

# Package ‘NMsim’

July 3, 2024

**Type** Package

**Title** Seamless 'Nonmem' Simulation Platform

**Version** 0.1.1

**Maintainer** Philip Delff <philip@delff.dk>

**Description** A complete and seamless 'Nonmem' simulation interface from 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 time spent and risks of re-implementation of models in other tools.

**License** MIT + file LICENSE

**RoxygenNote** 7.3.1

**Depends** R (>= 3.5.0)

**Imports** data.table, NMdata (>= 0.1.5), R.utils, MASS, fst, xfun

**Suggests** testthat, knitr, rmarkdown, ggplot2, patchwork, tracee, tidyvpc

**Encoding** UTF-8

**BugReports** <https://github.com/philipdelff/NMsim/issues>

**Language** en-US

**URL** <https://philipdelff.github.io/NMsim/>

**NeedsCompilation** no

**Author** Philip Delff [aut, cre]

**Repository** CRAN

**Date/Publication** 2024-07-03 12:20:02 UTC

## Contents

addEVID2 . . . . .	2
addResVar . . . . .	3
genPhiFile . . . . .	5
inputArchiveDefault . . . . .	6

NMcreateDoses . . . . .	6
NMexec . . . . .	8
NMreadSim . . . . .	10
NMsim . . . . .	11
NMsim_asis . . . . .	18
NMsim_default . . . . .	18
NMsim_EBE . . . . .	19
NMsim_typical . . . . .	20
NMsim_VarCov . . . . .	20
readParsWide . . . . .	21
simPopEtas . . . . .	23
unNMsimModTab . . . . .	23
unNMsimRes . . . . .	24

## Index 26

---

addEVID2	<i>Add simulation records to dosing records</i>
----------	---

---

### Description

Performs the simple job of adding simulation events to all subjects in a data set. Copies over columns that are not varying at subject level (i.e. non-varying covariates).

### Usage

```
addEVID2(doses, time.sim, CMT, EVID = 2, as.fun)
```

### Arguments

doses	dosing records Nonmem style (EVID==1 records from a data set)
time.sim	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 subjects found in 'doses'. The latter feature is experimental.
CMT	The compartment in which to insert the EVID=2 records. 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 preferred even for simulation.
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 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.sim=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.sim=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.sim=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.sim=dt.sample,CMT=2)
```

---

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

<code>data</code>	A data set containing individual predictions. Often a result of <code>NMsim</code> .
<code>path.ext</code>	Path to the ext file to take the parameter estimates from.
<code>prop</code>	Parameter number of parameter holding variance of the proportional error component. If <code>ERR(1)</code> is used for proportional error, use <code>prop=1</code> . Can also refer to a theta number.
<code>add</code>	Parameter number of parameter holding variance of the additive error component. If <code>ERR(1)</code> is used for additive error, use <code>add=1</code> . Can also refer to a theta number.
<code>log</code>	Should the error be added on log scale? This is used to obtain an exponential error distribution.
<code>par.type</code>	Use "sigma" if variances are estimated with the SIGMA matrix. Use "theta" if THETA parameters are used. See 'scale.par' too.
<code>trunc0</code>	If <code>log=FALSE</code> , truncate simulated values at 0? If <code>trunc0</code> , returned predictions can be negative.
<code>scale.par</code>	Denotes if parameter represents a variance or a standard deviation. Allowed values and default value depends on 'par.type'. <ul style="list-style-type: none"> <li>if <code>par.type="sigma"</code> only "var" is allowed.</li> <li>if <code>par.type="theta"</code> allowed values are "sd" and "var". Default is "sd".</li> </ul>
<code>subset</code>	A character string with an expression denoting a subset in which to add the residual error. Example: <code>subset="DVID=='A'"</code>
<code>seed</code>	A number to pass to <code>set.seed()</code> before simulating. Default is to generate a seed and report it in the console. Use <code>seed=FALSE</code> to avoid setting the seed (if you prefer doing it otherwise).
<code>col.ipred</code>	The name of the column containing individual predictions.
<code>col.ipredvar</code>	The name of the column to be created by <code>addResVar</code> to contain the simulated observations (individual predictions plus residual error).
<code>as.fun</code>	The default is to return data as a <code>data.frame</code> . Pass a function (say 'tibble::as_tibble') in <code>as.fun</code> to convert to something else. If <code>data.tables</code> are wanted, use <code>as.fun="data.table"</code> . The default can be configured using <code>NMdataConf</code> .

### 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)
```

---

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)
```

## Arguments

data	A dataset that contains "ID" and all ETAs. This can be obtained by 'NM-data::NMscanData'.
file	Path to the .phi file to be written.

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)

---

NMcreateDoses            *Easily generate dosing records*

---

**Description**

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

**Usage**

```
NMcreateDoses(  
  TIME,  
  AMT = NULL,  
  RATE = NULL,  
  SS = NULL,  
  CMT = 1,  
  EVID = 1,  
  add1 = NULL,  
  as.fun  
)
```

**Arguments**

TIME	The time of the dosing events
AMT	vector or data.frame with amounts amount
RATE	Optional infusion rate
SS	Optional steady-state flag
CMT	Compartment number. Default is to dose into CMT=1.
EVID	The event ID to use for doses. Default is to use EVID=1, but EVID might also be wanted.
add1	Optimal. A list of ADDL and II that will be applied to last dose
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**

Experimental. Please check output before use. AMT, RATE, SS, II, CMT are vectors 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. 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))
## Differentiate 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.
NMcreateDoses(TIME=c(0,12),AMT=c(2,1),add1=list(ADDL=c(NA,9*2),II=c(NA,12)))
dt.amt <- data.table(DOSE=c(100,400))
dt.amt[,AMT:=DOSE*1000]
dt.amt
doses.sd <- NMcreateDoses(TIME=0,AMT=dt.amt)
doses.sd$dose <- paste(doses.sd$DOSE,"mg")
doses.sd$regimen <- "SD"
doses.sd

### multiple dose regimens with loading are easily created with NMcreateDoses too
## 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

```

---

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 = 64,
  dir.data = NULL,
  wait = FALSE,
  args.psn.execute,
  update.only = FALSE,
  nmquiet = FALSE,
  method.execute = "psn",
  dir.psn,
  path.nonmem,
  system.type,
  files.needed,
  quiet = FALSE
)

```

## 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.



<code>sge</code>	Use the sge queing system. Default is TRUE. Disable for quick models not to wait for the queue to run the job.
<code>input.archive</code>	A function of the model file path to generate the path in which to archive the input data as RDS. Set to NULL not to archive the data.
<code>nc</code>	Number of cores to use if sending to the cluster. This will only be used if <code>method.execute="psn"</code> , and <code>sge=TRUE</code> . Default is 64.
<code>dir.data</code>	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.
<code>wait</code>	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.
<code>args.psn.execute</code>	A character string with arguments passed to execute. Default is <code>"-model_dir_name -nm_output=xml,ext,cov,cor,coi,phi,shk"</code> .
<code>update.only</code>	Only run model(s) if control stream or data updated since last run?
<code>nmquiet</code>	Suppress terminal output from 'Nonmem'. This is likely to only work on linux/unix systems.
<code>method.execute</code>	How to run Nonmem. Must be one of 'psn', 'nmsim', or 'direct'. <ul style="list-style-type: none"> <li>• <code>psn</code> PSN's execute is used. This supports parallel Nonmem runs. Use the <code>nc</code> 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 <code>dir.psn</code> argument too.</li> <li>• <code>nmsim</code> 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 <code>sge=TRUE</code>, the job will be submitted to a cluster, but parallel execution of the job itself is not supported. See <code>path.nonmem</code> argument too.</li> <li>• <code>direct</code> 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.</li> </ul> See 'sge' as well.
<code>dir.psn</code>	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).
<code>path.nonmem</code>	The path to the nonmem executable. Only used if <code>method.execute="direct"</code> or <code>method.execute="nmsim"</code> (which is not default). If this argument is not supplied, NMexec will try to run <code>nmfe75</code> , i.e. this has to be available in the path of the underlying shell. The default value can be modified using <code>NMdataConf</code> , like <code>NMdataConf(path.nonmem="/path/to/nonmem")</code>
<code>system.type</code>	A character string, either <code>"windows"</code> or <code>"linux"</code> - case insensitive. Windows is only experimentally supported. Default is to use <code>Sys.info()[["sysname"]]</code> .
<code>files.needed</code>	In case <code>method.execute="nmsim"</code> , 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.
<code>quiet</code>	Suppress messages on what NMexec is doing? Default is FALSE.

**Details**

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

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,
  as.fun
)
```

**Arguments**

<code>x</code>	Path to the simulation-specific rds file generated by NMsim, typically called <code>'NMsim_MetaData.rds'</code> . Can also be a table of simulation runs as stored in <code>'rds'</code> files by <code>'NMsim'</code> . The latter should almost never be used.
<code>check.time</code>	If found, check whether <code>'fst'</code> file modification time is newer than <code>'rds'</code> file. The <code>'fst'</code> is generated based on information in <code>'rds'</code> , but notice that some systems don't preserve the file modification times. Because of that, <code>'check.time'</code> is <code>'FALSE'</code> by default.
<code>dir.sims</code>	By default, <code>'NMreadSim'</code> will use information about the relative path from the results table file ( <code>'_MetaData.rds'</code> ) to the Nonmem simulation results. If these paths have changed, or for other reasons this doesn't work, you can use the <code>'dir.sims'</code> argument to specify where to find the Nonmem simulation results. If an <code>'fst'</code> file was already generated and is found next to the <code>'_MetaData.rds'</code> , the path to the Nonmem simulation results is not used.
<code>wait</code>	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. <code>'NMreadSim'</code> will be waiting for an <code>'lst'</code> file. If Nonmem fails, it will normally generate an <code>'lst'</code> file. But if <code>'NMTRAN'</code> fails (checks of control stream prior to running Nonmem), the <code>'lst'</code> file is not generated. Default is not to wait.
<code>quiet</code>	Turn off some messages about what is going on? Default is to report the messages.
<code>progress</code>	Track progress? Default is <code>'TRUE'</code> if <code>'quiet'</code> is <code>FALSE</code> 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.
<code>as.fun</code>	The default is to return data as a data.frame. Pass a function (say <code>'tibble::as_tibble'</code> ) in <code>as.fun</code> to convert to something else. If data.tables are wanted, use <code>as.fun="data.table"</code> . The default can be configured using <code>NMdataConf</code> .

**Value**

A data set of class defined by `as.fun`

---

NMsim

*Simulate from an estimated Nonmem model*

---

**Description**

Supply a data set and an estimation input control stream, and NMsim can create necessary 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,  
  dir.sims,  
  name.sim,  
  order.columns = TRUE,  
  file.ext = NULL,  
  script = NULL,  
  subproblems = NULL,  
  reuse.results = FALSE,  
  seed.R,  
  seed.nm,  
  args.psn.execute,  
  table.vars,  
  table.options,  
  text.sim = "",  
  method.sim = NMsim_default,  
  typical = FALSE,  
  execute = TRUE,  
  sge = FALSE,  
  nc = 1,  
  transform = NULL,  
  method.execute,  
  method.update.inits,  
  create.dirs = TRUE,  
  dir.psn,  
  modify.model,  
  sim.dir.from.scratch = TRUE,  
  col.row,  
  args.NMscanData,  
  path.nonmem = NULL,  
  nmquiet = TRUE,  
  progress,  
  as.fun,  
  suffix.sim,  
  text.table,  
  system.type = NULL,  
  dir.res,  
  file.res,  
  wait,  
  auto.dv = TRUE,  
  quiet = FALSE,  
  check.mod = TRUE,  
  seed,  
  list.sections,  
  format.data.complete = "rds",  
  ...  
)
```

)

**Arguments**

<code>file.mod</code>	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 <code>.lst</code> instead of <code>.mod</code> . The <code>.ext</code> file must also be present. If simulating known subjects, the <code>.phi</code> is necessary too.
<code>data</code>	The simulation data as a <code>data.frame</code> or a list of <code>data.frames</code> . If a list, the model(s) will be run on each of the data sets in the list.
<code>dir.sims</code>	The directory in which NMsim will store all generated files. Default is to create a folder called 'NMsim' next to 'file.mod'.
<code>name.sim</code>	Give all filenames related to the simulation a suffix. A short string describing the sim is recommended like "ph3_regimens".
<code>order.columns</code>	reorder columns by calling <code>NMdata::NMorderColumns</code> before saving dataset and running simulations? Default is TRUE.
<code>file.ext</code>	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 ' <code>.ext</code> '). If using <code>method.update.inits="psn"</code> , this argument cannot be used. If you want provide parameters to be used for the simulation, look at the ' <code>ext</code> ' argument to ' <code>NMsim_VarCov</code> '.
<code>script</code>	The path to the script where this is run. For stamping of dataset so results can be traced back to code.
<code>subproblems</code>	Number of subproblems to use as SUBPROBLEMS in \$SIMULATION block in Nonmem. The default is <code>subproblem=0</code> which means not to use SUBPROBLEMS.
<code>reuse.results</code>	If simulation results found on file, should they be used? If TRUE and reading the results fail, the simulations will still be rerun.
<code>seed.R</code>	A value passed to <code>set.seed()</code> . It may be better use <code>seed.R</code> rather than calling <code>set.seed()</code> manually because the seed can then be captured and stored by <code>NMsim()</code> for reproducibility. See <code>seed.nm</code> for finer control of the seeds that are used in the Nonmem control streams.
<code>seed.nm</code>	Control Nonmem seeds. If a numeric, a vector or a ' <code>data.frame</code> ', 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 <code>data.frame</code> will dictate the number of seeds in each Nonmem control stream. Use a list with elements ' <code>values</code> ', and ' <code>dist</code> ' and others for detailed control of the random sources. See <code>?NMseed</code> for details on what arguments can be passed this way. Default is to draw seeds between 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 <code>seed.nm="asis"</code> . In case <code>method.sim=NMsim_EBE</code> , seeds are not used.
<code>args.psn.execute</code>	A charachter string that will be passed as arguments PSN's ' <code>execute</code> '.

<code>table.vars</code>	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 <code>table.vars</code> 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 big difference in execution time. See <code>table.options</code> too.
<code>table.options</code>	A character vector or a string of space-separated options. Only used if <code>table.vars</code> is provided. If constructing a new output table with <code>table.vars</code> the default is to add two options, NOAPPEND and NOPRINT. You can modify that with <code>table.options</code> . Do not try to modify output filename - NMsim takes care of that.
<code>text.sim</code>	A character string to be pasted into \$SIMULATION. This must not contain seed or SUBPROBLEM which are handled separately. Default is to include "ONLYSIM". To avoid that, use <code>text.sim=""</code> .
<code>method.sim</code>	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 <code>NMsim_default</code> which will replace any estimation and covariance step by a simulation step. See details section on other methods, and see examples and especially vignettes on how to use the different provided methods.
<code>typical</code>	Run with all ETAs fixed to zero? Technically all ETAs=0 is obtained by replacing \$OMEGA by a zero matrix. Default is FALSE.
<code>execute</code>	Execute the simulation or only prepare it? <code>'execute=FALSE'</code> can be useful if you want to do additional tweaks or simulate using other parameter estimates.
<code>sge</code>	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.
<code>nc</code>	Number of cores used in parallelization. This is so far only supported with <code>method.execute="psn"</code> .
<code>transform</code>	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.
<code>method.execute</code>	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 <code>type.sim="NMsim_EBE"</code> . "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 <code>path.nonmem</code> is specified, and "psn" if not.
<code>method.update.inits</code>	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. The possible values are

- "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.
- "nmsim" Uses a simple internal method to update the parameter values based on the `ext` file. The advantages of "nmsim" are it does not require PSN, and that it does not rely on code-interpretation for generation of simulation control streams. "nmsim" fixes the whole OMEGA and SIGMA matrices as single blocks making the \$OMEGA and \$SIGMA sections of the control streams less easy to read. On the other hand, this method is robust because it avoids any interpretation of BLOCK structure or other code in the control streams.
- "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.  
On linux/mac, The default is to use "PSN" if found. On Windows, "nmsim" is the default.

<code>create.dirs</code>	If the directories specified in <code>dir.sims</code> and <code>dir.res</code> do not exist, should it be created? Default is TRUE.
<code>dir.psn</code>	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.
<code>modify.model</code>	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.
<code>sim.dir.from.scratch</code>	If TRUE (default) this will wipe the simulation directory before running new simulations. The directory that will be emptied is <code>_not_dir.sims</code> where you may keep many or all your simulations. It is the subdirectory named based on the run name and <code>name.sim</code> . 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 ran 500. If you choose FALSE here, you can end up with the results of 500 new and 500 old simulations.
<code>col.row</code>	Only used if data is not supplied (which is most likely for simulations for VPCs) A column name to use for a row identifier. If none is supplied, <code>NMdataConf()[['col.row']]</code> will be used. If the column already exists in the data set, it will be used as is, if not it will be added.
<code>args.NMscanData</code>	If <code>execute=TRUE&amp;sge=FALSE</code> , NMsim will normally read the results using <code>NMreadSim</code> . 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

	<code>'args.NMscanData=list(col.row="ROW")'</code> to merge by a column called <code>'ROW'</code> . This is only used in rare cases.
<code>path.nonmem</code>	The path to the Nonmem executable to use. The could be something like <code>"/usr/local/NONMEM/run/nmfe7"</code> (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.
<code>nmquiet</code>	Silent messages from Nonmem. The default is <code>'TRUE'</code> .
<code>progress</code>	Track progress? Default is <code>'TRUE'</code> if <code>'quiet'</code> is <code>FALSE</code> 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.
<code>as.fun</code>	The default is to return data as a data.frame. Pass a function (say <code>'tibble::as_tibble'</code> ) in <code>as.fun</code> to convert to something else. If data.tables are wanted, use <code>as.fun="data.table"</code> . The default can be configured using <code>NMdataConf</code> .
<code>suffix.sim</code>	Deprecated. Use <code>name.sim</code> instead.
<code>text.table</code>	A character string including the variables to export from Nonmem.
<code>system.type</code>	A character string, either <code>"windows"</code> or <code>"linux"</code> - case insensitive. Windows is only experimentally supported. Default is to use <code>Sys.info()[["sysname"]]</code> .
<code>dir.res</code>	Provide a path to a directory in which to save rds files with paths to results. Default is to use <code>dir.sims</code> . After running <code>'NMreadSim()'</code> on these files, the original simulation files can be deleted. Hence, providing both <code>'dir.sims'</code> and <code>'dir.res'</code> provides a structure that is simple to clean. <code>'dir.sims'</code> can be purged when <code>'NMreadSim'</code> has been run and only small <code>'rds'</code> and <code>'fst'</code> files will be kept in <code>'dir.res'</code> . Notice, in case multiple models are simulated, multiple <code>'rds'</code> (to be read with <code>'NMreadSim()'</code> ) files will be created by default. In cases where multiple models are simulated, see <code>'file.res'</code> to get just one file refering to all simulation results.
<code>file.res</code>	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 <code>'NMreadSim()'</code> . The default is to create a file called <code>'NMsim_..._MetaData.rds'</code> under the <code>dir.res</code> directory where <code>...</code> is based on the model name. However, if multiple models ( <code>file.mod</code> ) 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 <code>file.res</code> is supplied, <code>dir.res</code> is not used.
<code>wait</code>	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 <code>'rds'</code> files to read will be returned. Pass them through <code>'NMreadSim()'</code> (which also supports waiting for the simulations to finish).
<code>auto.dv</code>	Add a column called <code>'DV'</code> to simulation data sets if a column of that name is not found? Nonmem is generally dependent on a <code>'DV'</code> column in input data but this is typically uninformative in simulation data sets and hence easily forgotten when generating simulation data sets.
<code>quiet</code>	If <code>TRUE</code> , messages from what is going on will be suppressed.
<code>check.mod</code>	Check the provided control streams for contents that may cause issues for simulation. Default is <code>'TRUE'</code> , 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.



<code>seed</code>	Deprecated. See <code>seed.R</code> and <code>seed.nm</code> .
<code>list.sections</code>	Deprecated. Use <code>modify.model</code> instead.
<code>format.data.complete</code>	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.
<code>...</code>	Additional arguments passed to <code>method.sim</code> .

## 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 `modify.model` argument is handled in addition to the `method.sim`. The subproblems and seed 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_typical` Deprecated. Use `typical=TRUE` instead.
- `NMsim_EBE` Simulates `_known_` ETAs. By default, the ETA values are automatically taken from the estimation run. This is what is referred to as empirical 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 referring 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.

---

NMsim_asis	<i>Simulation method that uses the provided control stream as is</i>
------------	--

---

**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	<i>Transform an estimated Nonmem model into a simulation control stream</i>
---------------	---

---

**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,
  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.
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	<i>Use empirical Bayes estimates to simulate re-using ETAs</i>
-----------	--

---

**Description**

Simulation reusing ETA values from estimation run or otherwise specified ETA values. For observed subjects, this is referred to as empirical 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	See ?NMsim.
file.mod	See ?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_typical	<i>Typical subject simulation method</i>
---------------	--

---

**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	<i>Simulate with parameter values sampled from a covariance step</i>
--------------	--

---

**Description**

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. 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.

**Usage**

```
NMsim_VarCov(file.sim, file.mod, data.sim, nsims, ext, write.ext = NULL)
```

**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.
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.

**Value**

Character vector of simulation control stream paths

---

readParsWide	<i>Parameter data from csv</i>
--------------	--------------------------------

---

**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 = NULL,
  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	Name of the model counter, default is "model". 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.

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

## 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)
```

---

simPopEtas	<i>Generate a population based on a Nonmem model</i>
------------	--

---

**Description**

Generate a population based on a Nonmem model

**Usage**

```
simPopEtas(file.mod, N, seed, file.phi, as.fun)
```

**Arguments**

file.mod	Path to input control stream
N	Number of subjects to generate
seed	Optional seed. Will be passed to 'set.seed'. Same thing as running 'set.seed' just before calling 'simPopEtas()'.
file.phi	An optional phi file to write the generated subjects to.
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.

---

unNMsimModTab	<i>Remove NMsimModTab class and discard NMsimModTab meta data</i>
---------------	---

---

**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 an '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

addEVID2, [2](#)  
addResVar, [3](#)

cbind.NMsimModTab (unNMsimModTab), [23](#)  
cbind.NMsimRes (unNMsimRes), [24](#)

dimnames.NMsimModTab (unNMsimModTab), [23](#)  
dimnames.NMsimRes (unNMsimRes), [24](#)

genPhiFile, [5](#)

inputArchiveDefault, [6](#)  
is.NMsimModTab (unNMsimModTab), [23](#)  
is.NMsimRes (unNMsimRes), [24](#)

merge.NMsimModTab (unNMsimModTab), [23](#)  
merge.NMsimRes (unNMsimRes), [24](#)

NMcreateDoses, [6](#)  
NMexec, [8](#)  
NMreadSim, [10](#)  
NMsim, [11](#)  
NMsim\_asis, [18](#)  
NMsim\_default, [18](#)  
NMsim\_EBE, [19](#)  
NMsim\_typical, [20](#)  
NMsim\_VarCov, [20](#)  
NMsimModTabOperations (unNMsimModTab),  
[23](#)  
NMsimResOperations (unNMsimRes), [24](#)

rbind.NMsimModTab (unNMsimModTab), [23](#)  
rbind.NMsimRes (unNMsimRes), [24](#)  
readParsWide, [21](#)

simPopEtas, [23](#)

t.NMsimModTab (unNMsimModTab), [23](#)  
t.NMsimRes (unNMsimRes), [24](#)

unNMsimModTab, [23](#)  
unNMsimRes, [24](#)