Package ‘mrgsolve’

May 10, 2021

Type Package
Title Simulate from ODE-Based Models
Version 0.11.1
Maintainer Kyle T Baron <kyleb@metrumrg.com>
Description Fast simulation from ordinary differential equation (ODE) based models typically employed in quantitative pharmacology and systems biology.
License GPL (>= 2)
URL https://github.com/metrumresearchgroup/mrgsolve
BugReports https://github.com/metrumresearchgroup/mrgsolve/issues
Depends R (>= 3.1.2), methods
Imports Rcpp (>= 0.12.12), dplyr (>= 0.8.1), magrittr (>= 1.5), tibble (>= 2.1.1), rlang (>= 0.3.4), tidyselect (>= 0.2.5), lifecycle, glue
Suggests lattice, testthat, xml2 (>= 1.2.0), rmarkdown, yaml, knitr, data.table, pmxTools
LinkingTo Rcpp (>= 0.12.12), RcppArmadillo (>= 0.7.900.2.0), BH (>= 1.62.0-1)
RdMacros lifecycle
Encoding UTF-8
Language en-US
LazyLoad yes
NeedsCompilation yes
RoxygenNote 7.1.1
SystemRequirements C++11
Collate 'RcppExports.R' 'utils.R' 'package.R' 'generics.R'
'class_tgrid.R' 'class_numericlist.R' 'class_matlist.R'
'class_ev.R' 'classDerived.R' 'class_mrgmod.R'
'class_mrgsims.R' 'Aaaa.R' 'annot.R' 'chain.R' 'class_build.R'
R topics documented:

`model_include.R` `modlib.R` `modspec.R` `mread.R`

Author Kyle T Baron [aut, cre] (<https://orcid.org/0000-0001-7252-5656>),
Bill Gillespie [ctb],
Charles Margossian [ctb],
Devin Pastoor [ctb],
Bill Denney [ctb] (<https://orcid.org/0000-0002-5759-428X>),
Dilawar Singh [ctb],
Felicien Le Louedec [ctb] (<https://orcid.org/0000-0003-3699-2729>),
Timothy Waterhouse [ctb] (<https://orcid.org/0000-0002-0954-9660>),
Metrum Research Group [cph]

Repository CRAN

Date/Publication 2021-05-10 19:50:02 UTC
**R topics documented:**

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ev_rx</td>
<td>28</td>
</tr>
<tr>
<td>ev_seq</td>
<td>30</td>
</tr>
<tr>
<td>exdatasets</td>
<td>31</td>
</tr>
<tr>
<td>expand.idata</td>
<td>32</td>
</tr>
<tr>
<td>expand_observations</td>
<td>33</td>
</tr>
<tr>
<td>idata_set</td>
<td>34</td>
</tr>
<tr>
<td>init</td>
<td>36</td>
</tr>
<tr>
<td>inventory</td>
<td>37</td>
</tr>
<tr>
<td>is.mrgmod</td>
<td>38</td>
</tr>
<tr>
<td>is.mrgsims</td>
<td>38</td>
</tr>
<tr>
<td>lctran</td>
<td>39</td>
</tr>
<tr>
<td>loadso</td>
<td>39</td>
</tr>
<tr>
<td>matrix_helpers</td>
<td>40</td>
</tr>
<tr>
<td>mcode</td>
<td>41</td>
</tr>
<tr>
<td>mcRNG</td>
<td>42</td>
</tr>
<tr>
<td>modlib</td>
<td>43</td>
</tr>
<tr>
<td>modlib_details</td>
<td>44</td>
</tr>
<tr>
<td>modlib_pk</td>
<td>45</td>
</tr>
<tr>
<td>modlib_pkpd</td>
<td>46</td>
</tr>
<tr>
<td>modlib_tmdd</td>
<td>46</td>
</tr>
<tr>
<td>modlib_viral</td>
<td>47</td>
</tr>
<tr>
<td>mread</td>
<td>48</td>
</tr>
<tr>
<td>mrgsim</td>
<td>51</td>
</tr>
<tr>
<td>mrgsims_dplyr</td>
<td>55</td>
</tr>
<tr>
<td>mrgsims_modify</td>
<td>57</td>
</tr>
<tr>
<td>mrgsim_q</td>
<td>58</td>
</tr>
<tr>
<td>mrgsim_variants</td>
<td>59</td>
</tr>
<tr>
<td>mrgsolve</td>
<td>60</td>
</tr>
<tr>
<td>mutate.ev</td>
<td>62</td>
</tr>
<tr>
<td>names.mrgmod-method</td>
<td>63</td>
</tr>
<tr>
<td>nmext</td>
<td>64</td>
</tr>
<tr>
<td>nmxml</td>
<td>65</td>
</tr>
<tr>
<td>numerics_only</td>
<td>67</td>
</tr>
<tr>
<td>obsaug</td>
<td>67</td>
</tr>
<tr>
<td>obsonly</td>
<td>68</td>
</tr>
<tr>
<td>omega</td>
<td>68</td>
</tr>
<tr>
<td>outvars</td>
<td>70</td>
</tr>
<tr>
<td>param</td>
<td>70</td>
</tr>
<tr>
<td>PKMODEL</td>
<td>72</td>
</tr>
<tr>
<td>plot.batch_mrgsims,missing-method</td>
<td>73</td>
</tr>
<tr>
<td>plot_mrgsims</td>
<td>74</td>
</tr>
<tr>
<td>plot_sims</td>
<td>76</td>
</tr>
<tr>
<td>qsim</td>
<td>77</td>
</tr>
<tr>
<td>read_nmext</td>
<td>78</td>
</tr>
<tr>
<td>realize_addl</td>
<td>79</td>
</tr>
<tr>
<td>render</td>
<td>80</td>
</tr>
<tr>
<td>Req</td>
<td>81</td>
</tr>
<tr>
<td>reserved</td>
<td>82</td>
</tr>
</tbody>
</table>
About the lsoda differential equation solver used by mrgsolve

Description

The differential equation solver is a C++ translation of DLSODA from ODEPACK. The C++ translation was created by Dilawar Singh and hosted here https://github.com/dilawar/libsoda/. As we understand the history of the code, Heng Li was also involved in early versions of the code written in C. There was a potentially-related project hosted here https://github.com/sdwfrost/liblsoda/.

Details

The C++ translation by Dilawar Singh contains functions that appear to be based on BLAS and LAPACK routines. These functions have been renamed to be distinct from the respective BLAS and LAPACK function names. References are given in the section below.

History

The following history was recorded in the source code published by Dilawar Singh:

/*
 * HISTORY:
 * This is a CPP version of the LSODA library for integration into MOOSE
 * simulator.
 * The original was aquired from
 * http://www.ccl.net/cca/software/SOURCES/C/kinetics2/index.shtml and modified by
 * Heng Li <lh3lh3@gmail.com>. Heng merged several C files into one and added a
 * simpler interface. [Available
The original source code came with no license or copyright information. Heng Li released his modification under the MIT/X11 license. I maintain the same license. I have removed quite a lot of text/comments from this library. Please refer to the standard documentation.

Contact: Dilawar Singh <dilawars@ncbs.res.in>

References

1. LAPACK: https://netlib.org/lapack/
2. BLAS: https://netlib.org/blas/

---

### as.ev

*Coerce an object to class ev*

#### Description

Coerce an object to class ev

#### Usage

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

```r
## S4 method for signature 'data.frame'
as.ev(x, keep_id = TRUE, clean = FALSE, ...)
```

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

#### Arguments

- `x`  
  an object to coerce
- `...`  
  not used
- `keep_id`  
  if TRUE, ID column is retained if it exists
- `clean`  
  if TRUE, only dosing or ID information is retained in the result

#### Examples

```r
data <- data.frame(amt = 100)
as.ev(data)
```
as.list,mrgmod-method  
  Coerce a model object to list

Description

Coerce a model object to list

Usage

```r
## S4 method for signature 'mrgmod'
as.list(x, deep = FALSE, ...)
```

Arguments

- `x`: mrgmod object
- `deep`: if TRUE, extra information is returned (see details).
- `...`: not used

Details

If deep is TRUE, then the values for `trans`, `advan`, and `mindt` are returned as well as a summary of internal model functions (with a call to `mrgsolve:::funset`).

Slots

- `npar`: number of parameters
- `neq`: number of compartments or differential equations
- `pars`: names of model parameters
- `covariates`: names of parameters identified as covariates
- `cmt`: names of model compartments
- `param`: the parameter list
- `init`: initial condition list
- `omega`: SOMEGA matrices, as a matlist object
- `sigma`: SIGMA matrices, as a matlist object
- `fixed`: named list of $FIXED values
- `model`: model name
- `project`: model project directory
- `soloc`: directory where the model is being built
- `sodll`: complete path to the model shared object
- `cfile`: path for the model source code file
- `shlib`: list of compilation information
- `start`: simulation start time
Coerce an mrgsims object to list
Usage

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

Arguments

- `x`: an mrgsims object
- `...`: not used

Description

These are simple functions that may be helpful to create the matrix objects that mrgsolve expects. Functions are named based on whether they create a diagonal matrix (d), a block matrix (b), or a correlation matrix (c).

Usage

```r
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 = "*", cols = NULL, ...)

## 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_bmat

as_dmat(x, pat = "*", cols = NULL, ...)

as_cmat(x, ...)

Arguments

x         data frame or list
...
arguments passed to dmat or bmat
pat        regular expression, character
cols      column names to use instead of pat

Details

Use as_dmat to create a diagonal matrix, as_bmat to create a block matrix, and as_cmat to create a block matrix where diagonal elements are understood to be correlations rather than covariances. as_cmat uses as_bmat to form the matrix and then converts off-diagonal elements to covariances before returning.

The methods for data.frame will work down the rows of the data frame and make the appropriate matrix from the data in each row. The result is a list of matrices.

Value

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

See Also

bmat, dmat, cmat

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")
as_data_set  Create a simulation data set from ev objects

Description
Create a simulation data set from ev objects

Usage
as_data_set(x, ...)

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

## S4 method for signature 'data.frame'
as_data_set(x, ...)

Arguments
x  ev objects
...
more ev objects

Details
The goal is to take a series of event objects and combine them into a single data set that can be passed to \code{data_set}. Each event object is added to the data frame as an ID or set of IDs that are distinct from the IDs in the other event objects. Note that including ID argument to the \code{ev} call where \code{length(ID)} is greater than one will render that set of events for all of IDs that are requested.

To get a data frame with one row (event) per ID look at \code{expand.ev}.

Value
a data frame suitable for passing into \code{data_set}

Examples

```r
as_data_set(ev(amt=c(100,200), cmt=1, ID=1:3),
            ev(amt=300, time=24, ID=1:2),
            ev(amt=1000, ii=8, addl=10, ID=1:3))

# Instead of this, use expand.ev
as_data_set(ev(amt=100), ev(amt=200), ev(amt=300))
```
Create a list of designs from a data frame

Usage

```r
as_deslist(data, descol = "ID")
```

Arguments

- `data`: input data set; see details
- `descol`: character column name to be used for design groups

Details

The input data set must have a column with the same name as the value of `descol`. Other column names should be `start` (the time of the first observation), `end` (the time of the last observation), `delta` (the time steps to take between `start` and `end`), and `add` (other, ad-hoc times). Note that `add` might be a list-column to get a vector of times for each time grid object.

Value

The function returns a list of `tgrid` objects, one for each unique value found in `descol`.

Examples

```r
idata <- tibble::tibble(ID=1:4, end=seq(24,96,24), delta=6,
                        add=list(c(122,124,135),c(111),c(99),c(88)))

idata <- dplyr::mutate(idata, GRP = ID %%2)

idata

l <- as_deslist(idata,"GRP")
l
lapply(l,stime)
lapply(as_deslist(idata, "ID"),stime)
```
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)

-------------------------------

DESCRIPTION

Most of the basic blocks are listed in this help topic. But see also PKMODEL() which has more-involved options and is documented separately.

Usage

PARAM(
  x,
  env,
  pos = 1,
  annotated = FALSE,
  object = NULL,
  as_object = FALSE,
covariates = FALSE,

)  

FIXED(x, env, pos = 1, annotated = FALSE, ...)

THETA(
  x,
  env,
  pos = 1,
  annotated = FALSE,
  object = NULL,
  as_object = FALSE,
  name = "THETA",
  fill = NULL,
  ...
 )

INIT(x, env, pos = 1, annotated = FALSE, object = NULL, as_object = FALSE, ...)

CMT(x, env, pos = 1, annotated = FALSE, object = NULL, as_object = FALSE, ...)

CAPTURE(x, env, pos = 1, annotated = FALSE, ...)

HANDLEMATRIX(
  x,
  env,
  pos = 1,
  annotated = FALSE,
  object = NULL,
  as_object = FALSE,
  name = "...",
  type = NULL,
  oclass = ".",
  prefix = ".",
  labels = NULL,
  unlinked = FALSE,
  ...
  )

Arguments

x data
env parse environment
pos block position
annotated logical
object the name of an object in ENV
c,matlist-method

Operations with matlist objects

Description

Operations with matlist objects

Usage

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

Arguments

- `x` a matlist object
- `...` other matlist objects
- `recursive` not used

Details

When using `object` or `as_object` populate the block contents, the following types are required:

- `PARAM`: a named list
- `INIT`: a named list
- `THETA`: a numeric vector; names are ignored
- `CMT`: a character vector
- `OMEGA`: matrix; set rownames on the matrix to create ETA labels; setting rownames is the only way to specify labels when working through the `object` or `as_object` directives
- `SIGMA`: matrix; set rownames on the matrix to create EPS labels; setting rownames is the only way to specify labels when working through the `object` or `as_object` directives

See Also

`PKMODEL()`
Operations with tgrid objects

Usage

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

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

## 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

Arguments

x mrgmod object
...
... passed along to other methods
recursive not used
e1 tgrid or tgrids object
e2 numeric value

carry_out Select items to carry into simulated output

Description

When items named in this function are found in the input data set (either data_set or idata_set), they are copied into the simulated output. Special items like evid or amt or the like are not copied from the data set per se, but they are copied from datarecord objects that are created during the simulation.
Usage

carry_out(x, ...)

carry.out(x, ...)

Arguments

x model object
...

Details

There is also a carry.out argument to `mrgsim` that can be set to accomplish the same thing as a call to `carry_out` in the pipeline.

carry.out and carry_out. Using the underscore version is now preferred.

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
...
tag compartment name

Examples

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

*Extract the code from a model*

**Description**

Extract the code from a model

**Usage**

code(x)

**Arguments**

- **x**
  an mrgsolve model object

**Value**

a character vector of model code

---

**data_set**

*Select and modify a data set for simulation*

**Description**

The input data set (data_set) is a data frame that specifies observations, model events, and/or parameter values for a population of individuals.

**Usage**

data_set(x, data, ...)

```r
## S4 method for signature 'mrgmod,data.frame'
data_set(  
  x,  
  data,  
  .subset = TRUE,  
  .select = TRUE,  
  object = NULL,  
  need = NULL,  
  ...  
)
```

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

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

## S4 method for signature 'mrgmod,missing'
data_set(x, object, ...)  

**Arguments**  

- **x**: model object  
- **data**: data set  
- **...**: passed along  
- **.subset**: an unquoted expression passed to dplyr::filter; retain only certain rows in the data set  
- **.select**: passed to dplyr::select; retain only certain columns in the data set; this should be the result of a call to dplyr::vars()  
- **object**: character name of an object existing in $ENV to use for the data set  
- **need**: passed to inventory  

**Details**  

Input data sets are R data frames that can include columns with any valid name, however columns with selected names are treated specially 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. Along with ID, time is a required column in the input data set unless $PRED is in use. Upper case PK dosing column names including TIME, CMT, AMT, RATE, II, ADDL, SS are also recognized. However, an error will be generated if a mix of upper case and lower case columns in this family are found.  

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.  

An error will be generated when mrgsolve detects that the data set is not sorted by time within an individual.  

Only numeric data can be brought in to the problem. Any non-numeric data columns will be dropped with warning. See numerics_only, which is used to prepare the data set.  

An error will be generated if any parameter columns in the input data set contain NA. Likewise, and error will be generated if missing values are found in the following columns: ID, time/TIME, rate/RATE.  

See exdatasets for different example data sets.  

**See Also**  

idata_set, ev, valid_data_set, valid_idata_set
Examples

```r
mod <- mrgsolve::house()
data <- expand.ev(ID=1:3, amt=c(10,20))mod %>% data_set(data, ID > 1) %>% mrgsim
data(extran1)head(extran1)
mod %>% data_set(extran1) %>% mrgsim
mod %>% mrgsim(data=extran1)
```

---

**design**

*Set observation designs for the simulation*

**Description**

This function also allows you to assign different designs to different groups or individuals in a population.

**Usage**

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

**Arguments**

- `x` : model object
- `deslist` : a list of `tgrid` or `tgrids` objects or numeric vector to be used in place of ...
- `descol` : the `idata` column name (character) for design assignment
- `...` : not used

**Details**

This setup requires the use of an `idata_set`, with individual-level data passed in one ID per row. For each ID, specify a grouping variable in `idata` (`descol`). For each unique value of the grouping variable, make one `tgrid` object and pass them in order as ... or form them into a list and pass as `deslist`.

You must assign the `idata_set` before assigning the designs in the command chain (see the example below).
Examples

```r
peak <- tgrid(0,6,0.1)
sparse <- tgrid(0,24,6)

des1 <- c(peak,sparse)
des2 <- tgrid(0,72,4)

data <- expand.ev(ID = 1:10, amt=c(100,300))
data$GRP <- data$amt/100

idata <- data[,c("ID", "amt")]

mod <- mrgsolve::house()

mod %>%
  omat(dmat(1,1,1,1)) %>%
  carry_out(GRP) %>%
  idata_set(idata) %>%
  design(list(des1, des2),"amt") %>%
  data_set(data) %>%
  mrgsim %>%
  plot(RESP~time|GRP)
```

details

Extract model details

Description

Extract model details

Usage

details(x, complete = FALSE, values = TRUE, ...)

Arguments

- `x` a model object
- `complete` logical; if TRUE, un-annotated parameters and compartments will be added to the output
- `values` logical; if TRUE, a values column will be added to the output
- `...` not used

Details

This function is not exported. You will have to call it with mrgsolve::details().
**Examples**

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

---

**Description**

The `$ENV` block is a block of R code that can realize any sort of R object that might be used in running a model.

**Usage**

```r
env_eval(x, seed = NULL)
```

**Arguments**

- `x` : model object
- `seed` : passed to `set.seed` if a numeric value is supplied

**See Also**

`env_get`, `env_ls`

---

**env_get**  
*Return model environment*

---

**Description**

Return model environment

**Usage**

```r
env_get(x, tolist = TRUE)
env_get_env(x)
```

**Arguments**

- `x` : model object
- `tolist` : should the environment be coerced to `list`?
env_ls

List objects in the model environment

Description

Each model keeps an internal environment that allows the user to carry any R object along. Objects are coded in $ENV.

Usage

env_ls(x, ...)

Arguments

x  model object
...
... passed to ls

env_update

Update objects in model environment

Description

Update objects in model environment

Usage

env_update(.x, ..., .dots = list())

Arguments

.x  model object
...
... objects to update
.dots list of objects to updated
ev

Event objects for simulating PK and other interventions

Description

An event object specifies dosing or other interventions that get implemented during simulation. Event objects do similar things as `data_set`, but simpler and quicker.

Usage

`ev(x, ...)`

```r
## S4 method for signature 'mrgmod'
ev(x, object = NULL, ...)

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

```r
## S4 method for signature 'ev'
ev(x, realize_addl = FALSE, ...)
```

Arguments

- `x`: a model object
- `...`: other items to be incorporated into the event object; see details
- `object`: passed to show
- `time`: event time
- `amt`: dose amount
- `evid`: event ID
- `cmt`: compartment
- `ID`: subject ID
- `replicate`: logical; if `TRUE`, events will be replicated for each individual in `ID`
- `until`: the expected maximum **observation** time for this regimen
infusion time; if greater than zero, then the rate item will be derived as \( \frac{\text{amt}}{\text{tinf}} \).

**realize_addl** if FALSE (default), no change to addl doses. If TRUE, addl doses are made explicit with `realize_addl`.

### Details
- Required items in events objects include `time`, `amt`, `evid` and `cmt`.
  - If not supplied, `evid` is assumed to be 1.
  - If not supplied, `cmt` is assumed to be 1.
  - If not supplied, `time` is assumed to be 0.
  - If `amt` is not supplied, an error will be generated.
  - If `total` is supplied, then `addl` will be set to `total - 1`.
- Other items can include `ii`, `ss`, and `addl` (see `data_set` for details on all of these items).
- `ID` may be specified as a vector.
- If `replicate` is `TRUE` (default), then the events regimen is replicated for each ID; otherwise, the number of event rows must match the number of IDs entered.

### Value
- `events` object

### See Also
- `ev_rep`, `ev_days`, `ev_repeat`, `ev_assign`, `ev_seq`, `mutate.ev`, `as.ev`, `ev_methods`

### Examples
```r
mod <- mrgsolve::house()

mod <- mod %>% ev(amt=1000, time=0, cmt=1)

loading <- ev(time=0, cmt=1, amt=1000)

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

c(loading, maint)

loading$time
```
Replicate a list of events into a data set

Description
Replicate a list of events into a data set

Usage
```
ev_assign(l, idata, evgroup, join = FALSE)
assign_ev(...)
```

Arguments
- `l`: list of event objects
- `idata`: an idata set (one ID per row)
- `evgroup`: the character name of the column in `idata` that specifies event object to implement
- `join`: if `TRUE`, join `idata` to the data set before returning.
- `...`: used to pass arguments from `assign_ev` to `ev_assign`

Details
`ev_assign` connects events in a list passed in as the `l` argument to values in the data set identified in the `evgroup` argument. For making assignments, the unique values in the `evgroup` column are first sorted so that the first sorted unique value in `evgroup` is assigned to the first event in `l`, the second sorted value in `evgroup` column is assigned to the second event in `l`, and so on. This is a change from previous behavior, which did not sort the unique values in `evgroup` prior to making the assignments.

Examples
```
ev1 <- ev(amt=100)
ev2 <- ev(amt=300, rate=100, ii=12, addl=10)

idata <- data.frame(ID=1:10)
idata$arm <- 1+(idata$ID %%2)

ev_assign(list(ev1,ev2), idata, "arm", join=TRUE)
```
Description

This function lets you schedule doses on specific days of the week, allowing you to create dosing regimens on Monday/Wednesday/Friday, or Tuesday/Thursday, or every other day (however you want to define that) etc.

Usage

```r
ev_days(
ev = NULL,
days = "",
addl = 0,
ii = 168,
unit = c("hours", "days"),
...
)
```

Arguments

- **ev**: an event object
- **days**: comma- or space-separated character string of valid days of the the week (see details)
- **addl**: additional doses to administer
- **ii**: inter-dose interval; intended use is to keep this at the default value
- **unit**: time unit; the function can only currently handle hours or days
- **...**: event objects named by one the valid days of the week (see details)

Details

Valid names of the week are:

- m for Monday
- t for Tuesday
- w for Wednesday
- th for Thursday
- f for Friday
- sa for Saturday
- s for Sunday

The whole purpose of this function is to schedule doses on specific days of the week, in a repeating weekly schedule. Please do use caution when changing `ii` from it's default value.
Examples

# Monday, Wednesday, Friday x 4 weeks
ev_days(ev(amt=100), days="m,w,f", addl=3)

# 50 mg Tuesdays, 100 mg Thursdays x 6 months
ev_days(t=ev(amt=50), th=ev(amt=100), addl=23)

ev_rep
Replicate an event object

Description
An event sequence can be replicated a certain number of times in a certain number of IDs.

Usage
ev_rep(x, ID = 1, n = NULL, wait = 0, as.ev = FALSE, id = NULL)

Arguments
- x: event object
- ID: numeric vector if IDs
- n: passed to ev_repeat
- wait: passed to ev_repeat
- as.ev: if TRUE an event object is returned
- id: deprecated; use ID instead

Value
A single data.frame or event object as determined by the value of as.ev.

See Also
ev_repeat

Examples

e1 <- c(ev(amt=100), ev(amt=200, ii=24, addl=2, time=72))
ev_rep(e1, 1:5)
**ev_repeat**  
*Repeat a block of dosing events*

**Description**

Repeat a block of dosing events

**Usage**

```
ev_repeat(x, n, wait = 0, as.ev = FALSE)
```

**Arguments**

- `x`: event object or dosing data frame
- `n`: number of times to repeat
- `wait`: time to wait between repeats
- `as.ev`: if TRUE, an event object is returned; otherwise a data.frame is returned

**Value**

See as.ev argument.

---

**ev_rx**  
*Create intervention objects from Rx input*

**Description**

See details below for Rx specification. Actual parsing is done by `parse_rx`; this function can be used to debug Rx inputs.

**Usage**

```
ev_rx(x, y, ...)
```

```
## S4 method for signature 'mrgmod,character'
ev_rx(x, y, ...)
```

```
## S4 method for signature 'character,missing'
ev_rx(x, df = FALSE, ...)
```

```
pause_rx(x)
```

*Note: The `pause_rx` function is not included in the provided text.*
ev_rx

Arguments

- **x**: a model object or character Rx input
- **y**: character Rx input; see details
- **...**: not used at this time
- **df**: if TRUE then a data frame is returned

Value

The method dispatched on model object (mrgmod) returns another model object. The character method returns an event object. The parse_rx function return a list named with arguments for the event object constructor ev.

Rx specification

- The dose is found at the start of the string by sequential digits; this may be integer, decimal, or in scientific notation
- Use *in* to identify the dosing compartment number; must be integer
- Use *q* to identify the dosing interval; must be integer or decimal number (but not scientific notation)
- Use *over* to indicate an infusion and its duration; integer or decimal number
- Use *x* to indicate total number of doses; must be integer
- Use *then* or *,,* to separate dosing periods
- Use *after* to insert a lag in the start of a period; integer or decimal number (but not scientific notation)

Examples

```r
# example("ev_rx")
ev_rx("100")
ev_rx("100 in 2")
ev_rx("100 q12 x 3")
ev_rx("100 over 2")
ev_rx("100 q 24 x 3 then 50 q12 x 2")
ev_rx("100 then 50 q 24 after 12")
ev_rx("100.2E-2 q4")
ev_rx("100 over 2.23")
ev_rx("100 q 12 x 3")
parse_rx("100 mg q 24 then 200 mg q12")
```
ev_seq

Schedule a series of event objects

Description

Schedule a series of event objects

Usage

```
ev_seq(..., ID = NULL, .dots = NULL, id = NULL)
```

## S3 method for class 'ev'
seq(...)

Arguments

- `...`: event objects or numeric arguments named `wait`
- `ID`: numeric vector of subject IDs
- `.dots`: a list of event objects that replaces `...`
- `id`: deprecated; use `ID`

Details

The doses for the next event line start after all of the doses from the previous event line plus one
dosing interval from the previous event line (see examples).

When numerics named `wait` are mixed in with the event objects, a period with no dosing activity is
incorporated into the sequence, between the adjacent dosing event objects. Values for `wait` can be
negative.

Values for `time` in any event object act like a prefix time spacer wherever that event occurs in the
event sequence (see examples).

Use the generic `seq` when the first argument is an event object. If a waiting period is the first event,
you will need to use `ev_seq`. When an event object has multiple rows, the end time for that sequence
is taken to be one dosing interval after the event that takes place on the last row of the event object.

Value

A single event object.

Examples

```
e1 <- ev(amt=100, ii=12, addl=1)
e2 <- ev(amt=200)
seq(e1, e2)
```
exdatasets

seq(e1, .ii = 8, e2)
seq(e1, wait = 8, e2)
seq(e1, .ii = 8, e2, ID = 1:10)

ev_seq(.ii = 12, e1, .ii = 120, e2, .ii = 120, e1)

seq(ev(amt=100, ii=12), ev(time=8, amt=200))

<table>
<thead>
<tr>
<th>exdatasets</th>
<th>Example input data sets</th>
</tr>
</thead>
</table>

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

```r
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 template data sets for simulation

Description

Create template data sets for simulation

Usage

expand.idata(...)
expand.ev(...)
ev_expand(...)
Arguments

... passed to expand.grid

Details

An ID column is added as seq(nrow(ans)) if not supplied by the user. For expand.ev, defaults also added include cmt = 1, time = 0, evid = 1. If total is included, then addl is derived as total - 1. If tinf is included, then an infusion rate is derived for row where tinf is greater than zero.

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)
infusion <- expand.ev(amt = 100, tinf = 2)
Examples

```r
data <- expand.ev(amt = c(100, 200, 300))
expand_observations(data, times = seq(0, 48, 2))
```

Declaration

The individual data set (idata_set) is a data frame with one row for each individual in a population, specifying parameters and other individual-level data.

Usage

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

## S4 method for signature 'mrgmod, data.frame'

```r
idata_set(
  x,
  data,
  .subset = TRUE,
  .select = TRUE,
  object = NULL,
  need = NULL,
  ...
)
```

## S4 method for signature 'mrgmod, ANY'

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

## S4 method for signature 'mrgmod, missing'

```r
idata_set(x, object, ...)
```

Arguments

- **x**
  - model object
- **data**
  - a data set that can be coerced to data.frame
- **...**
  - passed along
- **.subset**
  - an unquoted expression passed to dplyr::filter; retain only certain rows in the data set
- **.select**
  - passed to dplyr::select; retain only certain columns in the data set; this should be the result of a call to dplyr::vars()
- **object**
  - character name of an object existing in $ENV to use for the data set
- **need**
  - passed to inventory
Details

The idata_set is a data.frame that specifies individual-level data for the problem. An ID column is required and there can be no more than one row in the data frame for each individual.

In most cases, the columns in the idata_set have the same names as parameters in the param list. When this is the case, the parameter set is updated as the simulation proceeds once at the start of each individual. The ‘idata_set’ can also be used to set initial conditions for each individual: for a compartment called CMT, make a column in idata_set called CMT_0 and make the value the desired initial value for that compartment. Note that this initial condition will be over-ridden if you also set the CMT_0 in $MAIN.

The most common application of idata_set is to specify a population or batch of simulations to do. We commonly use idata_set with an event object (see ev). In that case, the event gets applied to each individual in the idata_set.

It is also possible to provide both a data_set and a idata_set. In this case, the idata_set is used as a parameter lookup for IDs found in the data_set. Remember in this case, it is the data_set (not the idata_set) that determines the number of individuals in the simulation.

An error will be generated if any parameter columns in the input idata set contain NA.

See Also

data_set, ev

Examples

```r
mod <- mrgsolve::house()
data(exidata)
exidata

mod %>%
idata_set(exidata, ID <= 2) %>%
ev(amt = 100) %>%
mrgsim() %>%
plot()

mod %>%
idata_set(exidata) %>%
ev(amt = 100) %>%
mrgsim()

mod %>% ev(amt = 100) %>% mrgsim(idata=exidata)
```
Methods for working with the model compartment list

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.

Usage

```r
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, ...)```

Arguments

- `.x` the model object
- `...` passed 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

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 coerced to a list or numeric R object.

Value

an object of class `cmt_list` (see `numericlist`)
Examples
```r
## 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))
```

## inventory

Check whether all required parameters needed in a model are present in an object

Description
Check whether all required parameters needed in a model are present in an object

Usage
```
inventory(x, obj, ..., .strict = FALSE)
```

Arguments
```
x model object
obj data.frame to pass to idata_set or data_set
... capture dplyr-style parameter requirements
.strict whether to stop execution if all requirements are present (TRUE) or just warn (FALSE); see details
```

Details
If parameter requirements are not explicitly stated, the requirement defaults to all parameter names in x. Note that, by default, the inventory is not .strict unless the user explicitly states the parameter requirement. That is, if parameter requirements are explicitly stated, .strict will be set to TRUE if a value .strict was not passed in the call.

Value
original mrgmod
Examples

```r
## Not run:
inventory(mod, idata, CL:V) # parameters defined, inclusively, CL through Volume
inventory(mod, idata, everything()) # all parameters
inventory(mod, idata, contains("OCC")) # all parameters containing OCC
inventory(mod, idata, -F) # all parameters except F

## End(Not run)
```

---

is.mrgmod

Check if an object is a model object

Description

The function checks to see if the object is either `mrgmod` or `packmod`.

Usage

```r
is.mrgmod(x)
```

Arguments

- `x`: any object

Value

TRUE if `x` inherits `mrgsims`.

---

is.mrgsims

Check if an object is mrgsim output

Description

Check if an object is mrgsim output

Usage

```r
is.mrgsims(x)
```

Arguments

- `x`: any object

Value

TRUE if `x` inherits `mrgsims`. 
**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

---

**loadso**

_Load the model shared object_

**Description**

Once the model is compiled, the model object can be used to re-load the model shared object (the compiled code underlying the mode) when the simulation is to be done in a different R process.

**Usage**

`loadso(x, ...)`

````
## S3 method for class 'mrgmod'
loadso(x, ...)
```

**Arguments**

- `x`: the model object
- `...`: not used
Details

The ‘loadso’ function most frequently needs to be used when parallelizing simulations across worker nodes. The model can be run after calling ‘loadso’, without requiring that it is re-compiled on worker nodes. It is likely required that the model is built (and the shared object stored) in a local directory off of the working R directory (see the second example).

Value

The model object (invisibly).

Examples

```r
## Not run:
mod <- mread("pk1", modlib())
loadso(mod)

mod2 <- mread("pk2", modlib(), soloc = "build")
loadso(mod2)
## End(Not run)
```

matrix_helpers

Create matrices from vector input

Description

Create matrices from vector input

Usage

```r
bmatt(..., correlation = FALSE, digits = -1)
cmatt(..., digits = -1)
dmatt(...)
```

Arguments

```r
...
  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.

**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)

**Description**

This is a convenience function that ultimately calls mread. Model code is written to a file and read back in using mread.

**Usage**

mcode(model, code, project =getOption("mrgsolve.project", tempdir()), ...)

mcode_cache(
  model,
  code,
  project =getOption("mrgsolve.project", tempdir()),
  ...
)

**Arguments**

model model name
code character string specifying a mrgsolve model
project project name
... passed to mread; see that help topic for other arguments that can be set
mcRNG

Details

Note that the arguments are in slightly different order than \texttt{mread}. The default project is \texttt{tempdir()}. See the \texttt{mread} help topic for discussion about caching compilation results with \texttt{mcode_cache}.

See Also

\texttt{mread, mread_cache}

Examples

```r
## Not run:
code <- '  $CMT DEPOT CENT  
$PKMODEL ncmt=1, depot=TRUE  
$MAIN  
double CL = 1;  
double V = 20;  
double KA = 1;  
'

mod <- mcode("example",code)

## End(Not run)
```

---

mcRNG

\textit{Set RNG to use L'Ecuyer-CMRG}

Description

Set RNG to use L'Ecuyer-CMRG

Usage

\texttt{mcRNG()}
modlib

Internal model library

Description

Internal model library

Usage

modlib(model = NULL, ..., list = FALSE)

Arguments

model character name of a model in the library
...
  passed to mread_cache
list list available models

Details

See modlib_details, modlib_pk, modlib_pkpd, modlib_tmdd, modlib_viral for details.

Call modlib("<modelname>") to compile and load a mode from the library.

Call modlib(list=TRUE) to list available models. Once the model is loaded (see examples below),
call as.list(mod)$code to see model code and equations.

Examples

## Not run:
mod <- mread("pk1cmt", modlib())
mod <- mread("pk2cmt", modlib())
mod <- mread("pk3cmt", modlib())
mod <- mread("pk1", modlib())
mod <- mread("pk2", modlib())
mod <- mread("popex", modlib())
mod <- mread("irm1", modlib())
mod <- mread("irm2", modlib())
mod <- mread("irm3", modlib())
mod <- mread("irm4", modlib())
mod <- mread("emax", modlib())
mod <- mread("effect", modlib())
mod <- mread("tmdd", modlib())
mod <- mread("viral1", modlib())
mod <- mread("viral2", modlib())
mod <- mread("pred1", modlib())
mod <- mread("pbpk", modlib())
mod <- mread("1005", modlib()) # embedded NONMEM result
mrgsolve:::code(mod)
modlib: PK/PD Model parameters, compartments, and output variables

### Description

modlib: PK/PD Model parameters, compartments, and output variables

### Compartments

- EV1, EV2: extravascular dosing compartments
- CENT: central PK compartment
- PERIPH: peripheral PK compartment
- PERIPH2: peripheral PK compartment 2
- RESP: response PD compartment (irm models)

### Output variables

- CP: concentration in the central compartment (CENT/VC)
- RESP: response (emax model)

### PK parameters

- KA1, KA2: first order absorption rate constants from first and second extravascular compartment (1/time)
- CL: clearance (volume/time)
- VC: volume of distribution, central compartment (volume)
- VP: volume of distribution, peripheral compartment (volume)
- VP2: volume of distribution, peripheral compartment 2 (volume)
- Q: intercompartmental clearance (volume/time)
- Q2: intercompartmental clearance 2 (volume/time)
- VMAX: maximum rate, nonlinear process (mass/time)
- KM: Michaelis constant (mass/volume)
- K10: elimination rate constant (1/time); CL/VC
- K12: rate constant for transfer to peripheral compartment from central (1/time); Q/VC
- K21: rate constant for transfer to central compartment from peripheral (1/time); Q/VP
PD parameters

- $E_0$: baseline effect (emax model)
- $EMAX$, $IMAX$: maximum effect (response)
- $EC50$, $IC50$: concentration producing 50 percent of effect (mass/volume)
- $KIN$: zero-order response production rate (irm models) (response/time)
- $KOUT$: first-order response elimination rate (irm models) (1/time)
- $n$: sigmoidicity factor
- $KE0$: rate constant for transfer to effect compartment (1/time)

Description

modlib: Pharmacokinetic models

Arguments

... passed to update

Details

See modlib_details for more detailed descriptions of parameters and compartments.

The pk1cmt model is parameterized in terms of $CL$, $VC$, $KA1$ and $KA2$ and uses compartments EV1, EV2, and CENT. The pk2cmt model adds a PERIPH compartment and parameters $Q$ and $VP$ to that of the one-compartment model. Likewise, the three-compartment model (pk3cmt) adds PERIPH2 and parameters $Q2$ and $VP2$ to that of the two-compartment models. All pk models also have parameters $VMAX$ (defaulting to zero, no non-linear clearance) and $KM$.

Value

an object of class packmod

Model description

All pk models have two extravascular dosing compartments and potential for linear and nonlinear clearance.

- pk1cmt: one compartment pk model using ODEs
- pk2cmt: two compartment pk model using ODEs
- pk3cmt: three compartment pk model using ODEs
- pk1: one compartment pk model in closed-form
- pk2: two compartment pk model in closed-form
- popex: a simple population pk model
modlib: Pharmacokinetic / pharmacodynamic models

Description

modlib: Pharmacokinetic / pharmacodynamic models

Details

See modlib_details for more detailed descriptions of parameters and compartments.

All PK/PD models include 2-compartment PK model with absorption from 2 extravascular compartments and linear + nonlinear clearance. The PK models are parameterized with CL, VC, Q, VMAX, KM, KA1 and KA2 and implement compartments EV1, EV2, CENT, PERIPH. The indirect response models have compartment RESP and the emax model has output variable RESP. PD parameters include KIN, KOUT, IC50, EC50, IMAX, EMAX, E0, and n.

Also, once the model is loaded, use see method for mrgmod to view the model code.

Model description

- irm1 inhibition of response production
- irm2 inhibition of response loss
- irm3 stimulation of response production
- irm4 stimulation of response loss
- pd_effect effect compartment model
- emax sigmoid emax model

modlib: Target mediated disposition model

Description

modlib: Target mediated disposition model

Arguments

... passed to update
Parameters

- KEL: elimination rate constant
- KTP: tissue to plasma rate constant
- KPT: plasma to tissue rate constant
- VC: volume of distribution
- KA1, KA2: absorption rate constants
- KINT: internalization rate constant
- KON: association rate constant
- KOFF: dissociation rate constant
- KSYN: target synthesis rate
- KDEG: target degradation rate constant

Compartments

- CENT: unbound drug in central compartment
- TISS: unbound drug in tissue compartment
- REC: concentration of target
- RC: concentration of drug-target complex
- EV1, EV2: extravascular dosing compartments

Output variables

- CP: unbound drug in the central compartment
- TOTAL: total concentration of target (complexed and uncomplexed)

modlib_viral

modlib: HCV viral dynamics models

Description

modlib: HCV viral dynamics models

Models

- viral1: viral dynamics model with single HCV species
- viral2: viral dynamics model with wild-type and mutant HCV species
Parameters

- \(s\): new hepatocyte synthesis rate (cells/ml/day)
- \(d\): hepatocyte death rate constant (1/day)
- \(p\): viral production rate constant (copies/cell/day)
- \(\beta\): new infection rate constant (ml/copy/day)
- \(\delta\): infected cell death rate constant (1/day)
- \(c\): viral clearance rate constant (1/day)
- \(\text{fit}\): mutant virus fitness
- \(N\): non-target hepatocytes
- \(\mu\): forward mutation rate
- \(T_{\text{max}}\): maximum number of target hepatocytes (cells/ml)
- \(\rho\): maximum hepatocyte regeneration rate (1/day)

Compartments

- \(T\): uninfected target hepatocytes (cells/ml)
- \(I\): productively infected hepatocytes (cells/ml)
- \(V\): hepatitis C virus (copies/ml)
- \(\text{IM}\): mutant infected hepatocytes (cells/ml)
- \(\text{VM}\): mutant hepatitis C virus (copies/ml)
- \(\text{expos}\): exposure metric to drive pharmacodynamic model

---

**mread**

Read a model specification file

---

Description

`mread` reads and parses the `mrgsolve` model specification file, builds the model, and returns a model object for simulation. `mread_cache` does the same, but caches the compilation result for later use.

Usage

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

mread_cache(
  model = NULL,
  project = getOption("mrgsolve.project", getwd()),
  file = paste0(model, ".cpp"),
  code = NULL,
  soloc = getOption("mrgsolve.soloc", tempdir()),
  quiet = FALSE,
  preclean = FALSE,
  capture = NULL,
  ...
)

mread_file(file, ...)

Arguments

model            model name
project          location of the model specification file and any headers to be included; see also
                 the discussion about model; this argument can be set via options() library
                 under details as well as the modlib help topic
code             a character string with model specification code to be used instead of a model
                 file
file             the full file name (with extension, but without path) where the model is specified
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          logical; if TRUE, the model will be built
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            the directory location where the model shared object is built and stored; see
details; this argument can be set via options(); if the directory does not exist,
‘mread’ will attempt to create it.
capture a character vector or comma-separated string of additional model variables to capture; these variables will be added to the capture list for the current call to mread only
preclean logical; if TRUE, compilation artifacts are cleaned up first
recover if TRUE, an object will be returned in case the model shared object fails to build passed to update; also arguments passed to mread from mread_cache.

Details

The model argument is required. For typical use, the file argument is omitted and the value for file is generated from the value for model. To determine the source file name, mrgsolve will look for a file extension in model. A file extension is assumed when it finds a period followed by one to three alpha-numeric characters at the end of the string (e.g. mymodel.txt but not my.model). If no file extension is found, the extension .cpp is assumed (e.g. file is <model-name>.cpp). If a file extension is found, file is <model-name>.

Best practice is to avoid using . in model unless you are using model to point to the model specification file name. Otherwise, use mread_file.

Use the soloc argument to specify a directory location for building the model. This is the location where the model shared object will be stored on disk. The default is a temporary directory, so compilation artifacts are lost when R restarts when the default is used. Changing soloc to a persistent directory location will preserve those artifacts across R restarts. Also, if simulation from a single model is being done in separate processes on separate compute nodes, it might be necessary to store these compilation artifacts in a local directory to make them accessible to the different nodes. If the soloc directory does not exist, ‘mread’ will attempt to create it.

Similarly, using mread_cache will cache results in the temporary directory and the cache cannot be accessed after the R process is restarted.

Model Library

mrgsolve comes bundled with several pre coded PK, PK/PD, and other systems models that are accessible via the mread interface.

Models available in the library include:

- PK models: pk1cmt, pk2cmt, pk3cmt, pk1, pk2, popex, tmdd
- PKPD models: irm1, irm2, irm3, irm4, emax, effect
- Other models: viral1, viral2

When the library model is accessed, mrgsolve will compile and load the model as you would for any other model. It is only necessary to reference the correct model name and point the project argument to the mrgsolve model library location via modlib.

For more details, see modlib_pk, modlib_pkpd, modlib_tmdd, modlib_viral, and modlib_details for more information about the state variables and parameters in each model.

See Also

mcode, mcode_cache
mrgsim

### Examples

```r
## Not run:
code <-  
$PARAM CL = 1, VC = 5
$CMT CENT
$ODE dxdt_CENT = -(CL/VC)*CENT;
' 

mod <- mcode("ex_mread", code)

mod

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

mod <- mread("irm3", modlib())

mod

# if the model is in the file mymodel.cpp
mod <- mread("mymodel")

# if the model is in the file mymodel.txt
mod <- mread(file = "mymodel.txt")

or

mod <- mread_file("mymodel.txt")

## End(Not run)
```

---

**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. Use `mrgsim_q()` instead to benchmark mrgsolve or to do repeated quick simulation for tasks like parameter optimization, sensitivity analyses, or optimal design. See `mrgsim_variants` for other mrgsim-like functions that have more focused inputs. `mrgsim_df` coerces output to data.frame prior to returning.

**Usage**

```r
mrgsim(x, data = NULL, idata = NULL, events = NULL, nid = 1, ...)
```
mrgsim_df(..., output = "df")

do_mrgsim(
  x,
  data,
  idata = no_idata_set(),
  carry_out = carry.out,
  carry.out = character(0),
  recover = character(0),
  seed = as.integer(NA),
  Request = character(0),
  output = NULL,
  capture = NULL,
  obsonly = FALSE,
  obsaug = FALSE,
  tgrid = NULL,
  recsort = 1,
  deslist = list(),
  descol = character(0),
  filbak = TRUE,
  tad = FALSE,
  nocb = TRUE,
  skip_init_calc = FALSE,
  ss_n = 500,
  ss_fixed = FALSE,
  interrupt = 256,
  ...)

Arguments

x          the model object
data       NMTRAN-like data set (see data_set())
idata      a matrix or data frame of model parameters, one parameter per row (see idata_set())
events     an event object
nid        integer number of individuals to simulate; only used if idata and data are missing
...        passed to update() and do_mrgsim()
output     if NULL (the default) a mrgsims object is returned; otherwise, pass df to return a
data.frame or matrix to return a matrix
carry_out  numeric data items to copy into the output
carry.out  soon to be deprecated; use carry_out instead
recover    character column names in either data or idata to join back (recover) to simulated data; may be any class (e.g. numeric, character, factor, etc)
seed       deprecated
Request compartments or captured variables to retain in the simulated output; this is different than the request slot in the model object, which refers only to model compartments.

capture character file name used for debugging (not related to $CAPTURE).

obsonly if TRUE, dosing records are not included in the output.

obsaug augment the data set with time grid observations; 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.

tgrid a tgrid object; or a numeric vector of simulation times or another object with a stime method.

recsort record sorting flag. 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.

deslist a list of tgrid objects.

descol the name of a column for assigning designs.

filbak carry data items backward when the first data set row has time greater than zero.

tad when TRUE a column is added to simulated output is added showing the time since the last dose. Only data records with evid == 1 will be considered doses for the purposes of tad calculation. The tad can be properly calculated with a dosing lag time in the model as long as the dosing lag time (specified in $MAIN) is always appropriate for any subsequent doses scheduled through addl. This will always be true if the lag time doesn’t change over time. But it might (possibly) not hold if the lag time changes prior to the last dose in the addl sequence. This known limitation shouldn’t affect tad calculation in most common dosing lag time implementations.

nocb if TRUE, use next observation carry backward method; otherwise, use locf.

skip_init_calc don’t use $MAIN to calculate initial conditions.

ss_n maximum number of iterations for determining steady state for the PK system; a warning will be issued if steady state is not achieved within ss_n iterations when ss_fixed is TRUE.

ss_fixed if FALSE (the default), then a warning will be issued if the system does not reach steady state within ss_n iterations given the model tolerances rtol and atol; if TRUE, the number of iterations for determining steady state are capped at ss_n and no warning will be issued if steady state has not been reached within ss_n dosing iterations. To silence warnings related to steady state, set ss_fixed to TRUE and set ss_n as the maximum number of iterations to try when advancing the system for steady state determination.

interrupt integer check user interrupt interval; when interrupt is a positive integer, the simulation will check for the user interrupt signal every interrupt simulation records; pass a negative number to never check for the user interrupt interval.
Details

- Use `mrgsim_df()` to return a data frame rather than `mrgsims` object
- Both data and `idata` will be coerced 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 individual’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`
- `recover`: this is similar to `carry_out` with respect to end result, but it uses a different process. Columns to be recovered are cached prior to running the simulation, and then joined back on to the simulated data. So, whereas `carry_out` will only accept numeric data items, `recover` can handle data frame columns of any type. There is a small decrease in performance with `recover` compared to `carry_out`, but it is likely that the performance difference is difficult to perceive (when the simulation runs very fast) or only a small fractional increase in run time when the simulation is very large. And any performance hit is likely to be well worth it in light of the convenience gain. Just think carefully about using this feature when every millisecond counts.

Value

An object of class `mrgsims`

See Also

`mrgsim_variants`, `mrgsim_q()`

Examples

```r
## example("mrgsim")

e <- ev(amt = 1000)

mod <- mrgsolve::house()

out <- mod %>% ev(e) %>% mrgsim()

plot(out)

out <- mod %>% ev(e) %>% mrgsim(end=22)

out

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

gout

gout <- mod %>% mrgsim(data=exTheoph)

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

gout

gout <- mod %>% mrgsim(data=exTheoph, obsaug=TRUE, carry_out="a.u.g")

gout

gout <- mod %>% ev(e) %>% mrgsim(outvars="CP,RESP")

gout

g a <- ev(amt = 1000, group = 'a')
g b <- ev(amt = 750, group = 'b')
g data <- as_data_set(a,b)

gout <- mrgsim_d(mod, data, recover="group")

gout

mrgsims_dplyr  Methods for handling output with dplyr verbs

Description

These methods modify the data in a mrgsims object and return a data frame. Contrast with the functions in mrgsims_modify.

Usage

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

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

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

g## S3 method for class 'mrgsims'
distinct(.data, ..., .keep_all = FALSE)
Arguments

.data an mrgsims object; passed to various dplyr functions

... passed to other methods

add passed to dplyr::group_by (for dplyr < 1.0.0)

.add passed to dplyr::group_by (for dplyr >= 1.0.0)

.keep_all passed to dplyr::distinct

funs passed to dplyr::summarise_each

dots passed to various dplyr functions

data_ mrgsims object

x passed to dplyr::as.tbl

Details

For the select_sims function, the dots ... must be either compartment names or variables in $CAPTURE. An error will be generated if no valid names are selected or the names for selection are not found in the simulated output.

See Also

mrgsims_modify
Examples

```r
out <- mrgsim(house(), events = ev(amt = 100), end = 5, delta=1)
dplyr::filter(out, time==2)
dplyr::mutate(out, label = "abc")
dplyr::select(out, time, RESP, CP)
```

---

### mrgsims_modify

**Methods for modifying mrgsims objects**

**Description**

These functions modify the simulated data in an mrgsims object and return the modified object. Contrast with the functions in mrgsims_dplyr.

**Usage**

```r
mutate_sims(.data, ...)
select_sims(.data, ...)
filter_sims(.data, ...)
```

**Arguments**

- `.data` a mrgsims object
- `...` other arguments passed to the dplyr functions

**See Also**

- mrgsims_dplyr

**Examples**

```r
out <- mrgsim(house(), events = ev(amt = 100))
filter_sims(out, time > 2)
mutate_sims(out, label = "abc")
select_sims(out, RESP, CP)
```
**mrgsim_q**

*Simulate from a model object with quicker turnaround*

**Description**

Use the function when you would usually use `mrgsim_d`, but you need a quicker turnaround time. The timing differences might be difficult to detect for a single simulation run but could become appreciable with repeated simulation. See details for important differences in how `mrgsim_q` is invoked compared to `mrgsim` and `mrgsim_d`. This function should always be used for benchmarking simulation time with mrgsolve.

**Usage**

```r
mrgsim_q(
  x, data,
  recsort = 1,
  stime = numeric(0),
  output = "mrgsims",
  skip_init_calc = FALSE,
  simcall = 0
)
```

**Arguments**

- `x` a model object
- `data` a simulation data set
- `recsort` record sorting flag
- `stime` a numeric vector of observation times; these observation times will only be added to the output if there are no observation records in `data`
- `output` output data type; if `mrgsims`, then the default output object is returned; if "df" then a data frame is returned
- `skip_init_calc` don’t use $MAIN to calculate initial conditions
- `simcall` not used; only the default value of 0 is allowed

**Details**

This function does not support the piped simulation workflow. All arguments must be passed into the function except for `x`.

A data set is required for this simulation workflow. The data set can have only dosing records or doses with observations. When the data set only includes doses, a single numeric vector of observation times should be passed in.

This simulation workflow does not support Req (request) functionality. All compartments and captured variables will always be returned in the simulation output.
This simulation workflow does not support carry-out functionality. This simulation workflow does not accept arguments to be passed to `update`. This must be done by a separate call to `update`.

This simulation workflow does not support use of event objects. If an event object is needed, it should be converted to a data set prior to the simulation run (see `as_data_set` or `as.data.frame.ev`). This simulation workflow does not support idata sets or any feature enabled by idata set use. Individual level parameters should be joined onto the data set prior to simulation. Otherwise `mrgsim_i` or `mrgsim_ei` should be used.

By default, a mrgsims object is returned (as with `mrgsim`). Use the output="df" argument to request a plain data.frame of simulated data on return.

**See Also**

`mrgsim`, `mrgsim_variants`, `qsim`

**Examples**

```r
mod <- mrgsolve::house()
data <- expand.ev(amt = c(100, 300, 1000))
out <- mrgsim_q(mod, data)
out
```

**Description**

These functions are called by `mrgsim()` and have explicit input requirements written into the function name. The motivation behind these variants is to give the user a clear workflow with specific, required inputs as indicated by the function name. Use `mrgsim_q()` instead to benchmark mrgsolve or to do repeated quick simulation for tasks like parameter optimization, sensitivity analyses, or optimal design.

**Usage**

```r
mrgsim_e(x, events, idata = NULL, data = NULL, ...)
mrgsim_d(x, data, idata = NULL, events = NULL, ...)
mrgsim_ei(x, events, idata, data = NULL, ...)
mrgsim_di(x, data, idata, events = NULL, ...)
```
mrgsim_i(x, idata, data = NULL, events = NULL, ...)
mrgsim_0(x, idata = NULL, data = NULL, events = NULL, ...)

Arguments

x the model object
events an event object
idata a matrix or data frame of model parameters, one parameter per row (see idata_set())
data NMTRAN-like data set (see data_set())
... passed to update() and do_mrgsim()

Details

**Important**: all of these functions require that data, idata, and/or events be pass directly to the functions. They will not recognize these inputs from a pipeline.

- mrgsim_e simulate using an event object
- mrgsim_ei simulate using an event object and idata_set
- mrgsim_d simulate using a data_set
- mrgsim_di simulate using a data_set and idata_set
- mrgsim_i simulate using an idata_set
- mrgsim_0 simulate using just the model
- mrgsim_q simulate from a data set with quicker turnaround (see mrgsim_q())

See Also

mrgsim(), mrgsim_q(), qsim()

mrgsolve mrgsolve

Description

mrgsolve is an R package maintained under the auspices of Metrum Research Group 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. mrgsolve consists of computer code written in the R and C++ languages, providing an interface to a C++ translation of the lsoda differential equation solver. See aboutsolver for more information.
Resources

- Main mrgsolve resource page: https://mrgsolve.github.io
- User guide: https://mrgsolve.github.io/user_guide/
- Web vignettes: https://mrgsolve.github.io/vignettes/

Package-wide options

- `mrgsolve.project`: sets the default project director (mread)
- `mrgsolve.soloc`: sets the default package build directory (mread)
- `mrgsolve_mread_quiet`: don’t print messages during mread
- `mrgsolve.update.strict`: if TRUE, print warning when trying to update an item in the model object that doesn’t exist

Examples

```r
## example("mrgsolve")
mod <- mrgsolve::house(delta=0.1) %>% param(CL=0.5)

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

events

mod

see(mod)

## Not run:
stime(mod)

## End(Not run)
param(mod)

init(mod)

out <- mod %>% ev(events) %>% mrgsim(end=168)

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

plot(out, GUT+CP~.)

sims <- as.data.frame(out)

t72 <- dplyr::filter(sims, time==72)

str(t72)
```
ida <- data.frame(ID=c(1,2,3), CL=c(0.5,1,2), VC=12)
out <- mod %>% ev(events) %>% mrgsim(end=168, idata=ida, req="")
plot(out)

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

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

out <- mod %>% ev(events) %>% mrgsim(end=180, req="")
plot(out)

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

out <- mod %>% data_set(extran1) %>% mrgsim(end=200)
plot(out, CP~time|factor(ID))

## idata
data(exidata)

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

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

# Internal model library
## Not run:
mod <- mread("irm1", modlib())

mod

x <- mod %>% ev(amt=300, ii=12, addl=3) %>% mrgsim

## End(Not run)
names,mrgmod-method

Description

dplyr verbs for event objects

Usage

## S3 method for class 'ev'
mutate(.data, ...)

## S3 method for class 'ev'
select(.data, ...)

## S3 method for class 'ev'
filter(.data, ...)

Arguments

.data the event object

... passed to the dplyr function

names,mrgmod-method  Get all names from a model object

Description

Get all names from a model object

Usage

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

Arguments

x the model object

Examples

mod <- mrgsolve::house()
names(mod)
nmext

Import model estimates from a NONMEM ext file

Description

Import model estimates from a NONMEM ext file

Usage

nmext(
  run = NA_real_,
  project = getwd(),
  file = paste0(run, ".ext"),
  path = NULL,
  root = c("working", "cppfile"),
  index = "last",
  theta = TRUE,
  omega = TRUE,
  sigma = TRUE,
  olabels = NULL,
  slabels = NULL,
  oprefix = "",
  sprefix = "",
  tname = "THETA",
  oname = "...",
  sname = "...",
  read_fun = "data.table",
  env = NULL
)

Arguments

run run number
project project directory
file deprecated; use path instead
path full path to NONMEM ext file
root the directory that ‘path’ and ‘project’ are relative to; this is currently limited to
the ‘working’ directory or ‘cppdir’, the directory where the model file is located
index the estimation number to return; "last" will return the last estimation results;
otherwise, pass an integer indicating which estimation results to return
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
**nmxml**

Import model estimates from a NONMEM xml file

---

**Description**

Import model estimates from a NONMEM xml file

**Usage**

```r
nmxml(
    run = numeric(0),
    project = character(0),
    file = character(0),
    path = character(0),
    root = c("working", "cppfile"),
    theta = TRUE,
    omega = TRUE,
    sigma = TRUE,
    olabels = NULL,
    slabels = NULL,
    oprefix = "",
    sprefix = "",
    tname = "THETA",
    oname = "...",
    sname = "...",
    index = "last",
    xpath = ".//nm:estimation",
    env = NULL
)
```
Arguments

- **run**: run number
- **project**: project directory
- **file**: deprecated; use path instead
- **path**: the complete path to the run.xml file
- **root**: the directory that ‘path’ and ‘project’ are relative to; this is currently limited to the ‘working’ directory or ‘cppdir’, the directory where the model file is located
- **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
- **index**: the estimation number to return; "last" will return the last estimation results; otherwise, pass an integer indicating which estimation results to return
- **xpath**: xml path containing run results; if the default doesn’t work, consider using .//estimation as an alternative; see details
- **env**: internal

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.

This function requires that the xml2 package be installed and loadable. If requireNamespace("xml2") fails, an error will be generated.

nmxml usually expects to find run results in the xpath called .//nm:estimation. Occasionally, the run results are not stored in this namespace but no namespaces are found in the xml file. In this case, the user can specify the xpath containing run results. Consider trying .//estimation as an alternative if the default fails.

Value

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

See Also

nmext
Examples

```r
if(requireNamespace("xml2")) {
  proj <- system.file("nonmem", package = "mrgsolve")
  mrgsolve::nmxml(run = 1005, project = proj)
}
```

**numerics_only**

Prepare data.frame for input to mrgsim

**Description**

Prepare data.frame for input to mrgsim

**Usage**

```r
numerics_only(x, quiet = FALSE, convert_lgl = FALSE)
```

**Arguments**

- `x`: a input data set
- `quiet`: logical indicating whether or not warnings should be printed
- `convert_lgl`: if TRUE, convert logical columns with `as.integer`

**obsaug**

Augment observations in the simulated output

**Description**

Augment observations in the simulated output

**Usage**

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

**Arguments**

- `x`: model object
- `value`: the value for obsaug
- `...`: passed along There is also a obsaug argument to `mrgsim` that can be set to accomplish the same thing as a call to obsaug in the pipeline.
**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**

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

## S4 method for signature 'missing'

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

## S4 method for signature 'matrix'

```r
omat(.x, ..., labels = list())  
```

## S4 method for signature '\texttt{NULL}'

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

## S4 method for signature 'list'

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

---

**obsonly**

Collect only observations in the simulated output

**Description**

Collect only observations in the simulated output

**Usage**

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

**Arguments**

- `x`: model object
- `value`: the value for `obsonly`
- `...`: passed along

**Details**

There is also a `obsonly` argument to `mrgsim` that can be set to accomplish the same thing as a call to `obsonly` in the pipeline.
## S4 method for signature 'omegalist'
omat(.x, ...)

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

## 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
- `open` passed to `merge.list`
- `x` matlist object

### Examples

```r
## 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
  1 0.1 2

$OMEGA \@cor
\@ prefix ETA_
\@ labels CL VC KA
  0.1
```
outvars

Show names of current output variables

Description

Show names of current output variables

Usage

outvars(x, unlist = FALSE)

Arguments

x mrgmod object
unlist if TRUE then a character vector (rather than list) is returned

Value

When unlist is FALSE (default): a named list, with cmt showing names of output compartments and capture giving names of output variables in capture. When unlist is TRUE, then a single, unnamed character vector of outvar names is returned.

Examples

outvars(mrgsolve::house())

param

Create and work with parameter objects

Description

See numericlist for methods to deal with parameter_list objects.
Usage

    param(.x, ...)

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

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

## S4 method for signature 'missing'
    param(..., .strict = TRUE)

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

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

    allparam(.x)

Arguments

.x the 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
.strict if TRUE, all names to be updated must be found in the parameter list

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 coerced to a list or numeric R object.

Use allparam to get a parameter_list object including both model parameters and data items listed in $FIXED.

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

param(mod)[["KA"]]

as.list(param(mod))

as.data.frame(param(mod))

param(mod, CL = 1.2)

ew_values <- list(CL = 1.3, VC = 20.5)

param(mod, new_values)

PKMODEL

Parse PKMODEL BLOCK data

Description

Parse PKMODEL BLOCK data

Usage

PKMODEL(
  ncmt = 1,
  depot = FALSE,
  cmt = NULL,
  trans = pick_trans(ncmt, depot),
  env = list(),
  pos = 1,
  ...
)

Arguments

ncmt number of compartments; must be 1 (one-compartment, not including a depot dosing compartment) or 2 (two-compartment model, not including a depot dosing compartment)

depot logical indicating whether to add depot compartment

cmt compartment names as comma-delimited character

trans the parameterization for the PK model; must be 1, 2, 4, or 11
Details

When using `$PKMODEL` certain symbols must be defined in the model specification depending on the value of `ncmt`, `depot` and `trans`.

- `ncmt 1`, `depot FALSE`, `trans 2`: `CL`, `V`
- `ncmt 1`, `depot TRUE`, `trans 2`: `CL`, `V`, `KA`
- `ncmt 2`, `depot FALSE`, `trans 4`: `CL`, `V1`, `Q`, `V2`
- `ncmt 2`, `depot TRUE`, `trans 4`: `CL`, `V2`, `Q`, `V3`, `KA`

If `trans=11` is specified, use the symbols listed above for the `ncmt`/`depot` combination, but append `i` at the end (e.g. `CLi` or `Qi` or `KAi`).

If `trans=1`, the user must utilize the following symbols:

- `pred_CL` for clearance
- `pred_V` or `pred_V2` for central compartment volume of distribution
- `pred_Q` for intercompartmental clearance
- `pred_V3` for peripheral compartment volume of distribution
- `pred_KA` for absorption rate constant

See Also

`BLOCK_PARSE`

Description

Plot method for `mrgsims` objects

Usage

```r
## S4 method for signature 'batch_mrgsims,missing'
plot(x, yval = variables(x), auto.key = list(), mincol = 3, ...)

## S4 method for signature 'batch_mrgsims,formula'
plot(
  x,
  y,
```
plot_mrgsims

Generate a quick plot of simulated data

Arguments

x
  mrgsims object
yval
  y variables to plot
auto.key
  passed to xyplot
mincol
  minimum number of columns in key
...
  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
scales
  passed to xyplot

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,
  outer = TRUE,
  type = "l",
  lwd = 2,
  ylab = "value",
  show.grid = TRUE,
  lwd = 2,
  type = "l",
  yval = variables(x),
  auto.key = list(columns = 1),
  scales = list(y = list(relation = "free")),
  ...
)
plot_mrgsims

```r
groups = ID,
scales = list(y = list(relation = "free"),
logy = FALSE,
logbr = 1,
...
)
```

## S4 method for signature 'mrgsims,character'
plot(x, y, ...)

### 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
- **outer**: passed to xyplot
- **type**: passed to xyplot
- **lwd**: passed to xyplot
- **ylab**: passed to xyplot
- **groups**: passed to xyplot
- **scales**: passed to xyplot
- **logy**: plot the y variables on log scale
- **logbr**: log scale breaks indicator; use 1 for breaks every log unit; use 3 for breaks every half log unit; use 0 for default breaks

### Examples

```r
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)
```

## Not run:
```
plot(out, "CP RESP, GUT")

## End(Not run)
```
Description

Plot data as an mrgsims object

Usage

plot_sims(.data, ..., .f = NULL, .dots = list())

Arguments

- `.data` a data frame
- `...` unquoted column names to plot on y-axis
- `.f` a formula to plot
- `.dots` extra arguments passed to `lattice::xyplot`

Details

This function is only intended for use with data frames that were created by modifying an mrgsims object.

Examples

```r
mod <- mrgsolve::house() %>% ev(amt = 100)
out <- mrgsim(mod)
out_df <- dplyr::mutate(out, time <= 72)
plot(out)
plot_sims(out, CP, RESP)
```

```r
# Not run:
plot_sims(out, .f = CP + RESP)
plot_sims(out, .f = CP + RESP ~ time)
# End(Not run)
```
qsim  Basic, simple simulation from model object

Description

This is just a lighter version of mrgsim, with fewer options. See Details.

Usage

qsim(
  x,
  data,
  idata = no_idata_set(),
  obsonly = FALSE,
  tgrid = NULL,
  recsort = 1,
  tad = FALSE,
  Req = NULL,
  outvars = Req,
  skip_init_calc = FALSE,
  output = "mrgsims"
)

Arguments

x          the model object
data       can be either event object or data set
idata      a matrix or data frame of model parameters, one parameter per row (see idata_set())
obsonly    if TRUE, dosing records are not included in the output
tgrid      a tgrid object; or a numeric vector of simulation times or another object with an stime method
recsort    record sorting flag. 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.
tad        when TRUE a column is added to simulated output is added showing the time since the last dose. Only data records with evid == 1 will be considered doses for the purposes of tad calculation. The tad can be properly calculated with a dosing lag time in the model as long as the dosing lag time (specified in $MAIN) is always appropriate for any subsequent doses scheduled through addl. This will always be true if the lag time doesn’t change over time. But it might (possibly) not hold if the lag time changes prior to the last dose in the addl sequence. This known limitation shouldn’t affect tad calculation in most common dosing lag time implementations.
Req synonym for outvars
outvars output items to request; if missing, then only captured items will be returned in the output
skip_init_calc don’t use $MAIN to calculate initial conditions
output output data type; the default is mrgsims, which returns the default output object; other options include df (for data.frame) or matrix

Details

There is no pipeline interface for this function; all configuration options (see Arguments) must be passed as formal arguments to the function. You can’t carry_out, Request specific columns, or pass items in for update. Some other limitations, but only convenience-related. See Arguments for available options. Specifically, there is no ... argument for this function. Use the update() method to update the model object.

See Also

mrgsim_q, mrgsim, mrgsim_variants

Examples

mod <- mrgsolve::house()
dose <- ev(amt = 100)
out <- qsim(mod,dose)

read_nmext Extract estimates from NONMEM ext file

Description

Extract estimates from NONMEM ext file

Usage

read_nmext(
  run = NA_real_,
  project = getwd(),
  file = paste0(run, ".ext"),
  path = NULL,
  read_fun = c("data.table", "read.table"),
  index = "last"
)
Arguments

run a run number or run identifier
project the NONMEM project directory
file the ext file name
path full path and file name for ext file
read_fun function to read the ext file
index selects the table number whose results will be returned; use value "last" to select the last table in the .ext file; or pass an integer specifying the table number; in case there is exactly one table in the .ext file, pass the value "single" to bypass parsing the file to look for sub tables (this might be useful when BAYES analysis was performed as the only estimation method and there are 10000s of posterior samples in the file)

Value

A list with param, omega, and sigma in a format ready to be used to update a model object.

Examples

```r
project <- system.file("nonmem", package = "mrgsolve")
est <- read_nmext(1005, project = project)
est$param
est$omega
est$sigma
est <- read_nmext(2005, project = project, index = 3)
```

realize_addl

Make addl doses explicit in an event object or data set

Description

Make addl doses explicit in an event object or data set

Usage

```r
realize_addl(x, ...)
```

## S3 method for class 'data.frame'
```r
realize_addl(  
  x,
  ```
warn = FALSE,
mark_new = FALSE,
fill = c("inherit", "na", "locf"),
...
)

## S3 method for class 'ev'
realize_addl(x, ...)

 Arguments

 x a data_set data frame or an ev object (see details)
... not used
warn if TRUE a warning is issued if no ADDL or addl column is found
mark_new if TRUE, a flag is added to indicate new columns
fill specifies how to handle non-dose related data columns in new data set records; this option is critical when handling data sets with time-varying, non-dose-related data items; see details

 Details

 If no addl column is found the data frame is returned and a warning is issued if warn is true. If ii, time, or evid are missing, an error is generated.

 Use caution when passing in data that has non-dose-related data columns that vary within a subject and pay special attention to the fill argument. By definition, realize_addl will add new rows to your data frame and it is not obvious how the non-dose-related data should be handled in these new rows. When inherit is chosen, the new records have non-dose-related data that is identical to the originating dose record. This should be fine when these data items are not varying with time, but will present a problem when the data are varying with time. When locf is chosen, the missing data are filled in with NA and an last observation carry forward operation is applied to every column in the data set. This may not be what you want if you already had missing values in the input data set and want to preserve that missingness. When na is chosen, the missing data are filled in with NA and no locf operation is applied. But note that these missing values may be problematic for a mrgsolve simulation run. If you have any time-varying columns or missing data in your data set, be sure to check that the output from this function is what you were expecting.
Usage

render(x, ...)

## S4 method for signature 'character'
render(x, project = NULL, ...)

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

dorender(model, project, template = NULL, compile = TRUE, ...)

Arguments

x model object or the model name
...
project the directory containing the .cpp model file
model model name
template template document
compile logical; if true, the model will be compiled to run

Examples

## Not run:
mod <- mrgsolve::house()
mrgsolve:::render(mod)
mrgsolve:::render("irm2", modlib())

## End(Not run)

Req Request simulated output

Description

Use this function to select, by name, either compartments or derived variables that have been captured (see CAPTURE) into the simulated output.

Usage

Req(x, ...)

treq(x, ...)

## S3 method for class 'mrgmod'
treq(x, ...)

---

Raw Text:

**Usage**

```r
render(x, ...)

## S4 method for signature 'character'
render(x, project = NULL, ...)

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

dorender(model, project, template = NULL, compile = TRUE, ...)
```

**Arguments**

- `x`: model object or the model name
- `...`: passed to `rmarkdown::render`
- `project`: the directory containing the .cpp model file
- `model`: model name
- `template`: template document
- `compile`: logical; if true, the model will be compiled to run

**Examples**

```r
## Not run:
mod <- mrgsolve::house()
mrgsolve:::render(mod)
mrgsolve:::render("irm2", modlib())

## End(Not run)
```

---

**Description**

Use this function to select, by name, either compartments or derived variables that have been captured (see CAPTURE) into the simulated output.

**Usage**

```r
Req(x, ...)

treq(x, ...)

## S3 method for class 'mrgmod'
treq(x, ...)
```
Arguments

x model object

Details

There is also a `Req` argument to `mrgsim` that can be set to accomplish the same thing as a call to `Req` in the pipeline.

Note the difference between `req` and `Req`: the former only selects compartments to appear in output while the latter selects both compartments and captured items. Also, when there are items are explicitly listed in `Req`, all other compartments or captured items not listed there are ignored. But when compartments are selected with `req` all of the captured items are returned. Remember that `req` is strictly for compartments.

Examples

```r
mod <- mrgsolve::house()
mod %>% Req(CP, RESP) %>% ev(amt=1000) %>% mrgsim
```

---

`reserved`  
`Reserved words`

Description

Reserved words

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 covariances

Description

Get model random effect variances and covariances

Usage

revar(x, ...)

## S4 method for signature 'mrgmod'
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, raw = FALSE, ...)

Arguments

x  model object
...

Passed along
raw  return the raw code

Value

invisible NULL
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, open = FALSE)

## S4 method for signature `\`NULL``
smat(.x, ...)

## 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

open passed to merge.list

x matlist object
Examples

```r
## 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 calls to mrgsim*

Description

Access or clear arguments for calls to mrgsim

Usage

`simargs(x, which = NULL, clear = FALSE, ...)`

Arguments

- `x` model object
- `which` character with length 1 naming a single arg to get
- `clear` logical indicating whether or not to clear args from the model object
- `...` passed along

Value

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

Examples

```r
mod <- mrgsolve::house()
mod %>% Req(CP,RESP) %>% carry_out(evid,WT,FLAG) %>% simargs
```
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()
siloc(mod)

solversettings

Optional inputs for lsoda

Description

These are settings for the differential equation solver (lsoda) that can be accessed via the R interface. The code listing below is taken directly from the lsoda source code.

Details

The following items can be set

- hmax (HMAX below); decrease hmax when you want to limit how big of a step the solver can take when integrating from one time to the next time. However be aware that smaller hmax will result in longer run times.
- hmin (HMIN below); don’t fiddle with this unless you know what you’re doing.
- ixpr (IXPR below)
- maxsteps (MXSTEP below); increase this number when the solver has a long interval between two integration times (e.g. when observation records are far apart).
- mxhnile (MXHNIL below); don’t usually modify this one
• atol - the absolute solver tolerance; decrease this number (e.g. to 1E-10 or 1E-20 or 1E-50) when the value in a compartment can get extremely small; without this extra (lower) tolerance, the value can get so low that the number can randomly become negative. However be aware that more precision here will result in longer run times.

• rtol - the relative solver tolerances; decrease this number when you want a more precise solution. However be aware that more precision here will result in longer run times.

See Also

aboutsolver, update

summary.mrgmod

Print summary of a mrgmod object

Description

Print summary of a mrgmod object

Usage

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

Arguments

object a mrgmod object
...
not used

tscale

Re-scale time in the simulated output

Description

Re-scale time in the simulated output

Usage

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

Arguments

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

There is also a `tscale` argument to `mrgsim` that can be set to accomplish the same thing as a call to `tscale` in the pipeline.

Examples

```r
# The model is in hours:
mod <- mrgsolve::house()

# The output is in days:
mod %>% tscale(1/24) %>% mrgsim
```

---

**update**  
*Update the model object*

---

Description

After the model object is created, update various attributes.

Usage

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

## 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, ...)
```

Arguments

- **object**  
a model object
- **...**  
named items to update
- **merge**  
logical indicating to merge (rather than replace) new and existing attributes
- **open**  
logical; used only when merge is TRUE and parameter list or initial conditions list is being updated; if FALSE, no new items will be added; if TRUE, the parameter list may expand.
- **data**  
a list of items to update; this list is combined with any items passed in via .
- **strict**  
if TRUE, then an error will be generated if there is attempt to update a non-existent item
- **y**  
another object involved in update
- **.y**  
data to update
Details

Slots that can be updated:

- verbose
- debug
- preclean
- mindt
- digits
- atol - absolute solver tolerance; see solversettings
- rtol - relative solver tolerance; see solversettings
- ss_rtol - relative tolerance when finding steady state
- ss_atol - absolute tolerance when finding steady state
- ixpr - see IXPR in solversettings
- mxhnil - see MXHNIL in solversettings
- hmin - see HMIN in solversettings
- hmax - see HMAX in solversettings
- maxsteps - see MXSTEP in solversettings
- start, end, delta, add
- tscale
- request
- param
- init
- omega
- sigma
- outvars

Value

The updated model object is returned.

See Also

update, mrgmod-class, within

Examples

```r
## Not run:
mod <- mrgsolve::house()

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

## End(Not run)```
valid_data_set

Description

This function is called by mrgsim. Users may also call this function to pre-validate data when the same data set is used for repeated simulation.

Usage

valid_data_set(x, m = NULL, verbose = FALSE, quiet = FALSE)
valid_data_set.matrix(x, verbose = FALSE)

Arguments

x                data.frame or matrix
m                a model object
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.

See Also

valid_idata_set, idata_set, data_set

Examples

mod <- mrgsolve::house()
data(exTheoph)
d <- valid_data_set(exTheoph, mod)
valid_idata_set  Validate and prepare idata data sets for simulation

Description
Validate and prepare idata data sets for simulation

Usage
valid_idata_set(x, m, verbose = FALSE, quiet = FALSE)

Arguments
x  data.frame or matrix
m  a model object
verbose  logical
quiet  if TRUE, messages will be suppressed

Value
A numeric matrix with class valid_idata_set.

See Also
valid_data_set, idata_set, data_set

within  Update parameters, initials, and settings within a model object

Description
The main use case for using within rather than update or param or init is when you want to update to a new value that is calculated from the existing value. See the example in details

Usage
## S3 method for class 'mrgmod'
within(data, expr, ...)

Arguments
data  an object with class mrgmod
expr  expressions evaluated in an environment containing various model object components, including parameters, initial conditions, and others (see details)
...  not used
Details

Other model object slots that can be updated: `start`, `end`, `delta`, `add`, `rtol`, `atol`, `hmax`, `maxsteps`. These are included for convenience, but we expect that most of the time these will get updated through the update method.

See Also

`update`

Examples

```r
mod <- mrgsolve::house()
mod2 <- within(mod, {CL <- CL * 1.5})
mod$CL
mod2$CL
```

---

### zero_re

Zero out random effects in a model object

Description

Sets all elements of the OMEGA or SIGMA matrix to zero

Usage

```r
zero_re(.x, ...)
```

## S4 method for signature 'mrgmod'

```r
zero_re(.x, ...)
```

Arguments

- `.x` a model object
- `...` which matrix to zero out; pass `omega` to just zero out `omega`, `sigma` to just zero out `sigma`; passing nothing will zero out both

Value

An updated object.
### $ev-method

Select columns from an ev object

#### Description

Select columns from an ev object

#### Usage

```r
## S4 method for signature 'ev'
x$name

## S4 method for signature 'ev'
x[[i, exact = TRUE]]
```

#### Arguments

- `x`: ev object
- `name`: column to select
- `i`: an element to select
- `exact`: not used

### $mrgmod-method

Select parameter values from a model object

#### Description

The `$` and `[]` operators get the value of a single parameter in the model. The `[]` gets several values, returning a named list.
$mrgmod-method$

Usage

```r
## S4 method for signature 'mrgmod'
names

## S4 method for signature 'mrgmod'
x[[i, exact = TRUE]]

## S4 method for signature 'mrgmod'
x[i]
```

Arguments

- `x`: mrgmod object
- `name`: parameter to take
- `i`: an element to select
- `exact`: not used
Index

* datasets
  exdatasets, 31
* param
  param, 70
*,tgrid,numeric-method
  (c,tgrid-method), 15
*,tgrids,numeric-method
  (c,tgrid-method), 15
+,tgrid,numeric-method
  (c,tgrid-method), 15
+,tgrids,numeric-method
  (c,tgrid-method), 15
[,mrgmod-method
  ($,mrgmod-method), 93
][[,$,ev-method, 93
[,,$,ev-method, 93
$,ev-method, 93
$,mrgmod-method, 93

aboutsolver, 4, 60, 87
allparam (param), 70
as.data.frame.ev, 59
as.ev, 5, 24
as.ev.data.frame-method (as.ev), 5
as.ev-method (as.ev), 5
as.integer, 67
as.list,mrgmod-method, 6
as.list,mrgsims-method, 7
as.tbl.mrgsims (mrgsims_dplyr), 55
as_bmat, 8, 41
as_bmat,ANY-method (as_bmat), 8
as_bmat.data.frame-method (as_bmat), 8
as_bmat.list-method (as_bmat), 8
as_bmat.numeric-method (as_bmat), 8
as_cmat (as_bmat), 8
as_data.frame.mrgsims (mrgsims_dplyr), 55
as_data_set, 10
as_data_set.data.frame-method
  (as_data_set), 10
as_data_set,ev-method (as_data_set), 10
as_deslist, 11
as_dmat, 41
as_dmat (as_bmat), 8
as_dmat.data.frame-method (as_bmat), 8
as_dmat.list-method (as_bmat), 8
as_dmat.numeric-method (as_bmat), 8
as_tibble.mrgsims (mrgsims_dplyr), 55
assign.ev (ev_assign), 25

BLOCK_PARSE, 12, 73
blocks, 12
blocks.character-method (blocks), 12
blocks,mrgmod-method (blocks), 12
bmat, 9
bmat (matrix Helpers), 40

CMT (BLOCK_PARSE), 12
CMT (BLOCK_PARSE), 12
carry.out (carry_out), 15
carry_out, 15
cmat, 9
cmat (matrix Helpers), 40
CMT (BLOCK_PARSE), 12
cmtn, 16
cmtn,mrgmod-method (cmtn), 16
code, 17
da_set, 10, 15, 17, 23, 24, 35, 37, 90, 91
da_set(), 52, 60
da_set.mrgmod,ANY-method (data_set), 17
da_set.mrgmod.data.frame-method
  (data_set), 17
da_set.mrgmod.ev-method (data_set), 17
da_set.mrgmod.missing-method
  (data_set), 17
INDEX

mrgsim, 16, 51, 58, 59, 67, 68, 77, 78, 82, 88
mrgsim(), 59, 60
mrgsim_0 (mrgsim_variants), 59
mrgsim_d, 58
mrgsim_d (mrgsim_variants), 59
mrgsim_df (mrgsim), 51
mrgsim_df(), 54
mrgsim_di (mrgsim_variants), 59
mrgsim_e (mrgsim_variants), 59
mrgsim_ei, 59
mrgsim_ei (mrgsim_variants), 59
mrgsim_i, 59
mrgsim_i (mrgsim_variants), 59
mrgsim_q, 58, 58, 78
mrgsim_q(), 51, 54, 59, 60
mrgsim_variants, 51, 54, 59, 78
mrgsims, 54
mrgsims_dplyr, 55, 57
mrgsims_modify, 55, 56, 57
mrgsolve, 60
mutate.ev, 24, 62
mutate.mrgsims (mrgsims_dplyr), 55
mutate_sims (mrgsims_modify), 57

names.mrgmod-method, 63
nnext, 64
NMXML (nmxml), 65
nmxml, 65, 65
numericlist, 36, 70, 71
numerics_only, 18, 67

obsaugs, 67
obsonly, 68
omat (omega), 68
omat.list-method (omega), 68
omat.matrix-method (omega), 68
omat.missing-method (omega), 68
omat.mrgmod-method (omega), 68
omat.mrgsims-method (omega), 68
omat.NULL-method (omega), 68
omat.omegalist-method (omega), 68
OMEGA (omega), 68
omega, 68
outvars, 70

PARAM (BLOCK_PARSE), 12
param, 35, 70, 91
param, ANY-method (param), 70
param, list-method (param), 70

param, missing-method (param), 70
param, mrgmod-method (param), 70
param, mrgsims-method (param), 70
parse_rx, 28
parse_rx (ev_rx), 28
PKMODEL, 72
PKMODEL(). 12, 14
plot, batch_mrgsims, formula-method
  (plot, batch_mrgsims, missing-method), 73
plot, batch_mrgsims, missing-method, 73
plot, mrgsims, character-method
  (plot_mrgsims), 74
plot, mrgsims, formula-method
  (plot_mrgsims), 74
plot, mrgsims, missing-method
  (plot_mrgsims), 74
plot_mrgsims, 74
plot_sims, 76
pull.mrgsims (mrgsims_dplyr), 55
qism, 59, 77
qsim(), 60

read_nnext, 65, 78
realize_addl, 24, 79
render, 80
render, character-method (render), 80
render, mrgmod-method (render), 80
Req, 81
req (Req), 81
reserved, 82
revar, 83
revar, mrgmod-method (revar), 83

see, 46, 83
see, mrgmod-method (see), 83
select.ev (mutate.ev), 63
select.mrgsims (mrgsims_dplyr), 55
select_sims (mrgsims_modify), 57
seq, 30
seq.ev (ev_seq), 30
set.seed, 21
SIGMA (sigma), 84
sigma, 84
simargs, 85
slice.mrgsims (mrgsims_dplyr), 55
smat (sigma), 84
smat, list-method (sigma), 84
smat, matrix-method (sigma), 84
smat, missing-method (sigma), 84
smat, mrgmod-method (sigma), 84
smat, mrgsims-method (sigma), 84
smat, NULL-method (sigma), 84
smat, signalist-method (sigma), 84
soloc, 86
solversettings, 7, 86, 89
stime(), 53
summarise.each (mrgsims_dplyr), 55
summarise.mrgsims (mrgsims_dplyr), 55
summary.mrgmod, 87
tgrid, 19
tgrid_*_numeric (c, tgrid-method), 15
tgrid_*_numeric (c, tgrid-method), 15
tgrids_*_numeric (c, tgrid-method), 15
tgrids_*_numeric (c, tgrid-method), 15
THETA (BLOCK_PARSE), 12
tscale, 87
update, 50, 59, 87, 88, 89, 91, 92
update(), 52, 60, 78
update, mrgmod-method (update), 88
update, omegalist-method (update), 88
update, parameter_list-method (update), 88
update, signalist-method (update), 88
valid_data_set, 18, 90, 91
valid_idata_set, 18, 90, 91
within, 89, 91, 91
within, mrgmod-method (within), 91
within.mrgmod (within), 91
zero_re, 92
zero_re, mrgmod-method (zero_re), 92