Package 'mrgsolve'

April 6, 2016

Type Package

Version 0.5.12.9000

Title Simulation from ODE-Based Population PK/PD and Systems Pharmacology Models

Author Kyle T. Baron [cre,aut,cph], Alan C. Hindmarsh [ctb], Linda R. Petzold [ctb], Bill Gillespie [ctb], Charles Margossian [ctb], Metrum Research Group LLC [cre,aut,cph]

Maintainer Kyle T. Baron <kyleb@metrumrg.com>

URL http://metrumrg.com/opensourcetools.html

Copyright Metrum Research Group, LLC 2016

Description Facilitates simulation in R from hierarchical, ordinary differential equation (ODE) based models typically employed in drug development. The modeler creates a model specification file consisting of R and C++ code that is parsed, compiled, and dynamically loaded into the R session. Input data are passed in and simulated data are returned as R objects, so disk access is never required during the simulation cycle after compiling. Differential equations are solved with the DLSODA routine in ODEPACK [A. C. Hindmarsh, ``ODEPACK, A Systematized Collection of ODE Solvers," in Scientific Computing, R. S. Stepleman et al. (eds.), North-Holland, Amsterdam, 1983, pp.55-64.]. ACH and LRP are listed as authors of the DLSODA function in ODEPACK.

License CC BY-NC-ND 4.0

Depends R (>= 3.1.2), methods

Imports Rcpp (>= 0.12.1), dplyr (>= 0.4.3), magrittr (>= 1.5), lazyeval (>= 0.1.10)

LinkingTo Rcpp (>= 0.12.1), RcppArmadillo (>= 0.5.600.2.0),BH

Suggests lattice, testthat, XML

LazyLoad yes

NeedsCompilation yes

Collate 'classes.R' 'events.R' 'Ops.R' 'RcppExports.R' 'chain.R' 'utils.R' 'compile.R' 'complog.R' 'datasets.R' 'example.R' 'mrgsims.R' 'init.R' 'knobs.R' 'lockedmod.R' 'matlist.R' 'nmxml.R' 'modspec.R' 'mrgsolve.R' 'package.R' 'param.R' 'print.R' 'simtime.R' 'specdoc.R' 'update.R' 'zchain.R'

RoxygenNote 5.0.1

R topics documented:

`aboutsolver` *ODE solver.*

Description

About the `ODEPACK` differential equation solver used by `mrgsolve`.

DLSODA

```
C-----------------------------------------------------------------------
C This is the 12 November 2003 version of
C DLSODA: Livermore Solver for Ordinary Differential Equations, with
C         Automatic method switching for stiff and nonstiff problems.
C
C This version is in double precision.
C
C DLSODA solves the initial value problem for stiff or nonstiff
C systems of first order ODEs,
C     dy/dt = f(t,y) ,  or, in component form,
C     dy(i)/dt = f(i) = f(i,t,y(1),y(2),...,y(NEQ)) (i = 1,...,NEQ).
C
C This a variant version of the DLSODE package.
C It switches automatically between stiff and nonstiff methods.
C This means that the user does not have to determine whether the
C problem is stiff or not, and the solver will automatically choose the
C appropriate method.  It always starts with the nonstiff method.
C
C Authors:       Alan C. Hindmarsh
C                Center for Applied Scientific Computing, L-561
C                Lawrence Livermore National Laboratory
C                Livermore, CA 94551
C and
C                Linda R. Petzold
C                Univ. of California at Santa Barbara
C                Dept. of Computer Science
C                Santa Barbara, CA 93106
C
C References:
C 1.  Alan C. Hindmarsh,  ODEPACK, A Systematized Collection of ODE
C     Solvers, in Scientific Computing, R. S. Stepleman et al. (Eds.),
C     North-Holland, Amsterdam, 1983, pp. 55-64.
C 2.  Linda R. Petzold, Automatic Selection of Methods for Solving
C     Stiff and Nonstiff Systems of Ordinary Differential Equations,
C     Siam J. Sci. Stat. Comput. 4 (1983), pp. 136-148.
C-----------------------------------------------------------------------
```

`add.ev` *Add two events objects*

Description

Add two events objects

Usage

```
add.ev(e1, e2)
```

Arguments

e1	first events object
e2	second events object

`as.init` *Get and set model initial conditions.*

Description

Calling `init` with the model object as the first argument will return the model initial conditions as a `numericlist` object. See `numericlist` for methods to deal with `cmt_list` objects.

##' @export

Usage

```
as.init(.x, ...)

## S4 method for signature 'list'
as.init(.x, ...)

## S4 method for signature 'numeric'
as.init(.x, ...)

## S4 method for signature 'cmt_list'
as.init(.x, ...)

## S4 method for signature 'missing'
as.init(.x, ...)

## S4 method for signature '`NULL`'
as.init(.x, ...)

init(.x, ...)

## S4 method for signature 'mrgmod'
init(.x, .y = list(), ..., .pat = "*")

## S4 method for signature 'mrgsims'
init(.x, ...)

## S4 method for signature 'missing'
init(.x, ...)

## S4 method for signature 'list'
init(.x, ...)

## S4 method for signature 'ANY'
init(.x, ...)

## S4 method for signature 'cmt_list'
show(object)
```

Arguments

.x	the model object
...	passsed along
.y	list to be merged into parameter list
.pat	a regular expression (character) to be applied as a filter when printing compartments to the screen
object	to show

Details

Can be used to either get a compartment list object from a mrgmod model object or to update the compartment initial conditions in a model object. For both uses, the return value is a cmt_list object. For the former use, init is usually called to print the compartment initial conditions to the screen, but the cmt_list object can also be coreced to a list or numeric R object.

Value

an object of class cmt_list (see numericlist)

Examples

```
## example("init")
mod <- mrgsolve:::house()

init(mod)
init(mod, .pat="^C") ## may be useful for large models

class(init(mod))

init(mod)$CENT

as.list(init(mod))
as.data.frame(init(mod))
```

---

as.locked *Corece an mrgmod object to a lockedmod object.*

Description

Corece an mrgmod object to a lockedmod object.

Usage

```
as.locked(x, ...)

## S4 method for signature 'mrgmod'
as.locked(x, dllloc, dllname, src, include, ...)
```

Arguments

x	mrgmod model object
...	passed along
dllloc	directory location for the model shared object
dllname	the name of the model shared object
src	directory location of the model specification file
include	directory location for the header file

---

as.matrix.list *Coerce a list to a matrix*

Description

All elements of the list must be of length 1.

Usage

```
as.matrix.list(x, ..., nrow = 1)
```

Arguments

x	a named list, with each element of length 1
...	not used
nrow	the number of rows to make the matrix

Value

matrix with colnames set to the names of x

Author(s)

Kyle Baron

Examples

```
x <- list(a=1, b=2, c=3)
as.matrix(x,nrow=3)
```

as.packmod *Coerce an mrgmod object to packmod*

Description

Coerce an mrgmod object to packmod

Usage

```
as.packmod(x, ...)

## S4 method for signature 'mrgmod'
as.packmod(x, ...)
```

Arguments

x	mrgmod model object
...	passed along

as_bmat *Coerce R objects to block or diagonal matrices.*

Description

Coerce R objects to block or diagonal matrices.

Usage

```
as_bmat(x, ...)

## S4 method for signature 'list'
as_bmat(x, ...)

## S4 method for signature 'numeric'
as_bmat(x, pat = "*", ...)

## S4 method for signature 'data.frame'
as_bmat(x, pat = "*", ...)

## S4 method for signature 'ANY'
as_bmat(x, ...)

as_dmat(x, ...)

## S4 method for signature 'list'
as_dmat(x, ...)

## S4 method for signature 'ANY'
as_dmat(x, ...)
```

```
## S4 method for signature 'numeric'
as_dmat(x, pat = "*", ...)

## S4 method for signature 'data.frame'
as_dmat(x, pat = "*", ...)
```

Arguments

x	an R object
...	passed along
pat	regular expression, character

Value

A numeric matrix for list and numeric methods. For data.frames, a list of matrices are returned.

See Also

bmat, dmat

Examples

```
df <- data.frame(OMEGA1.1 = c(1,2),
                 OMEGA2.1 = c(11,22),
                 OMEGA2.2 = c(3,4),
                 SIGMA1.1 = 1,
                 FOO=-1)

as_bmat(df, "OMEGA")
as_dmat(df,"SIGMA")
as_dmat(df[1,],"OMEGA")
```

---

blocks *Return the code blocks from a model specification file.*

Description

Return the code blocks from a model specification file.

Usage

```
blocks(x, ...)

## S4 method for signature 'mrgmod'
blocks(x, ...)

## S4 method for signature 'character'
blocks(x, ...)
```

Arguments

x	model object or path to model specification file
...	passed along

Examples

```
mod <- mrgsolve:::house()
mod %>% blocks
mod %>% blocks(PARAM,TABLE)
```

---

bmat *Create matrices from vector input*

Description

Create matrices from vector input

Usage

```
bmat(..., correlation = FALSE, digits = -1)

BLOCK(..., correlation = FALSE, digits = -1)

cmat(..., digits = -1)

dmat(...)
```

Arguments

...	matrix data
correlation	logical; if TRUE, off diagonal elements are assumed to be correlations and converted to covariances
digits	if greater than zero, matrix is passed to signif (along with digits) prior to returning

Details

bmat makes a block matrix. cmat makes a correlation matrix. dmat makes a diagonal matrix. BLOCK is a synonym for bmat.

See Also

as_bmat

as_dmat

Examples

```
dmat(1,2,3)/10

bmat(0.5,0.01,0.2)

cmat(0.5, 0.87,0.2)
```

`carry.out` *Set the* `carry.out` *argument for* `mrgsim`.

Description

Set the `carry.out` argument for `mrgsim`.

Usage

```
carry.out(x, ...)
```

Arguments

`x`	model object
`...`	passed along

`cfile` *Return the name of the model specification file.*

Description

Return the name of the model specification file.

Usage

```
cfile(x, ...)

## S4 method for signature 'mrgmod'
cfile(x, ...)

## S4 method for signature 'lockedmod'
cfile(x, ...)
```

Arguments

`x`	model object
`...`	passed along

chain *Functions for chaining commands together.*

Description

Use these functions with chaining commands togehter with the

Details

Other functions that may be used in the chain of commands include: param, init, update,ev,limit or any other function that will take the output of the preceeding command as it's first argument.

Examples

```
mod <- mrgsolve:::house()

data(exidata)
data(exTheoph)

out <- mod %>% data_set(exTheoph) %>% mrgsim()
out <- mod %>% carry.out(evid) %>% ev(amt=100, cmt=1) %>% mrgsim()
out <- mod %>% Req(CP,RESP) %>% mrgsim()
```

cmt *Get the names of model compartments.*

Description

Get the names of model compartments.

Usage

```
cmt(x, ...)

## S4 method for signature 'mrgmod'
cmt(x, ...)
```

Arguments

x	model object
...	passed along

Examples

```
mod <- mrgsolve:::house()

cmt(mod)
```

cmtn — *Get the compartment number from a compartment name.*

Description

Get the compartment number from a compartment name.

Usage

```
cmtn(x, ...)

## S4 method for signature 'mrgmod'
cmtn(x, tag, ...)
```

Arguments

x	model object
...	passed along
tag	compartment name

Examples

```
mod <- mrgsolve:::house()
mod %>% cmtn("CENT")
```

cmt_list-class — *S4 cmt_list class*

Description

S4 cmt_list class

Details

cmt_list is a numericlist-class

complog *Functions for viewing and manipulating the compilation log.*

Description

complog displays all of the models in the compilation log. comp_forget removes models from the compilation log and attempts to unload the corresponding shared object.

Usage

```
complog(full = FALSE)

comp_forget(x)
```

Arguments

full	show a full display
x	not used

cvec *Create create character vectors.*

Description

Create create character vectors.

Usage

```
cvec(x, ...)

## S4 method for signature 'character'
cvec(x)

ch(...)

s(...)
```

Arguments

x	comma-separated quoted string (for cvec)
...	unquoted strings (for ch)

Examples

```
cvec("A,B,C")
ch(A,B,C)
s(A,B,C)
```

data_set *Set the* data *argument for* mrgsim.

Description

Set the data argument for mrgsim.

Usage

```
data_set(x, data, ...)

## S4 method for signature 'mrgmod,data.frame'
data_set(x, data, subset = TRUE,
  select = TRUE, ...)

## S4 method for signature 'mrgmod,ANY'
data_set(x, data, ...)
```

Arguments

x	model object
data	data set
...	passed along
subset	passed to dplyr::filter_
select	passed to dplyr::select_

Details

Input data sets are R data frames that can include columns with any valid name, however columns with selected names are recognized by mrgsolve and incorporated into the simulation.

ID specifies the subject ID and is required for every input data set.

When columns have the same name as parameters ($PARAM in the model specification file), the values in those columns will be used to update the corresponding parameter as the simulation progresses.

Input data set may include the following columns related to PK dosing events: time, cmt, amt, rate, ii, addl, ss. time and cmt (and ID) are required columns in the input data set. time is the observation or event time, cmt is the compartment number (see init), amt is the dosing amount, rate is the infusion rate, ii is the dosing interval, addl specifies additional doses to administer, and ss is a flag for steady state dosing. These column names operate similarly to other non-linear mixed effects modeling software, but note that (except for ID) the column names related to PK dosing must be lower case.

Only numeric data can be brought in to the problem. Any non-numeric data columns will be dropped with warning.

See exdatasets for different example data sets.

Examples

```
data <- expand.ev(ID=1:3, amt=c(10,20))

data <- expand.ev(amt=c(10,20), rate=c(1,2))
```

`design` *Set observation designs for the simulation.*

Description

Set observation designs for the simulation.

Usage

```
design(x, descol = character(0), ..., deslist = list())
```

Arguments

`x`	model object
`descol`	the `idata` column name for design assignment
`...`	`tgrid` or `tgrids` objects or `numeric` vector
`deslist`	a list of `tgrid` or `tgrids` objects or `numeric` vector to be used in place of ...

`dllname` *Return the model name.*

Description

Return the model name.

Usage

```
dllname(x, ...)

## S4 method for signature 'mrgmod'
dllname(x, ...)
```

Arguments

`x`	model object
`...`	passed along

See Also

`model`

Examples

```
mod <- mrgsolve:::house()
dllname(mod)
```

---

ev-class *S4 events class*

Description

S4 events class

Slots

`data` a data frame of events

---

events *Get model events*

Description

An accessor function for the `events` model attribute.

Events can either be specified when the model object is created (with `mrgmod`) or by updating an existing model object (with `update`).

Usage

```
events(x, ...)

## S4 method for signature 'mrgmod'
events(x, ...)

## S4 method for signature 'mrgsims'
events(x, ...)

ev(x, ...)

## S4 method for signature 'missing'
ev(evid = 1, time = 0, ID = numeric(0), cmt = 1,
  replicate = TRUE, until = NULL, ...)

## S4 method for signature 'ev'
ev(x, ...)

## S4 method for signature 'mrgmod'
ev(x, ...)

as.ev(x, ...)
```

```
## S4 method for signature 'data.frame'
as.ev(x, ...)

## S4 method for signature 'ev'
as.matrix(x, ...)

## S4 method for signature 'ev'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S4 method for signature 'ev'
show(object)
```

Arguments

x	mrgmodel object
...	passed on
evid	event ID
time	event time
ID	subject ID
cmt	compartment
replicate	logical; if TRUE, events will be replicated for each individual in ID
until	the expected maximum **observation** time for this regimen
row.names	passed to as.data.frame
optional	passed to as.data.frame
object	passed to show

Details

- Required input for creating events objects include time and cmt
- If not supplied, evid is assumed to be 1
- ID may be specified as a vector
- if replicate is TRUE (default), thenthe events regimen is replicated for each ID; otherwise, the number of event rows must match the number of IDs entered

Value

Returns a user-defined data frame of events that should be suitable for passing into lsoda. If events are stored as a data frame, events returns the data frame. If events are stored as a function that generates the data frame, events calls the function and passes return back to the user.

an object of class ev

Author(s)

Kyle Baron

Examples

```
mod <- mrgsolve:::house()
mod <- mod %>% ev(amt=1000, time=0, cmt=1)
events(mod)

loading <- ev(time=0, cmt=1, amt=1000)
maint <- ev(time=12, cmt=1, amt=500, ii=12, addl=10)
loading + maint


ev(ID=1:10, cmt=1, time=0, amt=100)
```

---

exdatasets *Example input data sets*

Description

Example input data sets

Usage

```
data(exidata)

data(extran1)

data(extran2)

data(extran3)

data(exTheoph)

data(exBoot)
```

Details

- `exidata` holds individual-level parameters and other data items, one per row
- `extran1` is a "condensed" data set
- `extran2` is a full dataset
- `extran3` is a full dataset with parameters
- `exTheoph` is the theophylline data set, ready for input into `mrgsolve`
- `exBoot` a set of bootstrap parameter estimates

Examples

```
mod <- mrgsolve:::house() %>% update(end=240) %>% Req(CP)

## Full data set
data(exTheoph)
```

```
out <- mod %>% data_set(exTheoph) %>% mrgsim
out
plot(out)
## Condensed: mrgsolve fills in the observations
data(extran1)
out <- mod %>% data_set(extran1) %>% mrgsim
out
plot(out)
## Add a parameter to the data set
stopifnot(require(dplyr))
data <- extran1 %>% distinct(ID) %>% select(ID) %>%
  mutate(CL=exp(log(1.5) + rnorm(nrow(.), 0,sqrt(0.1)))) %>%
  left_join(extran1,.)
data
out <- mod %>% data_set(data) %>% carry.out(CL) %>%  mrgsim
out
plot(out)
## idata
data(exidata)
out <- mod %>% idata_set(exidata) %>% ev(amt=100,ii=24,addl=10) %>% mrgsim
plot(out, CP~time|ID)
```

`expand.idata` *Create data sets.*

Description

Create data sets.

Usage

```
expand.idata(...)

expand.ev(...)
```

Arguments

`...`	passed to `expand.grid`

Details

An ID column is added as `1:nrow(ans)`

Examples

```
idata <- expand.idata(CL=c(1,2,3), VC=c(10,20,30))

doses <- expand.ev(amt=c(300,100), ii=c(12,24), cmt=1)
```

firstonly *This function is deprecated.*

Description

This function is deprecated.

Usage

```
firstonly()
```

idata *Create an idata data set*

Description

Create an idata data set

Usage

```
idata(..., KEEP.OUT.ATTRS = FALSE, stringsAsFactors = FALSE)
```

Arguments

`...`	passed to expand.grid
`KEEP.OUT.ATTRS`	passed to expand.grid
`stringsAsFactors`	passed to expand.grid

idata_set *Set the* idata *argument for* mrgsim.

Description

Set the idata argument for mrgsim.

Usage

```
idata_set(x, data, ...)

## S4 method for signature 'mrgmod,data.frame'
idata_set(x, data, subset = TRUE,
  select = TRUE, ...)

## S4 method for signature 'mrgmod,ANY'
idata_set(x, data, ...)
```

Arguments

x	model object
data	a data set coercable to data.frame
...	passed along
subset	passed to dplyr::filter_
select	passed to dplyr::select_

---

installed_models *Get path to example models*

Description

Get path to example models

Usage

```
installed_models()
```

---

knobs *Run sensitivity analysis on model settings*

Description

Knobs can be parameter values or PK dosing items (e.g. amt). By design, all combinations of specified knob/values are simulated.

Usage

```
knobs(x, y, ...)

## S4 method for signature 'mrgmod,missing'
knobs(x, ..., carry.out = character(0),
  drop = c("default", "none", "all"), update = list())

## S4 method for signature 'mrgmod,batch_mrgsims'
knobs(x, y, ...)

## S4 method for signature 'batch_mrgsims'
as.data.frame(x, row.names = NULL,
  optional = FALSE, ...)

## S4 method for signature 'batch_mrgsims'
as.matrix(x, y, ...)

batch(x, ...)

moving(x, ...)
```

```
## S4 method for signature 'batch_mrgsims'
batch(x, ...)

## S4 method for signature 'batch_mrgsims,ANY'
knobs(x, y, ...)

## S4 method for signature 'batch_mrgsims'
moving(x, ...)

## S4 method for signature 'batch_mrgsims'
show(object)
```

Arguments

x	the model object
y	batch_mrgsims object
...	knobs: named numeric vectors that identify knob names and knob values for a batch run. See details.
carry.out	passed to mrgsim
drop	defines which knobs to drop in the matrix of simulated data; with drop = "none", the values of all knobs appear in the simulated data matrix; with drop = "all", no knob names appear in the simulated data matrix; when drop is "default", selected non-moving columns related to PK dosing are dropped: cmt, time, addl, ii, ss, evid. In every case, the simulation run settings can be retreived with the batch method for the batch_mrgsims output object.
update	a list of arguments that are passed to update prior to running the knobs
row.names	passed to as.data.frame
optional	passed to as.data.frame
object	passed to show

Details

Valid knob names include: any parameter name (in param(mod)), time variables (start, end, delta), PK dosing items (amt, ii, rate, and others ...), and solver settings (atol, hmax, etc...).

Value

An object of class batch_mrgsims. Most methods for mrgsims objects also work on batch_mrgsims object.

Examples

```
## example("knobs")

mod <- mrgsolve:::house(end=72)

events <- ev(amt=1000, cmt=1, addl=3, ii=12)

out <- mod %>% ev(events) %>% knobs(CL=c(1,2,3))
plot(out)

out
```

```
moving(out)
batch(out)


out <- mod %>% ev(events) %>% knobs(CL=c(1,2,3), VC=c(5,20,50))
plot(out)
plot(out,CP~.)
plot(out, CP~time|VC, groups=CL, lty=2)

out <- knobs(mod, amt=c(100,300,500), cmt=1,time=0)
plot(out)

out <- mod %>% knobs(amt=c(100,300), CL=c(1,3),VC=c(5,20), cmt=1, time=0)
plot(out)
plot(out, CP~.)
plot(out, CP~time|CL*VC, groups=Amt)

out <- knobs(mod, CL=c(1,2,3), drop="all")
out

out <- knobs(mod, CL=c(1,2,3), drop="none")
out
```

`label` *Label simulation output.*

Description

Attaches a named column to the simulation output with a single numeric value

Usage

```
label(x, ...)

## S4 method for signature 'mrgsims'
label(x, ...)
```

Arguments

`x`	mrgsims object
`...`	name=value pairs; value must be numeric.

`lctran` *Convert select upper case column names to lower case to conform to* `mrgsolve` *data expectations.*

Description

Convert select upper case column names to lower case to conform to `mrgsolve` data expectations.

Usage

```
lctran(data)
```

Arguments

data	an nmtran-like data frame

Details

Columns that will be renamed with lower case versions: AMT, II, SS, CMT, ADDL, RATE, EVID, TIME. If a lower case version of these names exist in the data set, the column will not be renamed.

Value

A data.frame with renamed columns.

---

limit *Limit the scope of simulated output*

Description

Limit the scope of simulated output

Usage

```
limit(x, ...)

## S4 method for signature 'mrgsims'
limit(x, subset, select = TRUE, ...)
```

Arguments

x	mrgsims or batch mrgsims object
...	passed along
subset	rows to keep
select	columns to keep

loadso *Load the model shared object.*

Description

Load the model shared object.

Usage

```
loadso(x, ...)

## S4 method for signature 'mrgmod'
loadso(x, ...)
```

Arguments

x	the model object
...	passed along

lower2matrix *Create a square numeric matrix from the lower-triangular elements*

Description

Create a square numeric matrix from the lower-triangular elements

Usage

```
lower2matrix(x, prefix = NULL)
```

Arguments

x	numeric data
prefix	used to generate column names

Value

a square symmetric numeric matrix with column names

matlist *Various functions for and properties of* matlist *objects.*

Description

Various functions for and properties of matlist objects.

Usage

```
zero.re(.x, ...)

## S4 method for signature 'mrgmod'
zero.re(.x, ...)

drop.re(.x, ...)

## S4 method for signature 'mrgmod'
drop.re(.x, ...)

## S4 method for signature 'matlist'
as.list(x, ...)

## S4 method for signature 'matlist'
as.matrix(x, ...)

## S4 method for signature 'matlist'
names(x)

## S4 method for signature 'matlist'
length(x)

## S4 method for signature 'matlist'
dim(x)

## S4 method for signature 'matlist'
nrow(x)

## S4 method for signature 'matlist'
show(object)
```

Arguments

.x	a matlist object
...	passed along
x	a matlist object
object	passed to showmatlist

matlist-class *S4 class matlist.*

Description

S4 class matlist.

mcRNG *Set RNG to use L'Ecuyer-CMRG.*

Description

Set RNG to use L'Ecuyer-CMRG.

Usage

```
mcRNG()
```

merge.list *Merge two lists*

Description

Merge two lists

Usage

```
## S3 method for class 'list'
merge(x, y, ..., strict = TRUE, warn = TRUE,
  context = "object", wild = "...")
```

Arguments

x	the original list
y	the new list for merging
...	not used
strict	logical indicating whether or not new items should be allowed in the list upon merging.
warn	issue warning if nothing found to update
context	description of usage context
wild	wild-card name; see details

Details

Wild-card names (wild) are always retained in x and are brought along from y only when !strict.

mod *Return the model object.*

Description

Return the model object.

Usage

```
mod(x, ...)

## S4 method for signature 'mrgsims'
mod(x, ...)
```

Arguments

x	mrgsims object
...	passed along

model *Return the model name.*

Description

Return the model name.

Usage

```
model(x, ...)

## S4 method for signature 'mrgmod'
model(x, ...)
```

Arguments

x	model object
...	passed along

Examples

```
mod <- mrgsolve:::house()
model(mod)
```

modelparse *Parse model specification text*

Description

Parse model specification text

Usage

```
modelparse(txt, split = FALSE, ...)
```

Arguments

txt	model specification text
split	logical
...	arguments passed along

modelspec *Model Specification File*

Description

Model Specification File

CODE BLOCKS

The following list gives the names and usage for different code blocks in the mrgsolve model specification file. All code blocks start with $.

- $PROB include a description of the model; the value supplied here must resolve to a valid character string when parsed by the R parser
- $PARAM name/value pairs for parameters; these name/value pairs for inits must be valid input for `list()` when parsed. Parameters listed in $PARAM are numeric variables that can be used in $MAIN, $ODE, $TABLE, or $CAPTURE and, importantly, can be updated from R without recompiling the model.
- $FIXED include name/value pairs exactly as you would in $PARAM. However, these values are declared as constant doubles in the C++ code and are not able to be updated from the R side. This is usually only implemented when there are a very large number of parameters, some of which are never updated. Moving these parameters to $FIXED will decrease the "active" number of parameters resulting in minor efficiency gains during simulation but much better clarity when looking at the parameter list with `param`. A call to `param` will only show name/value pairs declared in $PARAM. Call `allparam` to get all name/values declared in $PARAM and $FIXED.
- $INIT name/value pars for compartments / initial conditions; these name/value pairs for inits must be valid input for `list()` when parsed.
- $CMT an alternative to $INIT; specify compartment names only; enter unquoted strings; may be separated by whitespace, comma, or newline. Initial conditions are assumed to be zero and result is stored and accessed as $INIT would be.

- $SET run settings, as name/value pairs; these name/value pairs must be valid input for list() when parsed; value must be a scalar numeric or logical value; modify settings for simulation start or end time, simulation time interval, solver settings (e.g. tolerances or max step size), number of significant digits in output, etc.
- $GLOBAL code that will get executed upon compile, outside of any other block; must be valid C++ code, may use C++ VARIABLES and MACROS provided by mrgsolve
- $MAIN main function (like NONMEM $PK); must be valid C++ code.
- $ODE differential equations (like NONMEM $DES); must be valid C++ code, may use C++ VARIABLES and MACROS provided by mrgsolve
- $TABLE table function (like NONMEM $ERROR + $TABLE); must be valid C++ code, may use C++ VARIABLES and MACROS provided by mrgsolve
- $ADVAN2 implement one-compartment PK model with first-order absorption, where compartment amounts are calculated from algebraic equations rather than ODEs. Initialize two compartments in $CMT, where the first compartment is the dosing depot. Set pred_CL, prec_VC, and pred_KA in $MAIN.
- $ADVAN4 implement two-compartment PK model with first-order absorption, where compartment amounts are calculated from algebraic equations rather than ODEs. Initialize three compartments in $CMT, where the first compartment is the dosing depot and the second is the central compartment. Set pred_CL, pred_V2, pred_Q, pred_V3, and pred_KA in $MAIN.
- $NMXML read in NONMEM modeling results; provide either project directory with run number or the full path to the .xml results file. Will read in final THETAs and add those to the parameter list. See nmxml for options that can be set in $NMXML. To load model elements directly from a NONMEM run with $NMXML, it is required to have the XML package installed in your packages library.
- $THETA numeric values that will be added to the parameter list.
- $OMEGA a matrix for subject-level random effects. See modMATRIX for options that can be set in $OMEGA.
- $SIGMA a matrix for residual unexplained variability. See modMATRIX for options that can be set in $SIGMA.
- $ENV environment variables; referencing environment variables is currently only implemented in $NMXML.
- $CMTN To get the number of specific compartments (CMT), specify names here as unquoted comma-separated list; they will be made available in the C++ code as _N_CMT.
- $CAPTURE names of C++ variables to capture in simulated output; separate by comma, space or newline. Each variable listed in this block will be passed to the capture macro discussed below.

Variables

- parameters are accessible by name as specified in param(mod); readonly except in main block iniated by BEGIN_mainW; TYPE: double
- amount in a compartment is accessible by compartment name as specified in init(mod); readonly. TYPE: double
- for the nth compartment CMT, set the bioavailability fraction to F_CMT. This is an alias to _F(n) (see below).
- for the nth compartment CMT, set the dosing lag time to ALAG_CMT. This is an alias to _ALAG(n) (see below).

- for the nth compartment CMT, set the infusion rate to R_CMT. This is an alias to _R(n) (see below).
- for the nth compartment CMT, set the infusion duration to D_CMT. This is an alias to _D(n) (see below).
- the initial amount in compartment CMT is accessible by CMT_0. TYPE: double
- EVID: event id (0: observation, 1: dosing event; 2: other type event; 3: reset; 8: replace). TYPE: int
- TIME: the time value of current record. TYPE: double
- SOLVERTIME: the current value of time within the ode solver. TYPE: double
- NEWIND: new individual flag (0: first record of data set; 1: first record of current individual; 2: subsequent records after and NEWIND 1 record). TYPE: int
- ID: current subject ID. TYPE: double.

Macros

- _F(n) bioavailability fraction for nth compartment; may be read or written to in main code block. See F_CMT above.
- _ALAG(n) dosing event (evid=1) lag time. See ALAG_CMT above.
- _R(n) infusion rate into the nth compartment. See R_CMT above.
- _D(n) infusion duration into the nth compartment. See D_CMT above.
- table(name) = value; save value to the output simulation matrix with name text; readonly. Only available in table code block.
- ETA(n) the value of the nth ETA drawn from normal distribution with mean zero and variance-covariance matrix omega; readonly.
- EPS(n) the value of the nth EPS drawn from normal distribution with mean zero and variance-covariance matrix sigma; readonly.
- SYSTEMSTOPADVANCING() stop advancing the problem for current individual; all compartments will retain the same value for all subsequent simulation times, and the table block will not be called. System will always be advancing when a new individual problem is started.
- SYSTEMNOTADAVANCING() returns bool (logical) if system is not currently advancing.
- _nEQ the number of compartments / differential equations in the problem
- capture(var) writes the C++ variable var to the table of simulated output.

User-defined variables

Users may define any valid C++ variable in the $GLOBAL code block. These variables will be global to the problem and may be read or written to in any other code block. User may also declare and use the C++ variables double, int, and bool in the $MAIN, $ODE, and $TABLE code blocks. These user variables will be declared in the global environment so that they will also be global to the problem. To use a varible that is local to one of these code blocks, use type localdouble, localint, and localbool for double, int, and bool, respectively, or create your own typedefs to avoid declaring the variables as double, int or bool.

Comments

- a double forward slash (//) will comment to the end of the line in any code block. This is the preferred commenting mechanism.

Reserved words

A listing of reserved words can be printed to the R console with the function reserved.

Examples

```
code <- '

$PROB 1-cmt model with first order absorption

$SET delta =0.1, end=120, verbose=TRUE
preclean=TRUE

$OMEGA block=TRUE
0.1
0.001 0.3

$OMEGA corr=TRUE
0.1
0.67 0.2

$SIGMA
1 0.1

$PARAM
CL=1, VC=10, KA=0.1

$INIT GUT=0
CENT=1

$GLOBAL
bool cool=true;
#define KE (CL/VC)

$ODE
double CP = CENT/VC;

dxdt_GUT = -KA*GUT;
dxdt_CENT = KA*GUT - KE*CENT;

$TABLE

table(ke) = CL/VC;

capture(CP);

'



mod <- mread(code=code, project=tempdir())

smat(mod)
omat(mod)
as.matrix(omat(mod))
```

```
see(mod)


code <- '
$PARAM CL = 1.5, VC=35, KA=1.2

$CMT DEPOT CENT

$ADVAN2

$MAIN
pred_CL = CL;
pred_VC = VC;
pred_KA = KA;
'




mod <- mread(code=code, "ADVAN2", tempdir())

mod %>%
  ev(amt=100,ii=24, addl=9) %>%
  mrgsim(delta=0.1,end=240) %>%
  plot
```

---

modMATRIX *Create a matrix.*

Description

Create a matrix.

Usage

```
modMATRIX(x, name = "", use = TRUE, block = FALSE, correlation = FALSE,
  digits = -1, ...)
```

Arguments

x	data for building the matrix. Data in x are assumed to be on-diagonal elements if block is FALSE and lower-triangular elements if block is TRUE
name	name
use	logical; if FALSE, all matrix elements are set to 0
block	logical; if TRUE, try to make a block matrix; diagonal otherwise
correlation	logical; if TRUE, off diagonal elements are assumed to be correlations and converted to covariances; if correlation is TRUE, then block is set to TRUE
digits	if value of this argument is greater than zero, the matrix is passed to signif (along with digits) prior to returning
...	passed along

Examples

```
modMATRIX("1 2.2 333")
modMATRIX("1 1.1 2.2", block=TRUE)
modMATRIX("23 234 234 5234", use=FALSE)

ans <- modMATRIX("1.1 0.657 2.2", correlation=TRUE, block=TRUE)
ans
cov2cor(ans)
```

`mread` *Read a model specification file*

Description

Read a model specification file

Usage

```
mread(model = character(0), project = getwd(), code = NULL, udll = TRUE,
  ignore.stdout = TRUE, raw = FALSE, compile = TRUE, audit = FALSE,
  quiet = getOption("mrgsolve_mread_quiet", FALSE), check.bounds = FALSE,
  warn = TRUE, soloc = tempdir(), preclean = FALSE, ...)
```

Arguments

`model`	model name
`project`	working directory
`code`	a character string with model specification code
`udll`	use unique name for shared object
`ignore.stdout`	passed to system call for compiling model
`raw`	if TRUE, return a list of raw output
`compile`	try to compile the model and load the shared object
`audit`	check the model specification file for errors
`quiet`	don't print messages when compiling
`check.bounds`	check boundaries of parameter list
`warn`	logical; if `TRUE`, print warning messages that may arise
`soloc`	directory where model shared object is stored
`preclean`	logical; if `TRUE`, compilation artifacts are cleaned up first
`...`	passed along

Examples

```
code <- '
$PARAM CL = 1, VC = 5
$CMT CENT
$ODE dxdt_CENT = -(CL/VC)*CENT;
'

mod <- mread(code=code)

mod

mod %>% init(CENT=1000) %>% mrgsim %>% plot
```

`mrgindata` *Prepare input data.frame or matrix*

Description

Prepare input data.frame or matrix

Usage

```
mrgindata(x, ...)

## S3 method for class 'data.frame'
mrgindata(x, m = NULL, verbose = FALSE,
  quiet = FALSE, ...)

## S3 method for class 'matrix'
mrgindata(x, verbose = FALSE, ...)
```

Arguments

`x`	data.frame or matrix
`...`	additional arguments
`m`	object that inherits from mrgmod
`verbose`	logical
`quiet`	if `TRUE`, messages will be suppressed

Value

a matrix with non-numeric columns dropped; if `x` is a data.frame with character `cmt` column comprised of valid compartment names and `m` is a model object, the `cmt` column will be converted to the corresponding compartment number.

mrgmod-class *S4 class for mrgsolve model object*

Description

S4 class for mrgsolve model object

Slots

`model` model name <character>

`project` working directory; must be writeable with no spaces <character>

`start` simulation start time <numeric>

`end` simulation end time <numeric>

`delta` simulation time interval <numeric>

`add` additional simulation times <numeric-vector>

`param` parameter_list

`fixed` a parameter_list of fixed value parameters; these are not updatable from R

`init` cmt_list

`events` events object

`digits` significant digits in simulated output; negative integer means ignore <numeric>

`hmin` passed to dlsoda <numeric>

`hmax` passed to dlsoda <numeric>

`mxhnil` passed to dlsoda <numeric>

`ixpr` passed to dlsoda <numeric>

`atol` passed to dlsoda <numeric>

`rtol` passed to dlsoda <numeric>

`maxsteps` passed to dlsoda <numeric>

`preclean` passed to R CMD SHLIB during compilation <logical>

`verbose` print run information to screen <logical>

`tscale` used to scale time in simulated output <numeric>

`omega` matlist for simulating individual-level random effects

`sigma` matlist for simulating residual error variates

`func` character vector of length 2 specifying symbol name and package for ode function; this is not normally set by the user

`init_fun` character vector of length 2 specifying symbol name and package for main function; this is not normally set by the user

`table_fun` character vector of length 2 specifying symbol name and package for table function; this is not normally set by the user

`args` <list> of arguments to be passed to mrgsim

`advan` either 2, 4, or 13 <numeric>

`request` vector of compartments to request <character>

`soloc` directory path for storing the model shared object <character>

`mindt` minimum time between simulation records <numeric>

Notes

- Spaces in paths (`project` and `soloc`) are prohibited.

`mrgsim` *Simulate from a model object.*

Description

This function sets up the simulation run from data stored in the model object as well as arguments passed in. Note that there are many non-formal arguments to this function that can be used to customize the simulation run and it's output.

Usage

```
mrgsim(x, ...)

## S4 method for signature 'mrgmod'
mrgsim(x, data = NULL, idata = NULL, nid = 1, ...)

## S4 method for signature 'mrgsims'
mrgsim(x, ...)
```

Arguments

`x`	the model objects
`...`	passed to `update`
`data`	NMTRAN-like data set
`idata`	a matrix or data frame of model parameters, one parameter per row
`nid`	integer number of individuals to simulate; only used if idata and data are missing

Details

- Both `data` and `idata` will be coreced to numeric matrix
- `carry.out` can be used to insert data columns into the output data set. This is partially dependent on the nature of the data brought into the problem.
- When using `data` and `idata` together, an error is generated if an ID occurs in `data` but not `idata`. Also, when looking up data in `idata`, ID in `idata` is assumed to be uniquely keyed to ID in `data`. No error is generated if ID is duplicated in `data`; parameters will be used from the first occurrence found in `idata`.
- `carry.out`: `idata` is assumed to be individual-level and variables that are carried from `idata` are repeated throughout the invidivual's simulated data. Variables carried from `data` are carried via last-observation carry forward. `NA` is returned from observations that are inserted into simulated output that occur prior to the first record in `data`.

Value

an object of class `mrgsims`

Additional arguments

- `mtime` numeric vector of times where the model is evaluated (with solver reset), but results are not included in simulated output
- `trequest` a vector of names of table data items to take in simulated ouput
- `Trequest` same as `trequest`, except that when Trequest is specified, all model compartments (in `request`) are dropped; this is just a shorter syntax for saying request="", trequest="name1,name2"
- `Request` a vector of compartment or table names to take in simulated output; if this is specified, `request`, `trequest`, and `Trequest` are ignored
- `obsonly` omit records with `evid` != 0 from simulated output
- `obsaug` logical; when `TRUE` and a full data set is used, the simulated output is augmented with an observation at each time in `stime`(). When using `obsaug`, a flag indicating augmented observations can be requested by including `a.u.g` in `carry.out`
- `recsort` Default value is 1. Possible values are 1,2,3,4: 1 and 2 put doses in a data set after padded observations at the same time; 3 and 4 put those doses before padded observations at the same time. 2 and 4 will put doses scheduled through `addl` after observations at the same time; 1 and 3 put doses scheduled through `addl` before observations at the same time. `recsort` will not change the order of your input data set if both doses and observations are given.
- `filbak` For each `ID`, carry the first record `data` backward to start of the simulation
- `mindt` The minimum allowable difference between `tto` and `tfrom`; if `(tto-tfrom)/denom` < `mindt` where `denom` is `tfrom` if `tfrom` is > 0 and 1 otherwise, then `tto` is set to `tfrom`. This adjustment is usually helpful when there are infusions in the problem and the end of an infusion is very close to an observation time. When this is the case, the solver may fail with the message `DLSODA- TOUT(=R1) too close to T(=R2) to start integration`. When `mindt==0` the adjustment is not attempted.

Examples

```
## example("mrgsim")

mod <- mrgsolve:::house() %>%  ev(amt=1000, cmt=1)
out <- mrgsim(mod)
plot(out)

out <- mrgsim(mod, end=22)
out


data(exTheoph)

out <- mrgsim(mod, data=exTheoph)
out

out <- mrgsim(mod, data=exTheoph, obsonly=TRUE)
out

out <- mrgsim(mod, data=exTheoph, obsaug=TRUE, carry.out="a.u.g")
out

out <- mrgsim(mod, req="CENT")
out
```

```
out <- mrgsim(mod, Req="CP,RESP")
out
```

---

mrgsims *Methods for working with* mrgsims *objects.*

Description

These methods help the user view simulation output and extract simulated data to work with further. The methods listed here for the most part have generics defined by R or other R packages. See the seealso section for other methods defined by mrgsolve that have their own documentation pages.

Usage

```
## S4 method for signature 'mrgsims'
x$name

## S4 method for signature 'mrgsims'
tail(x, ...)

## S4 method for signature 'mrgsims'
head(x, ...)

## S4 method for signature 'mrgsims'
dim(x)

## S4 method for signature 'mrgsims'
names(x)

## S4 method for signature 'mrgsims'
as.data.frame(x, row.names = NULL, optional = FALSE,
  ...)

## S3 method for class 'mrgsims'
as.tbl(x, ...)

## S3 method for class 'mrgsims'
filter_(.data, ..., .dots)

## S3 method for class 'mrgsims'
group_by_(.data, ..., .dots, add = FALSE)

## S3 method for class 'mrgsims'
mutate_(.data, ..., .dots)

summarise.each(.data, funs, ...)
```

```
## S3 method for class 'mrgsims'
summarise_(.data, ..., .dots)

## S3 method for class 'mrgsims'
do_(.data, ..., .dots)

## S3 method for class 'mrgsims'
select_(.data, ..., .dots)

## S3 method for class 'mrgsims'
slice_(.data, ...)

## S4 method for signature 'mrgsims'
as.matrix(x, ...)

## S4 method for signature 'mrgsims'
subset(x, ...)

## S4 method for signature 'mrgsims'
summary(object, ...)

## S4 method for signature 'mrgsims'
show(object)
```

Arguments

x	mrgsims object
name	name of column of simulated output to retain
...	passed to other functions
row.names	passed to as.data.frame
optional	passed to as.data.frame
.data	passed to various dplyr functions
.dots	passed to various dplyr functions
add	passed to dplyr::group_by_
funs	passed to dplyr::summarise_each
object	passed to show

Details

Most methods should behave as expected according to other method commonly used in R (e.g. head, tail, as.data.frame, etc ...)

- subset coreces simulated output to data.frame and passes to subset.data.frame
- $ selects a column in the simulated data and returns numeric
- head see head.matrix; returns simulated data
- tail see tail.matrix; returns simulated data
- dim, nrow, ncol returns dimensions, number of rows, and number of columns in simulated data
- as.data.frame coreces simulated data to data.frame and returns the data.frame

- `as.matrix` returns matrix of simulated data
- `as.tbl` coreces simulated to `tbl_df`; requires `dplyr`
- `summary` coreces simulated data to data.frame and passes to `summary.data.frame`
- `plot` plots simulated data; see `plot_mrgsims`

See Also

mod request variables label limit stime

Examples

```
## example("mrgsims")

mod <- mrgsolve:::house() %>% init(GUT=100)

out <- mrgsim(mod)
class(out)

out
head(out)
tail(out)

mod(out)

dim(out)
names(out)

mat <- as.matrix(out)
df <- as.data.frame(out)

df <- subset(out, time < 12) ## a data frame
out$CP

plot(out)
plot(out, CP~.)
plot(out, CP+RESP~time, scales="same", xlab="Time", main="Model sims")

out <- label(out, DOSE=100)
head(out)
```

---

mrgsims-class *S4 class for mrgsolve simulation output*

Description

S4 class for mrgsolve simulation output

Slots

`request` character vector of compartments requested in simulated output

`outnames` character vector of column names in simulated output coming from table step

`data` matrix of simulated data

`mod` the mrgmod model object

mrgsolve *mrgsolve*

Description

mrgsolve is an R package maintained under the auspices of Metrum Research Group, LLC, that facilitates simulation from models based on systems of ordinary differential equations (ODE) that are typically employed for understanding pharmacokinetics, pharmacodynamics, and systems biology and pharmacology. mrgsovle consists of computer code written in the R and C++ languages, providing an interface to the DLSODA differential equation solver (written in FORTRAN) provided through ODEPACK - A Systematized Collection of ODE Solvers.

Help with model specification file

See this help page: `modelspec`.

Example models

See `mrgsolve_example` to export example models into your own, writeable project directory.

Input data sets

See `data_set` for help creating input data sets. See `exdatasets` for example input data sets.

Package help

- Package index, including a listing of all functions
- Macros and variables in the model specification file: `modelspec`
- Reserved words in `mrgsolve`: `reserved`

About the model object

The model object has class `mrgmod`.

Handling simulated output

See `mrgsims` for methods to use with simulated output.

Operations between mrgsolve objects

See `mrgsolve_Ops` for details.

About the solver used by `mrgsolve`

See: `aboutsolver`

Examples

```
## example("mrgsolve")

mod <- mrgsolve:::house(delta=0.1)  %>% param(CL=0.5)

events <-  ev(amt=1000, cmt=1, addl=5, ii=24)

mod
cfile(mod)
see(mod)
events
stime(mod)

param(mod)
init(mod)

out <- mod %>% ev(events) %>% mrgsim(end=168)
out <- label(out, TRT=1)

out
head(out)
tail(out)
dim(out)

mod(out)
param(out)

plot(out, GUT+CP~.)

sims <- as.data.frame(out)

t72 <- subset(sims, time==72)
str(t72)



idata <- data.frame(ID=c(1,2,3), CL=c(0.5,1,2),VC=12)
out <- mod %>% ev(events) %>% mrgsim(end=168, idata=idata, req="")
plot(out)

out <- mod %>% ev(events) %>% mrgsim(carry.out="amt,evid,cmt,CL")
head(out)


out <- mod %>% ev() %>% knobs(CL=c(0.5, 1,2), amt=c(100,300,1000), cmt=1,end=48)
plot(out, CP~., scales="same")
plot(out, RESP+CP~time|CL, groups=Amt)


ev1 <- ev(amt=500, cmt=2,rate=10)
ev2 <- ev(amt=100, cmt=1, time=54, ii=8, addl=10)
events <- ev1+ev2
events

out <- mod %>% ev(ev1+ev2) %>% mrgsim(end=180, req="")
```

```
plot(out)



## Full NMTRAN data set
data(exTheoph)
head(exTheoph)

mod <- mrgsolve:::house(delta=0.1)

out <- mod %>% data_set(exTheoph) %>% mrgsim

plot(out,CP~time|factor(ID),type='b', scales="same")


## "Condensed" data set
data(extran1)
extran1

out <- mod %>% data_set(extran1) %>% mrgsim(end=200)

plot(out,CP~time|factor(ID))


## idata
data(exidata)
exidata

out <- mod %>% ev(amt=1000, cmt=1) %>% idata_set(exidata) %>%  mrgsim(end=72)

plot(out, CP~., as="log10")



code <- '
$PARAM CL=1, VC=10, KA=1.1
$INIT GUT=0, CENT=0
$SET end=48, delta=0.25

$MAIN
double CLi = CL*exp(ETA(1));
double VCi = VC*exp(ETA(2));
double ke = CLi/VCi;

$OMEGA corr=TRUE
0.04 0.6 0.09

$ODE
dxdt_GUT = -KA*GUT;
dxdt_CENT = KA*GUT - ke*CENT;

$TABLE
table(CP) = CENT/VC;

'
```

```
mod <- mread(code=code) %>% ev(amt=1000, cmt=1, addl=2, ii=8)

out <- mod %>% mrgsim

out

plot(out)
```

`mrgsolve_example` *Extract example model from system library*

Description

Extract example model from system library

Usage

```
mrgsolve_example(model = c("pkExample", "pkpdExample", "firstmodeExample",
  "viralExample", "popExample"), project = getwd(), overwrite = FALSE,
  quiet = FALSE, ...)
```

Arguments

`model`	name of model
`project`	working directory
`overwrite`	passed to file.copy
`quiet`	don't print any status messages to the screen
`...`	additional arguments

Examples

```
## example("mrgsolve_example", package="mrgsolve")

mrgsolve_example("pkpdExample", project=getwd())

mod <- mread("pkpdExample", project=getwd()) %>% ev(amt=1000, time=0, cmt=1)

see(mod)

out <- mod %>% mrgsim(end=48,delta=0.1)

out

plot(out)

out <- mod %>%
  ev(amt=1000, ii=24, cmt=1, addl=10)  %>%
  mrgsim(end=300)
```

```
plot(out)
plot(out, CP~time)
```

`mrgsolve_models` *Get the package models directory.*

Description

Get the package models directory.

Usage

```
mrgsolve_models()
```

`mrgsolve_Ops` *Operations for ev objects.*

Description

Operations for ev objects.

Usage

```
## S4 method for signature 'ev,ev'
e1 + e2

## S4 method for signature 'mrgmod,ev'
e1 + e2

## S4 method for signature 'mrgmod,data.frame'
e1 + e2

## S4 method for signature 'mrgmod,cmt_list'
e1 + e2

## S4 method for signature 'mrgmod,parameter_list'
e1 + e2

e1 %then% e2

## S4 method for signature 'ev,ev'
e1 %then% e2

## S4 method for signature 'ev,numeric'
e1 + e2
```

Arguments

e1	object on left hand side of operator (lhs)
e2	object on right hand side of operator (rhs)

Details

All operations involving mrgmod objects have been deprecated.

---

mrgsolve_template *Create model specification file from template*

Description

Create model specification file from template

Usage

```
mrgsolve_template(model = "template", project = getwd(),
  writeable = FALSE, overwrite = FALSE)
```

Arguments

model	name of the model to create
project	working directory
writeable	logical; if TRUE, parameters may be overwritten in the main block
overwrite	logical; if TRUE, an existing file with same stem will be overwritten

---

mvgauss *Simulate from a multivariate normal distribution with mean zero.*

Description

Simulate from a multivariate normal distribution with mean zero.

Usage

```
mvgauss(mat, n = 10, seed = NULL)
```

Arguments

mat	a positive-definite matrix
n	number of variates to simulate
seed	if not null, passed to set.seed

neq *Return the number of compartments / equations.*

Description

Return the number of compartments / equations.

Usage

```
neq(x, ...)

## S4 method for signature 'mrgmod'
neq(x, ...)
```

Arguments

x	model object
...	passed along

Examples

```
mod <- mrgsolve:::house()
neq(mod)
```

nmxml *Get THETA, OMEGA and SIGMA from a completed NONMEM run*

Description

Get THETA, OMEGA and SIGMA from a completed NONMEM run

Usage

```
nmxml(run = numeric(0), project = character(0), file = character(0),
  theta = TRUE, omega = FALSE, sigma = FALSE, olabels = NULL,
  slabels = NULL, oprefix = "", sprefix = "", tname = "THETA",
  oname = "...", sname = "...", ...)
```

Arguments

run	run number
project	project directory
file	the complete path to the run.xml file
theta	logical; if TRUE, the $THETA vector is returned
omega	logical; if TRUE, the $OMEGA matrix is returned
sigma	logical; if TRUE, the $SIGMA matrix is returned
olabels	labels for $OMEGA

slabels	labels for $SIGMA
oprefix	prefix for $OMEGA labels
sprefix	prefix for $SIGMA labels
tname	name for $THETA
oname	name for $OMEGA
sname	name for $SIGMA
...	passed along

Details

If run and project are supplied, the .xml file is assumed to be located in run.xml, in directory run off the project directory. If file is supplied, run and project arguments are ignored.

Value

a list with theta, omega and sigma elements, depending on what was requested

---

numeric2diag *Create a diagonal numeric matrix from diagonal elements*

Description

Create a diagonal numeric matrix from diagonal elements

Usage

```
numeric2diag(x, prefix = NULL)
```

Arguments

x	numeric data
prefix	used to generate column names

Value

a numeric diagonal matrix

numericlist *Methods for numericlist*

Description

These methods can be used to corece `param` and `init` objects into common `R` data structures.

Usage

```
## S4 method for signature 'numericlist'
as.list(x, ...)

## S4 method for signature 'numericlist'
as.numeric(x)

## S4 method for signature 'numericlist'
as.data.frame(x, row.names = NULL, optional = FALSE,
  ...)

## S4 method for signature 'numericlist'
length(x)

## S4 method for signature 'numericlist'
names(x)

## S4 method for signature 'numericlist'
x$name

## S4 method for signature 'numericlist'
x[i, j, ..., drop = TRUE]
```

Arguments

`x`	object
`...`	passed along to other methods
`row.names`	passed to `as.data.frame`
`optional`	passed to `as.data.frame`
`name`	column to take; used with `$`
`i`	elements to keep
`j`	not used
`drop`	not used

Examples

```
## Not run:
  mod <- mrgmod(...)
  as.list(param(mod))
  as.numeric(init(mod))

## End(Not run)
```

numericlist-class *S4 class numeric list.*

Description

S4 class numeric list.

Arguments

data	list of data
pattern	character of length 1 containing regular expression to be used as a filter when printing data to the console

obsaug *Set the* obsaug *argument for* mrgsim.

Description

Set the obsaug argument for mrgsim.

Usage

```
obsaug(x, value = TRUE, ...)
```

Arguments

x	model object
value	the value for obsaug
...	passed along

obsonly *Set the* obsonly *argument for* mrgsim.

Description

Set the obsonly argument for mrgsim.

Usage

```
obsonly(x, value = TRUE, ...)
```

Arguments

x	model object
value	the value for obsonly
...	passed along

omega *Manipulate* OMEGA *matrices.*

Description

The primary function is omat that can be used to both get the $OMEGA matrices out of a model object and to update $OMEGA matrices in a model object.

Usage

```
omat(.x, ...)

## S4 method for signature 'missing'
omat(.x, ...)

## S4 method for signature 'matrix'
omat(.x, ..., labels = list())

## S4 method for signature 'list'
omat(.x, ...)

## S4 method for signature 'omegalist'
omat(.x, ...)

## S4 method for signature 'mrgmod'
omat(.x, ..., make = FALSE, strict = TRUE)

## S4 method for signature 'mrgsims'
omat(.x, make = FALSE, ...)
```

Arguments

.x	a matrix, list of matrices or matlist object
...	passed to other functions, including modMATRIX
labels	character vector of names for $OMEGA elements; must be equal to number of rows/columns in the matrix
make	logical; if TRUE, matrix list is rendered into a single matrix
strict	passed to merge.list
x	matlist object

Examples

```
## example("omega")
mat1 <- matrix(1)
mat2 <- diag(c(1,2,3))
mat3 <- matrix(c(0.1, 0.002, 0.002, 0.5), 2,2)
mat4 <- dmat(0.1, 0.2, 0.3, 0.4)

omat(mat1)
omat(mat1, mat2, mat3)
```

```
omat(A=mat1, B=mat2, C=mat3)

mod <- mrgsolve:::house() %>% omat(mat4)

omat(mod)
omat(mod, make=TRUE)


## Not run:

$OMEGA
1 2 3

$OMEGA block=TRUE
1 0.1 2

$OMEGA cor=TRUE
prefix="ETA_"
labels=s(CL,VC,KA)
0.1
0.67 0.2
0 0 0.3


## End(Not run)
```

---

param *Get and set model parameters*

Description

An accessor function for the `param` model attribute.

See `numericlist` for methods to deal with `parameter_list` objects.

Usage

```
param(.x, ...)

## S4 method for signature 'mrgmod'
param(.x, .y = list(), ..., .pat = "*")

## S4 method for signature 'mrgsims'
param(.x, ...)

## S4 method for signature 'missing'
param(.x, ...)

## S4 method for signature 'list'
param(.x, ...)

## S4 method for signature 'ANY'
param(.x, ...)
```

```
as.param(.x, ...)

## S4 method for signature 'list'
as.param(.x, ...)

## S4 method for signature 'numeric'
as.param(.x, ...)

## S4 method for signature 'parameter_list'
as.param(.x, ...)

## S4 method for signature 'missing'
as.param(.x, ...)

## S4 method for signature 'parameter_list'
show(object)

allparam(.x)
```

Arguments

.x	the model object
...	passed along or name/value pairs to update the parameters in a model object
.y	list to be merged into parameter list
.pat	a regular expression (character) to be applied as a filter for which parameters to show when printing
object	passed to show

Details

Can be used to either get a parameter list object from a mrgmod model object or to update the parameters in a model object. For both uses, the return value is a parameter_list object. For the former use, param is usually called to print the parameters to the screen, but the parameter_list object can also be coreced to a list or numeric R object.

Value

An object of class parameter_list (see numericlist).

Examples

```
## example("param")
mod <- mrgsolve:::house()

param(mod)
param(mod, .pat="^(C|F)") ## may be useful when large number of parameters

class(param(mod))

param(mod)$KA

as.list(param(mod))
as.data.frame(param(mod))
```

parameter_list-class *S4 parameter_list class*

Description

S4 parameter_list class

Details

parameter_list is a numericlist-class

pars *Return the names of model parameters.*

Description

Return the names of model parameters.

Usage

```
pars(x, ...)

## S4 method for signature 'mrgmod'
pars(x, ...)
```

Arguments

x	model object
...	passed along

Examples

```
mod <- mrgsolve:::house()
pars(mod)
```

plot,batch_mrgsims,missing-method

Plot method for mrgsims objects.

Description

Plot method for mrgsims objects.

Usage

```
## S4 method for signature 'batch_mrgsims,missing'
plot(x, yval = variables(x), limit = 9,
  ...)

## S4 method for signature 'batch_mrgsims,formula'
plot(x, y, show.grid = TRUE, lwd = 2,
  type = "l", as = "raw", auto.key = list(columns = 1), scales = list(y
  = list(relation = "free")), ...)
```

Arguments

x	mrsims object
yval	variables to plot
limit	maximum number of yval to plot
...	arguments passed to xyplot
y	a formula passed to xyplot
show.grid	print grid in the plot
lwd	passed to xyplot
type	passed to xyplot
as	transformation for every yval that is plotted
auto.key	passed to xyplot
scales	passed to xyplot

plot_mrgsims

Generate a quick plot of simulated data.

Description

Generate a quick plot of simulated data.

Usage

```
## S4 method for signature 'mrgsims,missing'
plot(x, limit = 16, ...)

## S4 method for signature 'mrgsims,formula'
plot(x, y, limit = 16, show.grid = TRUE,
  as = "raw", outer = TRUE, type = "l", lwd = 2, ylab = "raw value",
  groups = ID, scales = list(y = list(relation = "free")), ...)
```

Arguments

x	mrgsims object
limit	limit the the number of panels to create
...	other arguments passed to xyplot
y	formula used for plotting
show.grid	logical indicating whether or not to draw panel.grid
as	transformations for plotting simulated values
outer	passed to xyplot
type	passed to xyplot
lwd	passed to xyplot
ylab	passed to xyplot
groups	passed to xyplot
scales	passed to xyplot

Details

Values for as argument: ; raw: raw simulated output; frac: each observation normalized to baseline value; cfb: change (difference) from baseline; cfblog: change from baseline of log10-transformed values; log10y: log10 transformation; lny: natural log transformed.

Examples

```
mod <- mrgsolve:::house(end=48, delta=0.2) %>% init(GUT=1000)

out <- mrgsim(mod)

plot(out)

plot(out, subset=time <=24)

plot(out, GUT+CP~.)

plot(out, CP+RESP~time, col="black", scales="same", lty=2)
```

---

plus.ev *Add an events object to a mrgmod object*

Description

Add an events object to a mrgmod object

Usage

```
plus.ev(e1, e2)
```

Arguments

e1	mrgmod object
e2	events object

project *Return the name of the project directory.*

Description

Return the name of the project directory.

Usage

```
project(x, ...)

## S4 method for signature 'mrgmod'
project(x, ...)

## S4 method for signature 'packmod'
project(x, ...)

## S4 method for signature 'mrgsims'
project(x, ...)
```

Arguments

x	model object or mrgsims object
...	passed along

Examples

```
mod <- mrgsolve:::house()
project(mod)
```

relocate *Update* model *or* project *in an* mrgmod *object.*

Description

Update model or project in an mrgmod object.

Usage

```
relocate(x, ...)

## S4 method for signature 'mrgmod'
relocate(x, model = NULL, project = NULL)
```

Arguments

x	mrgmod object
...	passed along
model	model name
project	project directory

Value

updated model object

`Req` *Set the* `Request` *argument for* `mrgsim`.

Description

Set the `Request` argument for `mrgsim`.

Usage

```
Req(x, ...)

## S4 method for signature 'mrgmod'
Req(x, ...)

req(x, ...)

## S4 method for signature 'mrgmod'
req(x, ...)
```

Arguments

`x`	model object
`...`	unquoted names of compartments or tabled items

Examples

```
mod <- mrgsolve:::house()

mod %>% Req(CP,RESP) %>% ev(amt=1000) %>%  mrgsim
```

`request` *Return the requested compartments from a simulation run.*

Description

Return the requested compartments from a simulation run.

Usage

```
request(x, ...)

## S4 method for signature 'mrgsims'
request(x, ...)
```

Arguments

x	mrgsims object
...	passed along

See Also

variables

---

reserved *Reserved words in* mrgsolve.

Description

Reserved words in mrgsolve.

Usage

```
reserved()
```

Details

Note: this function is not exported; you must go into the mrgsolve namespace by using the mrgsolve::: prefix.

Examples

```
mrgsolve:::reserved()
```

---

revar *Get model random effect variances and covariancnes.*

Description

Get model random effect variances and covariancnes.

Usage

```
revar(x, ...)

## S4 method for signature 'mrgmod'
revar(x, ...)

## S4 method for signature 'mrgsims'
revar(x, ...)
```

Arguments

x	model object
...	passed along

see *Print model code to the console.*

Description

Print model code to the console.

Usage

```
see(x, ...)

## S4 method for signature 'mrgmod'
see(x, ...)
```

Arguments

x	model object
...	passed along

Value

invisible NULL

shlib *Return information about model compilation.*

Description

Return information about model compilation.

Usage

```
shlib(x, ...)

## S4 method for signature 'mrgmod'
shlib(x, ...)
```

Arguments

x	model object
...	passed along

show,mrgmod-method *Print model details*

Description

Print model details

Usage

```
## S4 method for signature 'mrgmod'
show(object)
```

Arguments

object	the model object

sigma *Manipulate* SIGMA *matrices.*

Description

The primary function is smat that can be used to both get the $SIGMA matrices out of a model object and to update $SIGMA matrices in a model object.

Usage

```
smat(.x, ...)

## S4 method for signature 'missing'
smat(.x, ...)

## S4 method for signature 'matrix'
smat(.x, ..., labels = list())

## S4 method for signature 'list'
smat(.x, ...)

## S4 method for signature 'sigmalist'
smat(.x, ...)

## S4 method for signature 'mrgmod'
smat(.x, ..., make = FALSE, strict = TRUE)

## S4 method for signature 'mrgsims'
smat(.x, make = FALSE, ...)
```

Arguments

.x	a matrix, list of matrices or matlist object
...	passed to other functions, including modMATRIX
labels	character vector of names for $SIGMA elements; must be equal to number of rows/columns in the matrix
make	logical; if TRUE, matrix list is rendered into a single matrix
strict	passed to merge.list
x	matlist object

Examples

```
## example("sigma")
mat1 <- matrix(1)
mat2 <- diag(c(1,2))
mat3 <- matrix(c(0.1, 0.002, 0.002, 0.5), 2,2)
mat4 <- dmat(0.1, 0.2, 0.3, 0.4)

smat(mat1)
smat(mat1, mat2, mat3)
smat(A=mat1, B=mat2, C=mat3)

mod <- mrgsolve:::house() %>% smat(mat1)

smat(mod)
smat(mod, make=TRUE)
```

---

simargs *Access or clear arguments for* mrgsim.

Description

Access or clear arguments for mrgsim.

Usage

```
simargs(x, ...)

## S3 method for class 'mrgmod'
simargs(x, clear = FALSE, ...)
```

Arguments

x	model object
...	passed along
clear	logical indicating whether or not clear args from the model object

Value

If clear is TRUE, the argument list is cleared and the model object is returned. Otherwise, the argument list is returned.

simre *Simulate random effects from model.*

Description

Simulate random effects from model.

Usage

```
simre(x, ...)

## S4 method for signature 'mrgmod'
simre(x, seed = NULL, neta = 10, neps = 10, ...)

## S4 method for signature 'mrgsims'
simre(x, seed = NULL, ...)
```

Arguments

x	model object
...	passed along
seed	passed to set.seed
neta	number of etas to simulate
neps	number of epsilon values to simulate

sodll *Return the name of the shared object file.*

Description

Return the name of the shared object file.

Usage

```
sodll(x, ...)

## S4 method for signature 'mrgmod'
sodll(x, ...)

## S4 method for signature 'lockedmod'
sodll(x, ...)

## S4 method for signature 'packmod'
sodll(x, ...)
```

Arguments

x	model object
...	passed along

Examples

```
mod <- mrgsolve:::house()
sodll(mod)
```

---

soloc *Return the location of the model shared object.*

Description

Return the location of the model shared object.

Usage

```
soloc(x, short = FALSE)
```

Arguments

x	model object
short	logical; if TRUE, soloc will be rendered with a short path name

Examples

```
mod <- mrgsolve:::house()
soloc(mod)
```

---

stime,mrgmod-method *Create a simtime object.*

Description

simtime objects allow the user to specify simulation start and end times, along with the simulation time step.

##' @export

Usage

```
## S4 method for signature 'mrgmod'
stime(x, ...)

## S4 method for signature 'mrgsims'
stime(x, ...)

tgrid(start = 0, end = 24, delta = 1, add = numeric(0), .offset = 0,
  .scale = 1, ...)

## S4 method for signature 'tgrid'
c(x, ..., recursive = FALSE)
```

```
## S4 method for signature 'tgrids'
c(x, ..., recursive = FALSE)

## S4 method for signature 'tgrid'
stime(x, ...)

## S4 method for signature 'tgrids'
stime(x, ...)

## S4 method for signature 'numeric'
stime(x, ...)

## S4 method for signature 'tgrid,numeric'
e1 + e2

## S4 method for signature 'tgrid,numeric'
e1 * e2

## S4 method for signature 'tgrids,numeric'
e1 + e2

## S4 method for signature 'tgrids,numeric'
e1 * e2

## S4 method for signature 'tgrid'
show(object)

## S4 method for signature 'tgrids'
show(object)
```

Arguments

x	mrgmod object
...	passed on to other methods
start	simulation start time
end	simulation end time
delta	simulation time step
add	addition simulation times
.offset	the resulting set of times will be adjusted by this amount
.scale	the resulting set of times will be scaled by this factor
recursive	not used
e1	tgrid or tgrids object
e2	numeric value
object	passed to show

Examples

```
peak <- tgrid(0,6,0.2)
```

```
sparse <- tgrid(0,24,4)

day1 <- c(peak,sparse)

design <- c(day1, day1+72, day1+240)

mod <- mrgsolve:::house()

out <- mod %>% ev(amt=1000, ii=24, addl=10) %>% mrgsim(tgrid=design)

plot(out,CP~., type='b')
```

`tgrid-class` *Get the times at which the model will be evaluated.*

Description

Get the times at which the model will be evaluated.

Usage

```
stime(x, ...)
```

Arguments

x	object of class mrgmod
...	passed on

Details

Simulation times include the sequence of times created from `start`, `end`, and `delta` and the vector of times found in `add`. Making `end` negative will omit any `start` / `end` / `delta` sequence. Negative values are discarded from the result.

Value

a sorted vector of unique times

Examples

```
## example("stime", package="mrgsolve")

mod <- mrgsolve:::house(end=12, delta=2, add=c(11,13,15))

stime(mod)

out <- mrgsim(mod, end=-1, add=c(2,4,5))

stime(out)

out$time
```

touch_funs *Get inits from compiled function.*

Description

Get inits from compiled function.

Usage

```
touch_funs(x)
```

Arguments

x	mrgmod model object

tscale *Set the* tscale *argument for* mrgsim.

Description

Set the tscale argument for mrgsim.

Usage

```
tscale(x, value = 1, ...)
```

Arguments

x	model object
value	value by which time will be scaled
...	passed along

unloadso *Unload the model shared object.*

Description

Unload the model shared object.

Usage

```
unloadso(x, ...)

## S4 method for signature 'mrgmod'
unloadso(x, ...)
```

Arguments

x	model object
...	passed along

update *Update the model object*

Description

After the model object is created, update various attributes.

Usage

```
## S4 method for signature 'mrgmod'
update(object, ..., merge = TRUE, strict = TRUE,
  super.strict = FALSE, data = list())

## S4 method for signature 'omegalist'
update(object, y, ...)

## S4 method for signature 'sigmalist'
update(object, y, ...)

## S4 method for signature 'parameter_list'
update(object, y, ...)

## S4 method for signature 'ev'
update(object, y, ...)
```

Arguments

object	a model object
...	passed to other functions
merge	logical indicating to merge (rather than replace) new and existing attributes.
strict	logical; used only when merge is TRUE and parameter list or initial conditions list is being updated; if TRUE, no new items will be added; if FALSE, the parameter list may expand.
super.strict	logical; strict common area updating
data	a list of items to update; not used for now
y	another object involved in update

Details

See also mrgsolve_Ops for alternative ways to update events, parameters, and initial conditions in a model object.

Value

The updated model object is returned.

Examples

```
mod <- mrgsolve:::house()

mod <- update(mod, end=120, delta=4, param=list(CL=19.1))
```

variables — *Return the requested compartments and tabled items from a simulation run.*

Description

Return the requested compartments and tabled items from a simulation run.

Usage

```
variables(x, ...)

## S4 method for signature 'mrgsims'
variables(x, ...)
```

Arguments

x	mrgsims object
...	passed along

See Also

request

%>% — *Forward pipe. #'*

Description

Forward pipe. #'

Tee.

Index