Package: mapbayr (via r-universe)

September 9, 2024

Title MAP-Bayesian Estimation of PK Parameters

Version 0.10.0

Description Performs maximum a posteriori Bayesian estimation of individual pharmacokinetic parameters from a model defined in 'mrgsolve', typically for model-based therapeutic drug monitoring. Internally computes an objective function value from model and data, performs optimization and returns predictions in a convenient format. The performance of the package was described by Le Louedec et al (2021) <doi:10.1002/psp4.12689>.

License GPL-3

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

Imports dplyr, ggplot2 (>= 3.4.0), magrittr, mrgsolve (>= 1.0.8), progress, purrr, rlang, stringr, tidyr

URL https://github.com/FelicienLL/mapbayr

BugReports https://github.com/FelicienLL/mapbayr/issues

Suggests knitr, lubridate, minqa, scales, testthat (>= 3.0.0), tibble

Depends R (>= 2.10)

LazyData true

Config/testthat/edition 3

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

RemoteUrl https://github.com/felicienll/mapbayr

RemoteRef HEAD

RemoteSha 3ae0c7204737427b3cae9e2919dafd9b1b0ff8da

Contents

`add_covariates` *Add covariate columns to a dataset*

Description

The `add_covariates()` function adds an one or several covariate columns to a dataset provided as a proper data.frame or within a 'mrgsolve' model. Used in combination with `adm_rows()` and `obs_rows()`, it helps the creation of datasets in the proper format for simulations with 'mrgsolve' or parameter estimation with 'mapbayr', as explained in data_helpers.

Usage

```
add_covariates(x, ...)

## S3 method for class 'data.frame'
add_covariates(x, ..., covariates = list(), AOLA = FALSE, TOLA = FALSE)

## S3 method for class 'mrgmod'
add_covariates(x, ..., covariates = list(), AOLA = NULL, TOLA = NULL)
```

Arguments

x	either a data.frame or a 'mrgsolve' model object
...	covariates values to add to the data. For each variable, supply a vector of length 1 or with the same number of rows. Ignored if `covariates` argument is used.
covariates	Covariates passed as a single list of variables. Overrides ....
AOLA, TOLA	a logical. Should the "Amount Of Last Administration" and "Time Of Last Administration" variables be added into the dataset? Default if FALSE if x is a dataset, TRUE if x is a 'mrgsolve' model where AOLA and TOLA are defined as covariates

Value

a data.frame, or a 'mrgsolve' model with a dataset in the `@args$data` slot (accessible with `get_data()`).

See Also

data_helpers

Examples

```
# Cannot start from scratch
## Not run:
add_covariates(BW = 90, SEX = 0)

## End(Not run)

library(magrittr)
adm_rows(time = c(0, 24, 48), cmt = 1, amt = c(100, 200, 300)) %>%
  add_covariates(BW = c(90, 85, 80), SEX = 0)

# If covariates are stored in a list, use `covariates = `
adm_rows(time = c(0, 24, 48), cmt = 1, amt = c(100, 200, 300)) %>%
  add_covariates(covariates = list(BW = c(90, 85, 80), SEX = 0))

# Missing values are filled with the "next observation carried backward" rule
adm_rows(time = c(0, 24, 48), cmt = 1, amt = c(100, 200, 300)) %>%
  add_covariates(BW = c(90, 85, 80), SEX = 0) %>%
  obs_rows(time = 36, DV = .0123, cmt = 2)
# Always verify the output in case of time-varying covariates
```

```
# Possibility to add Time and Amount of last administration as covariates
adm_rows(time = c(0, 24, 48), amt = c(100, 200, 300), cmt = 1) %>%
  obs_rows(time = c(8, 16, 32, 40), cmt = 2, DV = runif(4)) %>%
  add_covariates(TOLA = TRUE, AOLA = TRUE) %>%
  obs_rows(time = 72, cmt = 2, DV = .123) # AOLA/TOLA re-updated afterwards

# Automatic inclusion of `TOLA`/`AOLA` if they are covariates of the model
library(mrgsolve)
model <- mcode("model", "
$PARAM @annotated @covariates
TOLA : 0 : Time Last Adm
AOLA : 0 : Amount Last Adm
", compile = FALSE)

model %>%
  adm_rows(time = c(0, 24, 48), amt = c(100, 200, 300), cmt = 1) %>%
  add_covariates() %>%
  get_data()
```

---

adm_rows *Add administration lines to a dataset*

Description

The adm_rows() function adds an one or several administration lines to a dataset provided as a proper data.frame or within a 'mrgsolve' model. Used in combination with obs_rows() and add_covariates(), it helps the creation of datasets in the proper format for simulations with 'mrgsolve' or parameter estimation with 'mapbayr', as explained in data_helpers.

Usage

```
adm_rows(x, ...)

## S3 method for class 'data.frame'
adm_rows(
  x,
  ID = NULL,
  time = NULL,
  evid = 1L,
  cmt,
  amt,
  mdv = 1L,
  addl = NULL,
  ss = NULL,
  ii = NULL,
  rate = NULL,
  .datehour = NULL,
  ...
```

```
)

## S3 method for class 'missing'
adm_rows(...)

## S3 method for class 'mrgmod'
adm_rows(x, cmt = adm_cmt(x), rate = NULL, ...)
```

Arguments

x	either a data.frame or a 'mrgsolve' model object
...	additional columns or arguments for mrgsolve::ev()
ID	subject ID (default is 1)
time	event time. Default is 0 if no previous events. Mind consistency with .datehour.
evid	event identification (default is 1 for administration, 0 for observation)
cmt	compartment (no default, except if [ADM] was tagged in the $CMT block in model code. See examples.)
amt	dose amount (for administration records only)
mdv	missing dependent value (default is 1 for administration records)
addl	additional dose (optional)
ss	steady-state (optional, is this dose the last of an infinity of administration? Yes, 1, or no, 0)
ii	inter-dose interval (optional, use it with ss and addl)
rate	rate of administration (optional, set to -2 if you model zero-order infusion. See examples.)
.datehour	a object of class POSIXct, a number or a character vector that can be passed to parse_datehour(). Using .datehour will update the value of time in the dataset, with units in hours. Mind consistency with the time argument.

Value

a data.frame, or a 'mrgsolve' model with a dataset in the @args$data slot (accessible with get_data()).

See Also

data_helpers

Examples

```
# Create a dataset from scratch
adm_rows(amt = 100, cmt = 1)

# Pipe-friendly addition of administration record to a pre-existing dataset
library(magrittr)
adm_rows(amt = 100, cmt = 1) %>%
  adm_rows(time = 3, amt = 200, cmt = 1, addl = 3, ii = 1)
```

```
# Inform times using the `.datehour` argument:
adm_rows(.datehour = "2020-01-01 11:11", amt = 100, cmt = 1) %>%
  adm_rows(.datehour = "2020-01-02 22:22", amt = 200, cmt = 1) %>%
  adm_rows(time = 48, amt = 300, cmt = 1)

# Start from a 'mrgsolve' model
library(mrgsolve)
house() %>%
  adm_rows(amt = 100, cmt = 1) %>%
  adm_rows(time = 3, amt = 200, cmt = 1, addl = 3, ii = 1) %>%
  mrgsim(delta = 1)

# Default administration compartments
# Set default administration compartments in the code with `[ADM]`
model <- mcode("model", "
$CMT @annotated
DEPOT : Depot [ADM]
CENTR : Central
", compile = FALSE)
adm_cmt(model)

# Thus, no need to manually specify `cmt = 1` anymore.
model %>%
  adm_rows(amt = 100) %>%
  adm_rows(time = 3, amt = 200, addl = 3, ii = 1) %>%
  get_data()

# Automatic lines duplication if multiple depot compartments
# Automatic `rate = -2` increment if model with 0-order absorption
model <- mcode("model", "
$PARAM DUR = 1.0
$CMT @annotated
DEPOT : Depot [ADM]
CENTR : Central [ADM]
$MAIN
D_CENTR = DUR ;
", compile = FALSE)
adm_cmt(model)

model %>%
  adm_rows(amt = 100) %>%
  adm_rows(time = 3, amt = 200, addl = 3, ii = 1) %>%
  get_data()
```

---

as.data.frame.mapbayests

Return the mapbay_tab as a data.frame

Description

Return the mapbay_tab as a data.frame

Usage

```
## S3 method for class 'mapbayests'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

x	A mapbayests object.
row.names, optional, ...	passed to as.data.frame

Value

a data.frame (the mapbay_tab from estimation)

---

augment *Compute full PK profile prediction from mapbayr estimates.*

Description

Compute full PK profile prediction from mapbayr estimates.

Usage

```
augment(x, ...)
```

Arguments

x	object to augment
...	additional arguments

Value

an augmented object (depending on the object passed).

`augment.mapbayests` *Compute full PK profile prediction from mapbayr estimates.*

Description

Compute full PK profile prediction from mapbayr estimates.

Usage

```
## S3 method for class 'mapbayests'
augment(
  x,
  data = NULL,
  start = NULL,
  end = NULL,
  delta = NULL,
  ci = FALSE,
  ci_width = 90,
  ci_method = "delta",
  ci_sims = 500,
  ...
)
```

Arguments

`x`	A `mapbayests` object.
`data`	dataset to pass to mrgsolve for simulation (default is dataset used for estimation)
`start`, `end`, `delta`	start, end and delta of simulation time passed to `mrgsim()` (see details)
`ci`	a logical. If TRUE, compute a confidence interval around the prediction (default is FALSE)
`ci_width`	a number between 0 and 100, width of the confidence interval (default is "90" for a 90%CI)
`ci_method`	method to compute the confidence interval. Can be "delta" (the default) to use the Delta approximation. Alternatively "simulations" for a more accurate approach, but also more time-consuming.
`ci_sims`	number of replicates to simulate in order to derive the confidence interval (default is 500)
`...`	additional arguments passed to `mrgsim()`

Details

This function is called in the background by `plot()` in order to simulate the full PK profile, and return a `mapbayests` object with an additional `aug_tab` data.frame inside. The latter is used with by the plot method. The time grid, for each PK profile (i.e. patient) is defaulted with the minimum time

in the dataset for `start` and the maximum time in the dataset +20% for `end`. `delta` is a power of 10 (e.g. 0.1, 1, 10 etc...), automatically chosen to render visually appealing graphs with a reasonable computing time (about 200 time points). Additional arguments can be passed to `mrgsim()` through `...`. Note that `recsort` is set to 3 (see mrgsolve documentation for more details).

Value

a `mapbayests` object, augmented of an `aug_tab` data.frame.

Examples

```
#x is the result of `mapbayest()`.
#Default plot is returned by:
# plot(x)
#Argument passed to `plot()` are passed to `augment()` in the background:
# plot(x, end = 240, ci = TRUE)
#Save the augmented object if simulation time is long
# x2 <- augment(x, ci = TRUE, ci_method = "simulations", ci_sims = 10000) %>%
# plot(x2)
```

---

check_mapbayr_model *Check if model is valid for 'mapbayr'*

Description

Checks that the model respects points related exclusively to 'mapbayr'. Useful at the time you wish to convert a "regular" 'mrgsolve' model you used for simulation into a model to perform MAP-Bayesian estimation. Note that some elements cannot be checked:

- In `$MAIN` block, make sure that you added `ETA1, ETA2...` in the code. For instance: `double CL = TVCL * exp(ETA(1) + ETA1) ;`.
- In `$OMEGA` block, make sure the order of the (diagonal) values is the same as for ETAs in `$PARAM`. For instance, if `ETA1` corresponds to clearance, the first value in `$OMEGA` must be the variance of clearance.
- In `$SIGMA` block, make sure the order is respected: proportional error first, and additive error secondly.

Usage

```
check_mapbayr_model(x, check_compile = TRUE)
```

Arguments

`x`	model file
`check_compile`	check if model is compiled

Value

TRUE (invisibly) if checks are passed, errors otherwise.

Examples

```
library(mapbayr)
library(mrgsolve)
## Not run: check_mapbayr_model(house())
```

---

compute_ofv *Compute the objective function value*

Description

Compute the objective function value

Usage

```
compute_ofv(
  eta,
  qmod,
  sigma,
  omega_inv,
  all_cmt,
  log_transformation,
  lambda = 1,
  idvaliddata,
  idDV,
  idcmt,
  idblq = NULL,
  idlloq = NULL,
  ...
)

do_compute_ofv(eta, argofv, ...)
```

Arguments

eta	a named vector/list of parameters
qmod, sigma, log_transformation, omega_inv, all_cmt, lambda	generated by preprocess.ofv.fix
idvaliddata, idDV, idcmt	generated by preprocess.ofv.id
idblq, idlloq	optionally generated by preprocess.ofv.id
...	for compatibility (not used)
argofv	above mentioned arguments as a list

Details

This function is called iteratively by the optimization function. Arguments should not be passed directly, but generated by the pre-processing functions (see `preprocess.ofv`).

Value

a single numeric value (the objective function value)

`data_helpers` *Data helpers: functions to build the dataset*

Description

Use `adm_rows()`, `obs_rows()` and `add_covariates()` to create or modify a dataset from scratch, from a pre-existing dataset, or from a dataset stored into a 'mrgsolve' model.

Details

Instead of importing a '.csv' file, or painfully build a data set with a call to `data.frame()` and mind how to format the data, you can pass information about:

- administrations with `adm_rows()`,
- observations with `obs_rows()`,
- covariates with `add_covariates()`,

all being called jointly with a pipe (`%>%` or `|>`). These functions can be used to create or modify a dataset as a proper data.frame, or to create or modify a dataset within a 'mrgsolve' model (`@args$data` slot). The latter is particularly useful in order to:

- automatically use default administration and observation compartments,
- automatically duplicate rows if there are several depot compartments,
- automatically set `rate = -2` if model has zero-order absorption pathways,
- automatically duplicate rows if concentrations of Parent drug and Metabolite are observed together,
- automatically add "Amount Of Last Administration" and "Time Of Last Administration" variables if these are covariates,
- subsequently call `mrgsim()` or `mapbayest()`.

Examples

```
library(magrittr)
# First option: work with a data.frame

adm_rows(amt = 1000, cmt = 1, addl = 4, ii = 12) %>%
  obs_rows(time = c(12.3, 45.6), DV = c(.111, .222), cmt = 2) %>%
  obs_rows(time = 48, cmt = 2) %>%
  add_covariates(BW = 90, SEX = 0, TOLA = TRUE)

# You can even inform "time" using date and hours:
adm_rows(amt = 1000, cmt = 1, addl = 4, ii = 12, .datehour = "2022-01-01 11:11:11") %>%
  obs_rows(.datehour = "2022-01-02 22:22:22", DV = 0.111, cmt = 2)

# Second option: work with a dataset within a 'mrgsolve' model

mod <- exmodel(add_exdata = FALSE)
# call `mrgsolve::see(mod)` to see how default compartment were coded
adm_cmt(mod)
obs_cmt(mod)

mod %>%
  adm_rows(amt = 10000) %>%
  obs_rows(time = c(1.5, 4.4, 7.5, 24.6), DV = c(91.2904, 110.826, 79.384, 20.6671)) %>%
  # get_data() # for curiosity, you can extract the data set at this step
  mapbayest()
```

`deprecations` *Deprecated functions*

Description

Deprecated functions

Usage

```
mbrest(...)

adm_lines(...)

obs_lines(...)
```

Arguments

`...`	passed to the corresponding function

Details

- mbrest() is now `mapbayest()`
- adm_lines() is now `adm_rows()`
- obs_lines() is now `obs_rows()`

`do_mapbayr_sim` *Simulate with mapbayr*

Description

A wrapper around `mrgsolve::mrgsim()` for results generated from `mapbayest()`. Exported for the purpose of utility but might be prone to changes.

Usage

```
do_mapbayr_sim(
  x,
  data,
  recsort = 3,
  output = "df",
  ...,
  eta = NULL,
  nrep = NULL,
  new_omega = NULL,
  new_sigma = NULL
)
```

Arguments

`x`	the model object
`data`	NMTRAN-like data set
`recsort`	record sorting flag. Defaulted to 3. See `mrgsolve::mrgsim()`.
`output`	type of object returned. Defaulted to `"df"` for a data.frame. See `mrgsolve::mrgsim()`.
`...`	passed to `mrgsolve::mrgsim()`.
`eta`	a matrix of individual point estimates of ETA. Most likely obtained with `get_eta()`.
`nrep`	number of replicates. If used, the original "ID" in the data will be replaced by unique identifiers.
`new_omega, new_sigma`	New "omega" and "sigma" matrices to use instead of those defined in "$OMEGA" and "$SIGMA".

Value

An output from `mrgsolve::mrgsim()`.

Examples

```
library(mrgsolve)
mod <- exmodel(1, exdata = FALSE)
dat <- exdata(ID = c(1,2))

# Classic framework
set.seed(123)
do_mapbayr_sim(x = mod, data = dat, Request = "DV")

# No random effect
do_mapbayr_sim(x = zero_re(mod), data = dat)
do_mapbayr_sim(x = mod, data = dat, new_omega = "zero_re")

# New random effects
## New omega matrix
do_mapbayr_sim(x = mod, data = dat, new_omega = dmat(0.1, 0.03, 0.01), nrep = 10)

## Matrix with "eta" as mean and "new_omega" as variance covariance matrix
etamat <- get_eta(est001, output = "num")[1:2,]

do_mapbayr_sim(
  x = mod, data = dat, nrep = 10,
  eta = etamat, new_omega = dmat(0.1, 0.03, 0.01)
)
```

`est001` *Estimation object*

Description

An example of `mapbayests` object, corresponding to the parameter estimation of the 8 subjects from model 1. Note that the model object within is not associated to a shared object, which make some features unavailable. This object can be re-generated by executing `est001 <- mapbayest(exmodel(ID = 1:8))`

Usage

```
est001
```

Format

An object of class `mapbayests` of length 9.

See Also

`mapbayest`

`eta` *Generate a vector of "ETA"*

Description

Generate a vector of "ETA" values. If `x` is a `mrgsolve` model, these will be extracted from values defined in `$PARAM`. Otherwise, any numeric values passed to `x` and `...` as vector(s) or list(s) will be coerced as a single vector. Alternatively, if `x` and `...` are missing, generate a vector of ETA equal to `val` of length `n`.

Usage

```
eta(x, ..., n, val = 0)
```

Arguments

`x`	either a `mrgsolve` model object, or a numeric
`...`	additional numeric(s)
`n`, `val`	generate a sequence of `val` of length `n`

Value

a single named vector of numeric

Examples

```
# Extract ETA from the model
mod <- exmodel()
eta(mod)

# Coerce numeric values
eta(0.1, 0.2, c(0.3, 0.4), list(0.5, 0.6))
eta(rnorm(4))

# Generate a sequence from scratch
eta(n = 3)
eta(n = 3, val = 0.001)
```

`exmodel_exdata` *Example model and data*

Description

A collection of example models and corresponding data to test and explore `mapbayr`.

Usage

```
exmodel(
  num = 1,
  add_exdata = TRUE,
  cache = TRUE,
  quiet = getOption("mrgsolve_mread_quiet", TRUE),
  ...,
  ID = 1,
  clean_data = TRUE
)

exdata(num = 1, ID = 1, clean_data = TRUE)
```

Arguments

`num`	model number (see details)
`add_exdata`	should data be automatically loaded with the model
`cache`	read the model with `mrgsolve::mread_cache()`
`quiet`	don't print messages when compiling
`...`	passed to `mrgsolve::mread()` or `mrgsolve::mread_cache()`
`ID`	individual number to include in the data (from 1 to 8)
`clean_data`	remove useless columns and rows from the original data

Details

Available models are:

- 1: Base model. A simple monocompartmental PK model with inter-individual variability on absorption constant (KA), volume of distribution (VC) and clearance (CL). The residual error model is proportional.
- 6: Complex absorption model. Dual 0- and 1st orders absorption phenomenons.
- 301: Time-varying covariates. A continuous covariate (body weight "BW") and a categorical one (sex "SEX") influence the clearance parameter. In the corresponding dataset, the values randomly changes from one record to another within a single individual.
- 401: Metabolite. The PK model of both a parent drug and its metabolite.

An example dataset of eight (simulated) individuals is available for each model. Individuals differ in terms of sampling times (sparse or rich) and dosing regimen (single or multiple dosing).

Model code and data files are stored at the location given by `system.file("exmodel", package = "mapbayr")`.

These models and data were created for the validation study of `mapbayr` published in CPT:Pharmacometrics & System Pharmacology. More models and full datasets can be accessed in a dedicated repository

Value

`exmodel()` reads and compiles code, and returns a (`mrgmod`) model object. `exdata()` returns a data.frame.

Source

https://github.com/FelicienLL/mapbayr-CPTPSP-2021

Examples

```
# Models can be loaded with data (the default), ready for parameter estimation
est <- mapbayest(exmodel())

# Number of subjects in dataset can be chosen up to 8 individuals
exdata(301, ID = c(5,8))
```

---

filter.mrgmod *Filter a dataset within a mrgmod*

Description

Filter a dataset within a mrgmod

Usage

```
## S3 method for class 'mrgmod'
filter(.data, ..., .preserve = FALSE)
```

Arguments

`.data`	a mrgmod
`..., .preserve`	additional arguments for `dplyr::filter()`

Value

a mrgmod

Examples

```
library(magrittr)
mod <- mrgsolve::mcode("mod", "$CMT FOO", compile = FALSE)
mod %>%
  adm_rows(amt = c(100, 200, 300), cmt = 1) %>%
  filter(amt != 200) %>%
  get_data()
```

---

get_x *Get content from object*

Description

Helpful functions to get content from a `mrgmod` object (i.e. data) or from a `mapbayests` object (`data`, `eta`, `cov`, `param`, `phi`).

Usage

```
get_data(x, ...)

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

## S3 method for class 'mapbayests'
get_data(x, ..., output = "df")

get_eta(x, ...)

## S3 method for class 'mapbayests'
get_eta(x, ..., output = NULL)

get_cov(x, ...)

## S3 method for class 'mapbayests'
get_cov(x, ..., simplify = TRUE)

get_param(x, ...)

## S3 method for class 'mapbayests'
get_param(x, ..., output = NULL, keep_ID = NULL, keep_names = NULL)

get_phi(x, ...)

## S3 method for class 'mapbayests'
get_phi(x, ...)
```

Arguments

x	mapbayests object
...	not used
output	either a data.frame ("df") or a vector of numeric ("num"). Default to "num" if only one ID
simplify	a logical. If TRUE (the default) and only one ID, one matrix is returned instead of a list of length 1
keep_ID	a logical. By default, the ID variable is dropped if one ID in data.
keep_names	a logical. By default, names are dropped if one parameter is requested, and output is not a data frame.

Value

the class of the object returned depends on the function, and on their arguments. Typically, a data.frame or a vector if the output can be reduced to one line.

Examples

```
# From a model object (mrgmod)
mod <- exmodel(ID = 1:2, cache = FALSE, capture = "CL")
get_data(mod)

# From an estimation object (mapbayests)
est <- mapbayest(mod)
get_data(est)
get_data(est, output = "list")

get_eta(est)
get_eta(est, output = "list")

get_cov(est)

get_param(est)

get_phi(est)
```

---

hist.mapbayests *Plot posterior distribution of bayesian estimates*

Description

Plot posterior distribution of bayesian estimates

Usage

```
## S3 method for class 'mapbayests'
hist(x, select_eta = x$arg.optim$select_eta, shk = c("sd", "var", NA), ...)
```

Arguments

x	A mapbayests object.
select_eta	a vector of numeric values, the numbers of the ETAs to show (default are estimated ETAs).
shk	method to compute the shrinkage if multiple subjects are analyzed. Possible values are "sd" (based on the ratio of standard deviation like in 'NONMEM'), "var" (based on the ratio of variances like 'Monolix'), or NA (do not show the shrinkage)
...	additional arguments (not used)

Details

Use this function to plot the results of the estimations, in the form of histograms with the *a priori* distribution in the background. For every parameter, the inter-individual variability is displayed, as well as the percentile of the patient in the corresponding distribution (if n = 1 patient). For additional modifications, you can add extra +function(...) in order to modify the plot as a regular ggplot2 object.

Value

a ggplot object.

Examples

```
est <- mapbayest(exmodel(ID = 1))

# Default Method
h <- hist(est)

# Can be modified with `ggplot2`
h +
  ggplot2::labs(title = "Awesome estimations")

# Select the ETAs
hist(est, select_eta = c(1,3))
```

---

mapbayest *Estimate parameters (maximum a posteriori)*

Description

The main function of the mapbayr package. Performs a *maximum a posteriori* Bayesian estimation of parameters, from a mrgsolve model object and a dataset containing information about administrations and observed concentrations.

Usage

```
mapbayest(
  x,
  data = NULL,
  method = c("L-BFGS-B", "newuoa"),
  hessian = stats::optimHess,
  select_eta = NULL,
  lambda = 1,
  lloq = NULL,
  force_initial_eta = NULL,
  quantile_bound = 0.001,
  control = list(),
  check = TRUE,
  verbose = TRUE,
  progress = TRUE,
  reset = 50,
  output = NULL,
  ...
)
```

Arguments

x	the model object
data	NMTRAN-like data set
method	optimization method; the default is "L-BFGS-B" (from stat::optim()), alternatively "newuoa" for minqa::newuoa()
hessian	function used to compute the Hessian and variance-covariance matrix with (default is stats::optimHess, alternatively use nlmixr::nlmixrHess)
select_eta	a vector of numeric values, the numbers of the ETAs to be estimated (default is NULL, all ETAs non-equal to zero)
lambda	a numeric value, the weight applied to the model prior (default is 1)
lloq	a numeric value, the lower limit of quantification. If not NULL, LLOQ and BLQ (below limit of quantification) variables will be added to the data. The related records will be censored with the M3 method. Ignored if LLOQ already in the data.
force_initial_eta	a vector of numeric values to start the estimation from (default to 0 for "L-BFGS-B")
quantile_bound	a numeric value representing the quantile of the normal distribution admitted to define the bounds for L-BFGS-B (default is 0.001, i.e. 0.1%)
control	a list passed to the optimizer (see stats::optim() or minqa::newuoa() documentation)
check	check model code for mapbayr specification (a logical, default is TRUE)
verbose	print a message whenever optimization is reset (a logical, default is TRUE)
progress	print a progress bar (a logical, default is TRUE)

reset	maximum allowed reset of the optimizer with new initial eta values if numerical difficulties, or with new bounds (L-BFGS-B) if estimate equal to a bound. (a numeric, default is 50)
output	if NULL (the default) a mapbayests object is returned; if df a *mapbay_tab* dataframe is returned
...	for compatibility (not used)

Value

a mapbayests object. Basically a list containing:

- model: the model object
- arg.ofv.optim, arg.ofv.fix, arg.ofv.id: arguments passed to the optimization function. Useful for debugging but not relevant for a basic usage. Access to the data with `get_data(x)`
- opt.value: the original output of the optimization function
- final_eta: a list of individual vectors of final estimates. Access it with `x$final_eta` or `get_eta(x)`.
- covariance: a list of individual variance-covariance matrix of estimation. Access it with `x$covariance` or `get_cov(x)`.
- mapbay_tab: an output table containing the results of your estimations (data, IPRED, PRED, covariates, captured items, ETA etc...). Access it with `x$mapbay_tab`, `as.data.frame(x)` or `as_tibble(x)`.
- information: run times and package versions.

See Also

`hist.mapbayests`

`plot.mapbayests`

`use_posterior`

Examples

```
# First, code a model
code1 <- "$PARAM ETA1 = 0, ETA2 = 0,
KA = 0.5, TVCL = 1.1, TVV = 23.3
$OMEGA 0.41 0.32
$SIGMA 0.04 0
$CMT DEPOT CENT
$PK
double CL=TVCL*exp(ETA1+ETA(1));
double V=TVV*exp(ETA2+ETA(2)) ;
$ERROR
double DV=CENT/V*(1+EPS(1))+EPS(2);
$PKMODEL ncmt = 1, depot = TRUE
$CAPTURE DV CL
"

my_model <- mrgsolve::mcode("my_model", code1)
```

```
# Then, import your data
my_data <- data.frame(ID = 1, TIME = c(0, 1.1, 5.2, 12.3), EVID = c(1,0,0,0), AMT = c(500, 0,0,0),
 CMT = c(1,2,2,2), DV = c(0, 15.1, 29.5, 22.3))
print(my_data)

# And estimate
my_est <- mapbayest(x = my_model, data = my_data)
print(my_est)
# see also plot(my_est) and hist(my_est)

# Use your estimation
get_eta(my_est)
get_param(my_est)
as.data.frame(my_est)
use_posterior(my_est)
```

---

mapbayr_plot *Make mapbayr plot*

Description

Make mapbayr plot

Usage

```
mapbayr_plot(
  aug_tab,
  obs_tab = NULL,
  PREDICTION = c("IPRED", "PRED"),
  MODEL_color = NULL
)
```

Arguments

aug_tab	a table of predictions, generated by augment(x) and available at x$aug_tab
obs_tab	a table of observations
PREDICTION	plot either "IPRED", "PRED" or both.
MODEL_color	a vector of strings interpretable as colors, with names that correspond to a value in aug_tab$MODEL

Value

a ggplot object

Examples

```
aug <- data.frame(
  ID = 1, name = factor("DV"), cmt = 2, time = rep(c(0,8,16,24), each = 2),
  type = rep(c("PRED", "IPRED"), 4), value = c(0,0, 1, 2, 4, 8, 2, 4)
  )

obs <- data.frame(
  ID = 1, time = c(6, 20), evid = 0,
  mdv = c(0,1), DV = c(0.5, 5), cmt = 2
  )

mapbayr_plot(aug, obs)
mapbayr_plot(aug, obs, PREDICTION = "IPRED")

aug2 <- dplyr::bind_rows(
  FOO = aug,
  BAZ = dplyr::mutate(aug, value = value * 2),
  BAR = dplyr::mutate(aug, value = value * 3),
  .id = "MODEL"
  )

mapbayr_plot(aug2, obs)
mapbayr_plot(aug2, obs, PREDICTION = "IPRED")
mapbayr_plot(aug2, obs, PREDICTION = "IPRED", MODEL_color = c(FOO = "black"))
```

---

mapbayr_vpc *Visual Predicted Checks*

Description

Visual Predicted Checks

Usage

```
mapbayr_vpc(
  x,
  data = NULL,
  nrep = 500,
  pcvpc = TRUE,
  idv = "time",
  stratify_on = NULL,
  start = NULL,
  end = NULL,
  delta = 1,
  ...
)
```

Arguments

x	the model object
data	NMTRAN-like data set
nrep	a numeric, the number of replicates for stochastic simulations. Default is 500.
pcvpc	a logical, if TRUE (the default) will output "prediction-corrected VPC" (see Details).
idv	a character indicating the variable used as independent variable. Default is "time", alternatively use "tad" to automatically compute the time after last dose.
stratify_on	a character (vector) indicating the variables of the data used to stratify the results. Variables must be numeric (as they are passed to `mrgsolve::carry_out()`)
start, end, delta, ...	passed to `mrgsolve::mrgsim()`

Details

- Prediction-corrected VPC

By default, VPC are prediction corrected (Bergstrand et al (2011) doi:10.1208/s122480119255z). This correction is advised if several levels of doses or covariates are in the dataset for instance. Note that the implemented correction formula does not take into account the 'lower bound' term (*lbij*), nor the log-transformed variables.

Value

a ggplot object, results of the VPC. The median and the 50%, 80% and 90% prediction intervals of the simulated distributions are reported.

Examples

```
library(mrgsolve)
library(magrittr)
# Define a model. Adding variability to house model because default is 0.
mod <- house() %>%
  omat(dmat(rep(0.2,4)))

# Creating dataset for the example
# Same concentration, but different dose (ID 2) and covariate (ID 3)
data <- adm_rows(ID = 1, amt = 1000, cmt = 1, addl = 6, ii = 12) %>%
  obs_rows(DV = 50, cmt = 2, time = 7 * 12) %>%
  adm_rows(ID = 2, time = 0, amt = 2000, cmt = 1, addl = 6, ii = 12) %>%
  obs_rows(DV = 50, cmt = 2, time = 7 * 12) %>%
  adm_rows(ID = 3, time = 0, amt = 1000, cmt = 1, addl = 6, ii = 12) %>%
  obs_rows(DV = 50, cmt = 2, time = 7 * 12) %>%
  add_covariates(SEX = c(0,0,0,0,1,1))

mapbayr_vpc(mod, data, nrep = 30) # prediction-corrected by default
mapbayr_vpc(mod, data, idv = "tad", start = 72, nrep = 30)
mapbayr_vpc(mod, data, pcvpc = FALSE, nrep = 30)
mapbayr_vpc(mod, data, stratify_on = "SEX", nrep = 30)
```

`model_averaging` *Average predictions from multiple models*

Description

Model Averaging consists in analyzing the same data with different models and to average their predictions. In order to perform weighted means of clearance predictions, (or concentrations, or any metric of interest), it is necessary to compute the "weight" of each estimation. It is informed by the likelihood of estimation. Two weighting scheme are currently implemented, one based on the log- likelihood ("LL", the default), the other on the Akaike criterion ("AIC"). The method was previously described by Uster et al (2021) doi:10.1002/cpt.2065.

Usage

```
model_averaging(
  ...,
  output_function = as.data.frame,
  scheme = c("LL", "AIC"),
  estlist = NULL
)

compute_weights(..., scheme = c("LL", "AIC"), estlist = NULL)

do_model_averaging(list_of_tabs, weights_matrix)
```

Arguments

`...`	estimation objects generated with `mapbayest()`, from which the weights will be computed
`output_function`	a unique function that takes any estimation object and returns a table with controlled variables, dimensions and attributes.
`scheme`	scheme weight, either "LL" or "AIC"
`estlist`	a list of estimation objects. Overrides `...`
`list_of_tabs, weights_matrix`	respectively outputs of the `output_function` and `compute_weights()`

Value

- `model_averaging()` and `do_model_averaging()`: a data.frame of the same dimensions and attributes as the outputs
- `compute_weights()`: a matrix with IDs as rows and estimation weights as columns

Examples

```
library(magrittr)

# Three different models: A, B, and C.
modA <- exmodel(1, add_exdata = FALSE)
modB <- mrgsolve::param(modA, TVCL = 2, TVVC = 30)
modC <- mrgsolve::param(modA, TVCL = 10)

# A common dataset that has 2 patients (ID 2 & 9)
data <- adm_rows(ID = 2, time = 0, amt = 200, addl = 3, ii = 24, cmt = 1) %>%
    obs_rows(ID = 2, time = 84, DV = 1.5, cmt = 2) %>%
    adm_rows(ID = 9, time = 0, amt = 100, addl = 3, ii = 24, cmt = 1) %>%
    obs_rows(ID = 9, time = 96, DV = 1, cmt = 2)

# Three different estimation objects: A, B and C.
estA <- mapbayest(modA, data)
as.data.frame(estA)
plot(estA) # Fit is pretty good

estB <- mapbayest(modB, data)
as.data.frame(estB)
plot(estB) # Excellent fit

estC <- mapbayest(modC, data)
as.data.frame(estC)
plot(estC) # Fit is worst

# Model averaging
model_averaging(A = estA, B = estB, C = estC)
# Weighted average of the table returned by as.data.frame(est))

# Internally, it first computes the "weight" of each model such as:
W <- compute_weights(A = estA, B = estB, C = estC)

# Then multiply the prediction table with each weight such as:
do_model_averaging(
  list_of_tabs = list(
    A = as.data.frame(estA),
    B = as.data.frame(estB),
    C = as.data.frame(estC)
    ),
  weights_matrix = W
  )

# If you do not want to perform an average of the full table, you can specify
# a function that takes the estimation object as an input and returns
# value(s) of interest: a single prediction, a clearance value, a full
# table of augmented predictions... as long as the structure of the final
# object is the same whatever the model.

reframe <- function(est){
  # From any estimation object, return a table with ID, time and predictions
```

```
  as.data.frame(est)[,c("ID", "time", "DV", "IPRED")]
}

model_averaging(A = estA, B = estB, C = estC, output_function = reframe)

# Make a plot that compares predictions
List_aug_tab <- lapply(
  X = list(A = estA, B = estB, C = estC),
  FUN = \(x) augment(x)$aug_tab
)
List_aug_tab$.AVERAGE <- do_model_averaging(List_aug_tab, W)

mapbayr_plot(
 aug_tab = dplyr::bind_rows(List_aug_tab, .id = "MODEL"),
 obs_tab = data,
 PREDICTION = "IPRED",
 MODEL_color = c(.AVERAGE = "black")
)
```

---

obs_rows *Add observation lines to a dataset*

Description

The obs_rows() function adds an one or several observation lines to a dataset provided as a proper data.frame or within a 'mrgsolve' model. Used in combination with adm_rows() and add_covariates(), it helps the creation of datasets in the proper format for simulations with 'mrgsolve' or parameter estimation with 'mapbayr', as explained in data_helpers.

Usage

```
obs_rows(x, ...)

## S3 method for class 'data.frame'
obs_rows(
  x,
  ID = NULL,
  time = NULL,
  evid = 0L,
  cmt,
  DV = NA_real_,
  mdv = NULL,
  .datehour = NULL,
  ...
)

## S3 method for class 'missing'
obs_rows(...)
```

```
## S3 method for class 'mrgmod'
obs_rows(x, cmt = NULL, DV = NA_real_, DVmet = NULL, ...)
```

Arguments

x	either a data.frame or a 'mrgsolve' model object
...	additional columns
ID	subject ID (default is 1)
time	event time. Default is 0 if no previous events. Mind consistency with .datehour.
evid	event identification (default is 1 for administration, 0 for observation)
cmt	compartment (no default, except if [OBS] was tagged in the $CMT block in model code. See examples.)
DV	dependent value, i.e. observed concentration.
mdv	missing dependent value (default is 0 a non-missing concentration value to take into account for parameter estimation, 1 otherwise)
.datehour	a object of class POSIXct, a number or a character vector that can be passed to parse_datehour(). Using .datehour will update the value of time in the dataset, with units in hours. Mind consistency with the time argument.
DVmet	second observation at the same time (e.g. a metabolite, "DVmet") observed jointly with parent drug ("DV"). Works only if x is a 'mrgsolve' model where two [OBS] compartments were defined (see examples)

Value

a data.frame, or a 'mrgsolve' model with a dataset in the @args$data slot (accessible with get_data()).

See Also

data_helpers

Examples

```
# Create a dataset from scratch
obs_rows(time = 12, DV = 0.12, cmt = 2)

# Pipe-friendly addition of observation record to a pre-existing dataset
library(magrittr)
obs_rows(time = 12, DV = 0.12, cmt = 2) %>%
  obs_rows(time = c(24, 36, 48), DV = c(0.34, 0.56, 0.78), mdv = c(0, 1, 0), cmt = 2)

# Inform times using the `.datehour` argument:
obs_rows(.datehour = "2020-01-01 11:11", DV = 0.12, cmt = 1) %>%
  obs_rows(.datehour = "2020-01-02 22:22", DV = 0.34, cmt = 1) %>%
  obs_rows(time = 48, DV = 0.56, cmt = 1)

# Start from a 'mrgsolve' model
library(mrgsolve)
```

```
house() %>%
  obs_rows(time = 12, DV = 0.12, cmt = 2) %>%
  obs_rows(time = c(24, 36, 48), DV = c(0.34, 0.56, 0.78), mdv = c(0, 1, 0), cmt = 2) %>%
  mrgsim()

# Default observation compartments
# Set default observation compartments in the code with `[OBS]`
model <- mcode("model", "
$CMT @annotated
DEPOT : Depot
CENTR : Central [OBS]
", compile = FALSE)
obs_cmt(model)

# Thus, no need to manually specify `cmt = 2` anymore.
model %>%
  obs_rows(time = 12, DV = 0.12) %>%
  obs_rows(time = c(24, 36, 48), DV = c(0.34, 0.56, 0.78), mdv = c(0, 1, 0)) %>%
  get_data()

# Automatic lines duplication if parent + metabolite defined in the model
model <- mcode("model", "
$CMT @annotated
DEPOT : Depot
CENTR : Central [OBS]
PERIPH : Periph
METABO : Metabo [OBS]
", compile = FALSE)
obs_cmt(model)

model %>%
  obs_rows(time = 12, DV = 0.12, DVmet = 120) %>%
  obs_rows(
    time = c(24, 36, 48), DV = c(0.34, 0.56, 0.78),
    mdv = c(0, 1, 0), DVmet = c(340, 560, 780)
  ) %>%
  get_data()
```

`parse_datehour` *Parse value to "POSIXct"*

Description

A wrapper around functions of `lubridate`, mainly in order to transform characters into a date-time ("POSIXct") format.

Usage

```
parse_datehour(
  x,
```

```
  orders = getOption("mapbayr.datehour", default = c("Ymd HMS", "Ymd HM", "dmY HMS",
    "dmY HM"))
)
```

Arguments

x	a numeric or a character.
orders	format specification for x, passed to lubridate::parse_date_time()

Value

a POSIXct

Examples

```
# POSITct are returned as is.
parse_datehour(x = as.POSIXct("2022-02-02 22:22:22", tz = "UTC"))

# Numerics are passed to `lubridate::as_datetime()`.
parse_datehour(1643840542)

# Characters are passed to `lubridate::parse_date_time()`.
# The format used will be the one defined in `orders`
parse_datehour(x = "2022-02-02 22:22:22", orders = "Ymd HMS")
parse_datehour(x = "02-02-2022 22:22", orders = "dmY HM")

# By default, the following formats will be subsequently tried:
# "Ymd HMS", "Ymd HM", "dmY HMS", "dmY HM"

# Alternatively, set a format through `options(mapbayr.datehour)`.
# Convenient for the use `.datehour` in  `adm_rows()` and `obs_rows()`.

# Following format will return NA:
adm_rows(.datehour = "22:22 02-02-2022", amt = 100, cmt = 1)

options(mapbayr.datehour = "HM dmY")
adm_rows(.datehour = "22:22 02-02-2022", amt = 100, cmt = 1)
options(mapbayr.datehour = NULL)
```

---

plot.mapbayests *Plot predictions from mapbayests object*

Description

Plot predictions from mapbayests object

Usage

```
## S3 method for class 'mapbayests'
plot(x, ..., PREDICTION = c("IPRED", "PRED"))
```

Arguments

x	A mapbayests object.
...	additional arguments (passed to augment.mapbayests)
PREDICTION	plot either "IPRED", "PRED" or both.

Details

Use this function to plot the results of the estimations, in the form of concentration vs time profiles for every patient of the data set. For additional modifications, you can:

- see augment.mapbayests to modify the simulation output.
- add extra +function(...) in order to modify the plot as a regular ggplot2 object.

Value

a ggplot object.

Examples

```
est <- mapbayest(exmodel(ID = 1))
plot(est, end = 48) +
  ggplot2::labs(title = "Awesome prediction")
```

---

preprocess.ofv *Preprocess model and data for ofv computation*

Description

Functions to generate arguments passed to compute_ofv. Arguments that are fixed between individuals are created once (preprocess.ofv.fix), while others are specific of each individual (preprocess.ofv.id).

Usage

```
preprocess.ofv.fix(x, data, select_eta = seq_along(eta(x)), lambda = 1)

preprocess.ofv.id(x, iddata)
```

Arguments

x	the model object
data, iddata	NMTRAN-like data set. iddata is likely a dataset of one individual
select_eta	numbers of the ETAs taken into account. Set the dimensions of the inversed OMEGA matrix
lambda	a numeric value, the weight applied to the model prior (default is 1)

Value

A list of arguments used to compute the objective function value.

The following arguments are fixed between individuals:

- qmod: model object, modified to simulate without random effects and with controlled outputs
- sigma: a single matrix object
- log_transformation: a logical, whether predictions need to be log-transformed for ofv computation
- omega_inv: a single matrix object
- all_cmt: a vector of compartment numbers where observations can be expected

The following arguments differs between individuals:

- idvaliddata: a matrix, individual data set (with administrations and covariates), validated with valid_data_set
- idDV: a vector of (possibly log-transformed) observations
- idcmt: a vector of compartments where observations belong to
- idblq,idlloq: optional, a logical and numerical vector indicating if the observation is below the lower limit of quantification, and the LLOQ value, respectively

Examples

```
mod <- exmodel(add_exdata = FALSE, compile = FALSE)
dat <- exdata(ID = c(1,4))

preprocess.ofv.fix(x = mod, data = dat)
preprocess.ofv.id(x = mod, iddata = dat[dat$ID == 1,])
preprocess.ofv.id(x = mod, iddata = dat[dat$ID == 4,])
```

`preprocess.optim` *Pre-process: arguments for optimization function*

Description

Pre-process: arguments for optimization function

Usage

```
preprocess.optim(
  x,
  method = c("L-BFGS-B", "newuoa"),
  select_eta = NULL,
  control = list(),
  force_initial_eta = NULL,
  quantile_bound = 0.001
)
```

Arguments

`x`	the model object
`method`	optimization method; the default is `"L-BFGS-B"` (from `stat::optim()`), alternatively `"newuoa"` for `minqa::newuoa()`
`select_eta`	a vector of numeric values, the numbers of the ETAs to be estimated (default is `NULL`, all ETAs non-equal to zero)
`control`	a list passed to the optimizer (see `stats::optim()` or `minqa::newuoa()` documentation)
`force_initial_eta`	a vector of numeric values to start the estimation from (default to 0 for "L-BFGS-B")
`quantile_bound`	a numeric value representing the quantile of the normal distribution admitted to define the bounds for L-BFGS-B (default is 0.001, i.e. 0.1%)

Value

a list of named arguments passed to optimizer (i.e. arg.optim)

print.mapbayests *Print a mapbayests object*

Description

Print a mapbayests object

Usage

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

Arguments

x	A mapbayests object.
...	additional arguments

Value

print the results of the estimation to the console, and returns it invisibly.

use_posterior *Use posterior estimation*

Description

Use posterior estimation

Usage

```
use_posterior(
  x,
  update_omega = FALSE,
  update_cov = TRUE,
  update_eta = TRUE,
  .zero_re = NULL,
  simplify = TRUE
)
```

Arguments

x	A mapbayests object.
update_omega	Update the OMEGA matrix with the variance-covariance matrix of estimation (a logical, default is FALSE).
update_cov	Update the values of covariates with the individual values (a logical, default is TRUE).
update_eta	Update the values of ETA with the final estimates (a logical, default is TRUE).
.zero_re	Set all elements of the OMEGA or SIGMA matrix to zero. Default is "both" if update_omega is FALSE, "sigma" otherwise. (possible values are "both", "sigma", "omega", "none")
simplify	a logical. If TRUE (the default) and only one ID, one mrgmod is returned instead of a list of length 1

Details

This function takes the results of an estimation (i.e. a mapbayests object) and return a modified mrgmod in order to perform *a posteriori* simulations. Modifications are:

- If update_eta is TRUE, the values of ETA are updated to the estimated values (instead of 0) in $PARAM.
- If update_cov is TRUE, the covariates values are updated to the values of the individual (instead of default model values) in $PARAM.
- If update_omega is TRUE, the values of OMEGA are updated with the variance-covariance matrix of estimation (i.e. an approximation of the *a posteriori* distribution) instead of the inter-individual variability (i.e. the *a priori* distribution). Use this command in order to derive a confidence interval of concentrations that reflects the uncertainty about parameter estimation when a large number of profiles are simulated. Note that if inter-individual variability was initially defined in multiple $OMEGA blocks in the model, they will be collapsed to a single full matrix (this is irreversible).
- Depending on the values of .zero_re, the elements of $OMEGA or $SIGMA can be set to zero, whether you want to simulate one profile, or several in order to derive confidence/prediction intervals. It does not handle time-varying covariates: only the first value will be used as the individual value.

Value

a mrgmod, or a list of mrgmod if there is more than 1 ID

Examples

```
library(magrittr)
est <- mapbayest(exmodel())
est %>%
  use_posterior() %>%
  mrgsolve::ev(amt = 50000) %>%
  mrgsolve::mrgsim()
```

vs_nonmem *Compare results to NONMEM .phi*

Description

Compare results to NONMEM .phi

Usage

```
read_nmphi(x)

merge_phi(mapbayr_phi, nonmem_phi)

plot_phi(merged_phi, only_ETA = TRUE)

summarise_phi(
  merged_phi,
  group,
  only_ETA = TRUE,
  levels = c(Excellent = 0, Acceptable = 0.001, Discordant = 0.1)
)

bar_phi(summarised_phi, xaxis = NULL, facet = NULL)
```

Arguments

x	full path to a .phi file generated by NONMEM
mapbayr_phi	results of mapbayr estimations, in the form of a tibble data.frame, typically obtained from `get_phi()`
nonmem_phi	results of NONMEM estimations, in the form of a tibble data.frame, typically obtained from `read_nmphi()`
merged_phi	merged results of estimations, typically obtained from `merge_phi()`
only_ETA	filter the data with `type=="ETA"` (a logical, default is `TRUE`)
group	one or multiple variables to `group_by()`
levels	a named vector of length 3 in order to classify the absolute differences. Default cut-offs are 0.1% and 10% in the parameters space.
summarised_phi	summarized results of estimations, typically obtained from `summarise_phi()`
xaxis	optional. A character value, that correspond to a variable in data, passed to the x-axis to plot multiple bars side-by-side.
facet	a formula, that will be passed to `ggplot2::facet_wrap()`

Details

These functions were made to easily compare the results of mapbayr to NONMEM. For instance, it could be useful in the case of the transposition of a pre-existing NONMEM model into mapbayr. For this, you need to code your model in both mapbayr and NONMEM, and perform the MAP-Bayesian estimation on the **same dataset**. Ideally, the latter contains a substantial number of patients. NONMEM returns the estimations results into a .phi file.

Use `read_nmphi()` to parse the NONMEM .phi file into a convenient tibble data.frame with the columns:

- `SUBJECT_NO`, `ID`: Subject identification.
- `ETA1`, `ETA2`, ..., `ETAn`: Point estimates of eta.
- `ETC1_1`, `ETC2_1`, `ETC2_2`, ..., `ETCn_n`: Variance-covariance matrix of estimation.
- `OBJ`: objective function value

Use `get_phi()` to access to the estimations of mapbayr with the same "phi" format.

Use `merge_phi()` to combine mapbayr and NONMEM "phi files" into a single long-form data.frame with the columns:

- `SUBJECT_NO`, `ID`: Subject identification.
- `variable` name and its `type`: ETA (point estimate), VARIANCE (on-diagonal element of the matrix), COVARIANCE (off-diagonal), and OBJ.
- `mapbayr` and `nonmem`: corresponding values
- `adiff`: absolute difference between `mapbayr` and `nonmem` values.

Use `plot_phi()` to graphically represent `adiff` *vs* `variable`. Alternatively, the table returned by `merge_phi()` is easy to play with in order to derive performance statistics or the graphical plot of your choice.

Use `summarise_phi()` to classify the estimation as "Excellent", "Acceptable" or "Discordant", over the whole dataset or by `group`.

Use `bar_phi()` to graphically represent the proportion of the aforementioned classification as bar plot.

Value

- read_nmphi: a tibble data.frame with a format close to the original .phi file
- merge_phi: a long-form tibble data.frame with results of mapbayr and NONMEM
- summarise_phi: a summarized tibble data.frame classifying the performance of mapbayr and NONMEM
- plot_phi, bar_phi: a `ggplot2` object

Examples

```
library(mapbayr)
nmphi <- read_nmphi(system.file("nm001", "run001.phi", package = "mapbayr"))
mapbayrphi <- get_phi(est001)
```

```
merged <- merge_phi(mapbayrphi, nmphi)
plot_phi(merged)

summarised <- summarise_phi(merged)
bar_phi(summarised)


# Analyse the results of multiple runs simultaneously

#Example dataset that represents 3 runs
merge3 <- dplyr::bind_rows(merged, merged, merged, .id = "RUN")
merge3$adiff <- 10 ^ runif(nrow(merge3), -8, 0)

summarised3 <- summarise_phi(merge3, group = RUN)
bar_phi(summarised3, xaxis = "RUN")
```

---

x_cmt *Read compartment options in a model*

Description

Read compartment options in a model

Usage

```
adm_cmt(x)

obs_cmt(x)
```

Arguments

x	model object

Details

In a mrgsolve model, it is possible to specify options in `$CMT`. If `[ADM]` or `[OBS]` are set, mapbayr will interpret these as defaults administration and observation compartments, respectively.

Value

a vector of compartment identified as default "administration" or "observation" compartments.

Examples

```
#Administration:  Both 1st and 0- order
model <- exmodel(6, compile = FALSE)
mrgsolve::see(model)
adm_cmt(model)
```

```
#Observation: Both parent drug and metabolite
model <- exmodel(401, compile = FALSE)
mrgsolve::see(model)
obs_cmt(model)
```

Index