.nlmerHdTheta . . . 50
rxUiGet.nlmerModel0 . . . 51
rxUiGet.nlmerParams . . . 51
rxUiGet.nlmerRandomFormula . . . 52
rxUiGet.nlmerSensModel . . . 52
rxUiGet.nlmerStart . . . 53
rxUiGet.nlmerThetaS . . . 53
rxUiGet.nlmerThetas . . . 54
saemixControl . . . 54
simplifyUnit . . . 58

Index **60**

`.nlmerNlmerFun` *Surrogate gradient function for lme4::nlmer*

Description

Computes predictions and the analytical gradient for every observation using the nlm C machinery (a single multithreaded `nlmixr2est::nlmerSolveGrad()` call against the model kept resident by `nlmixr2est::.nlmSetupEnv()`). Parameters arrive in positional order matching `.nlmerGlobal$nlmerEnv$paramNames` (`THETA[i]` order).

Usage

```
.nlmerNlmerFun(TIME, ...)
```

Arguments

`TIME`	Observation times vector (unused; kept for formula compatibility – the solver uses the resident per-subject event data)
`...`	Individual-level parameter vectors (one per param in `saemParamsToEstimate`)

Details

lme4 passes one value per observation for each nonlinear parameter; within a subject these are constant (phi = beta + b), so the per-subject parameter vector is read from each subject's first observation row.

Value

Numeric vector of predictions with `"gradient"` attribute

`.setupPopEDdatabase` *Setup the poped database*

Description

Setup the poped database

Usage

```
.setupPopEDdatabase(ui, data, control)
```

Arguments

`ui`	rxode2 ui function
`data`	babelmixr2 design data
`control`	PopED control

Value

PopED database

Author(s)

Matthew L. Fidler

`as.nlmixr2` *Convert an object to a nlmixr2 fit object*

Description

Convert an object to a nlmixr2 fit object

Usage

```
as.nlmixr2(
  x,
  ...,
  table = nlmixr2est::tableControl(),
  rxControl = rxode2::rxControl(),
  ci = 0.95
)

as.nlmixr(
  x,
  ...,
  table = nlmixr2est::tableControl(),
  rxControl = rxode2::rxControl(),
  ci = 0.95
)
```

Arguments

`x`	Object to convert
`...`	Other arguments
`table`	is the `nlmixr2est::tableControl()` options
`rxControl`	is the `rxode2::rxControl()` options, which is generally needed for how `addl` doses are handled in the translation
`ci`	is the confidence interval of the residual differences calculated (by default 0.95)

Value

nlmixr2 fit object

Author(s)

Matthew L. Fidler

Examples

```
# First read in the model (but without residuals)
mod <- nonmem2rx(system.file("mods/cpt/runODE032.ctl", package="nonmem2rx"),
                 determineError=FALSE, lst=".res", save=FALSE)

# define the model with residuals (and change the name of the
# parameters) In this step you need to be careful to not change the
# estimates and make sure the residual estimates are correct (could
# have to change var to sd).

 mod2 <-function() {
   ini({
     lcl <- 1.37034036528946
     lvc <- 4.19814911033061
     lq <- 1.38003493562413
     lvp <- 3.87657341967489
     RSV <- c(0, 0.196446108190896, 1)
     eta.cl ~ 0.101251418415006
     eta.v ~ 0.0993872449483344
     eta.q ~ 0.101302674763154
     eta.v2 ~ 0.0730497519364148
   })
   model({
     cmt(CENTRAL)
     cmt(PERI)
     cl <- exp(lcl + eta.cl)
     v <- exp(lvc + eta.v)
     q <- exp(lq + eta.q)
     v2 <- exp(lvp + eta.v2)
     v1 <- v
     scale1 <- v
     k21 <- q/v2
     k12 <- q/v
     d/dt(CENTRAL) <- k21 * PERI - k12 * CENTRAL - cl * CENTRAL/v1
     d/dt(PERI) <- -k21 * PERI + k12 * CENTRAL
     f <- CENTRAL/scale1
     f ~ prop(RSV)
   })
 }

# now we create another nonmem2rx object that validates the model above:

new <- as.nonmem2rx(mod2, mod)

# once that is done, you can translate to a full nlmixr2 fit (if you wish)
```

```
fit <- as.nlmixr2(new)

print(fit)
```

`babel.poped.database` *Expand a babelmixr2 PopED database*

Description

Expand a babelmixr2 PopED database

Usage

```
babel.poped.database(popedInput, ..., optTime = NA)
```

Arguments

`popedInput`	The babelmixr2 generated PopED database
`...`	other parameters sent to `PopED::create.poped.database()`
`optTime`	boolean to indicate if the global time indexer inside of babelmixr2 is reset if the times are different. By default this is `TRUE`. If `FALSE` you can get slightly better run times and possibly slightly different results. When `optTime` is `FALSE` the global indexer is reset every time the PopED rxode2 is setup for a problem or when a poped dataset is created. You can manually reset with `popedMultipleEndpointResetTimeIndex`

Value

babelmixr2 PopED database (with $babelmixr2 in database)

Author(s)

Matthew L. Fidler

References

Fidler ML, Denney W, Harrold J, Hooijmaijers R, Papathanasiou T, Schoemaker R, Taubert M, Trame M, Wilkins J (2024). babelmixr2 and PopED: Quick Conversion of NONMEM, Monolix and nlmixr2/rxode2 Models to PopED Optimal Design Analysis. American Conference on Pharmacometrics (ACoP) 2024. doi:10.70534/XUMG6226

babelBpopIdx *Get the bpop_idx by variable name for a poped database created by* babelmixr2

Description

This may work for other poped databases if the population parameters are named.

Usage

```
babelBpopIdx(popedInput, var)
```

Arguments

popedInput	The babelmixr2 created database
var	variable to query

Value

index of the variable

Author(s)

Matthew L. Fidler

Examples

```
if (requireNamespace("PopED", quietly=TRUE)) {

f <- function() {
  ini({
    tV <- 72.8
    tKa <- 0.25
    tCl <- 3.75
    tF <- fix(0.9)
    pedCL <- 0.8

    eta.v ~ 0.09
    eta.ka ~ 0.09
    eta.cl ~0.25^2

    prop.sd <- fix(sqrt(0.04))
    add.sd <- fix(sqrt(5e-6))

  })
  model({
    V<-tV*exp(eta.v)
    KA<-tKa*exp(eta.ka) * (pedCL**isPediatric) # add covariate for pediatrics
    CL<-tCl*exp(eta.cl)
```

```
    Favail <- tF

    N <-  floor(t/TAU)+1
    y <- (DOSE*Favail/V)*(KA/(KA - CL/V)) *
      (exp(-CL/V * (t - (N - 1) * TAU)) *
         (1 - exp(-N * CL/V * TAU))/(1 - exp(-CL/V * TAU)) -
         exp(-KA * (t - (N - 1) * TAU)) * (1 - exp(-N * KA * TAU))/(1 - exp(-KA * TAU)))

    y ~ prop(prop.sd) + add(add.sd)
  })
}

e <- et(c( 1,8,10,240,245))

babel.db <- nlmixr2(f, e, "poped",
                    popedControl(m = 2,
                                 groupsize=20,
                                 bUseGrouped_xt=TRUE,
                                 a=list(c(DOSE=20,TAU=24,isPediatric = 0),
                                        c(DOSE=40, TAU=24,isPediatric = 0))))

babelBpopIdx(babel.db, "pedCL")

}
```

---

bblDatToMonolix *Convert nlmixr2-compatible data to other formats (if possible)*

Description

Convert nlmixr2-compatible data to other formats (if possible)

Usage

```
bblDatToMonolix(
  model,
  data,
  table = nlmixr2est::tableControl(),
  rxControl = rxode2::rxControl(),
  env = NULL
)

bblDatToNonmem(
  model,
  data,
  table = nlmixr2est::tableControl(),
  rxControl = rxode2::rxControl(),
  env = NULL
)
```

```
bblDatToRxode(
  model,
  data,
  table = nlmixr2est::tableControl(),
  rxControl = rxode2::rxControl(),
  env = NULL
)

bblDatToMrgsolve(
  model,
  data,
  table = nlmixr2est::tableControl(),
  rxControl = rxode2::rxControl(),
  env = NULL
)

bblDatToPknca(
  model,
  data,
  table = nlmixr2est::tableControl(),
  rxControl = rxode2::rxControl(),
  env = NULL
)
```

Arguments

model	rxode2 model for conversion
data	Input dataset.
table	is the table control; this is mostly to figure out if there are additional columns to keep.
rxControl	is the rxode2 control options; This is to figure out how to handle the addl dosing information.
env	When NULL (default) nothing is done. When an environment, the function nlmixr2est::.foceiPreProce env, model, rxControl) is called on the provided environment.

Value

With the function bblDatToMonolix() return a list with:

- Monolix compatible dataset ($monolix)
- Monolix ADM information ($adm)

With the function nlmixrDataToNonmem() return a dataset that is compatible with NONMEM.

With the function nlmixrDataToMrgsolve() return a dataset that is compatible with mrgsolve. Unlike NONMEM, it supports replacement events with evid=8 (note with rxode2 replacement evid is 5).

With the function nlmixrDataToRxode() this will normalize the dataset to use newer evid definitions that are closer to NONMEM instead of any classic definitions that are used at a lower level

Author(s)

Matthew L. Fidler

Examples

```
pk.turnover.emax3 <- function() {
  ini({
    tktr <- log(1)
    tka <- log(1)
    tcl <- log(0.1)
    tv <- log(10)
    ##
    eta.ktr ~ 1
    eta.ka ~ 1
    eta.cl ~ 2
    eta.v ~ 1
    prop.err <- 0.1
    pkadd.err <- 0.1
    ##
    temax <- logit(0.8)
    tec50 <- log(0.5)
    tkout <- log(0.05)
    te0 <- log(100)
    ##
    eta.emax ~ .5
    eta.ec50  ~ .5
    eta.kout ~ .5
    eta.e0 ~ .5
    ##
    pdadd.err <- 10
  })
  model({
    ktr <- exp(tktr + eta.ktr)
    ka <- exp(tka + eta.ka)
    cl <- exp(tcl + eta.cl)
    v <- exp(tv + eta.v)
    emax = expit(temax+eta.emax)
    ec50 =  exp(tec50 + eta.ec50)
    kout = exp(tkout + eta.kout)
    e0 = exp(te0 + eta.e0)
    ##
    DCP = center/v
    PD=1-emax*DCP/(ec50+DCP)
    ##
    effect(0) = e0
    kin = e0*kout
    ##
    d/dt(depot) = -ktr * depot
    d/dt(gut) =  ktr * depot -ka * gut
    d/dt(center) =  ka * gut - cl / v * center
    d/dt(effect) = kin*PD -kout*effect
    ##
```

```
    cp = center / v
    cp ~ prop(prop.err) + add(pkadd.err)
    effect ~ add(pdadd.err) | pca
  })
}

bblDatToMonolix(pk.turnover.emax3, nlmixr2data::warfarin)

bblDatToNonmem(pk.turnover.emax3, nlmixr2data::warfarin)

bblDatToMrgsolve(pk.turnover.emax3, nlmixr2data::warfarin)

bblDatToRxode(pk.turnover.emax3, nlmixr2data::warfarin)
```

`fmeMcmcControl` *Control for fmeMcmc estimation method in nlmixr2*

Description

Control for fmeMcmc estimation method in nlmixr2

Usage

```
fmeMcmcControl(
  jump = NULL,
  prior = NULL,
  niter = 1000L,
  outputlength = niter,
  burninlength = 0,
  updatecov = niter,
  covscale = NULL,
  ntrydr = 1,
  drscale = NULL,
  verbose = FALSE,
  seed = 99,
  returnFmeMcmc = 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("none", "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 = TRUE,
  covMethod = c("mcmc", "r", ""),
  adjObf = TRUE,
  ci = 0.95,
  sigdig = 4,
  sigdigTable = NULL,
  eventSens = c("jump", "fd"),
  ...
)
```

Arguments

`jump`	jump length, either a *number*, a *vector* with length equal to the total number of parameters, a *covariance matrix*, or a *function* that takes as input the current values of the parameters and produces as output the perturbed parameters. See details.
`prior`	-2 * log(prior parameter probability), either a function called as `prior(p)` with the parameter vector `FME::modMCMC()` samples (which is the scaled space when `scaleType` rescales the parameters), or `NULL` for a non-informative prior. When the model declares priors in its `ini({})` block (for example `prior(tka) ~ dnorm(0, 10)`) this function is generated from them, evaluated on the natural (unscaled) parameter scale, and `prior` must be left `NULL`; supplying both is an error.
`niter`	number of iterations for the MCMC.
`outputlength`	number of iterations kept in the output; should be smaller or equal to `niter`.
`burninlength`	number of initial iterations to be removed from output.
`updatecov`	number of iterations after which the parameter covariance matrix is (re)evaluated based on the parameters kept thus far, and used to update the MCMC jumps.
`covscale`	scale factor for the parameter covariance matrix, used to perform the MCMC jumps.
`ntrydr`	maximal number of tries for the delayed rejection procedure. It is generally not a good idea to set this to a too large value.
`drscale`	for each try during delayed rejection, the cholesky decomposition of the proposal matrix is scaled with this amount; if `NULL`, it is assumed to be `c(0.2,0.25, 0.333, 0.333, ...)`
`verbose`	if `TRUE` or `1`: prints extra output, if numeric value `i > 1`, prints status information every `i` iterations.

`seed` an integer seed used for the mcmc chain. The chain is fully determined by this seed, so a fixed value makes the fit reproducible. The run is wrapped in `rxode2::rxWithSeed()`, which restores the prior random number generator state afterward, so seeding the fit does not disturb the calling session's stream. Vary this to explore different chains.

`returnFmeMcmc` return the fmeMcmc 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) a subject whose ODE solve had to be retried with a relaxed tolerance keeps that relaxed tolerance for the rest of the fit instead of resetting it every evaluation

`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,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.

`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	method used for the dosing-parameter (alag/F/rate/dur) sensitivities: "jump" routes them through rxode2's analytic event jumps; "fd" falls back to Shi2021 finite differences. See nlmixr2est::nlmControl().
...	Ignored parameters

Value

fmeMcmc 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="fmeMcmc")

print(fit2)

# you can also get the FME modMCMC output with
```

```
# fit2$fmeMcmc

# And use it in the summaries from FME, i.e.

summary(fit2$fmeMcmc)

pairs(fit2$fmeMcmc)

# and you can also use the coda package with `as.mcmc()`
coda::raftery.diag(coda::as.mcmc(fit2))
```

`getStandardColNames` *Determine standardized rxode2 column names from data*

Description

Determine standardized rxode2 column names from data

Usage

```
getStandardColNames(data)
```

Arguments

`data`	A data.frame as the source for column names

Value

A named character vector where the names are the standardized names and the values are either the name of the column from the data or `NA` if the column is not present in the data.

Examples

```
getStandardColNames(data.frame(ID=1, DV=2, Time=3, CmT=4))
```

`modelUnitConversion` *Unit conversion for pharmacokinetic models*

Description

Unit conversion for pharmacokinetic models

Usage

```
modelUnitConversion(
  dvu = NA_character_,
  amtu = NA_character_,
  timeu = NA_character_,
  volumeu = NA_character_
)
```

Arguments

dvu, amtu, timeu	The units for the DV, AMT, and TIME columns in the data
volumeu	The units for the volume parameters in the model

Value

A list with names for the units associated with each parameter ("amtu", "clearanceu", "volumeu", "timeu", "dvu") and the numeric value to multiply the modeled estimate (for example, cp) so that the model is consistent with the data units.

See Also

Other Unit conversion: simplifyUnit()

Examples

```
modelUnitConversion(dvu = "ng/mL", amtu = "mg", timeu = "hr", volumeu = "L")
```

---

monolixControl *Monolix Controller for nlmixr2*

Description

Monolix Controller for nlmixr2

Usage

```
monolixControl(
  nbSSDoses = 7,
  useLinearization = FALSE,
  stiff = FALSE,
  linCmt = c("pkmodel", "ode"),
  prune = "auto",
  addProp = c("combined2", "combined1"),
  exploratoryAutoStop = FALSE,
  smoothingAutoStop = FALSE,
  burnInIterations = 5,
```

```
  smoothingIterations = 200,
  exploratoryIterations = 250,
  simulatedAnnealingIterations = 250,
  exploratoryInterval = 200,
  exploratoryAlpha = 0,
  omegaTau = 0.95,
  errorModelTau = 0.95,
  variability = c("none", "firstStage", "decreasing"),
  runCommand = getOption("babelmixr2.monolix", ""),
  rxControl = NULL,
  sumProd = FALSE,
  optExpression = TRUE,
  calcTables = TRUE,
  compress = TRUE,
  ci = 0.95,
  sigdigTable = NULL,
  absolutePath = FALSE,
  modelName = NULL,
  muRefCovAlg = TRUE,
  run = TRUE,
  ...
)
```

Arguments

nbSSDoses	Number of steady state doses (default 7)
useLinearization	Use linearization for log likelihood and fim.
stiff	boolean for using the stiff ODE solver
linCmt	How a linCmt() model is written for Monolix: "pkmodel" uses Monolix's closed-form pkmodel() macro when the model allows it and otherwise translates the model to ODEs; "ode" always translates the model to ODEs (with rxode2::linToOde())
prune	How if/else statements are written for Monolix: "auto" (default) writes if/elseif/else statements as Monolix if blocks and prunes the if/else branches only when the model uses ifelse() (which Monolix cannot write); TRUE always prunes a model with if/else statements and FALSE never prunes. Pruning uses rxode2's branch pruning, which writes each branch as an arithmetic expression. A pruned fit contains the pruned model.
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).

`exploratoryAutoStop`
: logical to turn on or off exploratory phase auto-stop of SAEM (default 250)

`smoothingAutoStop`
: Boolean indicating if the smoothing should automatically stop (default `FALSE`)

`burnInIterations`
: Number of burn in iterations

`smoothingIterations`
: Number of smoothing iterations

`exploratoryIterations`
: Number of iterations for exploratory phase (default 250)

`simulatedAnnealingIterations`
: Number of simulating annealing iterations

`exploratoryInterval`
: Minimum number of iterations in the exploratory phase (default 200)

`exploratoryAlpha`
: Convergence memory in the exploratory phase (only used when `exploratoryAutoStop` is `TRUE`)

`omegaTau`
: Proportional rate on variance for simulated annealing

`errorModelTau`
: Proportional rate on error model for simulated annealing

`variability`
: This describes the methodology for parameters without variability. It could be: - Fixed throughout (none) - Variability in the first stage (firstStage) - Decreasing until it reaches the fixed value (decreasing)

`runCommand`
: is a shell command or function to run monolix; You can specify the default by `options("babelmixr2.monolix"="runMonolix")`. If it is empty and 'lixoft-Connectors' is available, use lixoftConnectors to run monolix. See details for function usage.

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

`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.

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

`compress`
: Should the object have compressed items

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

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

`absolutePath`
: Boolean indicating if the absolute path should be used for the monolix runs

`modelName`
: Model name used to generate the NONMEM output. If `NULL` try to infer from the model name (could be `x` if not clear). Otherwise use this character for outputs.

`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"`.

run	Should monolix be run and the results be imported to nlmixr2? (Default is TRUE)
...	Ignored parameters

Details

If runCommand is given as a string, it will be called with the system() command like:

runCommand mlxtran.

For example, if runCommand="'/path/to/monolix/mlxbsub2021' -p " then the command line used would look like the following:

'/path/to/monolix/mlxbsub2021' monolix.mlxtran

If runCommand is given as a function, it will be called as FUN(mlxtran, directory, ui) to run Monolix. This allows you to run Monolix in any way that you may need, as long as you can write it in R. babelmixr2 will wait for the function to return before proceeding.

If runCommand is NA, nlmixr() will stop after writing the model files and without starting Monolix.

Note that you can get the translated monolix components from a parsed/compiled rxode2 ui object with ui$monolixModel and ui$mlxtran

Value

A monolix control object

Author(s)

Matthew Fidler

---

nlmixr2Est.nlmer *nlmixr2 estimation using lme4::nlmer*

Description

nlmixr2 estimation using lme4::nlmer

Usage

```
## S3 method for class 'nlmer'
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

`nlmixr2Est.pknca` *Estimate starting parameters using PKNCA*

Description

Estimate starting parameters using PKNCA

Usage

```
## S3 method for class 'pknca'
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

Parameters are estimated as follows:

- `ka` 4 half-lives to Tmax but not higher than 3: `log(2)/(tmax/4)`
- `vc` Inverse of dose-normalized Cmax
- `cl` Estimated as the median clearance
- `vp`, `vp2`2- and 4-fold the `vc`, respectively by default, controlled by the `vpMult` and `vp2Mult` arguments to `pkncaControl`
- `q`, `q2` 0.5- and 0.25-fold the `cl`, respectively by default, controlled by the `qMult` and `q2Mult` arguments to `pkncaControl`

When both intravascular and extravascular doses are present (doses into the observation compartment are intravascular), `ka` is estimated from the extravascular doses and `vc` and `cl` are estimated from the intravascular doses; without extravascular-only doses, `ka` is not updated. Intravascular bolus doses have the concentration at the time of dosing back-extrapolated (replacing a predose concentration at the first dose), and other doses without a concentration at the time of dosing have it imputed as the predose concentration (or zero for the first dose).

The bounds for the parameter estimates are set to 10% of the first percentile and 10 times the 99th percentile. (For ka, the lower bound is set to the lower of 10% of the first percentile or 0.03 and the upper bound is not modified from 10 times the 99th percentile.)

Parameter estimation methods may be changed in a future version.

Value

A model with updated starting parameters. In the model a new element named "nca" will be available which includes the PKNCA results used for the calculation.

`nlmixr2NlmerControl` *Control for nlmer estimation method in nlmixr2*

Description

Control for nlmer estimation method in nlmixr2

Usage

```
nlmixr2NlmerControl(
  optimizer = "bobyqa",
  tolPwrss = NULL,
  optCtrl = list(),
  returnNlmer = FALSE,
  muRefCovAlg = TRUE,
  eventSens = c("jump", "fd"),
  eventType = c("central", "forward"),
  shiErr = (.Machine$double.eps)^(1/3),
  shi21maxFD = 20L,
  stickyRecalcN = 4,
  maxOdeRecalc = 5,
  odeRecalcFactor = 10^(0.5),
  indTolRelax = TRUE,
  useColor = NULL,
  printNcol = NULL,
  print = 1L,
  optExpression = TRUE,
  literalFix = TRUE,
  sumProd = FALSE,
  rxControl = NULL,
  calcTables = TRUE,
  compress = TRUE,
  adjObf = TRUE,
  ci = 0.95,
  sigdig = 4,
  sigdigTable = NULL,
  addProp = c("combined2", "combined1"),
  ...
)

nlmerControl(
  optimizer = "bobyqa",
```

```
  tolPwrss = NULL,
  optCtrl = list(),
  returnNlmer = FALSE,
  muRefCovAlg = TRUE,
  eventSens = c("jump", "fd"),
  eventType = c("central", "forward"),
  shiErr = (.Machine$double.eps)^(1/3),
  shi21maxFD = 20L,
  stickyRecalcN = 4,
  maxOdeRecalc = 5,
  odeRecalcFactor = 10^(0.5),
  indTolRelax = TRUE,
  useColor = NULL,
  printNcol = NULL,
  print = 1L,
  optExpression = TRUE,
  literalFix = TRUE,
  sumProd = FALSE,
  rxControl = NULL,
  calcTables = TRUE,
  compress = TRUE,
  adjObf = TRUE,
  ci = 0.95,
  sigdig = 4,
  sigdigTable = NULL,
  addProp = c("combined2", "combined1"),
  ...
)
```

Arguments

optimizer	passed to lme4::nlmerControl(): optimizer to use
tolPwrss	passed to lme4::nlmerControl(): tolerance for the penalized, weighted residual sum-of-squares (PWRSS) inner iterations. When NULL (the default) it is derived from sigdig, keeping the lme4 default of 1e-7 at the default sigdig = 4 and tightening/loosening it one order of magnitude per significant digit
optCtrl	a list of additional control parameters passed to the nonlinear optimizer via lme4::nlmerControl()
returnNlmer	logical; when TRUE return the raw lme4 nlmerMod object instead of the nlmixr2 fit
muRefCovAlg	logical; when TRUE apply mu2 covariate referencing algebraic transformation (same as saem)
eventSens	method used for the dosing-parameter (alag/F/rate/dur) sensitivities in the analytic gradient model: "jump" routes them through rxode2's analytic event jumps; "fd" falls back to Shi2021 finite differences. See nlmixr2est::nlmControl().
eventType	finite-difference type ("central" or "forward") used for event-related parameters when eventSens = "fd"

shiErr	epsilon used when optimizing the ideal finite-difference step size with the Shi2021 method
shi21maxFD	maximum number of Shi2021 step-size optimization iterations for the gradient
stickyRecalcN	number of bad ODE solves tolerated before the per-subject tolerance is stickily relaxed
maxOdeRecalc	maximum number of times to retry a bad ODE solve with a relaxed tolerance
odeRecalcFactor	factor by which atol/rtol are relaxed on an ODE solve retry
indTolRelax	when TRUE (default) a subject whose ODE solve had to be retried with a relaxed tolerance keeps that relaxed tolerance for the rest of the fit instead of resetting it every evaluation
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)'.
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()'
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'.

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).

... — Ignored parameters

Value

nlmer control structure

Author(s)

Matthew L. Fidler

Examples

```
nlmerControl()
nlmixr2NlmerControl()
```

---

nonmemControl *NONMEM estimation control*

Description

NONMEM estimation control

Usage

```
nonmemControl(
  est = c("focei", "imp", "its", "posthoc"),
  advanOde = c("advan13", "advan8", "advan6"),
  linCmt = c("advan", "ode"),
  prune = "auto",
  cov = c("r,s", "r", "s", ""),
  maxeval = 1e+05,
  tol = 6,
  atol = 12,
  sstol = 6,
  ssatol = 12,
  sigl = 12,
  sigdig = 3,
  print = 1,
  extension = getOption("babelmixr2.nmModelExtension", ".nmctl"),
```

```
  outputExtension = getOption("babelmixr2.nmOutputExtension", ".lst"),
  runCommand = getOption("babelmixr2.nonmem", ""),
  iniSigDig = 5,
  protectZeros = FALSE,
  protect = getOption("babelmixr2.nmProtect", TRUE),
  muRef = TRUE,
  addProp = c("combined2", "combined1"),
  rxControl = NULL,
  sumProd = FALSE,
  optExpression = TRUE,
  calcTables = TRUE,
  compress = TRUE,
  ci = 0.95,
  sigdigTable = NULL,
  readRounding = FALSE,
  readBadOpt = FALSE,
  niter = 100L,
  isample = 1000L,
  iaccept = 0.4,
  iscaleMin = 0.1,
  iscaleMax = 10,
  df = 4,
  seed = 14456,
  mapiter = 1,
  mapinter = 0,
  noabort = TRUE,
  modelName = NULL,
  muRefCovAlg = TRUE,
  run = TRUE,
  ...
)
```

Arguments

est	NONMEM estimation method: "focei" (METHOD=1 INTER), "imp" (importance sampling, METHOD=IMP INTERACTION), "its" (iterative two stage, METHOD=ITS INTERACTION) or "posthoc" (METHOD=0 MAXEVALS=0 POSTHOC)
advanOde	The ODE solving method for NONMEM
linCmt	How a linCmt() model is written for NONMEM: "advan" uses NONMEM's closed-form solutions (ADVAN1-ADVAN4, ADVAN11 or ADVAN12 with TRANS1 micro-constants) when the model allows it and otherwise translates the model to ODEs; "ode" always translates the model to ODEs (with rxode2::linToOde()) and solves it with advanOde
prune	How if/else statements are written for NONMEM: "auto" (default) writes a model whose if blocks are simple (no else, else if, nesting or ifelse()) as NONMEM IF blocks, and otherwise prunes the if/else branches; TRUE always prunes a model with if/else statements and FALSE never prunes (so NONMEM can only use simple if blocks). Pruning uses rxode2's branch pruning (the

same pruning nlmixr2 uses for its symbolic derivatives), which writes each branch as an arithmetic expression; this allows the nested if/else if/else statements NONMEM models cannot otherwise use. A pruned fit contains the pruned model.

cov — The NONMEM covariance method; "" skips the covariance step

maxeval — NONMEM's maxeval (for non posthoc methods)

tol — NONMEM tolerance for ODE solving advan

atol — NONMEM absolute tolerance for ODE solving

sstol — NONMEM tolerance for steady state ODE solving

ssatol — NONMEM absolute tolerance for steady state ODE solving

sigl — NONMEM sigl estimation option

sigdig — the significant digits for NONMEM

print — The print number for NONMEM

extension — NONMEM file extensions

outputExtension — Extension to use for the NONMEM output listing

runCommand — Command to run NONMEM (typically the path to "nmfe75") or a function. See the details for more information.

iniSigDig — How many significant digits are printed in $THETA and $OMEGA when the estimate is zero. Also controls the zero protection numbers

protectZeros — Add babelmixr2's own code to protect divide by zero (and the domains of log(), sqrt() and similar functions) in the control stream. This is only used when protect=FALSE (for example with NONMEM before 7.4); with protect=TRUE, NONMEM's $ABBR PROTECT protects these instead

protect — Add $ABBR PROTECT to the control stream so NM-TRAN replaces LOG, EXP, SQRT, division and powers with NONMEM's protected functions (PLOG, PEXP, PSQRT, PDZ); this requires NONMEM 7.4 or later. The default comes from the babelmixr2.nmProtect option, which is TRUE when unset

muRef — Automatically mu-reference the control stream

addProp, sumProd, optExpression, calcTables, compress, ci, sigdigTable — Passed to nlmixr2est::foceiControl

rxControl — Options to pass to rxode2::rxControl for simulations

readRounding — Try to read NONMEM output when NONMEM terminated due to rounding errors

readBadOpt — Try to read NONMEM output when NONMEM terminated due to an apparent failed optimization

niter — number of iterations in NONMEM estimation methods

isample — Isample argument for NONMEM IMP estimation method

iaccept — Iaccept for NONMEM IMP estimation method

iscaleMin — parameter for IMP NONMEM method (ISCALE_MIN)

iscaleMax — parameter for IMP NONMEM method (ISCALE_MAX)

df	degrees of freedom for IMP method
seed	is the seed for NONMEM methods
mapiter	the number of map iterations for IMP method
mapinter	is the MAPINTER parameter for the IMP method
noabort	Add the NOABORT option for $EST
modelName	Model name used to generate the NONMEM output. If NULL try to infer from the model name (could be x if not clear). Otherwise use this character for outputs.
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'.
run	Should NONMEM be run (and the files imported to nlmixr2); default is TRUE, but FALSE will simply create the NONMEM control stream and data file.
...	optional genRxControl argument controlling automatic rxControl generation.

Details

If runCommand is given as a string, it will be called with the system() command like:

runCommand controlFile outputFile.

For example, if runCommand="'/path/to/nmfe75'" then the command line used would look like the following:

'/path/to/nmfe75' one.cmt.nmctl one.cmt.lst

If runCommand is given as a function, it will be called as FUN(ctl, directory, ui) to run NONMEM. This allows you to run NONMEM in any way that you may need, as long as you can write it in R. babelmixr2 will wait for the function to return before proceeding.

If runCommand is NA, nlmixr() will stop after writing the model files and without starting NONMEM.

Value

babelmixr2 control option for generating NONMEM control stream and reading it back into babelmixr2/nlmixr2

Author(s)

Matthew L. Fidler

Examples

```
nonmemControl()
```

`pkncaControl` *PKNCA estimation control*

Description

PKNCA estimation control

Usage

```
pkncaControl(
  concu = NA_character_,
  doseu = NA_character_,
  timeu = NA_character_,
  volumeu = NA_character_,
  vpMult = 2,
  qMult = 1/2,
  vp2Mult = 4,
  q2Mult = 1/4,
  dvParam = "cp",
  groups = character(),
  sparse = FALSE,
  ncaData = NULL,
  ncaResults = NULL,
  rxControl = rxode2::rxControl()
)
```

Arguments

`concu`, `doseu`, `timeu`	concentration, dose, and time units from the source data (passed to `PKNCA::pknca_units_table()`).
`volumeu`	compartment volume for the model (if `NULL`, simplified units from source data will be used)
`vpMult`, `qMult`, `vp2Mult`, `q2Mult`	Multipliers for vc and cl to provide initial estimates for vp, q, vp2, and q2
`dvParam`	The parameter name in the model that should be modified for concentration unit conversions. It must be assigned on a line by itself, separate from the residual error model line.
`groups`	Grouping columns for NCA summaries by group (required if `sparse = TRUE`)
`sparse`	Are the concentration-time data sparse PK (commonly used in small nonclinical species or with terminal or difficult sampling) or dense PK (commonly used in clinical studies or larger nonclinical species)?

ncaData	Data to use for calculating NCA parameters. Typical use is when a subset of the original data are informative for NCA.
ncaResults	Already computed NCA results (a PKNCAresults object) to bypass automatic calculations. At least the following parameters must be calculated in the NCA: cmax.dn, cl.last, and tmax (without tmax, ka is not updated)
rxControl	Control options sent to `rxode2::rxControl()`

Value

A list of parameters

`popedControl` *Control for a PopED design task*

Description

Control for a PopED design task

Usage

```
popedControl(
  stickyRecalcN = 4,
  maxOdeRecalc = 5,
  odeRecalcFactor = 10^(0.5),
  maxn = NULL,
  rxControl = NULL,
  sigdig = 4,
  important = NULL,
  unimportant = NULL,
 iFIMCalculationType = c("reduced", "full", "weighted", "loc", "reducedPFIM", "fullABC",
    "largeMat", "reducedFIMABC"),
  iApproximationMethod = c("fo", "foce", "focei", "foi"),
  iFOCENumInd = 1000,
  prior_fim = matrix(0, 0, 1),
  d_switch = c("d", "ed"),
  ofv_calc_type = c("lnD", "d", "a", "Ds", "inverse"),
  strEDPenaltyFile = "",
  ofv_fun = NULL,
  iEDCalculationType = c("mc", "laplace", "bfgs-laplace"),
  ED_samp_size = 45,
  bLHS = c("hypercube", "random"),
  bUseRandomSearch = TRUE,
  bUseStochasticGradient = TRUE,
  bUseLineSearch = TRUE,
  bUseExchangeAlgorithm = FALSE,
  bUseBFGSMinimizer = FALSE,
```

```
bUseGrouped_xt = FALSE,
EACriteria = c("modified", "fedorov"),
strRunFile = "",
poped_version = NULL,
modtit = "PopED babelmixr2 model",
output_file = "PopED_output_summary",
output_function_file = "PopED_output_",
strIterationFileName = "PopED_current.R",
user_data = NULL,
ourzero = 1e-05,
dSeed = NULL,
line_opta = NULL,
line_optx = NULL,
bShowGraphs = FALSE,
use_logfile = FALSE,
m1_switch = c("central", "complex", "analytic", "ad"),
m2_switch = c("central", "complex", "analytic", "ad"),
hle_switch = c("central", "complex", "ad"),
gradff_switch = c("central", "complex", "analytic", "ad"),
gradfg_switch = c("central", "complex", "analytic", "ad"),
grad_all_switch = c("central", "complex"),
rsit_output = 5,
sgit_output = 1,
hm1 = 1e-05,
hlf = 1e-05,
hlg = 1e-05,
hm2 = 1e-05,
hgd = 1e-05,
hle = 1e-05,
AbsTol = 1e-06,
RelTol = 1e-06,
iDiffSolverMethod = NULL,
bUseMemorySolver = FALSE,
rsit = 300,
sgit = 150,
intrsit = 250,
intsgit = 50,
maxrsnullit = 50,
convergence_eps = 1e-08,
rslxt = 10,
rsla = 10,
cfaxt = 0.001,
cfaa = 0.001,
bGreedyGroupOpt = FALSE,
EAStepSize = 0.01,
EANumPoints = FALSE,
EAConvergenceCriteria = 1e-20,
bEANoReplicates = FALSE,
```

```
BFGSProjectedGradientTol = 1e-04,
BFGSTolerancef = 0.001,
BFGSToleranceg = 0.9,
BFGSTolerancex = 0.1,
ED_diff_it = 30,
ED_diff_percent = 10,
line_search_it = 50,
Doptim_iter = 1,
iCompileOption = c("none", "full", "mcc", "mpi"),
compileOnly = FALSE,
iUseParallelMethod = c("mpi", "matlab"),
MCC_Dep = NULL,
strExecuteName = "calc_fim.exe",
iNumProcesses = 2,
iNumChunkDesignEvals = -2,
Mat_Out_Pre = "parallel_output",
strExtraRunOptions = "",
dPollResultTime = 0.1,
strFunctionInputName = "function_input",
bParallelRS = FALSE,
bParallelSG = FALSE,
bParallelMFEA = FALSE,
bParallelLS = FALSE,
groupsize = NULL,
time = "time",
timeLow = "low",
timeHi = "high",
id = "id",
m = NULL,
x = NULL,
ni = NULL,
maxni = NULL,
minni = NULL,
maxtotni = NULL,
mintotni = NULL,
maxgroupsize = NULL,
mingroupsize = NULL,
maxtotgroupsize = NULL,
mintotgroupsize = NULL,
xt_space = NULL,
a = NULL,
maxa = NULL,
mina = NULL,
a_space = NULL,
x_space = NULL,
use_grouped_xt = FALSE,
grouped_xt = NULL,
use_grouped_a = FALSE,
```

```
  grouped_a = NULL,
  use_grouped_x = FALSE,
  grouped_x = NULL,
  our_zero = NULL,
  auto_pointer = "",
  user_distribution_pointer = "",
  minxt = NULL,
  maxxt = NULL,
  discrete_xt = NULL,
  discrete_a = NULL,
  fixRes = FALSE,
  script = NULL,
  overwrite = TRUE,
  literalFix = TRUE,
  opt_xt = FALSE,
  opt_a = FALSE,
  opt_x = FALSE,
  opt_samps = FALSE,
  optTime = TRUE,
  literalFixRes = FALSE,
  ...
)
```

Arguments

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
maxn	Maximum number of design points for optimization; By default this is declared by the maximum number of design points in the babelmixr2 dataset (when NULL)
rxControl	'rxode2' ODE solving options during fitting, created with 'rxControl()'
sigdig	Significant digits used to derive the ODE solver tolerances when rxControl is not supplied. The tolerances are fixed here rather than taken from rxode2::rxControl(sigdig=) so they do not change with the rxode2 version: atol/rtol are 0.5*10^(-sigdig-2), the sensitivity tolerances are 0.5*10^(-sigdig-1.5), the steady-state tolerances are 0.5*10^(-sigdig) and the steady-state sensitivity tolerances are 0.5*10^(-sigdig+0.625). PopED computes the FIM by finite differences (see hm1 and hm2), so the ODE solve must be much more accurate than those step sizes.
important	character vector of important parameters or NULL for default. This is used with Ds-optimality
unimportant	character vector of unimportant parameters or NULL for default. This is used with Ds-optimality

`iFIMCalculationType`
: can be either an integer or a named value of the Fisher Information Matrix type:
  - 0/"full" = Full FIM
  - 1/"reduced" = Reduced FIM
  - 2/"weighted" = weighted models
  - 3/"loc" = Loc models
  - 4/"reducedPFIM" = reduced FIM with derivative of SD of sigma as in PFIM
  - 5/"fullABC" = FULL FIM parameterized with A,B,C matrices & derivative of variance
  - 6/"largeMat" = Calculate one model switch at a time, good for large matrices
  - 7/"reducedFIMABC" = =Reduced FIM parameterized with A,B,C matrices & derivative of variance

`iApproximationMethod`
: Approximation method for model, 0=FO, 1=FOCE, 2=FOCEI, 3=FOI

`iFOCENumInd`
: integer; number of individuals in focei solve

`prior_fim`
: matrix; prior FIM

`d_switch`
: integer or character option:
  - 0/"ed" = ED design
  - 1/"d" = D design

`ofv_calc_type`
: objective calculation type:
  - 1/"d" = D-optimality". Determinant of the FIM: det(FIM)
  - 2/"a" = "A-optimality". Inverse of the sum of the expected parameter variances: 1/trace_matrix(inv(FIM))
  - 4/"lnD" = "lnD-optimality". Natural logarithm of the determinant of the FIM: log(det(FIM))
  - 6/"Ds" = "Ds-optimality". Ratio of the Determinant of the FIM and the Determinant of the uninteresting rows and columns of the FIM: det(FIM)/det(FIM_u)
  - 7/"inverse" = Inverse of the sum of the expected parameter RSE: 1/sum(get_rse(FIM,poped.db,use_pe

`strEDPenaltyFile`
: Penalty function name or path and filename, empty string means no penalty. User defined criterion can be defined this way.

`ofv_fun`
: User defined function used to compute the objective function. The function must have a poped database object as its first argument and have "..." in its argument list. Can be referenced as a function or as a file name where the function defined in the file has the same name as the file. e.g. "cost.txt" has a function named "cost" in it.

`iEDCalculationType`
: ED Integral Calculation type:
  - 0/"mc" = Monte-Carlo-Integration
  - 1/"laplace" = Laplace Approximation
  - 2/"bfgs-laplace" = BFGS Laplace Approximation

`ED_samp_size`
: Sample size for E-family sampling

bLHS How to sample from distributions in E-family calculations. 0=Random Sampling, 1=LatinHyperCube –

bUseRandomSearch

- ********START OF Optimization algorithm SPECIFICATION OPTIONS************

Use random search (1=TRUE, 0=FALSE)

bUseStochasticGradient

Use Stochastic Gradient search (1=TRUE, 0=FALSE)

bUseLineSearch Use Line search (1=TRUE, 0=FALSE)

bUseExchangeAlgorithm

Use Exchange algorithm (1=TRUE, 0=FALSE)

bUseBFGSMinimizer

Use BFGS Minimizer (1=TRUE, 0=FALSE)

bUseGrouped_xt Use grouped time points (1=TRUE, 0=FALSE).

EACriteria Exchange Algorithm Criteria:

- 1/"modified" = Modified
- 2/"fedorov" = Fedorov

strRunFile Filename and path, or function name, for a run file that is used instead of the regular PopED call.

poped_version

- ********START OF Labeling and file names SPECIFICATION OPTIONS************

The current PopED version

modtit The model title

output_file Filename and path of the output file during search

output_function_file

Filename suffix of the result function file

strIterationFileName

Filename and path for storage of current optimal design

user_data

- ********START OF Miscellaneous SPECIFICATION OPTIONS************

User defined data structure that, for example could be used to send in data to the model

ourzero Value to interpret as zero in design

dSeed The seed number used for optimization and sampling – integer or -1 which creates a random seed `as.integer(Sys.time())` or NULL.

line_opta Vector for line search on continuous design variables (1=TRUE,0=FALSE)

line_optx Vector for line search on discrete design variables (1=TRUE,0=FALSE)

bShowGraphs Use graph output during search

use_logfile If a log file should be used (0=FALSE, 1=TRUE)

m1_switch Method used to calculate M1:

- 1/"central" = Central difference
- 0/"complex" = Complex difference
- 20/"analytic" = Analytic derivative
- 30/"ad" = Automatic differentiation

`m2_switch` Method used to calculate M2:

- 1/"central" = Central difference
- 0/"complex" = Complex difference
- 20/"analytic" = Analytic derivative
- 30/"ad" = Automatic differentiation

`hle_switch` Method used to calculate linearization of residual error:

- 1/"central" = Central difference
- 0/"complex" = Complex difference
- 30/"ad" = Automatic differentiation

`gradff_switch` Method used to calculate the gradient of the model:

- 1/"central" = Central difference
- 0/"complex" = Complex difference
- 20/"analytic" = Analytic derivative
- 30/"ad" = Automatic differentiation

`gradfg_switch` Method used to calculate the gradient of the parameter vector g:

- 1/"central" = Central difference
- 0/"complex" = Complex difference
- 20/"analytic" = Analytic derivative
- 30/"ad" = Automatic differentiation

`grad_all_switch`
Method used to calculate all the gradients:

- 1/"central" = Central difference
- 0/"complex" = Complex difference

`rsit_output` Number of iterations in random search between screen output

`sgit_output` Number of iterations in stochastic gradient search between screen output

`hm1` Step length of derivative of linearized model w.r.t. typical values

`hlf` Step length of derivative of model w.r.t. g

`hlg` Step length of derivative of g w.r.t. b

`hm2` Step length of derivative of variance w.r.t. typical values

`hgd` Step length of derivative of OFV w.r.t. time

`hle` Step length of derivative of model w.r.t. sigma

`AbsTol` The absolute tolerance for the diff equation solver

`RelTol` The relative tolerance for the diff equation solver

`iDiffSolverMethod`
The diff equation solver method, NULL as default.

`bUseMemorySolver`
If the differential equation results should be stored in memory (1) or not (0)

`rsit` Number of Random search iterations

`sgit` Number of stochastic gradient iterations

`intrsit` Number of Random search iterations with discrete optimization.

`intsgit` Number of Stochastic Gradient search iterations with discrete optimization

`maxrsnullit` Iterations until adaptive narrowing in random search

`convergence_eps`
Stochastic Gradient convergence value, (difference in OFV for D-optimal, difference in gradient for ED-optimal)

`rslxt` Random search locality factor for sample times

`rsla` Random search locality factor for covariates

`cfaxt` Stochastic Gradient search first step factor for sample times

`cfaa` Stochastic Gradient search first step factor for covariates

`bGreedyGroupOpt`
Use greedy algorithm for group assignment optimization

`EAStepSize` Exchange Algorithm StepSize

`EANumPoints` Exchange Algorithm NumPoints

`EAConvergenceCriteria`
Exchange Algorithm Convergence Limit/Criteria

`bEANoReplicates`
Avoid replicate samples when using Exchange Algorithm

`BFGSProjectedGradientTol`
BFGS Minimizer Convergence Criteria Normalized Projected Gradient Tolerance

`BFGSTolerancef` BFGS Minimizer Line Search Tolerance f

`BFGSToleranceg` BFGS Minimizer Line Search Tolerance g

`BFGSTolerancex` BFGS Minimizer Line Search Tolerance x

`ED_diff_it` Number of iterations in ED-optimal design to calculate convergence criteria

`ED_diff_percent`
ED-optimal design convergence criteria in percent

`line_search_it` Number of grid points in the line search

`Doptim_iter` Number of iterations of full Random search and full Stochastic Gradient if line search is not used

`iCompileOption` Compile options for PopED

- "none"/-1 = No compilation
- "full/0 or 3 = Full compilation
- "mcc"/1 or 4 = Only using MCC (shared lib)
- "mpi"/2 or 5 = Only MPI,

When using numbers, option 0,1,2 runs PopED and option 3,4,5 stops after compilation.

When using characters, the option `compileOnly` determines if the model is only compiled (and PopED is not run).

`compileOnly` logical; only compile the model, do not run PopED (in conjunction with `iCompileOption`)

`iUseParallelMethod`
Parallel method to use

- 0/"matlab"= Matlab PCT
- 1/"mpi" = MPI

`MCC_Dep` Additional dependencies used in MCC compilation (mat-files), if several space separated

`strExecuteName` Compilation output executable name

`iNumProcesses` Number of processes to use when running in parallel (e.g. 3 = 2 workers, 1 job manager)

`iNumChunkDesignEvals`
Number of design evaluations that should be evaluated in each process before getting new work from job manager

`Mat_Out_Pre` The prefix of the output mat file to communicate with the executable

`strExtraRunOptions`
Extra options send to e$g. the MPI executable or a batch script, see execute_parallel$m for more information and options

`dPollResultTime`
Polling time to check if the parallel execution is finished

`strFunctionInputName`
The file containing the popedInput structure that should be used to evaluate the designs

`bParallelRS` If the random search is going to be executed in parallel

`bParallelSG` If the stochastic gradient search is going to be executed in parallel

`bParallelMFEA` If the modified exchange algorithm is going to be executed in parallel

`bParallelLS` If the line search is going to be executed in parallel

`groupsize` Vector defining the size of the different groups (num individuals in each group). If only one number then the number will be the same in every group.

`time` string that represents the time in the dataset (ie xt)

`timeLow` string that represents the lower design time (ie minxt)

`timeHi` string that represents the upper design time (ie maxmt)

`id` The id variable

`m` Number of groups in the study. Each individual in a group will have the same design.

`x` A matrix defining the initial discrete values for the model Each row is a group/individual.

`ni` Vector defining the number of samples for each group.

`maxni`
- ********START OF DESIGN SPACE OPTIONS************

Max number of samples per group/individual

`minni` Min number of samples per group/individual

`maxtotni` Number defining the maximum number of samples allowed in the experiment.

`mintotni` Number defining the minimum number of samples allowed in the experiment.

`maxgroupsize` Vector defining the max size of the different groups (max number of individuals in each group)

`mingroupsize` Vector defining the min size of the different groups (min num individuals in each group) –

`maxtotgroupsize`
The total maximal groupsize over all groups

`mintotgroupsize`
The total minimal groupsize over all groups

`xt_space` Cell array `cell` defining the discrete variables allowed for each xt value. Can also be a vector of values `c(1:10)` (same values allowed for all xt), or a list of lists `list(1:10, 2:23, 4:6)` (one for each value in xt in row major order or just for one row in xt, and all other rows will be duplicated).

`a` Matrix defining the initial continuous covariate values. n_rows=number of groups, n_cols=number of covariates. If the number of rows is one and the number of groups > 1 then all groups are assigned the same values.

`maxa` Vector defining the max value for each covariate. If a single value is supplied then all a values are given the same max value

`mina` Vector defining the min value for each covariate. If a single value is supplied then all a values are given the same max value

`a_space` Cell array `cell` defining the discrete variables allowed for each a value. Can also be a list of values `list(1:10)` (same values allowed for all a), or a list of lists `list(1:10, 2:23, 4:6)` (one for each value in a).

`x_space` Cell array `cell` defining the discrete variables for each x value.

`use_grouped_xt` Group sampling times between groups so that each group has the same values (`TRUE` or `FALSE`).

`grouped_xt` Matrix defining the grouping of sample points. Matching integers mean that the points are matched. Allows for finer control than `use_grouped_xt`

`use_grouped_a` Group continuous design variables between groups so that each group has the same values (`TRUE` or `FALSE`).

`grouped_a` Matrix defining the grouping of continuous design variables. Matching integers mean that the values are matched. Allows for finer control than `use_grouped_a`.

`use_grouped_x` Group discrete design variables between groups so that each group has the same values (`TRUE` or `FALSE`).

`grouped_x` Matrix defining the grouping of discrete design variables. Matching integers mean that the values are matched. Allows for finer control than `use_grouped_x`.

`our_zero` Value to interpret as zero in design.

`auto_pointer` Filename and path, or function name, for the Autocorrelation function, empty string means no autocorrelation

`user_distribution_pointer`
Filename and path, or function name, for user defined distributions for E-family designs

`minxt` Matrix or single value defining the minimum value for each xt sample. If a single value is supplied then all xt values are given the same minimum value

`maxxt` Matrix or single value defining the maximum value for each xt sample. If a single value is supplied then all xt values are given the same maximum value.

discrete_xt
: Cell array `cell` defining the discrete variables allowed for each xt value. Can also be a list of values `list(1:10)` (same values allowed for all xt), or a list of lists `list(1:10, 2:23, 4:6)` (one for each value in xt). See examples in `create_design_space`.

discrete_a
: Cell array `cell` defining the discrete variables allowed for each a value. Can also be a list of values `list(1:10)` (same values allowed for all a), or a list of lists `list(1:10, 2:23, 4:6)` (one for each value in a). See examples in `create_design_space`.

fixRes
: boolean; Fix the residuals to what is specified by the model

script
: write a PopED/rxode2 script that can be modified for more fine control. The default is NULL.

  When `script` is TRUE, the script is returned as a lines that would be written to a file and with the class `babelmixr2popedScript`. This allows it to be printed as the script on screen.

  When `script` is a file name (with an R extension), the script is written to that file.

overwrite
: [`logical(1)`]
  If `TRUE`, an existing file in place is allowed if it it is both readable and writable. Default is `FALSE`.

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

opt_xt
: boolean to indicate if this is meant for optimizing times

opt_a
: boolean to indicate if this is meant for optimizing covariates

opt_x
: boolean to indicate if the discrete design variables be optimized

opt_samps
: boolean to indicate if the sample optimizer is used (not implemented yet in `PopED`)

optTime
: boolean to indicate if the global time indexer inside of babelmixr2 is reset if the times are different. By default this is `TRUE`. If `FALSE` you can get slightly better run times and possibly slightly different results. When `optTime` is `FALSE` the global indexer is reset every time the PopED rxode2 is setup for a problem or when a poped dataset is created. You can manually reset with `popedMultipleEndpointResetTimeIndex`

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

...
: other parameters for PopED control

Value

popedControl object

Author(s)

Matthew L. Fidler

References

Fidler ML, Denney W, Harrold J, Hooijmaijers R, Papathanasiou T, Schoemaker R, Taubert M, Trame M, Wilkins J (2024). babelmixr2 and PopED: Quick Conversion of NONMEM, Monolix and nlmixr2/rxode2 Models to PopED Optimal Design Analysis. American Conference on Pharmacometrics (ACoP) 2024. doi:10.70534/XUMG6226

`popedGetMultipleEndpointModelingTimes`

Get Multiple Endpoint Modeling Times

Description

This function takes a vector of times and a corresponding vector of IDs, groups the times by their IDs, initializes an internal C++ global TimeIndexer, that is used to efficiently lookup the final output from the rxode2 solve and then returns the sorted unique times.

The `popedMultipleEndpointIndexDataFrame()` function can be used to visualize the internal data structure inside R, but it does not show all the indexes in the case of time ties for a given ID. Rather it shows one of the indexs and the total number of indexes in the data.frame

Usage

```
popedGetMultipleEndpointModelingTimes(times, modelSwitch, sorted = FALSE)

popedMultipleEndpointIndexDataFrame(print = FALSE)
```

Arguments

`times`	A numeric vector of times.
`modelSwitch`	An integer vector of model switch indicator corresponding to the times
`sorted`	A boolean indicating if the returned times should be sorted
`print`	boolean for `popedMultipleEndpointIndexDataFrame()` when `TRUE` show each id/index per time even though it may not reflect in the returned data.frame

Value

A numeric vector of unique times.

Examples

```
times <- c(1.1, 1.2, 1.3, 2.1, 2.2, 3.1)
modelSwitch <- c(1, 1, 1, 2, 2, 3)
sortedTimes <- popedGetMultipleEndpointModelingTimes(times, modelSwitch, TRUE)
print(sortedTimes)
```

```
# now show the output of the data frame representing the model
# switch to endpoint index

popedMultipleEndpointIndexDataFrame()

# now show a more complex example with overlaps etc.

times <- c(1.1, 1.2, 1.3, 0.5, 2.2, 1.1, 0.75,0.75)
modelSwitch <- c(1, 1, 1, 2, 2, 2, 3, 3)
sortedTimes <- popedGetMultipleEndpointModelingTimes(times, modelSwitch, TRUE)
print(sortedTimes)

popedMultipleEndpointIndexDataFrame(TRUE) # Print to show individual matching
```

`popedMultipleEndpointResetTimeIndex` *Reset the Global Time Indexer for Multiple Endpoint Modeling*

Description

This clears the memory and resets the global time indexer used for multiple endpoint modeling.

Usage

```
popedMultipleEndpointResetTimeIndex()
```

Value

NULL, called for side effects

Examples

```
popedMultipleEndpointResetTimeIndex()
```

pseudoOptimControl *Control for fmeMcmc estimation method in nlmixr2*

Description

Control for fmeMcmc estimation method in nlmixr2

Usage

```
pseudoOptimControl(
  npop = NULL,
  numiter = 10000,
  centroid = 3,
  varleft = NULL,
  verbose = FALSE,
  returnPseudoOptim = 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("none", "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 = TRUE,
  covMethod = c("r", ""),
  adjObf = TRUE,
  ci = 0.95,
  sigdig = 4,
  sigdigTable = NULL,
  eventSens = c("jump", "fd"),
  ...
)
```

Arguments

npop	Number of elements in the population. Defaults to max(5*length(p),50) which is calculated from the number of parameters in the model
numiter	Number of iterations to run the optimization. Defaults to 10000. The algorithm either stops when numiter iterations has been performed or when the remaining variation is less than varleft.
centroid	Number of elements from which to estimate a new parameter vector. The default is 3.
varleft	relative variation remaining; if below this value, the algorithm stops. When NULL (the default) it is derived from sigdig, keeping the FME::pseudoOptim() default of 1e-8 at the default sigdig = 4 and tightening/loosening it one order of magnitude per significant digit.
verbose	If TRUE, print information about the optimization from FME::pseudoOptim. Default is FALSE.
returnPseudoOptim	return the pseudoOptim 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) a subject whose ODE solve had to be retried with a relaxed tolerance keeps that relaxed tolerance for the rest of the fit instead of resetting it every evaluation
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,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.

`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` method used for the dosing-parameter (alag/F/rate/dur) sensitivities: `"jump"` routes them through rxode2's analytic event jumps; `"fd"` falls back to Shi2021 finite differences. See `nlmixr2est::nlmControl()`.

`...` Ignored parameters

Value

pseudoOptim 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({
   # This estimation method requires all parameters
   # to be bounded:
   E0 <- c(-100, 0.5, 100)
   Em <- c(0, 0.5, 10)
   E50 <- c(0, 2, 20)
   g <- fix(c(0.1, 2, 10))
```

```
 })
 model({
   v <- E0+Em*time^g/(E50^g+time^g)
   ll(bin) ~ DV * v - log(1 + exp(v))
 })
}

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

print(fit2)
```

---

rxToMonolix *Convert RxODE syntax to monolix syntax*

Description

Convert RxODE syntax to monolix syntax

Usage

```
rxToMonolix(x, ui)
```

Arguments

x	Expression
ui	rxode2 ui

Value

Monolix syntax

Author(s)

Matthew Fidler

`rxToNonmem` *Convert RxODE syntax to NONMEM syntax*

Description

Convert RxODE syntax to NONMEM syntax

Usage

```
rxToNonmem(x, ui)
```

Arguments

`x`	Expression
`ui`	rxode2 ui

Value

NONMEM syntax

Author(s)

Matthew Fidler

`rxUiGet.loadPruneNlmer` *Load pruned nlmer model into symengine (no sensitivity promotion)*

Description

Load pruned nlmer model into symengine (no sensitivity promotion)

Usage

```
## S3 method for class 'loadPruneNlmer'
rxUiGet(x, ...)
```

Arguments

`x`	rxUiGet list(ui)
`...`	additional arguments (currently ignored)

rxUiGet.loadPruneNlmerSens *Load pruned nlmer model into symengine (with sensitivity promotion)*

Description

Load pruned nlmer model into symengine (with sensitivity promotion)

Usage

```
## S3 method for class 'loadPruneNlmerSens'
rxUiGet(x, ...)
```

Arguments

x	rxUiGet list(ui)
...	additional arguments (currently ignored)

rxUiGet.nlmerEnv *Full nlmer symengine environment with sensitivities*

Description

Full nlmer symengine environment with sensitivities

Usage

```
## S3 method for class 'nlmerEnv'
rxUiGet(x, ...)
```

Arguments

x	rxUiGet list(ui)
...	additional arguments (currently ignored)

rxUiGet.nlmerFormula *Build the full 3-part nlmer formula*

Description

Build the full 3-part nlmer formula

Usage

```
## S3 method for class 'nlmerFormula'
rxUiGet(x, ...)
```

Arguments

x	rxUiGet list(ui)
...	additional arguments (currently ignored)

Value

3-part formula for lme4::nlmer

rxUiGet.nlmerHdTheta *Compute d(f)/d(THETA[i]) for all nlmer parameters*

Description

Compute d(f)/d(THETA[i]) for all nlmer parameters

Usage

```
## S3 method for class 'nlmerHdTheta'
rxUiGet(x, ...)
```

Arguments

x	rxUiGet list(ui)
...	additional arguments (currently ignored)

rxUiGet.nlmerModel0 *nlmer base model (saem-dropped etas,* THETA[i] *prefix, nlme error lines)*

Description

nlmer base model (saem-dropped etas, THETA[i] prefix, nlme error lines)

Usage

```
## S3 method for class 'nlmerModel0'
rxUiGet(x, ...)
```

Arguments

x	rxUiGet list(ui)
...	additional arguments (currently ignored)

rxUiGet.nlmerParams *params() string for the nlmer* THETA[i]*-indexed model*

Description

params() string for the nlmer THETA[i]-indexed model

Usage

```
## S3 method for class 'nlmerParams'
rxUiGet(x, ...)
```

Arguments

x	rxUiGet list(ui)
...	additional arguments (currently ignored)

rxUiGet.nlmerRandomFormula

Build the random-effects part (part 3) of the nlmer 3-part formula

Description

Build the random-effects part (part 3) of the nlmer 3-part formula

Usage

```
## S3 method for class 'nlmerRandomFormula'
rxUiGet(x, ...)
```

Arguments

x	rxUiGet list(ui)
...	additional arguments (currently ignored)

Value

Formula: param1 + param2 + ... + (rand1 + rand2 | ID)

rxUiGet.nlmerSensModel

Compiled nlmer sensitivity model (prediction + gradients)

Description

Returns a list with:

- `thetaGrad`: compiled rxode2 model computing rx_pred_ and rx__sens_rx_pred__BY_THETA_i___ for each estimated parameter. When `eventSens = "jump"` this model carries rxode2's analytic event-jump sensitivities (loaded/activated by `nlmixr2est::.nlmSetupEnv()`).
- `predOnly`: compiled rxode2 model for prediction only
- `eventTheta`: integer vector flagging `THETA[i]` that need Shi2021 finite differences for dosing parameters (all 0 under `eventSens = "jump"`)
- `paramNames`: character vector of parameter names in `THETA[i]` order

Usage

```
## S3 method for class 'nlmerSensModel'
rxUiGet(x, ...)
```

Arguments

x	rxUiGet list(ui)
...	additional arguments (currently ignored)

Details

The list follows the `modelInfo` contract of `nlmixr2est:::.nlmSetupEnv()` so the nlm C machinery can load it once and keep it resident.

---

rxUiGet.nlmerStart *Starting values for nlmer fixed effects*

Description

Starting values for nlmer fixed effects

Usage

```
## S3 method for class 'nlmerStart'
rxUiGet(x, ...)
```

Arguments

x	rxUiGet list(ui)
...	additional arguments (currently ignored)

Value

Named numeric vector: iniDf estimates for thetas, 0 for nonMuEtas

---

rxUiGet.nlmerThetaS *Compute d(state)/d(THETA[i]) sensitivities for nlmer*

Description

Compute d(state)/d(`THETA[i]`) sensitivities for nlmer

Usage

```
## S3 method for class 'nlmerThetaS'
rxUiGet(x, ...)
```

Arguments

x	rxUiGet list(ui)
...	additional arguments (currently ignored)

rxUiGet.nlmerThetas *Get the structural params to estimate for nlmer (= saemParamsToEstimate)*

Description

Get the structural params to estimate for nlmer (= saemParamsToEstimate)

Usage

```
## S3 method for class 'nlmerThetas'
rxUiGet(x, ...)
```

Arguments

x	rxUiGet list(ui)
...	additional arguments (currently ignored)

Value

Character vector of parameter names (mu-ref thetas + nonMuEtas)

saemixControl *Control for saemix estimation method in nlmixr2*

Description

Control for saemix estimation method in nlmixr2

Usage

```
saemixControl(
  map = TRUE,
  fim = TRUE,
  ll.is = TRUE,
  ll.gq = FALSE,
  nbiter.saemix = c(300, 100),
  nbiter.sa = NA,
  nbiter.burn = 5,
  nbiter.map = 5,
  nb.chains = 1,
  fix.seed = TRUE,
  seed = 23456,
  nmc.is = 5000,
  nu.is = 4,
```

```
  print.is = FALSE,
  nbdisplay = 100,
  displayProgress = FALSE,
  print = FALSE,
  save = FALSE,
  save.graphs = TRUE,
  directory = "newdir",
  warnings = FALSE,
  nbiter.mcmc = c(2, 2, 2, 0),
  proba.mcmc = 0.4,
  stepsize.rw = 0.4,
  rw.init = 0.5,
  alpha.sa = 0.97,
  nnodes.gq = 12,
  nsd.gq = 4,
  maxim.maxiter = 100,
  nb.sim = 1000,
  nb.simpred = 100,
  ipar.lmcmc = 50,
  ipar.rmcmc = 0.05,
  rxControl = NULL,
  stickyRecalcN = 4,
  maxOdeRecalc = 5,
  odeRecalcFactor = 10^(0.5),
  indTolRelax = TRUE,
  eventSens = c("jump", "fd"),
  useColor = NULL,
  printNcol = NULL,
  normType = c("rescale2", "mean", "rescale", "std", "len", "constant"),
  scaleType = c("none", "nlmixr2", "norm", "mult", "multAdd"),
  scaleCmax = 1e+05,
  scaleCmin = 1e-05,
  scaleC = NULL,
  scaleTo = 1,
  addProp = c("combined2", "combined1"),
  calcTables = TRUE,
  compress = TRUE,
  ci = 0.95,
  sigdigTable = NULL,
  sigdig = 4,
  ...
)
```

Arguments

map	logical, whether to compute MAP estimates (default TRUE)
fim	logical, whether to compute FIM (default TRUE)
ll.is	logical, whether to compute log-likelihood by Importance Sampling (default

TRUE)

ll.gq — logical, whether to compute log-likelihood by Gaussian Quadrature (default FALSE)

nbiter.saemix — integer vector of length 2, number of iterations for phase 1 (exploratory) and phase 2 (smoothing) of SAEM (default c(300, 100))

nbiter.sa — number of iterations for simulated annealing (default NA)

nbiter.burn — number of iterations for burn-in (default 5)

nbiter.map — number of iterations for MAP estimation (default 5)

nb.chains — number of chains (default 1)

fix.seed — logical, whether to fix the random seed (default TRUE)

seed — random seed (default 23456)

nmc.is — number of Monte Carlo samples for Importance Sampling (default 5000)

nu.is — number of degrees of freedom for the student distribution in Importance Sampling (default 4)

print.is — logical, whether to print progress during Importance Sampling (default FALSE)

nbdisplay — number of iterations between progress displays (default 100)

displayProgress — logical, whether to display graphical progress (default FALSE)

print — whether saemix prints algorithm progress (default FALSE). Accepts the legacy logical/integer scalar or a pre-built nlmixr2est::iterPrintControl() object; internally the control stores a single iterPrintControl sub-list (matching the nlmixr2est iteration-print unification), and any nonzero every enables the saemix progress output

save — logical, whether to save results to files (default TRUE)

save.graphs — logical, whether to save graphs (default TRUE)

directory — directory where results and graphs are saved (default "newdir")

warnings — logical, whether to show warnings (default FALSE)

nbiter.mcmc — integer vector of length 4, number of iterations for MCMC kernel updates (default c(2, 2, 2, 0))

proba.mcmc — probability for MCMC kernel selection (default 0.4)

stepsize.rw — stepsize for random walk kernel (default 0.4)

rw.init — initial standard deviation for random walk kernel (default 0.5)

alpha.sa — parameter for simulated annealing (default 0.97)

nnodes.gq — number of nodes for Gaussian Quadrature (default 12)

nsd.gq — number of standard deviations for Gaussian Quadrature (default 4)

maxim.maxiter — maximum number of iterations for maximization step (default 100)

nb.sim — number of simulations for visual predictive check (default 1000)

nb.simpred — number of simulations for predictions (default 100)

ipar.lmcmc — parameter for L-MCMC (default 50)

`ipar.rmcmc` parameter for R-MCMC (default `0.05`)

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

`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) a subject whose ODE solve had to be retried with a relaxed tolerance keeps that relaxed tolerance for the rest of the fit instead of resetting it every evaluation

`eventSens` method used for the dosing-parameter (alag/F/rate/dur) sensitivities when the final tables are built: `"jump"` routes them through rxode2's analytic event jumps; `"fd"` falls back to Shi2021 finite differences. See `nlmixr2est::foceiControl()`.

`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)'.

`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.

`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

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

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

`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

Value

saemix control structure

Author(s)

Matthew L. Fidler

Examples

```
# Example showing control creation:
ctrl <- saemixControl(seed = 123456)
```

`simplifyUnit` *Simplify units by removing repeated units from the numerator and denominator*

Description

Simplify units by removing repeated units from the numerator and denominator

Usage

```
simplifyUnit(numerator = "", denominator = "")
```

Arguments

numerator	The numerator of the units (or the whole unit specification)
denominator	The denominator of the units (or NULL if numerator is the whole unit specification)

Details

NA or "" for numerator and denominator are considered unitless.

Value

The units specified with units that are in both the numerator and denominator cancelled.

See Also

Other Unit conversion: modelUnitConversion()

Examples

```
simplifyUnit("kg", "kg/mL")
# units that don't match exactly are not cancelled
simplifyUnit("kg", "g/mL")
```

Index

∗ **Unit conversion**
    modelUnitConversion, 16
    simplifyUnit, 58
.nlmerNlmerFun, 3
.setupPopEDdatabase, 3

as.nlmixr *(*as.nlmixr2*)*, 4
as.nlmixr2, 4

babel.poped.database, 6
babelBpopIdx, 7
bblDatToMonolix, 8
bblDatToMrgsolve *(*bblDatToMonolix*)*, 8
bblDatToNonmem *(*bblDatToMonolix*)*, 8
bblDatToPknca *(*bblDatToMonolix*)*, 8
bblDatToRxode *(*bblDatToMonolix*)*, 8

cell, *39, 40*
create_design_space, *40*

FME::modMCMC(), *12*
fmeMcmcControl, 11

getStandardColNames, 16

lme4::nlmerControl(), *23*

modelUnitConversion, 16
modelUnitConversion(), *59*
monolixControl, 17

nlmerControl *(*nlmixr2NlmerControl*)*, 22
nlmixr2Est.nlmer, 20
nlmixr2Est.pknca, 21
nlmixr2est::.nlmSetupEnv(), *3*, *52, 53*
nlmixr2est::foceiControl(), *57*
nlmixr2est::iterPrintControl(), *56*
nlmixr2est::nlmControl(), *15*, *23*, *46*
nlmixr2est::nlmerSolveGrad(), *3*
nlmixr2NlmerControl, 22
nonmemControl, 25

pkncaControl, 29
popedControl, 30
popedGetMultipleEndpointModelingTimes, 41
popedMultipleEndpointIndexDataFrame *(*popedGetMultipleEndpointModelingTimes*)*, 41
popedMultipleEndpointResetTimeIndex, 42
pseudoOptimControl, 43

rxode2::rxWithSeed(), *13*
rxToMonolix, 47
rxToNonmem, 48
rxUiGet.loadPruneNlmer, 48
rxUiGet.loadPruneNlmerSens, 49
rxUiGet.nlmerEnv, 49
rxUiGet.nlmerFormula, 50
rxUiGet.nlmerHdTheta, 50
rxUiGet.nlmerModel0, 51
rxUiGet.nlmerParams, 51
rxUiGet.nlmerRandomFormula, 52
rxUiGet.nlmerSensModel, 52
rxUiGet.nlmerStart, 53
rxUiGet.nlmerThetaS, 53
rxUiGet.nlmerThetas, 54

saemixControl, 54
simplifyUnit, 58
simplifyUnit(), *17*