Package 'NMsim'

May 7, 2026

Type Package

Title Seamless 'Nonmem' Simulation Platform

Version 0.2.7

Maintainer Philip Delff <philip@delff.dk>

Description A complete and seamless 'Nonmem' simulation interface within R. Turns 'Nonmem' control streams into simulation control streams, executes them with specified simulation input data and returns the results. The simulation is performed by 'Nonmem', eliminating manual work and risks of re-implementation of models in other tools.

License MIT + file LICENSE

RoxygenNote 7.3.3

Depends R (>= 3.5.0)

Imports data.table, NMdata (>= 0.2.2), R.utils, MASS, fst, xfun

Suggests testthat, knitr, rmarkdown, ggplot2, ggstance, patchwork, stringr, tracee, tidyvpc, kableExtra, coveffectsplot, NMcalc, waldo

Enhances simpar

Encoding UTF-8

Additional_repositories https://mpn.metworx.com/snapshots/stable/2024-09-23

BugReports https://github.com/nmautoverse/NMsim/issues

Language en-US

URL https://nmautoverse.github.io/NMsim/

NeedsCompilation no

Author Philip Delff [aut, cre],
Brian Reilly [ctb],
Sanaya Shroff [ctb],
Boris Grinshpun [ctb]

Repository CRAN

Date/Publication 2026-03-19 08:10:19 UTC

Contents

add *Create function that adds text elements to vector*

Description

Namely used to feed functions to modify control streams using ‘NMsim()‘ arguments such as ‘modify‘. Those functions are often onveniently passed a function. ‘add‘ and ‘overwrite‘ are simple shortcuts to creating such functions. Make sure to see examples.

Usage

```
add(..., .pos = "bottom")
```

Arguments

...	Elements to add.
.pos	Either "top" or "bottom". Decides if new text is prepended or appended to existing text.

Value

A function that adds the specified text to character vectors

Examples

```
myfun <- add("b","d")
myfun("a")
## If more convenient, you can add a vector instead.
myfun2 <- add(c("b","d"))
myfun2("a")
myfun3 <- add("b","d",.pos="top")
myfun3("a")
```

addEVID2 *Add simulation records to dosing records*

Description

Deprecated, use ‘NMaddSampples()‘. Adds simulation events to all subjects in a data set. Copies over columns that are not varying at subject level (i.e. non-variying covariates). Can add simulation events relative to previous dosing time.

Usage

```
addEVID2(
  data,
  TIME,
  TAPD,
  CMT,
  EVID,
  DV,
  col.id = "ID",
  args.NMexpandDoses,
  unique = TRUE,
  extras.are.covs = TRUE,
  as.fun,
  doses,
  time.sim
)
```

Arguments

data	Nonmem-style data set. If using 'TAPD' an 'EVID' column must contain 1 for dosing records.
TIME	A numerical vector with simulation times. Can also be a data.frame in which case it must contain a 'TIME' column and is merged with 'data'.
TAPD	A numerical vector with simulation times, relative to previous dose. When this is used, 'data' must contain rows with 'EVID=1' events and a 'TIME' column. 'TAPD' can also be a data.frame in which case it must contain a 'TAPD' column and is merged with 'data'.
CMT	The compartment in which to insert the EVID=2 records. Required if 'CMT' is a column in 'data'. If longer than one, the records will be repeated in all the specified compartments. If a data.frame, covariates can be specified.
EVID	The value to put in the 'EVID' column for the created rows. Default is 2 but 0 may be prefered even for simulation.
DV	Optionally provide a single value to be assigned to the 'DV' column. The default is to assign nothing which will result in 'NA' as samples are stacked ('rbind') with 'data'. If you assign a different value in 'DV', the default value of 'EVID' changes to '0', and 'MDV' will be '0' instead of '1'. An example where this is useful is when generating datasets for '$DESIGN' where 'DV=0' is often used.
col.id	The name of the column in 'data' that holds the unique subject identifier.
args.NMexpandDoses	Only relevant - and likely not needed - if data contains ADDL and II columns. If those columns are included, 'addEVID2()' will use 'NMdata::NMexpanDoses()' to evaluate the time of each dose. Other than the 'data' argument, 'addEVID2()' relies on the default 'NMexpanDoses()' argument values. If this is insufficient, you can specify other argument values in a list, or you can call 'NMdata::NMexpanDoses()' manually before calling 'addEVID2()'.

unique	If 'TRUE' (default), events are reduced to unique time points before insertion. Sometimes, it's easier to combine sequences of time points that overlap (maybe across 'TIME' and 'TAPD'), and let 'addEVID2()' clean them. If you want to keep your duplicated events, use 'unique=FALSE'.
extras.are.covs	If 'TIME' and/or 'TAPD' are 'data.frame's and contain other columns than 'TIME' and/or 'TAPD', those are by default assumed to be covariates to be merged with data. More specifically, they will be merged by when the sample times are added. If 'extras.are.covs=FALSE', they will not be merged by. Instead, they will just be kept as additional columns with specified values, aligned with the sample times.
as.fun	The default is to return data as a 'data.frame'. Pass a function (say 'tibble::as_tibble') in as.fun to convert to something else. If data.tables are wanted, use 'as.fun="data.table"'. The default can be configured using 'NMdataConf()'.
doses	Deprecated. Use 'data'.
time.sim	Deprecated. Use 'TIME'.

Details

The resulting data set is ordered by ID, TIME, and EVID. You may have to reorder for your specific needs.

Value

A data.frame with dosing records

Examples

```
(doses1 <- NMcreateDoses(TIME=c(0,12,24,36),AMT=c(2,1)))
addEVID2(doses1,TIME=seq(0,28,by=4),CMT=2)

## two named compartments
dt.doses <- NMcreateDoses(TIME=c(0,12),AMT=10,CMT=1)
seq.time <- c(0,4,12,24)
dt.cmt <- data.frame(CMT=c(2,3),analyte=c("parent","metabolite"))
res <- addEVID2(dt.doses,TIME=seq.time,CMT=dt.cmt)

## Separate sampling schemes depending on covariate values
dt.doses <- NMcreateDoses(TIME=data.frame(regimen=c("SD","MD","MD"),TIME=c(0,0,12)),AMT=10,CMT=1)

seq.time.sd <- data.frame(regimen="SD",TIME=seq(0,6))
seq.time.md <- data.frame(regimen="MD",TIME=c(0,4,12,24))
seq.time <- rbind(seq.time.sd,seq.time.md)
addEVID2(dt.doses,TIME=seq.time,CMT=2)

## an observed sample scheme and additional simulation times
df.doses <- NMcreateDoses(TIME=0,AMT=50,addl=list(ADDL=2,II=24))
dense <- c(seq(1,3,by=.1),4:6,seq(8,12,by=4),18,24)
trough <- seq(0,3*24,by=24)
sim.extra <- seq(0,(24*3),by=2)
```

```
time.all <- c(dense,dense+24*3,trough,sim.extra)
time.all <- sort(unique(time.all))
dt.sample <- data.frame(TIME=time.all)
dt.sample$isobs <- as.numeric(dt.sample$TIME%in%c(dense,trough))
dat.sim <- addEVID2(dt.doses,TIME=dt.sample,CMT=2)

## TAPD - time after previous dose
df.doses <- NMcreateDoses(TIME=c(0,12),AMT=10,CMT=1)
seq.time <- c(0,4,12,24)
addEVID2(df.doses,TAPD=seq.time,CMT=2)

## TIME and TAPD
df.doses <- NMcreateDoses(TIME=c(0,12),AMT=10,CMT=1)
seq.time <- c(0,4,12,24)
addEVID2(df.doses,TIME=seq.time,TAPD=3,CMT=2)

## Using a custom DV value affects EVID and MDV
df.doses <- NMcreateDoses(TIME=c(0,12),AMT=10,CMT=1)
seq.time <- c(4)
addEVID2(df.doses,TAPD=seq.time,CMT=2,DV=0)
```

---

addResVar *Add residual variability based on parameter estimates*

Description

Add residual variability based on parameter estimates

Usage

```
addResVar(
  data,
  path.ext,
  prop = NULL,
  add = NULL,
  log = FALSE,
  par.type = "SIGMA",
  trunc0 = TRUE,
  scale.par,
  subset,
  seed,
  col.ipred = "IPRED",
  col.ipredvar = "IPREDVAR",
  as.fun
)
```

Arguments

data	A data set containing indiviudual predictions. Often a result of NMsim.
path.ext	Path to the ext file to take the parameter estimates from.
prop	Parameter number of parameter holding variance of the proportional error component. If ERR(1) is used for proportional error, use prop=1. Can also refer to a theta number.
add	Parameter number of parameter holding variance of the additive error component. If ERR(1) is used for additive error, use add=1. Can also refer to a theta number.
log	Should the error be added on log scale? This is used to obtain an exponential error distribution.
par.type	Use "sigma" if variances are estimated with the SIGMA matrix. Use "theta" if THETA parameters are used. See ‘scale.par‘ too.
trunc0	If log=FALSE, truncate simulated values at 0? If trunc0, returned predictions can be negative.
scale.par	Denotes if parmeter represents a variance or a standard deviation. Allowed values and default value depends on ‘par.type‘. • if par.type="sigma" only "var" is allowed. • if par.type="theta" allowed values are "sd" and "var". Default is "sd".
subset	A character string with an expression denoting a subset in which to add the residual error. Example: subset="DVID==’A’"
seed	A number to pass to set.seed() before simulating. Default is to generate a seed and report it in the console. Use seed=FALSE to avoid setting the seed (if you prefer doing it otherwise).
col.ipred	The name of the column containing individual predictions.
col.ipredvar	The name of the column to be created by addResVar to contain the simulated observations (individual predictions plus residual error).
as.fun	The default is to return data as a data.frame. Pass a function (say ‘tibble::as_tibble‘) in as.fun to convert to something else. If data.tables are wanted, use as.fun="data.table". The default can be configured using NMdataConf.

Value

An updated data.frame

Examples

```
## Not run:
## based on SIGMA
simres.var <- addResVar(data=simres,
                        path.ext = "path/to/model.ext",
                        prop = 1,
                        add = 2,
                        par.type = "SIGMA",
                        log = FALSE)
```

```
## If implemented using THETAs
simres.var <- addResVar(data=simres,
                        path.ext = "path/to/model.ext",
                        prop = 8, ## point to elements in THETA
                        add = 9,  ## point to elements in THETA
                        par.type = "THETA",
                        log = FALSE)


## End(Not run)
```

`deleteTmpDirs` *clean up temporary directories left by PSN and NMsim.*

Description

clean up temporary directories left by PSN and NMsim.

Usage

```
deleteTmpDirs(dir, methods, recursive = FALSE, delete = TRUE, as.fun)
```

Arguments

`dir`	The directory in which to look for contents to clean
`methods`	The sources to delete temporary content from. This is a character vector, and the defailt is ‘c("nmsim","psn","psnfit","backup")‘. Each of these correspond to a preconfigured pattern.
`recursive`	Look recursively in folder? Notice, matches will be deleted recursively (they are often directories). ‘recursive‘ controls whether they are searched for recursively.
`delete`	Delete the found matches? If not, the matches are just reported, but nothing deleted.
`as.fun`	Pass a function (say tibble::as_tibble) in as.fun to convert to something else. If data.tables are wanted, use as.fun="data.table". The default is to return data as a data.frame. Modify the defaul using ‘NMdataConf()‘.

Value

data.table with identified items for deletion

expandCovs — *Create data set where each covariate is univariately varied (see 'forestDefineCovs()')*

Description

Create data set where each covariate is univariately varied (see 'forestDefineCovs()')

Usage

```
expandCovs(...)
```

Arguments

...	Passed to 'forestDefineCovs()'

Value

A data.frame

forestDefineCovs — *Create data set where each covariate is univariately varied*

Description

Each covariate is univariately varied while other covariates are kept at reference values. This structure is often used for forest-plot type simulations.

Usage

```
forestDefineCovs(
  ...,
  data,
  col.id = "ID",
  sigdigs = 2,
  reduce.ref = TRUE,
  as.fun
)
```

Arguments

...	Covariates provided as lists - see examples. The name of the arguement must match columns in data set. An element called ref must contain either a reference value or a function to use to derive the reference value from data (e.g. 'median'). Provide either 'values' or 'quantiles' to define the covariate values of interest (typically, the values that should later be simulated and maybe shown in a forest plot). 'label' is optional - if missing, the argument name will be used. If quantiles are requested, they are derived after requiring unique values for each subject.
data	A data set needed if the reference(s) value of one or more covariates is/are provided as functions (like median), or if covariate values are provided as quantiles.
col.id	The subject ID column name. Necessary because quantiles sould be quantiles of distribution of covariate on subjects, not on observations (each subject contributes once).
sigdigs	Used for rounding of covariate values if using quantiles or if using a function to find reference.
reduce.ref	If 'TRUE' (default), only return one row with all reference values. If 'FALSE' there will be one such row for each covariate. When reduced to one line, all columns related to covariate-level information such as covariate name will contain 'NA' for the reference.
as.fun	The default is to return data as a data.frame. Pass a function (say 'tibble::as_tibble') in as.fun to convert to something else. If data.tables are wanted, use as.fun="data.table". The default can be configured using NMdataConf.

Value

A data.frame

Examples

```
## Not run:
file.mod <- system.file("examples/nonmem/xgxr134.mod",package="NMdata")
res <- NMdata::NMscanData(file.mod)
forestDefineCovs(
    WEIGHTB=list(ref=70,values=c(40,60,80,100),label="Bodyweight (kg)"),
## notice, values OR quantiles can be provided
    AGE=list(ref=median, quantiles=c(10,25,75,90)/100, label="Age (years)"
             ),
    data=res
)

## End(Not run)
```

`forestSummarize` *Summarize simulated exposures relative to reference subject*

Description

Summarize simulated exposures relative to reference subject

Usage

```
forestSummarize(data, funs.exposure, cover.ci = 0.95, by, as.fun)
```

Arguments

`data`	Simulated data to process. This data.frame must contain must contain multiple columns, as defined by 'NMsim::forestDefineCovs()'.
`funs.exposure`	A named list of functions to apply for derivation of exposure metrics.
`cover.ci`	The coverage of the confidence intervals. Default is 0.95.
`by`	a character vector of column names to perform all calculations by. This could be sampling subsets or analyte.
`as.fun`	The default is to return data as a 'data.frame'. Pass a function (say 'tibble::as_tibble') in as.fun to convert to something else. If data.tables are wanted, use 'as.fun="data.table"'. The default can be configured using 'NMdataConf()'.

Details

This function is part of the workflow provided by NMsim to generate forest plots - a graphical representation of the estimated covariate effects and the uncertainty of those effect estimates. 'forestDefineCovs()' helps construct a set of simulations to perform, simulation methods like 'NMsim_VarCov' and 'NMsim_NWPRI' can perform siulations with parameter uncertainty, and 'forestSummarize()' can then summarize those simulation results into the numbers to plot in a forest plot. See the NMsim vignette on forest plot generation available on the NMsim website for a step-by-step demonstration.

The following columns are generated by 'forestDefineCovs()' and are expected to be present. Differences within any of them will lead to separate summarizing (say for as covariate value to be plotted):

- 'model': A model identifier - generated by 'NMsim()'.
- 'type': The simulation type. "ref" for reference subject, "value" for any other. This is generated by 'forestDefineCovs()'.
- 'covvar': The covariate (of interest) that is different from the reference value in the specific simulation. Example: "WT"
- 'covlabel': Label of the covariate of interest. Example: "Bodyweight (kg)"
- 'covref': Reference value of the covariate of interest. Example: 80
- 'covval': Value of the covariate of interest (not reference). Example 110.

Value

A data.frame

`genPhiFile` *Generate a .phi file for further simulation with Nonmem*

Description

This will typically be used in a couple of different situations. One is if a number of new subjects have been simulated and their ETAs should be reused in subsequent simulations. Another is internally by NMsim when simulating new subjects from models estimated with SAEM.

Usage

```
genPhiFile(data, file, overwrite = FALSE)
```

Arguments

`data`	A dataset that contains "ID" and all 'ETA's. This can be obtained by 'NMdata::NMscanData'.
`file`	Path to the .phi file to be written.
`overwrite`	If 'file' exists already, overwrite it? Default is 'FALSE'.

Value

Invisibly, character lines (strings) optionally written to file

See Also

simPopEtas

`inputArchiveDefault` *Default location of input archive file*

Description

Default location of input archive file

Usage

```
inputArchiveDefault(file)
```

Arguments

`file`	Path to input or output control stream.

Value

A file name (character)

`modTab` *Get NMsim model metadata*

Description

Get NMsim model metadata

Usage

```
modTab(res)
```

Arguments

`res`	NMsim results (class 'NMsimRes').

Details

- ROWMODEL (integer): A unique row identifier
- file.mod (character): Path to the originally provided input control stream, relative to current working directory.
- path.sim (character): Path to the simulation input control stream, relative to current working directory.
- path.rds (character): Path to the results meta data file (_path.rds0)
- model (character): The name of the original model, no extension. Derived from file.mod. If file.mod is named, the provided name is used.;
- model.sim (character): A unique and cleaned (no special characters) name for the derived model, without extension. Notice if a simulation method generates multiple models, model.sim will be distinct for those. This is unlike model and name.sim.
- name.sim (character): The value of the NMsim() argument of the same name at function call.
- fn.sim (character): Name of the mod file to be simulated. Has .mod extension. It will differ from file mod in being derived from model.sim so it is unique and cleaned.
- dir.sim (character): Relative path from point of execution to simulation directory. Cleaned.
- path.mod.exec (character): Path to the control stream executed by Nonmem, relative to current working directory.

Value

A table with model details

NMaddSamples *Add simulation (sample) records to dosing records*

Description

Adds simulation events to all subjects in a data set. Copies over columns that are not varying at subject level (i.e. non-variying covariates). Can add simulation events relative to previous dosing time. This function was previously called 'addEVID2()'.

Usage

```
NMaddSamples(
  data,
  TIME,
  TAPD,
  CMT,
  EVID,
  DV,
  col.id = "ID",
  args.NMexpandDoses,
  unique = TRUE,
  by,
  quiet = FALSE,
  as.fun,
  doses,
  time.sim,
  extras.are.covs
)
```

Arguments

data	Nonmem-style data set. If using 'TAPD' an 'EVID' column must contain 1 for dosing records.
TIME	A numerical vector with simulation times. Can also be a data.frame in which case it must contain a 'TIME' column and is merged with 'data'.
TAPD	A numerical vector with simulation times, relative to previous dose. When this is used, 'data' must contain rows with 'EVID=1' events and a 'TIME' column. 'TAPD' can also be a data.frame in which case it must contain a 'TAPD' column and is merged with 'data'.
CMT	The compartment in which to insert the EVID=2 records. Required if 'CMT' is a column in 'data'. If longer than one, the records will be repeated in all the specified compartments. If a data.frame, covariates can be specified.
EVID	The value to put in the 'EVID' column for the created rows. Default is 2 but 0 may be prefered even for simulation.

`DV` Optionally provide a single value to be assigned to the ‘DV‘ column. The default is to assign nothing which will result in ‘NA‘ as samples are stacked (‘rbind‘) with ‘data‘. If you assign a different value in ‘DV‘, the default value of ‘EVID‘ changes to ‘0‘, and ‘MDV‘ will be ‘0‘ instead of ‘1‘. An example where this is useful is when generating datasets for ‘$DESIGN‘ where ‘DV=0‘ is often used.

`col.id` The name of the column in ‘data‘ that holds the unique subject identifier. Currently, this is needed to be non-‘NULL‘.

`args.NMexpandDoses` Only relevant - and likely not needed - if data contains ADDL and II columns. If those columns are included, ‘NMaddSamples()‘ will use ‘NMdata::NMexpanDoses()‘ to evaluate the time of each dose. Other than the ‘data‘ argument, ‘NMaddSamples()‘ relies on the default ‘NMexpanDoses()‘ argument values. If this is insufficient, you can specify other argument values in a list, or you can call ‘NMdata::NMexpanDoses()‘ manually before calling ‘NMaddSamples()‘.

`unique` If ‘TRUE‘ (default), events are reduced to unique time points before insertion. Sometimes, it’s easier to combine sequences of time points that overlap (maybe across ‘TIME‘ and ‘TAPD‘), and let ‘NMaddSamples()‘ clean them. If you want to keep your duplicated events, use ‘unique=FALSE‘.

`by` If `TIME` and/or ‘TAPD‘ are ‘data.frame‘s and contain other columns than ‘TIME‘ and/or ‘TAPD‘, those will by default follow the ‘TIME‘/‘TAPD‘ records. Think of them as record-level variables, like ‘VISIT‘. The exception is ‘col.id‘ - if the subject identifier is present, it will be merged by. If additional columns should be used to merge by, you can use the ‘by‘ argument. This is useful to generate differentiated sampling schemes for subsets of subjects (like regimen="SAD" and regimen="MAD"). If no columns in ‘TIME‘ and/or ‘TAPD‘ should not be merged by, use ‘by=FALSE‘. You can also specify selected ‘by‘ variables like ‘by="ID"‘ or ‘by=c("ID","regimen")‘ See examples.

`quiet` Suppress messages? Default is ‘FALSE‘.

`as.fun` The default is to return data as a ‘data.frame‘. Pass a function (say ‘tibble::as_tibble‘) in as.fun to convert to something else. If data.tables are wanted, use ‘as.fun="data.table"‘. The default can be configured using ‘NMdataConf()‘.

`doses` Deprecated. Use ‘data‘.

`time.sim` Deprecated. Use ‘TIME‘.

`extras.are.covs` Deprecated. Use ‘by‘.

Details

The resulting data set is ordered by ID, TIME, and EVID. You may have to reorder for your specific needs.

Value

A data.frame with dosing records only using column names in covs.data (from data) that are not in TIME.

All rows in TIME get reused for all matches by column names common with covs.data - the identified subject-level covariates (and col.id). This is with the exception of the TIME column itself, because in case of single dose, TIME would be carried over.

Examples

```
(doses1 <- NMcreateDoses(TIME=c(0,12,24,36),AMT=c(2,1)))
NMaddSamples(doses1,TIME=seq(0,28,by=4),CMT=2)

## two named compartments
dt.doses <- NMcreateDoses(TIME=c(0,12),AMT=10,CMT=1)
seq.time <- c(0,4,12,24)
dt.cmt <- data.frame(CMT=c(2,3),analyte=c("parent","metabolite"))
res <- NMaddSamples(dt.doses,TIME=seq.time,CMT=dt.cmt)

## Separate sampling schemes depending on covariate values
dt.doses <- NMcreateDoses(TIME=data.frame(regimen=c("SD","MD","MD"),TIME=c(0,0,12)),AMT=10,CMT=1)

seq.time.sd <- data.frame(regimen="SD",TIME=seq(0,3))
seq.time.md <- data.frame(regimen="MD",TIME=c(0,12,24))
seq.time <- rbind(seq.time.sd,seq.time.md)
NMaddSamples(dt.doses,TIME=seq.time,CMT=2,by="regimen")

## All subjects get all samples
NMaddSamples(dt.doses,TIME=seq.time,by=FALSE,CMT=2)

## an observed sample scheme and additional simulation times
df.doses <- NMcreateDoses(TIME=0,AMT=50,addl=list(ADDL=2,II=24))
dense <- c(seq(1,3,by=.1),4:6,seq(8,12,by=4),18,24)
trough <- seq(0,3*24,by=24)
sim.extra <- seq(0,(24*3),by=2)
time.all <- c(dense,dense+24*3,trough,sim.extra)
time.all <- sort(unique(time.all))
dt.sample <- data.frame(TIME=time.all)
dt.sample$isobs <- as.numeric(dt.sample$TIME%in%c(dense,trough))
dat.sim <- NMaddSamples(dt.doses,TIME=dt.sample,CMT=2)

## TAPD - time after previous dose
df.doses <- NMcreateDoses(TIME=c(0,12),AMT=10,CMT=1)
seq.time <- c(0,4,12,24)
NMaddSamples(df.doses,TAPD=seq.time,CMT=2)

## TIME and TAPD
df.doses <- NMcreateDoses(TIME=c(0,12),AMT=10,CMT=1)
seq.time <- c(0,4,12,24)
NMaddSamples(df.doses,TIME=seq.time,TAPD=3,CMT=2)

## Using a custom DV value affects EVID and MDV
df.doses <- NMcreateDoses(TIME=c(0,12),AMT=10,CMT=1)
seq.time <- c(4)
NMaddSamples(df.doses,TAPD=seq.time,CMT=2,DV=0)
```

NMcreateDoses *Easily and flexibly generate dosing records*

Description

Columns will be extended by repeating last value of the column if needed in order to match length of other columns. Combinations of different columns can be generated by specifying covariates on the columns where the regimens differ.

Usage

```
NMcreateDoses(
  TIME,
  AMT,
  EVID = 1,
  CMT = 1,
  ADDL = NULL,
  II = NULL,
  RATE = NULL,
  SS = NULL,
  addl = NULL,
  N,
  addl.lastonly = TRUE,
  col.id = "ID",
  as.fun
)
```

Arguments

TIME	The time of the dosing events. Required.
AMT	vector or data.frame with amounts amount. Required.
EVID	The event ID to use for doses. Default is to use EVID=1, but EVID might also be wanted.
CMT	Compartment number. Default is to dose into CMT=1. Use 'CMT=NA' or 'CMT=NULL' to omit in result.
ADDL	Number of additional dose events. Must be in combination with and consistent with II. Notice if of length 1, only applied to last event in each regimen.
II	Dosing frequency of additional events specified in 'ADDL'. See 'ADDL' too.
RATE	Infusion rate. Optional.
SS	steady-state flag. Optional.
addl	A list of ADDL and II that will be applied to last dose. This may be prefered if II and ADDL depend on covariates - see examples. Optional.
N	Number of replications. Default is 1. If 'N=1' results in two distinct subjects, 'N=100' will result i 200 distinct subjects. The ID column will automatically be recoded to contain distinct ID's.

addl.lastonly	If ADDL and II are of length 1, apply only to last event of a regimen? The default is 'TRUE'.
col.id	Default is to denote the dosing regimens by an ID column. The name of the column can be modified using this argument. Use 'col.id=NA' to omit the column altogether. The latter may be wanted if repeating the regimen for a number of subjects after running 'NMcreateDoses()'.
as.fun	The default is to return data as a data.frame. Pass a function (say 'tibble::as_tibble') in as.fun to convert to something else. If data.tables are wanted, use as.fun="data.table". The default can be configured using NMdataConf.

Details

Only TIME and AMT are required. AMT, RATE, SS, II, ADDL, CMT are of length 1 or longer. Those not of max length 1 are repeated. If TIME is longer than those, they are extended to match length of TIME. All these arguments can be data.frames with additional columns that define distinct dosing regimens - with distinct subject ids. However, if covariates are applied to ADDL+II, see the addl argument and see examples.

Allowed combinations of AMT, RATE, SS, II here: https://ascpt.onlinelibrary.wiley.com/doi/10.1002/psp4.12404

Value

A data.frame with dosing events

Examples

```
library(data.table)
## Users should not use setDTthreads. This is for CRAN to only use 1 core.
data.table::setDTthreads(1)
## arguments are expanded - makes loading easy
NMcreateDoses(TIME=c(0,12,24,36),AMT=c(2,1))
## Different doses by covariate
NMcreateDoses(TIME=c(0,12,24),AMT=data.table(AMT=c(2,1,4,2),DOSE=c(1,2)))
## Make Nonmem repeat the last dose. This is a total of 20 dosing events.
## The default, addl.lastonly=TRUE means if ADDL and II are of
## length 1, they only apply to last event.
NMcreateDoses(TIME=c(0,12),AMT=c(2,1),ADDL=9*2,II=12)
dt.amt <- data.table(DOSE=c(100,400))
## multiple dose regimens.
## Specifying the time points explicitly
dt.amt <- data.table(AMT=c(200,100,800,400)*1000,DOSE=c(100,100,400,400))
doses.md.1 <- NMcreateDoses(TIME=seq(0,by=24,length.out=7),AMT=dt.amt)
doses.md.1$dose <- paste(doses.md.1$DOSE,"mg")
doses.md.1$regimen <- "QD"
doses.md.1
## or using ADDL+II
dt.amt <- data.table(AMT=c(200,100,800,400)*1000,DOSE=c(100,100,400,400))
doses.md.2 <- NMcreateDoses(TIME=c(0,24),AMT=dt.amt,addl=data.table(ADDL=c(0,5),II=c(0,24)))
doses.md.2$dose <- paste(doses.md.2$DOSE,"mg")
doses.md.2$regimen <- "QD"
```

```
doses.md.2
## ADDL and II can be wrapped in a data.frame. This allows including covariates
NMcreateDoses(TIME=c(0,12),AMT=c(2,1),addl=data.frame(ADDL=c(NA,9*2),II=c(NA,12),trt=c("A","B")))
```

NMexec *Execute Nonmem and archive input data with model files*

Description

Execute Nonmem from within R - optionally but by default in parallel. Archiving the input data ensures that postprocessing can still be reproduced if the input data files should be updated.

Usage

```
NMexec(
  files,
  file.pattern,
  dir,
  sge = TRUE,
  input.archive,
  nc,
  dir.data = NULL,
  wait = FALSE,
  path.nonmem,
  update.only = FALSE,
  fun.post,
  method.execute,
  nmfe.options,
  dir.psn,
  args.psn.execute,
  files.needed,
  clean = 1,
  backup = TRUE,
  quiet = FALSE,
  nmquiet = FALSE,
  system.type
)
```

Arguments

files	File paths to the models (control streams) to run nonmem on. See file.pattern too.
file.pattern	Alternatively to files, you can supply a regular expression which will be passed to list.files as the pattern argument. If this is used, use dir argument as well. Also see data.file to only process models that use a specific data file.
dir	If file.pattern is used, dir is the directory to search for control streams in.

`sge` Use the sge queing system. Default is TRUE. Disable for quick models not to wait for the queue to run the job.

`input.archive` A function of the model file path to generate the path in which to archive the input data as RDS. Set to FALSE not to archive the data.

`nc` Number of cores to use if sending to the cluster. This will only be used if `method.execute="psn"`, and `sge=TRUE`. Default is 64.

`dir.data` The directory in which the data file is stored. This is normally not needed as data will be found using the path in the control stream. This argument may be removed in the future since it should not be needed.

`wait` Wait for process to finish before making R console available again? This is useful if calling NMexec from a function that needs to wait for the output of the Nonmem run to be available for further processing.

`path.nonmem` The path to the nonmem executable. Only used if `method.execute="direct"` or `method.execute="nmsim"` (which is not default). If this argument is not supplied, NMexec will try to run nmfe75, i.e. this has to be available in the path of the underlying shell. The default value can be modified using `NMdata::NMdataConf`, like `NMdataConf(path.nonmem="/path/to/nonmem")`

`update.only` Only run model(s) if control stream or data updated since last run?

`fun.post` A function of the path to the control stream (‘file.mod‘) that generates bash code to be evaluated once Nonmem is done. This can be used to automatically run a goodness-of-fit script or a simulation script after model estimation.

`method.execute` How to run Nonmem. Must be one of ’psn’, ’nmsim’, or ’direct’.

- psn PSN’s execute is used. This supports parallel Nonmem runs. Use the `nc` argument to control how many cores to use for each job. For estimation runs, this is most likely the better choice, if you have PSN installed. See `dir.psn` argument too.
- nmsim Creates a temporary directory and runs Nonmem inside that directory before copying relevant results files back to the folder where the input control stream was. If `sge=TRUE`, the job will be submitted to a cluster, but parallel execution of the job itself is not supported. See `path.nonmem` argument too.
- direct Nonmem is called directly on the control stream. This is the simplest method and is the least convenient in most cases. It does not offer parallel runs and leaves all the Nonmem output files next to the control streams.

See ‘sge‘ as well.

`nmfe.options` additional options that will be passed to nmfe. It is only used when path.nonmem is available (directly or using ‘NMdataConf()‘). Default is "-maxlim=2" For PSN, see ‘args.psn.execute‘.

`dir.psn` The directory in which to find PSN executables. This is only needed if these are not searchable in the system path, or if the user should want to be explicit about where to find them (i.e. want to use a specific installed version of PSN).

`args.psn.execute`
A character string with arguments passed to execute. Default is "-model_dir_name -nm_output=coi,cor,cov,ext,phi,shk,xml".

files.needed	In case method.execute="nmsim", this argument specifies files to be copied into the temporary directory before Nonmem is run. Input control stream and simulation input data does not need to be specified.
clean	The degree of cleaning (file removal) to do after Nonmem execution. If 'method.execute=="psn"', this is passed to PSN's 'execute'. If 'method.execute=="nmsim"' a similar behavior is applied, even though not as granular. NMsim's internal method only distinguishes between 0 (no cleaning), any integer 1-4 (default, quite a bit of cleaning) and 5 (remove temporary dir completely).
backup	Before running, should existing results files be backed up in a sub directory? If not, the files will be deleted before running.
quiet	Suppress messages on what NMexec is doing? Default is FALSE.
nmquiet	Suppress terminal output from 'Nonmem'. This is likely to only work on linux/unix systems.
system.type	A charachter string, either \"windows\" or \"linux\" - case insensitive. Windows is only experimentally supported. Default is to use `Sys.info()[["sysname"]]`.

Details

Use this to read the archived input data when retrieving the nonmem results: `NMdataConf(file.data=inputArchiveDefaul`

Since 'NMexec' will typically not be used for simulations directly ('NMsim' is the natural interface for that purpose), the default method for 'NMexec' is currently to use 'method.execute="psn"' which is at this point the only of the methods that allow for multi-core execution of a single Nonmem job (NB: 'method.execute="NMsim"' can run multiple jobs in parallel which is normally sufficient for simulations).

Value

NULL (invisibly)

Examples

```
file.mod <- "run001.mod"
## Not run:
## run locally - not on cluster
NMexec(file.mod,sge=FALSE)
## run on cluster with 16 cores. 64 cores is default
NMexec(file.mod,nc=16)
## submit multiple models to cluster
multiple.models <- c("run001.mod","run002.mod")
NMexec(multiple.models,nc=16)
## run all models called run001.mod - run099.mod if updated. 64 cores to each.
NMexec(file.pattern="run0..\\.mod",dir="models",nc=16,update.only=TRUE)

## End(Not run)
```

NMreadSim *Read simulation results based on NMsim's track of model runs*

Description

Read simulation results based on NMsim's track of model runs

Usage

```
NMreadSim(
  x,
  check.time = FALSE,
  dir.sims,
  wait = FALSE,
  quiet = FALSE,
  progress,
  skip.missing = FALSE,
  rm.tmp = FALSE,
  as.fun
)
```

Arguments

x	Path to the simulation-specific rds file generated by NMsim, typically called 'NMsim_MetaData.rds'. Can also be a table of simulation runs as stored in 'rds' files by 'NMsim'. The latter should almost never be used.
check.time	If found, check whether 'fst' file modification time is newer than 'rds' file. The 'fst' is generated based on information in 'rds', but notice that some systems don't preserve the file modification times. Becasue of that, 'check.time' is 'FALSE' by default.
dir.sims	By default, 'NMreadSim' will use information about the relative path from the results table file ('_MetaData.rds') to the Nonmem simulation results. If these paths have changed, or for other reasons this doesn't work, you can use the 'dir.sims' argument to specify where to find the Nonmem simulation results. If an '.fst' file was already generated and is found next to the '_MetaData.rds', the path to the Nonmem simulation results is not used.
wait	If simulations seem to not be done yet, wait for them to finish? If not, an error will be thrown. If you choose to wait, the risk is results never come. 'NMreadSim' will be waiting for an 'lst' file. If Nonmem fails, it will normally generate an 'lst' file. But if 'NMTRAN' fails (checks of control stream prior to running Nonmem), the 'lst' file is not generated. Default is not to wait.
quiet	Turn off some messages about what is going on? Default is to report the messages.
progress	Track progress? Default is 'TRUE' if 'quiet' is FALSE and more than one model is being read. The progress tracking is based on the number of models completed/read, not the status of the individual models.

skip.missing	Skip models where results are not available? Default is 'FALSE' meaning an error will be thrown if one or more models do not have completed results.
rm.tmp	If results are read successfully, remove temporary simulation results files? This can be useful after a script is developed and intermediate debugging information is not needed. It cleans up and saves significant disk space.
as.fun	The default is to return data as a data.frame. Pass a function (say 'tibble::as_tibble') in as.fun to convert to something else. If data.tables are wanted, use as.fun="data.table". The default can be configured using NMdataConf.

Value

A data set of class defined by as.fun

NMsim *Simulate from a Nonmem model*

Description

Supply a data set and an estimation input control stream, and NMsim can create neccesary files (control stream, data files), run the simulation and read the results. It has additional methods for other simulation types available, can do multiple simulations at once and more. Please see vignettes for an introduction to how to get the most out of this.

Usage

```
NMsim(
  file.mod,
  data,
  subproblems = NULL,
  reuse.results = FALSE,
  seed.R,
  seed.nm,
  name.sim,
  table.vars,
  table.options,
  table.format = "s1PE16.9",
  carry.out = TRUE,
  method.sim = NMsim_default,
  typical = FALSE,
  inits,
  modify,
  filters,
  sizes,
  path.nonmem = NULL,
  sge = FALSE,
  nc = 1,
```

```
  execute = TRUE,
  script = NULL,
  transform = NULL,
  order.columns = TRUE,
  method.execute,
  nmfe.options,
  nmrep,
  col.flagn = FALSE,
  dir.psn,
  args.psn.execute,
  args.NMscanData,
  as.fun,
  system.type = NULL,
  dir.sims,
  dir.res,
  file.res,
  dir.sim.sub = TRUE,
  wait,
  text.sim = "",
  auto.dv = TRUE,
  clean,
  sim.dir.from.scratch = TRUE,
  create.dirs = TRUE,
  quiet = FALSE,
  nmquiet,
  progress,
  check.mod = TRUE,
  format.data.complete = "rds",
  text.table,
  suffix.sim,
  seed,
  file.ext = NULL,
  method.update.inits,
  modify.model,
  list.sections,
  ...
)
```

Arguments

file.mod	Path(s) to the input control stream(s) to run the simulation on. The output control stream is for now assumed to be stored next to the input control stream and ending in .lst instead of .mod. The .ext file must also be present. If simulating known subjects, the .phi is necessary too.
data	The simulation data as a data.frame or a list of data.frames. If a list, the model(s) will be run on each of the data sets in the list.
subproblems	Number of subproblems to use as SUBPROBLEMS in $SIMULATION block in Nonmem. The default is subproblem=0 which means not to use SUBPROBLEMS.

`reuse.results` If simulation results found on file, should they be used? If TRUE and reading the results fail, the simulations will still be rerun.

`seed.R` A value passed to `set.seed()`. It is recommended to use `seed.R` rather than calling `set.seed()` manually because the seed can then be captured and stored by `NMsim()` for reproducibility. See `seed.nm` for finer control of the seeds that are used in the Nonmem control streams.

`seed.nm` Control Nonmem seeds. If a numeric, a vector or a 'data.frame', these are used as the the seed values (a single value or vector will be recycled so make sure the dimesnsions are right, the number of columns in a `data.frame` will dictate the number of seeds in each Nonmem control stream. Use a list with elements 'values', and 'dist' and others for detailed control of the random sources. See `?NMseed` for details on what arguments can be passed this way.

Default is to draw seeds betwen 0 and 2147483647 (the values supported by Nonmem) for each simulation. You can pass a function that will be evaluated (say to choose a different pool of seeds to draw from).

To avoid changing an exisiting seed in a control stream, use `seed.nm="asis"`.

In case `method.sim=NMsim_EBE`, seeds are not used.

`name.sim` Give all filenames related to the simulation a suffix. A short string describing the sim is recommended like "ph3_regimens".

`table.vars` Variables to be printed in output table as a character vector or a space-separated string of variable names. The default is to export the same tables as listed in the input control stream. If `table.vars` is provided, all output tables in estimation control streams are dropped and replaced by a new one with just the provided variables. If many variables are exported, and much fewer are used, it can speed up NMsim significantly to only export what is needed (sometimes this is as little as "PRED IPRED"). Nonmem writes data slowly so reducing output data can make a very big difference in execution time. See `table.options` too.

`table.options` A character vector or a string of space-separated options. Only used if `table.vars` is provided. If constructing a new output table with `table.vars` the default is to add two options, `NOAPPEND` and `NOPRINT`. You can modify that with `table.options`. Do not try to modify output filename - `NMsim` takes care of that. See 'table.format' too.

`table.format` A format for '$TABLE'. Only used if 'table.vars' is provided. Default is "s1PE16.9". NMsim needs a high-resolution format. The Nonmem default "s1PE11.4" is insufficient for simulation data sets of 1e5 rows or more.

`carry.out` Variables from input data that should be included in results. Default is to include everything. If working with large data sets, it may be wanted to provide a subset of the columns here. If doing very large simulations, this may also be a way to save memory.

`method.sim` A function (not quoted) that creates the simulation control stream and other necessary files for a simulation based on the estimation control stream, the data, etc. The default is called `NMsim_default` which will replace any estimation and covariance step by a simulation step. See details section on oter methods, and see examples and especially vignettes on how to use the different provided methods.

`typical` Run with all ETAs fixed to zero? Technically all ETAs=0 is obtained by replacing `$OMEGA` by a zero matrix. Default is 'FALSE'. Instead of a logical 'TRUE/FALSE', a character vector can be used to specify what parameter types to set to zero and fix. Examples: 'typical=c("OMEGA","SIGMA")', 'typical=c("THETAPV","OMEGA","OMEGAP","OMEGAPD")'. In fact, if 'typical=TRUE', both '$OMEGA' itself and - if found - their priors will be fixed at zero.

`inits` Control the parameter values. 'inits' is a list and contains (any of) the 'method' used to edit the parameters, and what modifications to do.

Using the defaul 'method', all other list elements are passed as arguments to 'NMwriteInits()'. Please see '?NMwriteInits' and the examples on the NMsim website for how to edit the parameter values: https://nmautoverse.github.io/NMsim/articles/NMsim-modify-model.html

The 'method' element controls which method is used to do this, and this corresponds to the old 'method.update.initxfgs' argument. Normally, the user should not need to deal with this as the default 'nmsim' method is very flexible and powerful. If using the new 'method=nmsim' you can specify parameter values, fix/unfix them, and edit lower and upper limits for estimation.

- 'method="nmsim"' (default) A highly flexible internal method, allows for modification of the parameter values. All other elements in 'inits' are passed to 'NMwriteInits()'. Example where 'THETA(2)' is customized: 'inits=list("THETA(2)"=list(init=1.3))'. See '?NMwriteInits' too.
- 'method="psn"' Uses PSN's "update_inits". Requires a functioning PSN installation and possibly that `dir.psn` is correctly set. The advantages of this method are that it keeps comments in the control stream and that it is a method known to many.
- 'method="simple"' Uses a simple internal method to update the parameter values based on the ext file. The advantages are it does not require PSN, and that it does not rely on code-interpretation for generation of simulation control streams. "simple" fixes the whole OMEGA and SIGMA matrices as single blocks which is robust because it avoids any interpretation of BLOCK structure or other code in the control streams. The downside is it strips all comments, and generally makes the $OMEGA and $SIGMA sections of the simulation control streams less easy to read. "simple" can be used as a fallback in case of any issues with 'method="nmsim"'.
- 'method="none"' Do nothing. This is useful if the model to simulate has not been estimated but parameter values have been manually put into the respective sections in the control stream.

See also 'file.ext' which can now be handled by 'inits' too. This change collects the update of the "initial" parameter values into one interface rather than multiple arguments.

`modify` Named list of additional control stream section edits. Note, these can be functions that define how to edit sections. This is an advanced feature which is not needed to run most simulations. It is however powerful for some types of analyses, like modifying parameter values. See vignettes for further information.

`filters` Edit data filters ('IGNORE'/'ACCEPT' statements) before running model. This should normally only be used if no data set is provided. It can be useful if

simulating for a VPC but a different subset of data needs to be simulated than the one used for estimation. A common example on this is inclusion of BLQ's in the VPC even if they were excluded in the estimation. See '?NMreadFilters' which returns a table you can edit and pass to 'filters'. You can also just pass a string representing the full set of filters to be used. If you pass a string, consider including "IGN=@" to avoid character rows, like the column headers.

`sizes` If needed, adjust the '$SIZES' section by providing a list of arguments to 'NMupdateSizes()'. Example: 'sizes=list(PD=80)'. See '?NMupdateSizes' for details. Don't use arguments like 'file.mod' and 'newfile' which are handled internally.

`path.nonmem` The path to the Nonmem executable to use. The could be something like "/usr/local/NONMEM/run/nmfe7 (which is a made up example). No default is available. You should be able to figure this out through how you normally execute Nonmem, or ask a colleague.

`sge` Submit to cluster? Default is not to, but this is very useful if creating a large number of simulations, e.g. simulate with all parameter estimates from a bootstrap result.

`nc` Number of cores used in parallelization. Only used if 'sge=TRUE'.

`execute` Execute the simulation or only prepare it? 'execute=FALSE' can be useful if you want to do additional tweaks or simulate using other parameter estimates.

`script` The path to the script where this is run. For stamping of dataset so results can be traced back to code.

`transform` A list defining transformations to be applied after the Nonmem simulations and before plotting. For each list element, its name refers to the name of the column to transform, the contents must be the function to apply.

`order.columns` reorder columns by calling `NMdata::NMorderColumns` before saving dataset and running simulations? Default is TRUE.

`method.execute` Specify how to call Nonmem. Options are "psn" (PSN's execute), "nmsim" (an internal method similar to PSN's execute), and "direct" (just run Nonmem directly and dump all the temporary files). "nmsim" has advantages over "psn" that makes it the only supported method when type.sim="NMsim_EBE". "psn" has the simple advantage that the path to nonmem does not have to be specified if "execute" is in the system search path. So as long as you know where your Nonmem executable is, "nmsim" is recommended. The default is "nmsim" if path.nonmem is specified, and "psn" if not.

`nmfe.options` additional options that will be passed to nmfe. It is only used when path.nonmem is available (directly or using 'NMdataConf()'). Default is "-maxlim=2" For PSN, see 'args.psn.execute'.

`nmrep` Include 'NMREP' as counter of subproblems? The default is to do so if 'subproblems>0'. This will insert a counter called 'NMREP' in the '$ERROR' section and include that in the output table(s). At this point, nothing is done to avoid overwriting existing variables.

`col.flagn` Only used if 'data' is provided. Use this if you are including an exclusion flag column in data. However, what NMsim will then do is to require that column to equal '0' (zero) for the rows to be simulated. It is often better to subset the data before simulation. See 'filters' too.

`dir.psn` The directory in which to find PSN's executables ('execute' and 'update_inits'). The default is to rely on the system's search path. So if you can run 'execute' and 'update_inits' by just typing that in a terminal, you don't need to specify this unless you want to explicitly use a specific installation of PSN on your system.

`args.psn.execute`
A charachter string that will be passed as arguments PSN's 'execute'. The default is "-model_dir_name -nm_output=coi,cor,cov,ext,phi,shk,xml -nmfe_options=\"-maxlim=2\"" in addition to the "-clean" based on the 'clean' argument. Notice, if 'path.nonmem' is provided, the default is not to use PSN.

`args.NMscanData`
If `execute=TRUE&sge=FALSE`, NMsim will normally read the results using `NMreadSim`. Use this argument to pass additional arguments (in a list) to that function if you want the results to be read in a specific way. This can be if the model for some reason drops rows, and you need to merge by a row identifier. You would do 'args.NMscanData=list(col.row="ROW")' to merge by a column called 'ROW'. This is only used in rare cases.

`as.fun` The default is to return data as a data.frame. Pass a function (say 'tibble::as_tibble') in as.fun to convert to something else. If data.tables are wanted, use as.fun="data.table". The default can be configured using NMdataConf.

`system.type` A charachter string, either "windows" or "linux" - case insensitive. Windows is only experimentally supported. Default is to use `Sys.info()[["sysname"]]`.

`dir.sims` The directory in which NMsim will store all generated files. Default is to create a folder called 'NMsim' next to 'file.mod'.

`dir.res` Provide a path to a directory in which to save rds files with paths to results. Default is to use dir.sims. After running 'NMreadSim()' on these files, the original simulation files can be deleted. Hence, providing both 'dir.sims' and 'dir.res' provides a structure that is simple to clean. 'dir.sims' can be purged when 'NMreadSim' has been run and only small 'rds' and 'fst' files will be kept in 'dir.res'. Notice, in case multiple models are simulated, multiple 'rds' (to be read with 'NMreadSim()') files will be created by default. In cases where multiple models are simulated, see 'file.res' to get just one file refering to all simulation results.

`file.res` Path to an rds file that will contain a table of the simulated models and other metadata. This is needed for subsequently retrieving all the results using 'NMreadSim()'. The default is to create a file called 'NMsim_..._MetaData.rds' under the `dir.res` directory where ... is based on the model name. However, if multiple models (`file.mod`) are simulated, this will result in multiple rds files. Specifying a path ensures that one rds file containing information about all simulated models will be created. Notice if `file.res` is supplied, `dir.res` is not used.

`dir.sim.sub` If 'TRUE' (default) a dedicated subdirectory will be created for eac model run. This is normally the cleanest way to run simulations. However, when 'NMsim()' is used for estimation, it may be better to provide model results in the same folder as the input control stream (like PSN would do). Use 'dir.sim.sub=FALSE' to get this behavior.

`wait` Wait for simulations to finish? Default is to do so if simulations are run locally but not to if they are sent to the cluster. Waiting for them means that the results

will be read when simulations are done. If not waiting, path(s) to 'rds' files to read will be returned. Pass them through 'NMreadSim()'. Conveniently, NMreadSim() also takes the 'wait' argument too, allowing flexibility to run Nonmem in the background, and then read the results, still waiting for Nonmem to finish.

`text.sim` A character string to be pasted into $SIMULATION. This must not contain seed or SUBPROBLEM which is handled separately. Default is to include "ONLYSIM". You cannot avoid that using 'text.sim'. If needed, you can use 'onlysim=FALSE' which will be passed to 'NMsim_default()'.

`auto.dv` Add a column called 'DV' to input data sets if a column of that name is not found? Nonmem is generally dependent on a 'DV' column in input data but this is typically uninformative in simulation data sets and hence easily forgotten when generating simulation data sets. If `auto.dv=TRUE` and no 'DV' column is found, 'DV=NA' will be added. In this case ('auto.dv=TRUE' and no 'DV' column found) a 'MDV=1' column will also be added if none found.

`clean` The degree of cleaning (file removal) to do after Nonmem execution. If 'method.execute=="psn"', this is passed to PSN's 'execute'. If 'method.execute=="nmsim"' a similar behavior is applied, even though not as granular. NMsim's internal method only distinguishes between 0 (no cleaning), any integer 1-4 (default, quite a bit of cleaning) and 5 (remove temporary dir completely).

`sim.dir.from.scratch`
If TRUE (default) this will wipe the simulation directory before running new simulations. The directory that will be emptied is _not_ dir.sims where you may keep many or all your simulations. It is the subdirectory named based on the run name and `name.sim`. The reason it is advised to wipe this directory is that if you in a previous simulation created simulation runs that are now obsolete, you could end up reading those too when collecting the results. NMsim will delete previously generated simulation control streams with the same name, but this option goes further. An example where it is important is if you first ran 1000 replications, fixed something and now rand 500. If you choose FALSE here, you can end up with the results of 500 new and 500 old simulations.

`create.dirs` If the directories specified in dir.sims and dir.res do not exists, should it be created? Default is TRUE.

`quiet` If TRUE, messages from what is going on will be suppressed.

`nmquiet` Silent console messages from Nonmem? The default behaviour depends. It is FALSE if there is only one model to execute and 'progress=FALSE'.

`progress` Track progress? Default is 'TRUE' if 'quiet' is FALSE and more than one model is being simulated. The progress tracking is based on the number of models completed, not the status of the individual models.

`check.mod` Check the provided control streams for contents that may cause issues for simulation. Default is 'TRUE', and it is only recommended to disable this if you are fully aware of such a feature of your control stream, you know how it impacts simulation, and you want to get rid of warnings.

`format.data.complete`
For development purposes - users do not need this argument. Controls what format the complete input data set is saved in. Possible values are 'rds' (default),

	'fst' (experimental) and 'csv'. 'fst' may be faster and use less disk space but factor levels may be lost from input data to output data. 'csv' will also lead to loss of additional information such as factor levels.
`text.table`	Deprecated. Use 'table.vars' and 'table.options' instead.
`suffix.sim`	Deprecated. Use name.sim instead.
`seed`	Deprecated. See `seed.R` and `seed.nm`.
`file.ext`	Deprecated. Use 'inits=list(file.ext="path/to/file.ext")' instead. Optionally provide a parameter estimate file from Nonmem. This is normally not needed since 'NMsim' will by default use the ext file stored next to the input control stream (replacing the file name extension with '.ext'). If using method.update.inits="psn", this argument cannot be used.
`method.update.inits`	Deprecated, please migrate to 'inits' instead. The initial values of all parameters are by updated from the estimated model before running the simulation. NMsim can do this with a native function or use PSN to do it - or the step can be skipped to not update the values.
`modify.model`	Deprecated. Use modify instead.
`list.sections`	Deprecated. Use modify instead.
`...`	Additional arguments passed to `method.sim`.

Details

Loosely speaking, the argument `method.sim` defines _what_ NMsim will do, `method.execute` define _how_ it does it. `method.sim` takes a function that converts an estimation control stream into whatever should be run. Features like replacing '$INPUT', '$DATA', '$TABLE', and handling seeds are NMsim features that are done in addition to the `method.sim`. Also the `modeify.model` argument is handled in addition to the `method.sim`. The `subproblems` and `seed.nm` arguments are available to all methods creating a `$SIMULATION` section.

Notice, the following functions are internally available to 'NMsim' so you can run them by say `method.sim=NMsim_EBE` without quotes. To see the code of that method, type `NMsim_EBE`.

- `NMsim_default` The default behaviour. Replaces any $ESTIMATION and $COVARIANCE sections by a $SIMULATION section.
- `NMsim_asis` The simplest of all method. It does nothing (but again, `NMsim` handles '$INPUT', '$DATA', '$TABLE' and more. Use this for instance if you already created a simulation (or estimation actually) control stream and want NMsim to run it on different data sets.
- `NMsim_EBE` Simulates _known_ ETAs. By default, the ETA values are automatically taken from the estimation run. This is what is refered to as emperical Bayes estimates, hence the name of the method "NMsim_EBE". However, the user can also provide a different '.phi' file which may contain simulated ETA values (see the 'file.phi' argument). ID values in the simulation data set must match ID values in the phi file for this step to work. If refering to estimated subjects, the .phi file from the estimation run must be found next to the .lst file from the estimation with the same file name stem (say 'run1.lst' and 'run1.phi'). Again, ID values in the (simulation) input data must be ID values that were used in the estimation too. The method Runs an `$ESTIMATION MAXEVAL=0` but pulls in ETAs for the ID's found in data. No `$SIMULATION` step is run which unfortunately means no residual error will be simulated.

- NMsim_VarCov Like NMsim_default but '$THETA', '$OMEGA', and 'SIGMA' are drawn from distribution estimated in covariance step. This means that a successful covariance step must be available from the estimation. NB. A multivariate normal distribution is used for all parameters, including '$OMEGA' and '$SIGMA' which is not the correct way to do this. In case the simulation leads to negative diagonal elements in $OMEGA and $SIGMA, those values are truncated at zero. This method is only valid for simulation of '$THETA' variability. The method accepts a table of parameter values that can be produced with other tools than 'NMsim'. For simulation with parameter variability based on bootstrap results, use NMsim_default.

Value

A data.frame with simulation results (same number of rows as input data). If 'sge=TRUE' a character vector with paths to simulation control streams.

NMsimTestConf *Summarize and test NMsim configuration*

Description

Summarize and test NMsim configuration

Usage

```
NMsimTestConf(
  path.nonmem,
  dir.psn,
  method.execute,
  must.work = FALSE,
  system.type
)
```

Arguments

path.nonmem	See ?NMsim
dir.psn	See ?NMsim
method.execute	See ?NMsim
must.work	Throw an error if the configuration does not seem to match system.
system.type	See ?NMsim

Value

A list with configuration values

NMsim_asis *Simulation method that uses the provided control stream as is*

Description

The simplest of all method. It does nothing (but again, NMsim handles '$INPUT', '$DATA', '$TABLE' and more. Use this for instance if you already created a simulation (or estimation actually) control stream and want NMsim to run it on different data sets.

Usage

```
NMsim_asis(file.sim, file.mod, data.sim)
```

Arguments

file.sim	See ?NMsim.
file.mod	See ?NMsim.
data.sim	See ?NMsim.

Value

Path to simulation control stream

NMsim_default *Transform an estimated Nonmem model into a simulation control stream*

Description

The default behaviour of NMsim. Replaces any $ESTIMATION and $COVARIANCE sections by a $SIMULATION section.

Usage

```
NMsim_default(
  file.sim,
  file.mod,
  data.sim,
  nsims = 1,
  onlysim = TRUE,
  replace.sim = TRUE,
  return.text = FALSE
)
```

Arguments

file.sim	See ?NMsim.
file.mod	See ?NMsim.
data.sim	See ?NMsim.
nsims	Number of replications wanted. The default is 1. If greater, multiple control streams will be generated.
onlysim	Include 'ONLYSIM' in '$SIMULATION'? Default is 'TRUE'. Only applied when 'replace.sim='TRUE'.
replace.sim	If there is a $SIMULATION section in the contents of file.sim, should it be replaced? Default is yes. See the list.section argument to NMsim for how to provide custom contents to sections with NMsim instead of editing the control streams beforehand.
return.text	If TRUE, just the text will be returned, and resulting control stream is not written to file.

Value

Character vector of simulation control stream paths

NMsim_EBE *Use emperical Bayes estimates to simulate re-using ETAs*

Description

Simulation reusing ETA values from estimation run or otherwise specified ETA values. For observed subjects, this is refered to as emperical Bayes estimates (EBE). The .phi file from the estimation run must be found next to the .lst file from the estimation.This means that ID values in the (simulation) input data must be ID values that were used in the estimation too. Runs an $ESTIMATION MAXEVAL=0 but pulls in ETAs for the ID's found in data. No $SIMULATION step is run which may affect how for instance residual variability is simulated, if at all. You can also specify a different .phi file which can be a simulation result.

Usage

```
NMsim_EBE(file.sim, file.mod, data.sim, file.phi, return.text = FALSE)
```

Arguments

file.sim	The path to the control stream to be edited. This function overwrites the contents of the file pointed to by file.sim.
file.mod	Path to the path to the original input control stream provided as 'file.mod' to 'NMsim()'.
data.sim	See ?NMsim.

file.phi	A phi file to take the known subjects from. The default is to replace the filename extension on file.mod with .phi. A different .phi file would be used if you want to reuse subjects simulated in a previous simulation.
return.text	If TRUE, just the text will be returned, and resulting control stream is not written to file.

Value

Path to simulation control stream

See Also

simPopEtas

NMsim_NWPRI *Simulate with parameter variability using the NONMEM NWPRI subroutine*

Description

Modify control stream for simulation with uncertainty using inverse-Wishart distribution for OMEGA and SIGMA parameters

This function does not run any simulations. To simulate, using this method, see 'NMsim()'. See examples.

Usage

```
NMsim_NWPRI(file.sim, file.mod, data.sim, PLEV = 0.999, add.diag, ...)
```

Arguments

file.sim	The path to the control stream to be edited. This function overwrites the contents of the file pointed to by file.sim.
file.mod	Path to the path to the original input control stream provided as 'file.mod' to 'NMsim()'.
data.sim	Included for compatibility with 'NMsim()'. Not used.
PLEV	Used in `$PRIOR NWPRI PLEV=0.999`. This is a NONMEM argument to the NWPRI subroutine. When PLEV < 1, a value of THETA will actually be obtained using a truncated multivariate normal distribution, i.e. from an ellipsoidal region R1 over which only a fraction of mass of the normal occurs. This fraction is given by PLEV.
add.diag	A umeric value to add to the diagonal of the covariance matrix. This can be used in case of negative eigenvaluen in variance-covariance matrix.
...	Additional arguments passed to 'NMsim_default()'.

Details

Simulate with parameter uncertainty. THETA parameters are sampled from a multivariate normal distribution while OMEGA and SIGMA are simulated from the inverse-Wishart distribution. Correlations of OMEGA and SIGMA parameters will only be applied within modeled "blocks".

Value

Path to simulation control stream

Author(s)

Brian Reilly, Philip Delff

References

inverse-Wishart degrees of freedom calculation for OMEGA and SIGMA: NONMEM tutorial part II, supplement 1, part C.

See Also

NMsim_VarCov

Examples

```
## Not run:
simres <- NMsim(file.path,method.sim=NMsim_WPRI,typical=TRUE,subproblems=500)

## End(Not run)
```

NMsim_typical *Typical subject simiulation method*

Description

Like `NMsim_default` but with all ETAs=0, giving a "typical subject" simulation. Do not confuse this with a "reference subject" simulation which has to do with covariate values. Technically all ETAs=0 is obtained by replacing `$OMEGA` by a zero matrix.

Usage

```
NMsim_typical(file.sim, file.mod, data.sim, return.text = FALSE)
```

Arguments

`file.sim`	See `?NMsim`.
`file.mod`	See `?NMsim`.
`data.sim`	See `?NMsim`.
`return.text`	If TRUE, just the text will be returned, and resulting control stream is not written to file.

Value

Path to simulation control stream

`NMsim_VarCov` *Simulate with parameter values sampled from a covariance step*

Description

Like `NMsim_default` but '$THETA', '$OMEGA', and 'SIGMA' are drawn from distribution estimated in covariance step. A successful covariance step must be available from the estimation. In case the simulation leads to negative diagonal elements in $OMEGA and $SIGMA, those values are truncated at zero. For simulation with parameter variability based on bootstrap results, use `NMsim_default`.

This function does not run any simulations. To simulate, using this method, see 'NMsim()'.

Usage

```
NMsim_VarCov(
  file.sim,
  file.mod,
  data.sim,
  nsims,
  method.sample = "mvrnorm",
  ext,
  write.ext = NULL,
  ...
)
```

Arguments

`file.sim`	The path to the control stream to be edited. This function overwrites the contents of the file pointed to by file.sim.
`file.mod`	Path to the path to the original input control stream provided as 'file.mod' to 'NMsim()'.
`data.sim`	Included for compatibility with 'NMsim()'. Not used.
`nsims`	Number of replications wanted. The default is 1. If greater, multiple control streams will be generated.
`method.sample`	When 'ext' is not used, parameters are sampled, using 'samplePars()'. 'method' must be either 'mvrnorm' or 'simpar'. Only used when 'ext' is not provided.
`ext`	Parameter values in long format as created by 'readParsWide' and 'NMdata::NMreadExt'.
`write.ext`	If supplied, a path to an rds file where the parameter values used for simulation will be saved.
`...`	Additional arguments passed to 'NMsim_default()'.

Value

Character vector of simulation control stream paths

`NMwriteInits` *Writes a parameter values to a control stream*

Description

Edit parameter values, fix/unfix them, or edit lower/upper bounds.

Usage

```
NMwriteInits(
  file.mod,
  lines,
  update = TRUE,
  file.ext = NULL,
  ext,
  inits.tab,
  values,
  newfile,
  ...
)
```

Arguments

`file.mod`	Path to control stream.
`lines`	Control stream as character vector. Use either ‘file.mod‘ or ‘lines‘, not both.
`update`	If ‘TRUE‘ (default), the parameter values are updated based on the ‘.ext‘ file. The path to the ‘.ext‘ file can be specified with ‘file.ext‘ but that is normally not necessary.
`file.ext`	Optionally provide the path to an ‘.ext‘ file. If not provided, the default is to replace the file name extention on ‘file.mod‘ with ‘.ext‘. This is only used if ‘update=TRUE‘.
`ext`	An long-format parameter table as returned by ‘NMreadExt()‘. Can contain multiple models if ‘file.mod‘ does not.
`inits.tab`	A wide-format parameter table, well suited for customizing initial values, limits, and for fixing parameters. For multiple custom parameter specifications, this may be the most suitable argument.
`values`	A list of lists. Each list specifies a parameter with named elements. Must be named by the parameter name. ‘lower‘, ‘upper‘ and ‘fix‘ can be supplied to modify the parameter. See examples. Notice, you can use ‘...‘ instead. ‘values‘ may be easier for programming but other than that, most users will find ‘...‘ more intuitive.
`newfile`	If provided, the results are written to this file as a new input control stream.
`...`	Parameter specifications. See examples,

Details

Limitations:

- 'NMwriteInits()' can only update specifications of existing parameters. It cannot insert new parameters.
- lower, init, upper, and FIX must be on same line in control stream.
- If using something like CL=(.1,4,15) in control stream, two of those cannot be on the same line.

In Nonmem an entire block is either fixed or not. 'NMwriteInits()' fixes/unfixes the entire block based on the top-left element in the block. This means, if OMEGA(2,2)-OMEGA(3,3) is a block, the 'FIX' status of OMEGA(2,2) determines whether the block is fixed. 'FIX' of all other elements in the block has no effect.

Value

a control stream as lines in a character vector.

Examples

```
## Not run:
file.mod <- system.file("examples/nonmem/xgxr021.mod",package="NMdata")
## specify parameters using ...
NMwriteInits(file.mod,
  "theta(2)"=list(init=1.4),
  "THETA(3)"=list(FIX=1),
  "omega(2,2)"=list(init=0.1)
)
## or put them in a list in the values argument
NMwriteInits(file.mod,
values=list( "theta(2)"=list(init=1.4),
             "THETA(3)"=list(FIX=1),
             "omega(2,2)"=list(init=0.1))
)


## End(Not run)
```

NMwriteSizes *Create or update $SIZES in a control stream*

Description

Update $SIZES parameters in a control stream. The control stream can be in a file or provided as a character vector (file lines).

Usage

```
NMwriteSizes(
  file.mod = NULL,
  newfile,
  lines = NULL,
  wipe = FALSE,
  write = !is.null(newfile),
  ...
)
```

Arguments

file.mod	A path to a control stream. See also alternative 'lines' argument. Notice, if 'write' is 'TRUE' (default) and 'newfile' is not provided, 'file.mod' will be overwritten.
newfile	An optional path to write the resulting control stream to. If nothing is provided, the default is to overwrite 'file.mod'.
lines	Control stream lines as a character vector. If you already read the control stream - say using 'NMdata::NMreadSection()', use this to modify the text lines.
wipe	The default behavior ('wipe=FALSE') is to add the '$SIZES' values to any existing values found. If SIZES parameter names are overlapping with existing, the values will be updated. If 'wipe=TRUE', any existing '$SIZES' section is disregarded.
write	Write results to 'newfile'?
...	The $SIZES parameters. Provided anything, like 'PD=40' See examples.

Value

Character lines with updated control stream

Examples

```
## No existing SIZES in control stream
## Not run:
file.mod <- system.file("examples/nonmem/xgxr132.mod",package="NMdata")
newmod <- NMwriteSizes(file.mod,LTV=50,write=FALSE)
head(newmod)

## End(Not run)
## provide control stream as text lines
## Not run:
file.mod <- system.file("examples/nonmem/xgxr032.mod",package="NMdata")
lines <- readLines(file.mod)
newmod <- NMwriteSizes(lines=lines,LTV=50,write=FALSE)
head(newmod)

## End(Not run)
## By default (wipe=FALSE) variabels are added to SIZES
```

```
## Not run:
lines.mod <- NMwriteSizes(file.mod,LTV=50,write=FALSE)
newmod <- NMwriteSizes(lines=lines.mod,PD=51,write=FALSE)
head(newmod)

## End(Not run)
```

`overwrite` *Create function that modifies text elements in a vector Namely used to feed functions to modify control streams using 'NMsim()' arguments such as 'modify'. Those functions are often onveniently passed a function. 'add' and 'overwrite' are simple shortcuts to creating such functions. Make sure to see examples.*

Description

Create function that modifies text elements in a vector Namely used to feed functions to modify control streams using 'NMsim()' arguments such as 'modify'. Those functions are often onveniently passed a function. 'add' and 'overwrite' are simple shortcuts to creating such functions. Make sure to see examples.

Usage

```
overwrite(..., fixed = TRUE)
```

Arguments

`...`	Passed to 'gsub()'
`fixed`	This is passed to gsub(), but 'overwrite()''s default behavior is the opposite of the one of 'gsub()'. Default is 'FALSE' which means that strings that are exactly matched will be replaced. This is useful because strings like 'THETA(1)' contains special characters. Use 'fixed=FALSE' to use regular expressions. Also, see other arguments accepted by 'gsub()' for advanced features.

Value

A function that runs 'gsub' to character vectors

Examples

```
myfun <- overwrite("b","d")
myfun(c("a","b","c","abc"))
## regular expressions
myfun2 <- overwrite("b.*","d",fixed=FALSE)
myfun2(c("a","b","c","abc"))
```

print.summary_NMsimRes *print method for NMsimRes summaries*

Description

print method for NMsimRes summaries

Usage

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

Arguments

x	The summary object to be printed. See ?summary.NMsimRes
...	Arguments passed to other print methods.

Value

NULL (invisibly)

prioritizePaths *first path that works*

Description

When using scripts on different systems, the Nonmem path may change from run to run. With this function you can specify a few paths, and it will return the one that works on the system in use.

Usage

```
prioritizePaths(paths, must.work = FALSE)
```

Arguments

paths	vector of file paths. Typically to Nonmem executables.
must.work	If TRUE, an error is thrown if no paths are valid.

Examples

```
library(NMdata)
NMdataConf(path.nonmem = prioritizePaths(c(
  "/opt/NONMEM/nm75/run/nmfe75",
  "C:/nm75g64/run/nmfe75.bat")
))
```

`readParsWide` *Parameter data from csv*

Description

Reads output table from simpar and returns a long format data.table. This is the same format as returned by NMreadExt() which can be used by NMsim.

Usage

```
readParsWide(
  data,
  col.model,
  col.model.sim,
  strings.par.type = c(THETA = "^T.*", OMEGA = "^O.*", SIGMA = "^S."),
  as.fun
)
```

Arguments

`data`	A data.frame or a path to a delimited file to be read using 'data.table::fread'.
`col.model`	Column containing name of the original model. By default a column called "model" will contain "Model1".
`col.model.sim`	Name of the model counter, default is "model.sim". If the provided name is not found in data, it will be created as a row counter. Why needed? Each row in data represents a set of parameters, i.e. a model. In the long format result, each model will have multiple rows. Hence, a model identifier is needed to distinguish between models in results.
`strings.par.type`	Defines how column names get associated with THETA, OMEGA, and SIGMA. Default is to look for "T", "O", or "S" as starting letter. If customizing, make sure each no column name will be matched by more than one criterion.
`as.fun`	The default is to return data as a data.frame. Pass a function (say `tibble::as_tibble`) in as.fun to convert to something else. If data.tables are wanted, use as.fun="data.table". The default can be configured using NMdataConf.

Details

The wide data format read by 'readParsWide' is not a Nonmem format. It is used to bridge output from other tools such as simpar, and potentially PSN.

This function reads a data that is "wide" in parameters - it has a column for each parameter, and one row per parameter set or "model". It returns a data set that is "long" in model and parameters. The long format contains

- id.model.par The unique model-parameter identifier. The row-identifier.
- model Model identifier.

- par.type ("THETA", "OMEGA", "SIGMA")
- i and j indexes for the parameters (j is NA for par.type=="THETA").
- value The parameter value
- parameter Nonmem-style parameter names. THETA1, OMEGA(1,1) etc. Notice the inconsistent naming of THETA vs others.
- name.wide The column name in the wide data where this value was taken

The columns or "measure variables" from which to read values are specified as three regular expressions, called THETA, OMEGA, and SIGMA. The default three regular expressions will associate a column name starting with "T" with THETAs, while "O" or "S" followed by anything means "OMEGA" or "SIGMA".

readParsWide extracts i and j indexes from sequences of digits in the column names. TH.1 would be TETA1, SG1.1 is SIGMA(1,1).

Value

a long-format data.frame of model parameters

Examples

```
## Not run:
tab.ext <- readParsCsv("simpartab.csv")
## or
tab.simpar <- fread("simpartab.csv")
tab.ext <- readParsCsv(tab.simpar)
NMsim(...,method.sim=NMsim_VarCov,tab.ext=tab.ext)

## End(Not run)
```

---

sampleCovs *Sample subject-level covariates from an existing data set*

Description

Repeats a data set with just one subject by sampling covariates from subjects in an existing data set. This can conveniently be used to generate new subjects with covariate resampling from an studied population.

Usage

```
sampleCovs(
  data,
  Nsubjs,
  col.id = "ID",
  col.id.covs = "ID",
  data.covs,
  covs,
```

```
  seed.R,
  as.fun
)
```

Arguments

data	A simulation data set with only one subject
Nsubjs	The number of subjects to be sampled. This can be greater than the number of subjects in data.covs.
col.id	Name of the subject ID column in ‘data‘ (default is "ID").
col.id.covs	Name of the subject ID column in ‘data.covs‘ (default is "ID").
data.covs	The data set containing the subjects to sample covariates from.
covs	The name of the covariates (columns) to sample from ‘data.covs‘.
seed.R	If provided, passed to ‘set.seed()‘.
as.fun	The default is to return data as a data.frame. Pass a function (say ‘tibble::as_tibble‘) in as.fun to convert to something else. If data.tables are wanted, use as.fun="data.table". The default can be configured using NMdataConf.

Value

A data.frame. Includes sampled covariates. The subject ID’s the covariates are sampled from will be included in a column called ‘IDCOVS‘.

Examples

```
library(NMdata)
data.covs <- NMscanData(system.file("examples/nonmem/xgxr134.mod",package="NMsim"))
dos.1 <- NMcreateDoses(TIME=0,AMT=100)
data.sim.1 <- NMaddSamples(dos.1,TIME=c(1,4),CMT=2)
sampleCovs(data=data.sim.1,Nsubjs=3,col.id.covs="ID",data.covs=data.covs,covs=c("WEIGHTB","eff0"))
```

---

samplePars *Sample model parameters using ‘mvrnorm‘ or the ‘simpar‘ package*

Description

Sample model parameters using ‘mvrnorm‘ or the ‘simpar‘ package

Usage

```
samplePars(file.mod, nsims, method, seed.R, format = "ext", as.fun)
```

Arguments

file.mod	Path to model control stream. Will be used for both 'NMreadExt()' and 'NMreadCov()', and extension will automatically be replaced by '.ext' and '.cov'.
nsims	Number of sets of parameter values to generate. Passed to 'simpar'.
method	The sampling method. Options are "mvrnorm" and "simpar". Each have pros and cons. Notice that both methods are fully automated as long as ".ext" and ".cov" files are available from model estimation.
seed.R	seed value passed to set.seed().
format	The returned data set format "ext" (default) or "wide". "ext" is a long-format, similar to what 'NMdata::NMreadExt()' returns.
as.fun	The default is to return data as a data.frame. Pass a function (say 'tibble::as_tibble') in as.fun to convert to something else. If data.tables are wanted, use as.fun="data.table". The default can be configured using NMdataConf.

Details

samplePars() uses internal methods to sample using mvrnorm or simpar. Also be aware of NMsim_NWPRI which is based on the Nonmem-internal NWPRI subroutine. NMsim_NWPRI is much faster to execute. Simulation with paramater uncertainty on variance components ('OMEGA' and 'SIGMA') is only reliable starting from Nonmem 7.6.0.

mvrorm: The multivariate normal distribution does not ensure non-negative variances. Negative variances are not allowed and can not be simulated. To avoid this, 'method=mvrnorm' truncates negative variance diagonal elements at zero.

simpar: simpar must be installed.

Please refer to publications and vignettes for more information on sampling methods.

Value

A table with sampled model parameters

Author(s)

Sanaya Shroff, Philip Delff

---

sampleParsSimpar *Sample model parameters using the 'simpar' package*

Description

Sample model parameters using the 'simpar' package

Usage

```
sampleParsSimpar(file.mod, nsim, format = "ext", seed.R, as.fun)
```

Arguments

file.mod	Path to model control stream. Will be used for both 'NMreadExt()' and 'NMreadCov()', and extension will automatically be replaced by '.ext' and '.cov'.
nsim	Number of sets of parameter values to generate. Passed to 'simpar'.
format	"ext" (default) or "wide".
seed.R	seed value passed to set.seed().
as.fun	The default is to return data as a data.frame. Pass a function (say 'tibble::as_tibble') in as.fun to convert to something else. If data.tables are wanted, use as.fun="data.table". The default can be configured using NMdataConf.

Value

A table with sampled model parameters

Author(s)

Sanaya Shroff, Philip Delff

---

simPopEtas *Generate a population based on a Nonmem model*

Description

Generate a population based on a Nonmem model

Usage

```
simPopEtas(
  file,
  N,
  seed.R,
  pars,
  file.phi,
  overwrite = FALSE,
  as.fun,
  file.mod,
  seed,
  ...
)
```

Arguments

file	Passed to 'NMdata::NMreadExt()'. Path to ext file. By default, 'NMreadExt()' uses a'auto.ext=TRUE' which means that the file name extension is replaced by '.ext'. If your ext file name extension is not '.ext', add 'auto.ext=FALSE' (see ...).
N	Number of subjects to generate
seed.R	Optional seed. Will be passed to 'set.seed'. Same thing as running 'set.seed' just before calling 'simPopEtas()'.
pars	A long-format parameter table containing par.type and i columns. If this is supplied, the parameter values will not be read from an ext file, and file has no effect. If an ext file is available, it is most likely better to use the file argument.
file.phi	An optional phi file to write the generated subjects to.
overwrite	If 'file.phi' exists already, overwrite it? Default is 'FALSE'.
as.fun	The default is to return data as a data.frame. Pass a function (say 'tibble::as_tibble') in as.fun to convert to something else. If data.tables are wanted, use as.fun="data.table". The default can be configured using NMdataConf.
file.mod	Deprecated. Use file instead.
seed	Deprecated. Use seed.R instead.
...	Additional arguments passed to NMdata::NMreadExt().

Value

A data.frame

---

summarizeCovs *Summarize simulated exposures relative to reference subject (see 'forestSummarize()')*

Description

Summarize simulated exposures relative to reference subject (see 'forestSummarize()')

Usage

```
summarizeCovs(...)
```

Arguments

...	Passed to 'forestSummarize()'

Value

A data.frame

summary.NMsimRes *summary method for NMsim results (NMsimRes objects)*

Description

summary method for NMsim results (NMsimRes objects)

Usage

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

Arguments

object	An NMsimRes object (from NMsim).
...	Not used

Value

A list with summary information on the NMsimRes object.

unNMsimModTab *Remove NMsimModTab class and discard NMsimModTab meta data*

Description

Remove NMsimModTab class and discard NMsimModTab meta data

Check if an object is 'NMsimModTab'

Basic arithmetic on NMsimModTab objects

Usage

```
unNMsimModTab(x)

is.NMsimModTab(x)

## S3 method for class 'NMsimModTab'
merge(x, ...)

## S3 method for class 'NMsimModTab'
t(x, ...)

## S3 method for class 'NMsimModTab'
dimnames(x, ...)
```

```
## S3 method for class 'NMsimModTab'
rbind(x, ...)

## S3 method for class 'NMsimModTab'
cbind(x, ...)
```

Arguments

x	an NMsimModTab object
...	arguments passed to other methods.

Details

When 'dimnames', 'merge', 'cbind', 'rbind', or 't' is called on an 'NMsimModTab' object, the 'NMsimModTab' class is dropped, and then the operation is performed. So if and 'NMsimModTab' object inherits from 'data.frame' and no other classes (which is default), these operations will be performed using the 'data.frame' methods. But for example, if you use 'as.fun' to get a 'data.table' or 'tbl', their respective methods are used instead.

Value

x stripped from the 'NMsimModTab' class

logical if x is an 'NMsimModTab' object

An object that is not of class 'NMsimModTab'.

---

unNMsimRes *Remove NMsimRes class and discard NMsimRes meta data*

Description

Remove NMsimRes class and discard NMsimRes meta data

Check if an object is 'NMsimRes'

Basic arithmetic on NMsimRes objects

Usage

```
unNMsimRes(x)

is.NMsimRes(x)

## S3 method for class 'NMsimRes'
merge(x, ...)

## S3 method for class 'NMsimRes'
t(x, ...)
```

```
## S3 method for class 'NMsimRes'
dimnames(x, ...)

## S3 method for class 'NMsimRes'
rbind(x, ...)

## S3 method for class 'NMsimRes'
cbind(x, ...)
```

Arguments

x	an NMsimRes object
...	arguments passed to other methods.

Details

When 'dimnames', 'merge', 'cbind', 'rbind', or 't' is called on an 'NMsimRes' object, the 'NMsimRes' class is dropped, and then the operation is performed. So if and 'NMsimRes' object inherits from 'data.frame' and no other classes (which is default), these operations will be performed using the 'data.frame' methods. But for example, if you use 'as.fun' to get a 'data.table' or 'tbl', their respective methods are used instead.

Value

x stripped from the 'NMsimRes' class

logical if x is an 'NMsimRes' object

An object that is not of class 'NMsimRes'.

Index