Package: amp.sim (via r-universe)

May 26, 2026

Title Flexible Simulation Utilities for Pharmacometric Modeling

Version 0.1.2

Description The goal of 'amp.sim' is to transform 'NONMEM' models into R syntax so they can be used for simulations using the 'deSolve', 'nlmixr2' or 'mrgsolve' package. Additionally, functionality is included to aid simulations performed directly in 'NONMEM' and to automatically create shiny apps for simulation models.

URL https://leidenadvancedpkpd.github.io/amp.sim/, https://github.com/LeidenAdvancedPKPD/amp.sim

BugReports https://github.com/LeidenAdvancedPKPD/amp.sim/issues

License GPL (>= 3)

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

VignetteBuilder quarto

Depends R (>= 4.1.0)

Imports nonmem2rx, stats, utils, MASS, whisker, rlang, cli, lifecycle, NMdata (>= 0.2.3)

Suggests tidyr, shiny, shinyjs, bslib, rstudioapi, deSolve, rxode2, mrgsolve, R3port, ggplot2, dplyr, testthat, quarto

Config/roxygen2/version 8.0.0

Config/pak/sysreqs libfontconfig1-dev libfreetype6-dev make libxml2-dev

Repository https://leidenadvancedpkpd.r-universe.dev

Date/Publication 2026-05-26 13:23:14 UTC

RemoteUrl https://github.com/leidenadvancedpkpd/amp.sim

RemoteRef HEAD

RemoteSha e12c85369ef9afc420975b712cfc33d803d059a9

Contents

`conv_pow` *Convert infix power notation to prefix notation*

Description

This function will transform the power notation that can be used in R to the one needed by mrgsolve

Usage

```
conv_pow(x)
```

Arguments

x	character vector with the formulas to convert

Value

a vector with the transformed power notations

Author(s)

Richard Hooijmaijers

Examples

```
conv_pow("y = par1*(par2/par3)^xy + a - par4**(2/par5)")
```

`convert_nmsyntax` *Convert NONMEM specific syntax to R syntax*

Description

This function converts a NONMEM syntax to R syntax, mainly for functions and operators

Usage

```
convert_nmsyntax(x, type = "mrgsolve")
```

Arguments

`x`	character vector with the syntax to be converted
`type`	character with the type of syntax to convert to (currently "deSolve" and "mrgsolve" and "rxode2" are supported)

Value

character vector with the converted syntax

Author(s)

Richard Hooijmaijers

Examples

```
convert_nmsyntax("IF(VAR.GT.0) VAR2 = PHI(1)")
```

convert_nonmem *Convert NONMEM model to R syntax*

Description

This function converts a NONMEM model to syntax useable in R simulations. Currently deSolve, rxode2 (nonmem2rx) and mrgsolve are available syntaxes to use Additionally a code to control the simulations is created to directly test the simulations

Usage

```
convert_nonmem(
  model,
  out,
  ext = NULL,
  mod_return = NULL,
  type_return = "mrgsolve",
  overwrite = FALSE,
  control = "file",
  verbose = TRUE
)
```

Arguments

model	character with the model file to be read in and converted
out	character with the name of the output file without a file extension
ext	character with the name of the NONMEM ext file (if not provided estimates are read directly from control stream)
mod_return	a character vector indicating which items should be returned from the model function. For more information see details
type_return	character indicating the type of model that should be created. Currently "deSolve", "rxode2", "nonmem2rx" and "mrgsolve" are accepted
overwrite	logical indicating if the output model should be overwritten
control	character indicating how the model control code should be returned. Currently "file", "console", "string", "script" and "model" (only for rxode2/DeSolve) are accepted
verbose	logical indicating if additional information is written to the console

Details

For the mod_return argument, the additional variables are added to the output in case the type_return is either mrgsolve or deSolve.

Value

a converted file is generated and a message is returned

Author(s)

Richard Hooijmaijers

Examples

```
mod  <- system.file("example_models","PK.1CMT.ORAL.mod", package = "amp.sim")
outf <- tempfile()

convert_nonmem(mod, outf, verbose = FALSE)
readLines(paste0(outf,".cpp")) |> head(n=15) |> cat(sep="\n")

convert_nonmem(mod, outf, verbose = FALSE, type_return = "nonmem2rx") |> suppressMessages()
readLines(paste0(outf,".r")) |> head(n=15) |> cat(sep="\n")
```

---

cut_equal *cut function for (appr.) equal intervals*

Description

This function uses cut to make categories based on approximately equal number of bins

Usage

```
cut_equal(x, n, type = 1, ntries = 1000)
```

Arguments

x	the vector that should be cut into equal bins
n	the number of bins that should be used
type	the type of algorithm to be used (see details)
ntries	number of samples/tries for type 2

Details

Generating equal bins can be quite difficult. There are multiple ways of assessing if bins are equal. This function provides a method based on quantiles (type=1) or a method based on sampling proposed by M. Ruppert (type=2). In general type 1 is faster but less accurate, while type 2 is slower and more accurate (in case of a reasonable ntries)

Value

a character vector with the categories/bins

Author(s)

Richard Hooijmaijers

Examples

```
table(cut_equal(1:20,5))
```

`dose_func` *Create data.frame with dosing for usage in simulation with deSolve*

Description

This function creates a data.frame that can be used for events of a deSolve simulation

Usage

```
dose_func(cmt, value, tinf, tau, ndose, times)
```

Arguments

`cmt`	number of compartment or differential equation where the dosing should be given
`value`	the value of the dosing that should be given
`tinf`	in case value is set an infusion is assumed with tinf as infusion time
`tau`	dosing interval to be used
`ndose`	number of doses to be used
`times`	In case tau and ndose cannot be used (unequal dosing), this parameter can be used to set times of dosing (e.g c(0,24,168))

Value

a data frame that can be used as an event dataset within deSolve

Author(s)

Richard Hooijmaijers

Examples

```
dose_func(8,100,tau=48,ndose=5,tinf=2)
```

dput2 — *Wrapper function around dput to provide more options*

Description

This function wraps around the dput function and provide more options to save result in a vector

Usage

```
dput2(x, comment = FALSE, obj = NULL, collapse = NULL, ...)
```

Arguments

x	An object passed to dput
comment	logical indicating if comment characters should be prepended to results
obj	character of length one indicating the name of the object that should be prepended to result
collapse	character of length one with the collapse character. If provided the result will be pasted with this collapse character
...	additional arguments passed to dput

Value

a vector with the result from dput

Author(s)

Richard Hooijmaijers

Examples

```
dput2(setNames(runif(5),letters[1:5]))
```

get_est — *Get model estimates from NONMEM ext or model file*

Description

This function gets the estimates (THETA, ETA and OMEGA) from a NONMEM ext or model file to be used within the simulations in R. The model file is included as option as the names of THETAs can be obtained in case this is set as comment in the model file

Usage

```
get_est(from)
```

Arguments

from	the model or ext file (or data.frame/model text string from results object) to be read in to obtain estimates. extension or class of object of file determines the actions to be taken

Details

the function will return a list with theta, eta, omega and naming of theta values. In case a model is used as input, the values represent the initial values from a model. In case the ext file is used, the final estimates are taken. The eta values are all set to 0. The omega values are returned as a matrix so it can be used for sampling (e.g. using mvrnorm). naming of thetas is taken from the model comments in the THETA block or in case an ext file is used naming is set to THETA1:n. In case the model is used as input, there are some assumptions within the function on how the model is coded. For the omega block the value of omega must always be placed on a separate line (e.g. $OMEGA 0.1 is not permitted as 0.1 should be placed on the next line. Also covariance within the omega block should be placed on the same line separated by spaces (e.g. for a BLOCK(2) the first line should state variance eta1 and the second line should state covariance eta1, eta2 followed by variance eta2). For the theta block, it is assumed that in case lower and upper boundaries are available they are separated by commas.

Value

a list with theta, eta and omega values

Author(s)

Richard Hooijmaijers

Examples

```
# get the initial estimates from the model or final estimates from ext file
mod  <- system.file("example_models","PK.1CMT.ORAL.mod", package = "amp.sim")
ext  <- system.file("example_models","PK.1CMT.ORAL.ext", package = "amp.sim")
get_est(mod)
get_est(ext)
```

---

get_inits *get the initial states for the differential equations*

Description

This function will extracts the initial states for the differential equations from a NONMEM model

Usage

```
get_inits(lstblock)
```

Arguments

lstblock	list with each item being a separate structured dollar block, usually obtain from nmlistblock

Value

a named vector with the state values

Author(s)

Richard Hooijmaijers

Examples

```
mod    <- system.file("example_models","PK.1CMT.ORAL.mod", package = "amp.sim")
mdll   <- get_nmblock(mod,block=c("PK","DES"))
mdlls  <- nmlistblock(mdll)
get_inits(mdlls)
```

---

get_nmblock *Get information from dollar blocks inside NONMEM control streams*

Description

This function returns a list with indices or content of dollar blocks

Usage

```
get_nmblock(model, block, ret = "content", omitbn = TRUE)
```

Arguments

model	character vector of length 1 with filename of model, in case length is greater than 1 it is assumed to be a vector with model code
block	character vector with names of the model blocks. Take into account that grep is used with respect to partial matching
ret	character with the type of return value can be either "content" or "index"
omitbn	logical indicating if the name of the block should be omitted when return (has only effect if ret="content")

Value

a list with either a numeric vector with the indices or a character vector with the content of the dollar blocks

Author(s)

Richard Hooijmaijers

Examples

```
mod  <- system.file("example_models","PK.1CMT.ORAL.mod", package = "amp.sim")
get_nmblock(mod,"OMEGA")
```

---

get_param *Get parameter values from model or ext file*

Description

This function gets the parameter values from a NONMEM model or ext file including naming and additional variables if present

Usage

```
get_param(model, lstblock, ext = NULL, addparam = TRUE)
```

Arguments

model	character vector with the model content
lstblock	list with each item being a separate structured dollar block, usually obtain from nmlistblock
ext	character with the name of the NONMEM ext file (if not provided estimates are read directly from the list block)
addparam	logical indicating if the function should try to add parameters (besides THETAs and the ones defined in covariates) the additional parameters are always returned so it can be used for warnings

Value

a list with parameters, names and matrices

Author(s)

Richard Hooijmaijers

Examples

```
mod    <- system.file("example_models","PK.1CMT.ORAL.COV.mod", package = "amp.sim")
mdll   <- get_nmblock(mod,block=c("PK","DES"))
mdlls  <- nmlistblock(mdll)
get_param(mod, mdlls)
```

get_parmvar — *Get the variables within a NONMEM model that should be passed to the simulation model*

Description

This function go through all formulas and control flows to check if a parameter is created ad-hoc or taken from the input file. This is important as these variables need to be set for simulation

Usage

```
get_parmvar(lstblock, returnall = FALSE)
```

Arguments

lstblock	list with each item being a separate structured dollar block, usually obtain from nmlistblock
returnall	logical indicating if all variables should be returned or just the ones that are not defined

Value

a vector with model variables for the simulation model

Author(s)

Richard Hooijmaijers

Examples

```
mod    <- system.file("example_models","PK.1CMT.ORAL.COV.mod", package = "amp.sim")
mdll   <- get_nmblock(mod,block=c("PK","DES"))
mdlls  <- nmlistblock(mdll)
get_parmvar(mdlls)
```

make_nmsimmodel — *Create a simulation model from original model*

Description

This function creates a simulation model from an original NONMEM model where dollar blocks are replaced and/or commented based on input file and other simulation functions within package

Usage

```
make_nmsimmodel(omod, smod, data, subprobs = 1, table = NULL, sigma_ext = NULL)
```

Arguments

omod	character string for the original model
smod	character string for the simulation model (for writing the model to disk)
data	character string for the input dataset for the simulation
subprobs	numeric indicating the number of subproblems in simulation model
table	character of length 1 with the items for the dollar table block (if null it will use items in input dataset)
sigma_ext	character with the name of the ext file which includes the sigma value (in case residual error is coded in the dollar sigma block)

Details

The function will adapt an original model in such a way that it can be used directly for simulations the assumptions are that an input dataset is created using sample_par and simdata. This means that all THETA and ETA values are available in the simulation dataset as respectively STHETAN and SETAN (simulated THETA/ETA n). Furthermore it is assumed that the input dataset is a csv file. The function will first comment all applicable dollar blocks. Then a OMEGA is added with 0 FIX and the INPUT, DATA and TABLE blocks are appended with information based on the input dataset. Finally all THETAs and ETAs are replaced with the items in the dataset and the simulation model is written to disk.

Value

a simulation model (file) is created

Author(s)

Richard Hooijmaijers

Examples

```
nmmod   <- system.file("example_models","PK.1CMT.ORAL.mod", package = "amp.sim")
dat     <- simdata(0:24, dosetime = 0, doseheight = 10, addl = 2, ii = 24,
                   numid = 50, STHETA1= 1, STHETA2 = 2, STHETA3 = 1,
                   SETA1 = 0, SETA2 = 0)
tmp_out <- tempfile(fileext = ".csv")
mod_out <- tempfile(fileext = ".mod")

write.csv(dat,file=tmp_out, na=".", quote=FALSE, row.names = FALSE)
make_nmsimmodel(nmmod, mod_out, data=tmp_out)
readLines(mod_out) |> head(n=15) |> cat(sep="\n")
```

mdose *Performs multiple dosing in case of analytical solution of model*

Description

This function takes a function that defines a model in analytical solution and performs multiple dosing for it

Usage

```
mdose(Dose, tau, ndose, t, func, ...)
```

Arguments

Dose	numeric vector with the dosing height
tau	numeric vector with the tau of dosing
ndose	numeric vector with the number of doses
t	numeric vector with the time-points that should be outputted
func	name of the function for which multiple dosing should be applied
...	arguments for func

Details

This function will create a list that can be used to perform superposition which is necessary in case of an analytical solution in a multiple dose setting. The function will check if there is an overlap in arguments and will use the arguments given to mdose for the function given in func if applicable (e.g. it is likely that func has an argument for Dose, in this case it will use the Dose argument provide in mdose) The function can have any number of arguments that can be passed using "...". However there should at least be an argument t which is the time vector for which simulations are necessary.

Value

a data frame with the superimposed results

Author(s)

Richard Hooijmaijers

Examples

```
ana1CMTiv <- function(Dose,pars,t){
  Dose * pars['C'] * exp(-pars['L']*t)
}
res <- mdose(10, tau = 24, ndose = 5, t = 0:240,
             func = ana1CMTiv, pars = c(L=.01,C=5))
plot(res$time,res$y, type="l")
```

mod2shiny *Creates a basic simulation app for a given R model*

Description

This function fills in a default template for simulations within shiny

Usage

```
mod2shiny(
  parvector,
  modfile,
  evnt,
  init = NULL,
  naming = NULL,
  apptitle = "Shiny app title",
  outloc = ".",
  omega = NULL,
  sigma = NULL,
  delloc = FALSE,
  framework = "deSolve",
  logo = paste0(system.file(package = "amp.sim"), "/logo.png"),
  times = NULL
)
```

Arguments

parvector	named vector with the model parameters, should contain all parameters used by the model
modfile	A script with the model defined in it, it is assumed that this file is created using convert_nonmem
evnt	dataframe with the events used by the model (this is saved as rds together with the app)
init	vector with the compartment initialization (only applicable for deSolve framework)
naming	named vector with the names in parvector and the new values to use within the shiny app
apptitle	string with the title to be used for the app
outloc	character with location where the resulting shiny app should be saved
omega	vector with the omega matrix for the model (only applicable for rxode2 framework)
sigma	vector with the sigma matrix for the model (only applicable for rxode2 framework)
delloc	logical indicating if the location should be deleted first

framework	character indicating the simulation framework that was used, currently "deSolve", "rxode2", "nonmem2rx" and "mrgsolve" are supported
logo	character with a png of the company logo added in the header of the shiny app
times	character vector with the times to simulate or set to NULL to use a default time vector

Details

This function creates a default shiny app that can be used as a starting point for further development. There are already some basic features available but it is intended to be adapted when used for production. For the automatic creation of an app it is assumed that the model is created using the convert_nonmem function. Although it is not strictly necessary, the information provided to this function will be in the correct format when this function is used. Some of the arguments in this function are only necessary in case a certain conversion framework is used. In general a single subject is simulated, but be aware that for the deSolve and rxode2 framework OMEGA/ETA is implemented.

Value

creates necessary app files for a shiny simulation app

Author(s)

Richard Hooijmaijers

Examples

```
nmmod <- system.file("example_models","PK.1CMT.ORAL.mod", package = "amp.sim")
outf  <- tempfile()
convert_nonmem(nmmod, out=outf, verbose = FALSE)
prm   <- c(THETA1 = 0.3, THETA2 = 2, THETA3 = 5)
evnt  <- mrgsolve::ev(amt = 100, ii = 24, addl = 1)
nams  <- c(THETA1 = "KA (1/h)", THETA2 = "CL (l/h)", THETA3 = "V (l)")
mod2shiny(prm, modfile= paste0(outf,".cpp"), evnt = evnt,
          naming = nams, framework = "mrgsolve", outloc=tempdir())
```

---

model_validation *Validates NONMEM estimation model with an mrgsolve simulation model*

Description

This function uses the estimates from a NONMEM run and compares this with the results of a simulation run. This function is inspired by a blog post from mrgsolve and mainly looks at the differences in population predictions

Usage

```
model_validation(
  nmtable,
  simmodel,
  rounding = 4,
  comppred = "CP",
  out = NULL,
  ...
)
```

Arguments

nmtable	either a character with a file or a data frame including the NONMEM table output
simmodel	character with the file including the mrgsolve model
rounding	numeric with the rounding applied for comparing
comppred	character with the variable in mrgsolve model that should be compared with PRED variable in NONMEM
out	character with the name of the output to create
...	additional arguments passed through to mrgsim function

Details

For a correct comparison, the nmtable should include all variables related to dosing (e.g. AMT/CMT/EVID). The simulation model should be available as a separate file that can be read in using mrgsolve::mread. To use the function, the packages R3port, ggplot2, mrgsolve and dplyr should be installed. Be aware that no variables are renamed in $TABLE in the NONMEM control stream (e.g. AMT2=AMT). This can have unexpected results when comparing.

Value

a file with a PDF report is returned

Author(s)

Richard Hooijmaijers

Examples

```
res  <- model_validation(system.file("testfiles/compareParfile",package="amp.sim"),
                         system.file("testfiles/compareModel.cpp",package="amp.sim"),
                         out=NULL)
```

nmlistblock *convert a block to a list*

Description

This function will convert a NONMEM block to a list including the type and separate formula parts

Usage

```
nmlistblock(dollmodel)
```

Arguments

dollmodel	list with each item being a separate dollar block, usually obtain from get_nmblock

Value

a list with the structured code

Author(s)

Richard Hooijmaijers

Examples

```
nmmod <- system.file("example_models","PK.1CMT.ORAL.mod", package = "amp.sim")
lst   <- get_nmblock(nmmod, block = "PROB")
nmlistblock(lst)
```

overlaying *Performs overlaying of simulations*

Description

This function basically appends the simulation output and add this to a reactiveValue

Usage

```
overlaying(out, input, savedsims = NULL)
```

Arguments

out	dataframe with the results from a simulation to be appended for overlaying
input	list with the input elements from a shiny app
savedsims	reactiveValue or list that contains the saved simulations

Value

a list with a dataframe with the appended simulations and settings

Author(s)

Richard Hooijmaijers

Examples

```
if(requireNamespace("shiny")){
  out   <- data.frame(time=0:4, A1=rnorm(5), numsim=1)
  input <- shiny::reactiveValues(CL=2, KA=0.3, updOpts="appsim")
  overlayres <- overlaying(out,input)
  savedsims  <- list(res = overlayres$results, sett = overlayres$settings)
  overlayres <- overlaying(input=input,out=out,savedsims=savedsims)
  head(overlayres$result)
}
```

---

par_delete *Delete parenthesis including a numeric within a vector*

Description

This function deletes all parenthesis from a character vector that has a numeric value inside. This is applicable as THETA(n), ETA(n), DADT(n) is not always allowed in R coding

Usage

```
par_delete(vect, excfun = TRUE)
```

Arguments

vect	the vector that should be scanned for parenthesis
excfun	exclude functions from deleting parenthesis (taken from the globally defined function list in the package)

Value

a vector with the stripped parenthesis

Author(s)

Richard Hooijmaijers

Examples

```
par_delete(c("LOG(1)","ETA(1)","EXP(2)/ETA(3)+THETA(4)"))
```

pos_clpar *Get position of closing parenthesis*

Description

This function gets the position of the closing parenthesis after the first opening one

Usage

```
pos_clpar(x)
```

Arguments

x	character string for which the parenthesis should be searched

Value

a numeric with the position of the closing parenthesis (will be -1 if no match is present)

Author(s)

Richard Hooijmaijers

Examples

```
tst <- "IF (test == A(1)) a(1)=(1*5)/2"
pos_clpar(tst)
substring(tst,1,pos_clpar(tst))
```

sample_par *Sample model parameters from a multivariate normal distribution*

Description

This function samples model parameters from NONMEMs covariate matrix (*.cov) and final parameter estimates (*.ext). Furthermore etas can also be sampled where eta blocks are taken into account (see details)

Usage

```
sample_par(
  ext,
  covmat = NULL,
  bootstrap = NULL,
  seed = NULL,
  nrepl = 10,
  inc_theta = TRUE,
  inc_eta = FALSE,
  verbose = FALSE,
  dropfixed = FALSE,
  uncert = FALSE,
  restheta = NULL
)
```

Arguments

ext	character string with location of ext file with final model parameters
covmat	character string with location of cov file with covariance matrix or data.frame of cov file
bootstrap	character string with location of ext files from bootrstrap or vector with ext files from bootstrap (relevant in case ext files should be excluded because of minimization issues)
seed	a numeric with the seed number used in set.seed to enable reproducibility, when not provided the seed from the global environment will be used (e.g. using base::set.seed)
nrepl	numeric with the number of replicates to sample
inc_theta	logical indicating if THETAs should be added to result
inc_eta	logical indicating if ETAs should be added to result
verbose	logical indicating if additional information should be added to result (e.g. OMEGA/SIGMA values)
dropfixed	logical indicating if parameters that are fixed should be dropped (can only be done in case covmat is provided)
uncert	logical indicating if the uncertainty should be sampled
restheta	character with the theta that describes residual error (e.g. "THETA3") in case uncertainty is sampled this parameter will be set to the population value

Details

In general the function can be used to sample from covariance matrix so the different parameters can be added to a simulation dataset or model to enable uncertainty simulations. In most cases it is not necessary to include OMEGAs or SIGMAs in these type of simulations. It can be convenient to add ETAs in the simulation dataset to perform a simulation where no $THETA or $OMEGA information is necessary. In case inc_eta is TRUE, the ETAs from the ext files are used and placed in a matrix to take into account covariance or 'BLOCKS'. A matrix from the ext file is constructed based on the naming of OMEGA values (e.g. OMEGA.2.1. will be added to row 2, column 1 and column 1, row 2). The matrix is used as Sigma for the mvrnorm function with mu=0.

Value

a dataframe with sampled values

Author(s)

Richard Hooijmaijers

See Also

MASS::mvrnorm

Examples

```
ext <- system.file("example_models","PK.1CMT.ORAL.COV.ext", package = "amp.sim")
cov <- system.file("example_models","PK.1CMT.ORAL.COV.cov", package = "amp.sim")
sample_par(ext, inc_eta = TRUE, nrepl = 5)
sample_par(ext, cov, uncert = TRUE, nrepl = 5)
```

---

sample_sim *Sample model parameters for a simulation*

Description

This function samples parameters for a simulation. It wraps around the sample_par function on the background

Usage

```
sample_sim(nrepl = 2, nsub = 3, type = "noIIV", ...)
```

Arguments

nrepl	Number of replicates for the simulation
nsub	Number of subjects for the simulation
type	character with the type of simulation to perform (see details)
...	Additional arguments passed to sample_par mainly for passing information for ext and cov files

Details

This function is a high level function wrapper for the sample_par function specified for different types of simulations that might occur. Currently the following situations can be sampled:

- noIIV: Sample without uncertainty and without IIV
- sameIIV: Sample without uncertainty and with the same IIV values within subjects
- varIIV: Sample without uncertainty and with different IIV values within subjects

- unc_noIIV: Sample with uncertainty and without IIV
- unc_sameIIV: Sample with uncertainty and and with the same IIV values within subjects
- unc_varIIV: Sample with uncertainty and with different IIV values within subjects

In all the cases above where uncertainty is sampled, this is done only for THETA values.

Value

a dataframe with sampled values

Author(s)

Richard Hooijmaijers

Examples

```
ext <- system.file("example_models","PK.1CMT.ORAL.COV.ext", package = "amp.sim")
cov <- system.file("example_models","PK.1CMT.ORAL.COV.cov", package = "amp.sim")
sample_sim(nrepl=2,nsub=3,type="unc_varIIV", ext=ext,cov=cov)
sample_sim(nrepl=2,nsub=3,type="sameIIV", ext=ext)
```

---

settings2df *Add settings to dataframe*

Description

This function adds settings or input elements to a dataframe for displaying in app

Usage

```
settings2df(savedsims, leaveout = c("go", "updOpts", "sett", "refr", "tabsel"))
```

Arguments

savedsims	reactiveValue that contains the saved simulations
leaveout	character vector with the the elements that should be left out the dataframe

Value

a dataframe with settings

Author(s)

Richard Hooijmaijers

Examples

```
if(requireNamespace("tidyr")){
  sim1 <- list(THETA1  = 0.5, THETA2 = 1,
               alllabs =c("THETA1%=%DUMMY1","THETA2%=%DUMMY2"))
  sim2 <- list(THETA1  = 0.9, THETA2 = 1.5)
  settings2df(list(sim1 = sim1, sim2 = sim2))
}
```

`simdata` *Create a simulation dataset for NONMEM simulation*

Description

This function creates a simulation dataset including information for dosing and sampling. The function is setup to return a data frame to be used within NONMEM or other frameworks

Usage

```
simdata(time, dosetime, doseheight, addl, ii, rate = NA, numid = 5, ...)
```

Arguments

`time`	a vector with all sampling times to be used
`dosetime`	a vector with the different dosing times
`doseheight`	a vector with the different dosing heights (to be added to AMT)
`addl`	a vector with the additional dose levels (must be same length as doseheight)
`ii`	a vector with the interdose interval (must be same length as doseheight)
`rate`	a vector with the dosing rate (must be same length as doseheight)
`numid`	a vector with the number of IDs to be created
`...`	additional variables to be added to the resulting dataset

Value

a dataframe that can be used for NONMEM simulations

Author(s)

Richard Hooijmaijers

Examples

```
# Include additional variables
sim1 <- simdata(seq(0,24,1),0.5,100,10,12,NA,2, WEIGHT=70, ETA=0)

# unequal dosing scheme
sim2 <- simdata(seq(0,24,1), dosetime = c(0.5,1), doseheight = c(100,200),
                addl = c(10,5), ii = c(12,24), numid = 2)

# Directly create a sequence of different dose levels
sim3 <- lapply(seq(25,60,5),function(x){
  simdata(time=1:12,dosetime=0,doseheight=x,addl=139,ii=120,rate=0,numid=10)
})
sim3 <- do.call(rbind,sim3)
```

---

split_sim *Split a large NONMEM simulation in chunks*

Description

This function splits a large simulation dataset and accompanying model in chunks to prevent memory problems within the simulation or to enable simulating over multiple cores

Usage

```
split_sim(data, model, locout, splitby = "ID", numout = 4)
```

Arguments

data	character string for input dataset or a dataframe with the simulation data (see details)
model	character string for the simulation model
locout	character string for the location of the split input dataset and models
splitby	character string with the variable in the data to be split on (if this is not ID it could lead to unexpected results)
numout	the number of 'equal length' outputs to be created.

Details

In general the data can be a character string defining the input dataset (csv file) or a dataframe it is proposed to use a dataframe as in many cases the splitting should take place for a large problem. In case a dataframe is used (e.g. from loading an Rdata object) it is not necessary to import a huge csv file.

Value

split CSV and models files are written to disk

Author(s)

Richard Hooijmaijers

Examples

```
nmmod   <- system.file("example_models","PK.1CMT.ORAL.mod", package = "amp.sim")
dat     <- simdata(0:24, dosetime = 0, doseheight = 10, addl = 2, ii = 24,
                   numid = 50, STHETA1= 1, STHETA2 = 2, STHETA3 = 1,
                   SETA1 = 0, SETA2 = 0)
tmp_out <- tempfile(fileext = ".csv")
mod_out <- tempfile(fileext = ".mod")

write.csv(dat,file=tmp_out, na=".", quote=FALSE, row.names = FALSE)
make_nmsimmodel(nmmod, mod_out, data=tmp_out)

split_sim(data = tmp_out, model = mod_out, locout=tempdir())
list.files(tempdir(), pattern="\\.mod$")
```

---

tmpl_model *Get coding for template models*

Description

This function returns the code for some template models

Usage

```
tmpl_model(tmpl, ret = "console")
```

Arguments

tmpl	character with template
ret	character indicating how the result should be returned. Currently "console", "string" and "script" are accepted

Details

There are templates available for 1-2 CMT PK models for IV/bolus/oral administration as both analytical as closed form and parameterized with constants or as CL/V. To see which templates are available in the package run the function without arguments

Value

model syntax is returned either to console or script (within Rstudio) or a character string

Author(s)

Richard Hooijmaijers

Examples

```
tmpl_model()
tmpl_model("ana1CMTbolusK.tmp")
```

Index