Package: Pmetrics (via r-universe)

August 30, 2026

Type Package

Title Pmetrics for Population Modeling and Simulation

Version 3.2.6

Description Pmetrics is short for pharmacometrics, the discipline of modeling drug kinetics and effects on biologic systems, including disease progression. Models can be used to simulate novel therapeutic dosing regimens and/or populations, informing trial design and dosing strategies. The package supports primarily non-parametric but aims to support parametric pharmacometric modeling and simulation with functions to run and analyze the output from all three components of the Pmetrics software suite for population pharmacometric data analysis.

License GPL (>= 3)

URL https://lapkb.github.io/Pmetrics/

BugReports https://github.com/LAPKB/Pmetrics/issues

Depends R (>= 4.2)

Imports bslib, cli, clipr, crayon, dplyr, DT, flextable, ggplot2, glue, graphics, grDevices, grid, htmltools, htmlwidgets, jsonlite, knitr, lifecycle, lubridate, magrittr, MASS, openxlsx, parallel, plotly, progress, purrr, R6, RColorBrewer, readr, reticulate, rlang, rmarkdown, shiny, stats, stringr, tibble, tidyr, tools, trelliscopejs, utils, withr

Additional_repositories https://lapkb.r-universe.dev

Suggests covr, devtools, mclust, PmetricsReports, PmetricsLitSim, PmetricsPlot, PmetricsModelLib, npde, testthat

VignetteBuilder knitr

Config/Needs/website r-lib/pkgdown, quarto

Config/rextendr/version 0.5.0

Config/roxygen2/version 8.1.0

Config/testthat/edition 3

Encoding UTF-8

LazyData true

LazyDataCompression xz

Roxygen list(markdown = TRUE)

SystemRequirements Cargo (>= 1.82) (Rust's package manager), rustc

Config/pak/sysreqs
libcairo2-dev cmake libfontconfig1-dev libfreetype6-dev libfribidi-dev make libharfbuzz-dev libmagick++-dev gsfonts libicu-dev libjpeg-dev libpng-dev libsodium-dev libtiff-dev libuv1-dev libwebp-dev libxml2-dev libssl-dev python3 libx11-dev zlib1g-dev libclang-dev

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

Date/Publication 2026-08-30 18:19:06 UTC

RemoteUrl https://github.com/LAPKB/Pmetrics

RemoteRef v3.2.6

RemoteSha 56132879ede7194972d19f8d20acc586e65a9352

Contents

ab . . . 5
ab_line . . . 5
add_renal . . . 6
add_shapes . . . 9
add_smooth . . . 10
additive . . . 12
all_is_numeric . . . 12
apps . . . 13
badData . . . 14
cdc_bmi . . . 14
check_updates . . . 15
click_plot . . . 15
cor2cov . . . 16
dataEx . . . 17
downloadR . . . 17
export_plotly . . . 18
ger_bmi . . . 19
getCov . . . 20
getDefaultColors . . . 21
getFixedColNum . . . 22
getPalettes . . . 22
getPMoptions . . . 23
growth . . . 24
interp . . . 24
latestR . . . 25
locales . . . 25

make_AUC . . . 26
make_NCA . . . 28
makeErrorPoly . . . 30
makePTAtarget . . . 31
mic1 . . . 32
model_lib . . . 33
modEx . . . 33
msd . . . 34
mtsknn.eq . . . 34
NM2PM . . . 35
NPex . . . 37
one_comp_bolus . . . 38
one_comp_bolus_cl . . . 38
one_comp_iv . . . 38
one_comp_iv_cl . . . 39
opposite_color . . . 39
plot.PM_cov . . . 40
plot.PM_cycle . . . 44
plot.PM_cycle_data . . . 47
plot.PM_data . . . 48
plot.PM_data_data . . . 53
plot.PM_final . . . 54
plot.PM_final_data . . . 59
plot.PM_model . . . 59
plot.PM_op . . . 61
plot.PM_op_data . . . 66
plot.PM_opt . . . 67
plot.PM_pop . . . 68
plot.PM_post . . . 72
plot.PM_pta . . . 76
plot.PM_sim . . . 81
plot.PM_valid . . . 86
plot.PMvalid . . . 91
plotlygg . . . 93
PM_build . . . 93
PM_compare . . . 94
PM_cov . . . 95
PM_cycle . . . 98
PM_data . . . 100
PM_final . . . 104
PM_help . . . 108
PM_load . . . 109
PM_manual . . . 110
PM_model . . . 111
PM_op . . . 121
PM_opt . . . 124
PM_pop . . . 127
PM_post . . . 130

PM_report . . . 132
PM_step . . . 132
PM_tree . . . 133
PM_tutorial . . . 134
PM_valid . . . 135
PMcheck . . . 138
PMmatrixRelTime . . . 141
PMpatch . . . 142
PMreadMatrix . . . 143
PMwriteMatrix . . . 144
PMwrk2csv . . . 145
print.PM_compare . . . 146
print.PMerr . . . 146
print.PMnpc . . . 147
print.summary.PM_cycle . . . 148
print.summary.PM_data . . . 149
print.summary.PM_final . . . 150
print.summary.PM_op . . . 151
print.summary.PM_sim . . . 152
proportional . . . 153
qgrowth . . . 153
rgba_to_rgb . . . 154
setPMoptions . . . 155
simEx . . . 155
ss.PK . . . 156
sub_plot . . . 157
summary.PM_cov . . . 159
summary.PM_cycle . . . 160
summary.PM_cycle_data . . . 161
summary.PM_data . . . 162
summary.PM_final . . . 163
summary.PM_final_data . . . 165
summary.PM_op . . . 165
summary.PM_op_data . . . 167
summary.PM_pop . . . 167
summary.PM_post . . . 169
summary.PM_pta . . . 170
summary.PM_sim . . . 171
summary.PM_valid . . . 173
three_comp_bolus . . . 173
three_comp_bolus_cl . . . 174
three_comp_iv . . . 174
three_comp_iv_cl . . . 174
two_comp_bolus . . . 175
two_comp_bolus_cl . . . 175
two_comp_iv . . . 175
two_comp_iv_cl . . . 176
zBMI . . . 176

Index **178**

---

ab *Initial range for primary parameter values*

Description

[Stable]

Define primary model parameter initial values as range. For nonparametric, this range will be absolutely respected. For parametric, the range serves to define the mean (midpoint) and standard deviation (1/6 of the range) of the initial parameter value distribution.

Usage

```
ab(min, max)
```

Arguments

`min`	Minimum value.
`max`	Maximum value.

---

ab_line *Add lines to plotly plot*

Description

[Stable]

Analogous to abline, draws horizontal, vertical or sloped reference lines.

Usage

```
ab_line(a = NULL, b = NULL, h = NULL, v = NULL, line = TRUE)
```

Arguments

`a`	Intercept y value in relative coordinates, i.e. 0 (bottom) to 1 (top). The x value is 0.
`b`	Slope
`h`	Y-intercept of horizontal line, in absolute coordinates
`v`	X-intercept of vertical line, in absolute coordinates
`line`	Controls characteristics of lines. This argument maps to plotly line attributes. `TRUE` will plot default lines. `FALSE` will suppress lines. If a list, can control many line characteristics, including overriding defaults. Use the plotly `plotly::schema()` command in the console and navigate to traces > scatter > attributes > line to see all the ways the line can be formatted. Most common will be:

- `color` Line color.
- `dash` Plotting character. See `plotly::schema()`, traces > scatter > attributes > line > dash > values.
- `width` Thickness in points.

Example: `line = list(color = "red", dash = "longdash", width = 2)` . Default is `line = list(color = "black", width = 1, dash = "dash")`.

Details

This function creates a line shape that can be added a plotly plot. See schema() > layout > layoutAttributes > shapes for details. Use only one of the following:

- a and b to specify a line with intercept and slope
- h to specify a horizontal line with y-intercept at h
- v to specify a vertical line with x-intercept at v

If using this function to add a line to an existing plot, it must be used with add_shapes. If used for a new plot, it can be included as an element in the layout list.

See Also

add_shapes

Examples

```
## Not run:
# add to an existing plot
NPex$op$plot() |>
  add_shapes(shapes = ab_line(v = 12))
# add to a new plot
plotly::plot_ly(x = 1:10, y = 1:10, type = "scatter", mode = "lines+markers") |>
  plotly::layout(shapes = ab_line(h = 5, line = list(color = "red", dash = "solid")))

## End(Not run)
```

---

add_renal *Estimate renal function using various equations*

Description

[Stable]

Adds columns to PM_data object for creatinine clearance or estimated glomerular filtration rate (eGFR).

Creatinine clearance can be estimated by:

- The Jelliffe equation for paired creatinines
- The Cockcroft-Gault equation using a single creatinine.

eGFR can be estimated by:

- The MDRD equation
- The CKD-EPI equation
- The Schwartz equation for children

Usage

```
add_renal(
  x,
  method,
  id = "id",
  wt = "wt",
  ht = "ht",
  male = "male",
  age = "age",
  black = "black",
  scr = "scr",
  bun = "bun",
  cysC = "cysC",
  SI = FALSE
)
```

Arguments

x	A PM_data data object
method	A character vector defining the method to use. Options are • "jelliffe" or "jel" • "cockcroft-gault" or "cg" • "mdrd" • "ckd-epi" or "ckd" • "schwartz" For the fuller versions, Only the first 3 letters are required
id	A character vector with the name of the id column in x. The default is "id".
wt	A character vector with the name of the weight column in x. The default is "wt". Only required for "jelliffe" and "cockcroft-gault" methods. Values must be in **kilograms**.
ht	A character vector with the name of the height column in x. The default is "ht". Only required for the "schwartz" method. Values must be in **meters**
male	A character vector with the name of the column defining sex in x. Male should be 1 and female should be 0. The default is "male".
age	A character vector with the name of the age column in x. The default is "age". Values must be in **years**.
black	A character vector with the name of a column defining black race in x. Male should be 1 and female should be 0. The default is "male".
scr	A character vector with the name of the serum creatinine column in x. Default units are **mg/dL**, unless SI = TRUE below. The default name is "scr".

bun	A character vector with the name of the blood urea nitrogen column in x. Default units are **mg/dL**, unless SI = TRUE below. Default name is "bun". Optional for the Schwartz method.
cysC	A character vector with the name of the cystatin C column in x. Units are always **mg/L**. Default is "cysC". Optional for the Schwartz method.
SI	Boolean value, if true, will expect serum creatinine to be in **micromol/L** and BUN reported as **micromol/L** of urea. Default is FALSE.

Details

Note: In all the equations below, age is in years, weight is in kilograms, height is in meters, and cystatin C (cysC) is in mg/L. Serum creatinine (Scr) is in mg/dL and blood urea nitrogen (BUN) is in mg/L. However, if SI = TRUE, then serum creatinine is in micromol/L and BUN is in micromol/L of urea.

Missing covariate values are interpolated using linear interpolation within each subject.

- The **Jelliffe** equation depends on age, sex, weight, and serum creatinine. It uniquely uses two serum creatinine values separated by time to estimate changing renal function.
  - $ESS = wt * (29.3 - (0.203 * age))$ for males
  - $ESS = wt * (25.1 - (0.175 * age))$ for females
  - $scrAve = (Scr1 + Scr2)/2$
  - $ESS_cor = ESS * (1.035 - (0.0337 * scrAve))$
  - $E = ESS_cor - 4 * wt * (Scr2 - Scr1)/(time2 - time1)$
  - $CRCL = E/(14.4 * scrAve)$ in ml/min/1.73m^2
- The **Cockroft-Gault** equation is a point estimate of creatinine clearance. It depends on the same variables as Jelliffe except it does not require paired creatinines.
  - $CRCL = ((140 - age) * weight * K)/(72 * Scr)$
  - K is 1 for males and 0.85 for females
- The **MDRD** equation estimates GFR. It depends on age, sex, black race, and serum creatinine.
  - $eGFR = 175 * (Scr)^{-1.154} * (age)^{-0.203}$
  - Multiply by 1.212 if black.
- The **CKD-EPI** equation estimates GFR. It depends on the same variables as MDRD, except black is no longer required in the 2021 version used here.
  - $eGFR = 141 * min(Scr/k, 1)^a * max(Scr/k, 1)^{-1.209} * 0.993^{age}$
  - Multiply by 1.1018 if female.
- The **Schwartz** equation estimates GFR in children. The equation used depends on which covariates are included in the data.
  - height and serum creatinine: $eGFR = 41.3 * height/scr$ (updated Schwartz equation)
  - height, serum creatinine, and cystatin C: $eGFR = 41.6 * (height/scr)^{0.599} * (1.8/cysC)^{0.317}$ (Equation 1a)
  - height, serum creatinine, and BUN: $eGFR = 40.7 * (height/scr)^{0.640} * (30/BUN)^{0.202}$ (Equation 1b)
  - height, serum creatinine, cystatin C, and BUN: $eGFR = 41.1 * (height/scr)^{0.510} * (1.8/cysC)^{0.272} * (30/BUN)^{0.171}$ (Equation II)

- height, serum creatinine, cystatin C, BUN, and male: $eGFR = 39.1 * (height/scr)^{0.516} * (1.8/cysC)^{0.294} * (30/BUN)^{0.169} * 1.099^{male} * (height/1.4)^{1.88}$ (Equation III)

Discussion of the strengths and weaknesses of these equations are beyond the scope of this documentation.

Value

A column added to the x data object with the name of the method used and calcuated values.

Author(s)

Michael Neely

`add_shapes` *Add shapes to plotly plot*

Description

[Stable]

Modifies the layout object of an existing plot to include a new shape.

Usage

```
add_shapes(p = plotly::last_plot(), shapes)
```

Arguments

`p`	The plot to which the shape should be added. Default is the `last_plot()`.
`shapes`	A list of attributes that specify a shape. Note that only one shape can be added for each call, but to be consistent with the `shapes` argument to `plotly::layout()`, we use the same plural.

Details

This function adds a shape to the layout element of a plotly plot. Type `schema()` in the console and expand the list for layout > layoutAttributes > shapes for details on how to specify a shape. A convenient Pmetrics helper function to add line shapes is ab_line. Other shapes such as circles, rectangles and paths can be added, but must be done manually as outlined in the plotly schema documentation. An example of a circle shape is below in examples.

See Also

ab_line

Examples

```
#'
## Not run:
NPex$op$plot() |>
  add_shapes(shapes = ab_line(v = 12))

NPex$data$plot() |>
  add_shapes(shapes = list(type = "circle", x0 = 125, y0 = 10, x1 = 135, y1 = 15))

## End(Not run)
```

`add_smooth` *Add regression to plotly plot*

Description

[Stable]

Modifies an existing plot to include a regression line with confidence interval.

Usage

```
add_smooth(
  p = plotly::last_plot(),
  x = NULL,
  y = NULL,
  data = NULL,
  method = c("lm", "loess"),
  span = 0.75,
  line = TRUE,
  ci = 0.95,
  stats = TRUE
)
```

Arguments

`p`	The plot to which the shape should be added. Default is the `plotly::last_plot()`.
`x`	X value for regression if not the original x value of the plot. Be sure to specify as a formula, e.g. `x = ~pred`.
`y`	Y value for regression if not the original y value of the plot. Be sure to specify as a formula, e.g. `y = ~obs`.
`data`	A secondary data object to use for regression other than the original plot data.
`method`	The tegression method, currently either "lm" (the default) for linear regression, or "loess" for loess regression.
`span`	Only used for `method = "loess"`. The span parameter to control the degree of smoothing. Default is 0.75.

`line` Controls characteristics of lines. This argument maps to plotly line attributes. `TRUE` will plot default lines. `FALSE` will suppress lines. If a list, can control many line characteristics, including overriding defaults. Use the plotly `plotly::schema()` command in the console and navigate to traces > scatter > attributes > line to see all the ways the line can be formatted. Most common will be:

- `color` Line color.
- `dash` Plotting character. See `plotly::schema()`, traces > scatter > attributes > line > dash > values.
- `width` Thickness in points.

Example: `line = list(color = "red", dash = "longdash", width = 2)` Default is `list(color = "blue", width = 2)`. The confidence interval will have the same color but at opacity 0.2.

`ci` Confidence interval for regressions. Default is 0.95. It can be suppressed by setting to 0.

`stats` Add the statistics from linear regression to the plot. Ignored if `method = "loess"`. If missing or `FALSE`, will be suppressed. Can be set to `TRUE` which results in default format of `list(x= 0.8, y = 0.1, bold = F, font = list(color = "black", family = "Arial", size = 14))`. The coordinates are relative to the plot with lower left = (0,0), upper right = (1,1). This argument maps to `plotly::add_text()`. It is also an option that can be set in setPMoptions to avoid specifying this argument for every plot. The default option is `TRUE`. If specified as a Pmetrics option, it can be overridden for specific plots by supplying a value.

Details

This function adds a regression line to an existing plotly plot. The default is to use the `x` and `y` values in the plot, but this can be overridden by specifying a data object. If another `data` object is used, values for `x` and `y` must also be specified. Alternatively, the original data can be used and new columns selected for regression by omitting the `data` argument and specifying new `x` and `y` values. The intent of this function is to replicate the behavior of `ggplot::geom_smooth()`.

See Also

add_shapes

Examples

```
## Not run:
plotly::plot_ly(mtcars,
  x = ~hp, y = ~mpg,
  type = "scatter", mode = "markers", showlegend = FALSE
) |>
  add_smooth()
plotly::plot_ly(iris,
  x = ~Sepal.Length, y = ~Petal.Length,
  type = "scatter", mode = "markers", showlegend = FALSE
) |>
  add_smooth(method = "loess", ci = 0.9, line = list(color = "red", dash = "dash"))
```

```
## End(Not run)
```

`additive` *Additive error model*

Description

[Stable]

Create an additive (lambda) error model

Usage

```
additive(initial, coeff, fixed = FALSE, outeq = 1)
```

Arguments

`initial`	Initial value for lambda
`coeff`	Vector of coefficients defining assay error polynomial
`fixed`	Estimate if `FALSE` (default).
`outeq`	The output equation this error model applies to, given either as the number used in the model's `out` block (e.g. `outeq = 2` for `Y[2]`) or as an output label declared in a DSL model. Defaults to `1`.

`all_is_numeric` *Check if all values are numeric*

Description

[Stable] Checks if all values in a vector are numeric.

Usage

```
all_is_numeric(x, what = c("test", "vector", "nonnum"), extras = c(".", "NA"))
```

Arguments

`x`	A vector to check.
`what`	A character string indicating what to return. Can be "test", "vector", or "nonnum". The default is "test".
`extras`	A character vector of extra values to exclude from the check. The default is c(".", "NA").

Details

The function checks if all values in a vector are numeric. It can be used to check if a vector contains only numeric values. It can also be used to check if a vector contains any non-numeric values.

Value

A logical value indicating if all values are numeric. If `what` is "vector", a numeric vector is returned. If `what` is "nonnum", a character vector of non-numeric values is returned. If `what` is "test", a logical value is returned.

Examples

```
## Not run:
all_is_numeric(c("1", "2", "3"))
all_is_numeric(c("1", "2", "a"))
all_is_numeric(c("1", "2", "3"), what = "vector")
all_is_numeric(c("1", "2", "a"), what = "nonnum")

## End(Not run)
```

---

apps *Launch standalone Pmetrics apps*

Description

`lit_sim()`, `model_lib()`, and `pm_plot()` launch standalone golem app packages.

Usage

```
lit_sim(launch.browser = TRUE)

pm_plot(launch.browser = TRUE, data_env = .GlobalEnv)

apps(launch.browser = TRUE)
```

Arguments

`launch.browser`	Logical. Should the app launch in a browser?
`data_env`	Environment passed to `pm_plot()` containing the data objects the app should discover and use. Defaults to `.GlobalEnv`. Ignored by launchers that do not accept this argument.

Details

`apps()` shows an interactive numbered menu (via `cli`) and launches the selected app using `readline()` input.

Value

Invisibly returns the value from the standalone app package's `run_app()` function for launcher functions. `apps()` returns invisibly `NULL` on cancel/invalid input.

`badData` *Pmetrics data file with errors*

Description

Example data set for an NPAG/IT2B run, which has been corrupted with errors.

Usage

```
badData
```

Format

PM_data

Details

Errors include missing covariate on first line for subject 1, alphanumeric covariate for gender, and trailing dose for subject 1.

Author(s)

Michael Neely

`cdc_bmi` *CDC Pediatric and Adolescent BMI Table*

Description

Centers for Disease Control Pediatric and Adolescent BMI Table

Usage

```
cdc_bmi
```

Format

A data frame with the following 9 columns: Sex (1 = male), Agemos; L, M, S (coefficients for calculating z-scores), P3, P5, P10, P25, P50, P75, P85, P90, P95, P97: age and sex specific BMI percentiles

Details

Coefficients to calculate sex-specific BMI z-scores and percentiles. Downloaded from . Tables were last updated in 2000, based on data through 1994. Definitions of overweight and obese come from these data, based on BMI percentile >=85 for overweight and >=95 for obese. See ger_bmi for percentiles based on more modern NHANES data.

Author(s)

Michael Neely

---

check_updates *Check for Pmetrics and R updates*

Description

[Stable] Performs an on-demand check for newer Pmetrics and R releases. This function is intended for interactive use and avoids running network checks automatically during package attach.

Usage

```
check_updates(verbose = interactive(), timeout = 2)
```

Arguments

verbose	Logical. If TRUE, emits a user-facing CLI summary.
timeout	Numeric scalar. Network timeout in seconds used for this check.

Value

An invisible list with installed/latest versions and outdated flags.

---

click_plot *Click on plotly plot to highlight traces*

Description

[Stable] Adds click functionality to a plotly plot to highlight traces when clicked.

Usage

```
click_plot(p, highlight_color = "#1f77b4")
```

Arguments

p	The plotly plot to which the click functionality should be added. Default is the `plotly::last_plot()`.
highlight_color	The color to use for traces that are highlighted. Colors for non-highlighted traces will be preserved but dimmed to 20% opacity. Default highlight color is `orange()`.

Details

This function modifies a plotly plot to allow clicking on traces to highlight them. Clicking on a trace will highlight it in orange (default) and dim all other traces. Clicking on the same trace again will deselect it and restore the original colors. Clicking on the background will also restore all traces to their original colors. The function uses the `htmlwidgets::onRender` function from the `htmlwidgets` package to add JavaScript code that handles the click events on the plotly plot.

Author(s)

Michael Neely

---

cor2cov *Convert correlation matrix to covariance matrix*

Description

[Stable] Converts a correlation matrix to a covariance matrix using standard deviations.

Usage

```
cor2cov(cor, sd)
```

Arguments

cor	A correlation matrix.
sd	A vector of standard deviations corresponding to the variables in the correlation matrix.

Details

Uses matrix multiplication to convert a correlation matrix to a covariance matrix.

Value

A covariance matrix.

Author(s)

Michael Neely

dataEx *Pmetrics data file*

Description

Example data set for an NPAG/IT2B run.

Usage

```
dataEx
```

Format

PM_data

Details

Data are kindly supplied by Chuck Peloquin, PharmD. They consist of multiple rifapentine oral doses followed by 6-7 concentrations in 20 adult subjects. Covariates include weight (kg), africa (origin), age (years), gender (1 = male), and height (cm).

Author(s)

Michael Neely

downloadR *Download the latest platform-specific R installer*

Description

[Stable] Downloads the latest R installer (or source tarball on Linux) for the current platform to the user's Downloads folder.

Usage

```
downloadR(r_info = latestR(), destdir = path.expand("~/Downloads"))
```

Arguments

r_info	Optional API response list. Defaults to latestR().
destdir	Destination directory. Defaults to the user's Downloads folder.

Value

The file path of the downloaded installer/tarball.

export_plotly *Export plotly plot*

Description

[Stable]

Wrapper around `plotly::save_image()`.

Usage

```
export_plotly(
  p,
  file,
  width = NULL,
  height = NULL,
  scale = NULL,
  units = "px",
  show = TRUE
)
```

Arguments

p	The plot to which the shape should be added. Default is the `plotly::last_plot()`.
file	A file path with a suitable file extension (png, jpg, jpeg, webp, svg, or pdf). Unlike `save_image`, the `file` argument may include the full path and filename other than in the current working directory.
width, height	The width/height of the exported image in pixels, mutliplied by `scale`.
scale	The scale factor to use when exporting the figure. Default is 1.0. A scale factor larger than 1.0 will increase the image size, and less than 1.0 will decrease the image size. Note that the documentation for `save_image` says that this argument changes the resolution, but that is not true. The resolution will remain at 72 pixels/inch (28.3 px/cm), which is the default for R. To increase the resolution, export to .pdf or .svg and use an external program, such as Adobe Acrobat, Acrobat Reader, or Mac Preview.
units	Units for `width` and `height`. Default is pixels ("px"). Alternatives are inches ("in") or centimeters ("cm")
show	Show the exported image in R via `base::file.show()`. If export format is pdf, it will open the system default pdf viewer. Default is `TRUE`.

Details

This function improves the experience with the native plotly method of exporting plots to static images. Much of the online documentation points towards using the orca package, but the R help indicates that this method has been superseded by the kaleido python package, made accesible in R via the reticulate package and installation of the miniconda package manager.

These steps are all outlined in the help for plotly::save_image(), but one step is neglected. It is necessary to execute the following line of code at the end: reticulate::py_run_string("import sys").

This function will check to see that all installations are in place and offer to install if not.

Many of the arguments are the same as for save_image and are passed directly to that function.

Value

Plot p will be exported to file with format determined by the extension for file.

Author(s)

Michael Neely

See Also

plotly::save_image()

Examples

```
## Not run:
NPex$op$plot(stats = list(x = 0.9)) |>
  export_plotly(file = "op.png", width = 12, height = 6, units = "in")

## End(Not run)
```

---

ger_bmi *CDC Pediatric and Adolescent BMI Table*

Description

Revised Pediatric and Adolescent BMI Table

Usage

```
ger_bmi
```

Format

A data frame with the following 9 columns: Sex (1 = male), Agemos; L, M, S (coefficients for calculating z-scores), P3, P5, P10, P25, P50, P75, P85, P90, P95, P97: age and sex specific BMI percentiles

Details

Coefficients to calculate sex-specific BMI z-scores and percentiles based on the supplemental data published by Gerhart et al: Gerhart, Jacqueline G., Fernando O. Carreño, Andrea N. Edginton, Jaydeep Sinha, Eliana M. Perrin, Karan R. Kumar, Aruna Rikhi, et al. "Development and Evaluation of a Virtual Population of Children with Obesity for Physiologically Based Pharmacokinetic Modeling." Clinical Pharmacokinetics 61, no. 2 (February 2022): 307–20. doi:10.1007/s40262021-010724. These data are in the same format as cdc_bmi but are derived from more recent NHANES data.

Author(s)

Michael Neely

---

getCov *Extract covariate information*

Description

[Stable]

Extracts covariate information from a Pmetrics data object.

Usage

```
getCov(mdata)
```

Arguments

mdata	A PM_data object. It can be other data frames but the results will likely not be meaningful.

Details

The function subtracts the number of fixed columns in a standard data format, currently 14, from the total number of columns in `mdata` and queries the remaining columns. When given a PM_data object, will return a list with the number of covariates, their names, and the starting and ending column numbers

Value

A list with named items: *ncov, covnames, covstart, covend.*

Author(s)

Michael Neely

Examples

```
## Not run:
getCov(dataEx)

## End(Not run)
```

`getDefaultColors` *Get a list of default colors*

Description

[Stable]

Generate list of default color names.

Usage

```
getDefaultColors(n)
```

Arguments

`n`	The number of colors to return from the list.

Details

Used for Pmetrics plots. The following list is recycled as necessary to generate the requested number of colors: `c(red(), green(), blue(), brown(), black(), purple(), pink(), gold(), orange(), gray())`. Each value is a function that returns a color name.

Value

A character vector of color names, which is recycled as needed.

Author(s)

Michael Neely

Examples

```
## Not run:
getDefaultColors(6)

## End(Not run)
```

getFixedColNum *Number of fixed columns*

Description

[Stable]

Returns the number of fixed columns (non-covariate) in Pmetrics data objects.

Usage

```
getFixedColNum()
```

Value

An integer with the number of fixed columns.

Author(s)

Michael Neely

Examples

```
## Not run:
getFixedColNum()

## End(Not run)
```

getPalettes *Get color palette*

Description

[Stable]

Generate list of palettes for plots.

Usage

```
getPalettes()
```

Details

If RColorBrewer package is installed, will return the list of palette names from RColorBrewer::brewer.pal.info. If not, will return the current list as of April 2024.

Value

A character vector of palette names.

Author(s)

Michael Neely

Examples

```
## Not run:
getPalettes()

## End(Not run)
```

`getPMoptions` *Get Pmetrics User Options*

Description

[Stable]

Get user options for Pmetrics

Usage

```
getPMoptions(opt, warn = TRUE, quiet = FALSE)
```

Arguments

`opt`	The option to retrieve. If omitted, all option values will be returned.
`warn`	Warn if options file doesn't exist. Default `TRUE`.
`quiet`	Suppress warning messages. Default `FALSE`.

Details

This function will get user options for Pmetrics. It will look for a *PMoptions.json* file in a hidden folder outside of the Pmetrics package. If that does not exist, it will look for a default options file in the package options folder. See setPMoptions for details on where the options file is stored and how to set options.

Value

A list with the current options.

Author(s)

Michael Neely

growth *CDC Pediatric and Adolescent Growth Data Table*

Description

Centers for Disease Control Pediatric and Adolescent Growth Data Table

Usage

```
growth
```

Format

A data frame with the following 9 columns: KNOT (integer age in months); A, B1, B2, B3 (coefficients for calculating percentiles), SEX, AGE, PERCENTILE, and CHART (length x age, wt x age, wt x length, hc x age, or ht x age).

Details

Coefficients to calculate sex-specific percentiles of length, weight and head cicumference data in children from 0 to 18 years. Downloaded and combined from . Used with the `qgrowth()` function to generate height and weight percentiles for the purposes of simulation.

Author(s)

Michael Neely

interp *Model covariate declaration*

Description

[Stable]

Declare whether covariates in the data are to have interpolation between values or not.

Usage

```
interp(type = "lm")
```

Arguments

`type`	If `type = "lm"` (the default) or `type = "linear"`, the covariate value will be linearly interpolated between values when fitting the model to the data. in a model list `cov` item. To fix covariate values to the value at the last time point, set `type = "none"`.

Value

A value of 1 for "lm" and 0 for "none", which will be passed to Rust.

Examples

```
## Not run:
cov <- c(
  wt = interp(), # same as interp("lm") or interp("linear")
  visit = interp("none")
)

## End(Not run)
```

---

latestR *Get latest platform-specific R release metadata*

Description

[Stable] Retrieves metadata for the latest R release available for the current platform from the r-hub rversions API.

Usage

```
latestR()
```

Value

A list containing all fields returned by the API response.

---

locales *Pmetrics locales*

Description

Pmetrics locales

Usage

```
locales
```

Format

Dataframe with languages and iso693 two- and three-letter codes.

Author(s)

Michael Neely

make_AUC *Calculation of AUCs*

Description

[Stable]

Calculates AUC from a variety of inputs

Usage

```
make_AUC(
  data = NULL,
  formula = NULL,
  include = NULL,
  exclude = NULL,
  start = 0,
  end = Inf,
  icen = "median",
  outeq = 1,
  block = 1,
  method = "linear",
  addZero = FALSE
)
```

Arguments

data	A suitable data object, i.e. PM_pop, PM_post, PM_op, PM_sim, or some other suitable dataframe with at least time/observation columns referred to by formula.
formula	A formula of the form obs ~ time | group. Default value for group is "id", so when if the data contain an "id" column, the formula can simply be obs ~ time. If the data contain a grouping variable, it can be specified as obs ~ time | group, where group is the name of the grouping variable. This is only required with data that is not of class PM_pop, PM_post, PM_op or PM_sim.
include	A vector of subject IDs to include in the plot, e.g. c(1:3,5,15)
exclude	A vector of subject IDs to exclude in the plot, e.g. c(4,6:14,16:20)
start	Specify the time to begin AUC calculations. Default is 0.
end	Specify the time to end AUC calculations so that AUC is calculated from start to end. Default for end is the maximum observation time for each subject. Subjects with insufficient data for a specified interval will have AUC calculated for the available data, not to exceed the specified interval.
icen	Can be either "median" for the predictions based on medians ofpred.type parameter value distributions, or "mean". Default is "median". Only relevant for PMpost or PMpop objects.

outeq	Which output equation to plot. Default is 1.
block	Which block to plot, where blocks are defined by dose reset events (EVID = 4) in the data.
method	Default is "linear" for AUC trapezoidal calculation. Any other value will result in linear up, log down.
addZero	Boolean to add a zero concentration at time 0. Default is FALSE.

Details

Calculates the area under the time concentration curve using the trapezoidal approximation from a variety of inputs. If a PM_pop, PM_post, PM_op, or PMsim object is specified, formula is not required.

Value

A dataframe of class *PMauc*, which has 2 columns:

- group - Subject identification, usually "id"
- tau - AUC from start to end

Author(s)

Michael Neely

See Also

PM_result, PM_sim, PM_op

Examples

```
## Not run:
NPex$cov$plot(V ~ wt)
NPex$cov$plot(Ke ~ wt, line = list(lm = TRUE, ref = FALSE, loess = FALSE))
NPex$cov$plot(Ke ~ wt, line = list(loess = list(ci = 0.9, color = "green")))
NPex$cov$plot(V ~ time, marker = list(color = "blue"))
NPex$cov$plot(V ~ wt,
  line = list(lm = TRUE, loess = FALSE),
  stats = list(x = 0.5, y = 0.2, font = list(size = 7, color = "blue"))
)

## End(Not run)
```

make_NCA *Non-compartmental analysis*

Description

Performs a non-compartmental analysis from a PM_result object using observed concentrations in the raw data file `PM_result$data$standard_data` or from an individual Bayesian posterior predicted time-observation profiles `PM_result$post$data` generated automatically after an NPAG run and loaded with PM_load .

Usage

```
make_NCA(
  x,
  postPred = FALSE,
  include,
  exclude,
  input = 1,
  icen = "median",
  outeq = 1,
  block = 1,
  start = 0,
  end = Inf,
  first = NA,
  last = NA,
  terminal = 3
)
```

Arguments

x	PM_data object to analyze.
postPred	Boolean switch to use the posterior predictions rather than the observed. concentrations. Default is `FALSE`. Ignored if an IT2B run is used to supply the raw data file.
include	A vector of subject IDs to include in the NCA, e.g. `c(1:3,5,15)`
exclude	A vector of subject IDs to exclude in the NCA, e.g. `c(4,6:14,16:20)`. When `postPred` is `TRUE`, any subject(s) excluded from the IT2B/NPAG run will be excluded as well.
input	The number of the input (e.g. drug) to analyze; default 1.
icen	If `postPred` is `TRUE`, use predictions based on median or mean of each subject's Bayesian posterior parameter distribution. Default is "median", but could be "mean".
outeq	The number of the output equation to analyze; default 1
block	The number of the observation block within subjects, with each block delimited by EVID=4 in the data file; default 1

start	The beginning of the time interval to look for doses and observations, e.g. 120. It can be a vector to allow for individual start times per subject, e.g. c(120,120,144,168). If the length of start is less than the number of subjects, the last value will be recycled as needed. If the start time is not 0 (default), then it is assumed that steady state (multiple dose) conditions apply.
end	Analogous to start, set this equal to the end of the dosing interval. It too can be a vector, with the last value recycled as necessary. Default is Inf, i.e. all data used.
first	Alternative way to specify time interval for NCA by choosing dose number, e.g. 1 or 3. May be a numeric vector, like start and end, e.g. c(1,1,1,3,1,...) to allow for individualization by subject. The last value will be recycled to ensure length equal to the number of subjects. Default is NA, which means start will be used.
last	The complement to first, specifying the last dose to end the time interval. If NA, which is the default, then the maximum time per subject will be the upper bound of the time interval.Like first, last can be a vector, with the last value recycled as necessary. Use NA in the vector to signify maximum time for that subject.
terminal	Number of observations to use for terminal curve fitting (i.e. to estimate k). Default is 3.

Details

If concentrations from multiple dose intervals are included in the start to end time interval, make_NCA will superpose the concentrations using the time after dose. An error will be generated if different doses are within this interval as superposition would no longer be valid.

A minimum of 5 concentrations must be available to perform NCA for any given subject. Fewer than this will suppress all results for that subject.

Value

A dataframe of class *PMnca* with columns

id	Subject identification
auc	Area under the time-observation curve, using the trapezoidal approximation, from time 0 until the second dose, or if only one dose, until the last observation
aumc	Area under the first moment curve
k	Slope by least-squares linear regression of the final 3 log-transformed observations vs. time. If the final 3 concentrations are not decreasing such that linear regression results in a positive slope, this value and all others that depend on k will be suppressed.
auclast	Area under the curve from the time of the last observation to infinity, calculated as Final obs/k. This value will be suppressed if start != 0.
aumclast	Area under the first moment curve from the time of the last observation to infinity. This value will be suppressed if start != 0.
aucinf	Area under the curve from time 0 to infinity, caluculated as auc + auclast

aumcinf	Area under the first moment curve from time 0 to infinity
mrt	Mean residence time, calculated as 1/k
cmax	Maximum predicted concentration after the first dose
tmax	Time to cmax
cl	Clearance, calculated as dose/aucinf
vdss	Volume of distribution at steady state, calculated as cl*mrt
thalf	Half life of elimination, calculated as ln(2)/k
dose	Dose for each subject

Author(s)

Michael Neely

`makeErrorPoly` *Assay error polynomial coefficients*

Description

[Stable]

Estimate coefficients for the polynomial to describe assay error.

Usage

```
makeErrorPoly(
  obs,
  sd,
  data,
  outeq = 1,
  col = "red",
  cex = 3,
  pch = "+",
  lcol = "blue",
  lwd = 2,
  ref = T,
  legend = T,
  ...
)
```

Arguments

obs	A vector of observations
sd	A vector of standard deviations obtained from repeated measurements at each observation in obs

data	A Pmetrics data file. From this, the maximum and mininimum observations will be retrieved. This is useful to ensure that calculated standard deviations are not negative at any observation in the dataset. If not specified, the default is the maximum *obs*.
outeq	The output equation in *data*. Default is 1.
col	Color of the data points. Default is red.
cex	Relative size of the data points. Default is 3. See par.
pch	Ploting symbol. Default is "+". See par.
lcol	Color of the fitted polynomial lines. Default is blue.
lwd	Width of the lines. Default is 2.
ref	Add a reference line at SD 0 to help evaluate that all fitted SDs are >0. Default is true.
legend	Boolean argument to plot legend. Default is TRUE.
...	Other plotting parameters as in plot.default and par

Details

This function plots first, second, and third order polynomial functions fitted to pairs of observations and associated standard deviations for a given output assay. In this way, the standard deviation associated with any observation may be calculated and used to appropriately weight that observation in the model building process. Observations are weighted by the reciprocal of the variance, or squared standard deviation.

Value

A plot of the measured observations and fitted polynomial curves and a list with the first, second, and third order coefficients

Author(s)

Michael Neely

Examples

```
makeErrorPoly(obs = c(0, 5, 50, 100, 250, 500, 1000), sd = c(1, 0.4, 4.5, 12, 34, 60, 190))
```

---

makePTAtarget *Make a Percent Target Attainment (PTA) Target*

Description

[Stable]

Generates an object of class *PMpta.targ* which can be used in the $new() method for PM_pta or $pta() method for PM_sim for targets sampled from a distribution.

Usage

```
makePTAtarget(x)
```

Arguments

x	A data.frame or name of .csv file in working directory whose first two columns are targets and the number of samples for each target. An example can be seen for Staphylococcus aureus susceptibility to vancomycin at EUCAST.

Value

A data frame with two columns named targets and n, of class *PMpta.targ*.

See Also

PM_pta

Examples

```
## Not run:
makePTAtarget(mic1)

## End(Not run)
```

---

mic1 *Example MIC data*

Description

Example MIC data

Usage

```
mic1
```

Format

An R data frame containing example MIC distribution data in two columns:

- mic Minimum inhibitory concentration
- n Number of organisms with the given MIC

Details

This data frame contains MIC data for vancomycin against S. aureus. It was obtained from the EUCAST website at . Select the organism or drug, and then select the desired row of the resulting table to see a histogram (top) and table (bottom) of MIC distributions.

Copy the table into excel, save as a .csv file, and read into R using a function like `read.csv()` or `readr::read_csv()`. Then use `makePTAtarget()`.

Author(s)

Michael Neely

`model_lib` *Model Library*

Description

Launches a Shiny app for browsing all available pharmacokinetic model templates in the Pmetrics library. Select a model to view its parameters, compartment structure, differential equations, and ready-to-use `PM_model$new()` code. Use the Copy button to send the code to the clipboard. Individual creator functions such as `one_comp_iv()` return a compiled `PM_model` and copy the code automatically.

Usage

```
model_lib(show = TRUE, launch.browser = TRUE)
```

Arguments

`show`	Logical. If `TRUE` (default), launches the Shiny browser app. If `FALSE`, returns the model tibble invisibly without launching the app.
`launch.browser`	Logical. Passed to the standalone app package's `run_app()` function when `show = TRUE`.

Value

Invisibly, a tibble of all model templates.

`modEx` *Pmetrics model object*

Description

Example model for an NPAG/IT2B run. There are 4 parameters in the model: lag time of absorption (Tlag1), rate constant of absorption (Ka), volume (V) and rate constant of elmination (Ke). There are 5 covariates: weight in kg (WT), whether from Africa or not (AFRICA), age in years (AGE), 1 for male (GENDER), and height in cm (HEIGHT). There is one output equation, and the model uses gamma plus an error polynomial derived from the assay.

Usage

```
modEx
```

Format

R6 PM_model

Author(s)

Michael Neely

`msd` *Initial mean/SD for primary parameter values*

Description

[Stable]

Define primary model parameter initial values as mean and standard deviation, which translate to a range. The mean serves as the midpoint of the range, with 3 standard deviations above and below the mean to define the min and max of the range. For nonparametric, this range will be absolutely respected. For parametric, values can be estimated beyond the range.

Usage

```
msd(mean, sd)
```

Arguments

`mean`	Initial mean.
`sd`	Initial standard deviation.

`mtsknn.eq` *Compare discrete distributions*

Description

[Stable]

Multivariate two-sample test based on k-nearest neighbors

Usage

```
mtsknn.eq(x, y, k, clevel = 0.05, getpval = TRUE, print = TRUE)
```

Arguments

x	A matrix or data frame.
y	A matrix or data frame.
k	k An integer.
clevel	The confidence level. Default value is 0.05.
getpval	Logic value. If it is set to be TRUE the p value of test will be calcuated and reported; if it is set to be false the p value will not be calculated.
print	Boolean value. If it is set to be TRUE the test result will be reported; if it is set to be FALSE the test result will not be reported.

Details

This function tests whether two samples share the same underlying distribution based on k-nearest-neighbors approach. Matrices or data frames x and y are the two samples to be tested. Each row consists of the coordinates of a data point. The integer k is the number of nearest neighbors to choose in the testing procedure. This approach is robust in the unbalanced case.

Value

A list consists of the test statistics, normalized Z score and corresponding P value.

Author(s)

Lisha Chen (Yale), Peng Dai (Stonybrook) and Wei Dou (Yale)

References

Schilling, M. F. (1986). Multivariate two-sample tests based on nearest neighbors. *J. Amer. Statist. Assoc.*, 81 799-806. Henze, N. (1988). A multivariate two-sample test based on the number of nearest neighbor type coincidences.*Ann. Statist.*, 16 772-783. Chen, L. and Dou W. (2009). Robust multivariate two-sample tests based on k nearest neighbors for unbalanced designs. *manuscripts*.

NM2PM *Convert NONMEM to Pmetrics Data Files*

Description

[Stable]

NM2PM will convert NONMEM .csv data files to Pmetrics csv data files.

Usage

```
NM2PM(data, ctl)
```

Arguments

data	The name and extension of a NONMEM data (e.g. .csv) file in the working directory, or the full path to a file.
ctl	The name and extension of a NONMEM control (e.g. .ctl) file in the working directory, or the full path to a file.

Details

The format of NONMEM and Pmetrics data .csv files are similar, but not quite identical. A major difference is that the order of the columns are fixed in Pmetrics (not including covariates), while they are user-determined in NONMEM, and specified in a control (.ctl) file.

A list of other differences follows by data item.

- ID This item is the same in both formats and is required.
- EVID This is the same in both formats but is not required in NONMEM. Doses have an EVID of 1 and observations 0. EVID=4 (dose/time reset) is the same in Pmetrics and NONMEM. EVID=2 (other event) and EVID=3 (dose reset) are not directly supported in Pmetrics, but if included in a NONMEM file, will be converted into covariate values. Specifically the value in the CMT variable will be the covariate value for EVID=2, while for EVID=3, the covariate will be 1 at the time of the EVID=3 entry and 0 othewise. This allows for handling of these events in the Pmetrics model file using conditional statements.
- DATE Pmetrics does not use dates, but will convert all NONMEM dates and times into relative times.
- TIME Pmetrics uses relative times (as does NONMEM), but the NONMEM pre-processor will convert clock times to relative times, as does NM2PM.
- RATE NONMEM RATE items are converted by this function to Pmetrics DURation values.
- AMT becomes DOSE in Pmetrics
- ADDL is supported in both formats. However, if NONMEM files contain an SS flag, it will be incorporated as ADDL=-1 according to Pmetrics style.
- II is the same in both formats.
- INPUT in Pmetrics is similar to CMT in NONMEM for doses.
- DV in NONMEM becomes OUT in Pmetrics. Ensure that the units of OUT are consistent with the units of DOSE.
- OUTEQ In Pmetrics, this is roughly equivalent to CMT in NONMEM for observation events. The lowest CMT value for any observation becomes OUTEQ=1; the next lowest becomes OUTEQ=2, etc.
- SS Steady state dosing is incorporated into Pmetrics as ADDL=-1.
- MDV Missing DV in NONMEM become OUT=-99 in Pmetrics.
- Covariates These are copied from NONMEM to Pmetrics. Note that Pmetrics does not allow missing covariates at time 0 for each subject.
- DROP Items marked as DROP in the NONMEM control file will not be included in the Pmetric data file.

It is strongly suggested to run PMcheck on the returned object for final adjusting.

Value

A Pmetrics style PMmatrix data.frame.

Author(s)

Michael Neely

See Also

PMcheck, PMwriteMatrix, PMwrk2csv

NPex *Example NPAG Output with validation*

Description

Example output from an NPAG run with validation.

Usage

```
NPex
```

Format

R6 PM_result

Details

This is an R6 Pmetrics PM_result object created with `PM_load()` after an NPAG run. The run consisted of a model with an absorptive compartment and a central compartment. There were 4 parameters in the model: lag time of absorption (Tlag1), rate constant of absorption (Ka), volume (V) with weight as a covariate, and rate constant of elmination (Ke). There were 20 subjects in the dataset. The run was 100 cycles long and did not converge. It was then validated with the `$validate` method for PM_result objects.

Author(s)

Michael Neely

one_comp_bolus *One-compartment bolus model template*

Description

Create a PM_model object for the one_comp_bolus model template.

Usage

```
one_comp_bolus()
```

Value

A PM_model object.

one_comp_bolus_cl *One-compartment bolus clearance model template*

Description

Create a PM_model object for the one_comp_bolus_cl model template.

Usage

```
one_comp_bolus_cl()
```

Value

A PM_model object.

one_comp_iv *One-compartment IV model template*

Description

Create a PM_model object for the one_comp_iv model template.

Usage

```
one_comp_iv()
```

Value

A PM_model object.

one_comp_iv_cl *One-compartment IV clearance model template*

Description

Create a PM_model object for the one_comp_iv_cl model template.

Usage

```
one_comp_iv_cl()
```

Value

A PM_model object.

opposite_color *Find opposite color with max contrast*

Description

[Stable] Finds an opposite color with maximum contrast to the input color.

Usage

```
opposite_color(
  col,
  method = c("complement", "complement_maxcontrast", "bw_maxcontrast"),
  degrees = 180
)
```

Arguments

col	A color name or hex string (e.g. "red", "#FF0000", "#FF0000FF").
method	The method to use for finding the opposite color. One of "complement", "complement_maxcontrast", or "bw_maxcontrast". Default is "complement".
degrees	The degree offset for the hue complement. Default is 180.

Details

This function takes a color input (name or hex) and returns an opposite color using one of three methods:

- "complement": strict 180 degrees hue complement
- "complement_maxcontrast": 180 degrees hue complement adjusted for maximum contrast
- "bw_maxcontrast": black or white, whichever has higher contrast The function uses the WCAG relative luminance and contrast ratio formulas to determine contrast.

Value

A hex color string in the format "#RRGGBBAA".

`plot.PM_cov` *Plot Pmetrics Covariate objects*

Description

[Stable]

Plot PMcov objects

Usage

```
## S3 method for class 'PM_cov'
plot(
  x,
  formula,
  line = list(lm = NULL, loess = NULL, ref = NULL),
  marker = TRUE,
  colors,
  icen = "median",
  include = NULL,
  exclude = NULL,
  legend,
  log = FALSE,
  grid = TRUE,
  xlab,
  ylab,
  title,
  stats = TRUE,
  xlim,
  ylim,
  print = TRUE,
  ...
)
```

Arguments

`x`	The name of an PM_cov data object and loaded with PM_load as a PM_result, e.g. `PM_result$cov`.
`formula`	This is a mandatory formula of the form `y ~ x`, where `y` and `x` are the two `data` parameters to plot.
`line`	Controls characteristics of lines. Unlike some other Pmetrics plots, but like plot.PM_op, `line` is a list of three elements:

- `lm` If set to `TRUE` or a list of plotly line attributes, will generate a linear regression of the form y ~ x Line attributes will control the appearance of the regression line and the confidence interval around the line. If set to `FALSE`, no linear regression will be generated. The default values for the elements of the `lm` list, all of which can be overriden are:
  - `ci` Confidence interval around the regression, default 0.95.
  - `color` Color of the regression line and the confidence area around the line, but at opacity = 0.2. Default is "dodgerblue".
  - `width` Width of the regression line, default 1.
  - `dash` See `plotly::schema()`, traces > scatter > attributes > line > dash > values. Default is "solid". Example: `line = list(lm = list(color = "red", dash = "longdash", width = 2))`
- `loess` If set to `TRUE` or a list of plotly line attributes, will generate a loess regression of the form y ~ x The list elements and default values in the `loess` list are the same as for `lm` except the default style is "dash". Example: `line = list(lm = FALSE, loess = TRUE)`
- `ref` If set to `TRUE` or a list of plotly line attributes, will generate a reference line with slope = 1 and intercept = 0. The default values for the elements of the `ref` list are:
  - `color` "grey".
  - `width` 1.
  - `dash` "dot". Note that there is no *ci* argument for the *ref* list. Example: `line = list(lm = FALSE, loess = TRUE, ref = list(color = "lightgrey"))` If the `line` argument is missing, it will be set to `line = list(lm = FALSE, loess = TRUE, ref = FALSE)`, i.e. there will be a linear regression with reference line, but no loess regression. If *time* is chosen as the x variable in the formula, linear, loess and reference lines will be suppressed, although formatting specified in the loess line (except color, see below) will be applied to the lines joining the subject values.

`marker` Controls the plotting symbol for observations. This argument maps to the plotly marker object. It can be boolean or a list. `TRUE` will plot the marker with default characteristics. `FALSE` will suppress marker plotting. If a list, can control many marker characteristics, including overriding defaults. Use the plotly `plotly::schema()` command in the console and navigate to traces > scatter > attributes > marker to see all the ways the marker can be formatted. Most common will be:

- `color` Marker color.
- `symbol` Plotting character. See `plotly::schema()`, traces > scatter > attributes > marker > symbol > values.
- `size` Character size in points.
- `opacity` Ranging between 0 (fully transparent) to 1 (fully opaque).
- `line` A list of additional attributes governing the outline for filled shapes, most commonly color and width.

Example: `marker = list(color = "red", symbol = "triangle", opacity = 0.8, line = list(color = "black", width = 2))` Default is `marker = list(color = orange, shape = "circle", size = 10, opacity = 0.5, line = list(color = black, width = 1))`.

colors — to use for subjects when *time* is set as the x parameter. This can be a palette or a vector of colors. For accepted palette names see `RColorBrewer::brewer.pal.info`. Examples include "BrBG", or "Set2". An example vector could be `c("red", "green", "blue")`. It is not necessary to specify the same number of colors as groups within `color`, as colors will be interpolated to generate the correct number. The default is the "Spectral" palette. This will override any color in the `marker` or `line`.

icen — Can be either "median" for the predictions based on medians of`pred.type` parameter value distributions, or "mean". Default is "median".

include — A vector of subject IDs to include in the plot, e.g. c(1:3,5,15)

exclude — A vector of subject IDs to exclude in the plot, e.g. c(4,6:14,16:20)

legend — Not used for this function.

log — Boolean operator to plot the y axis in log base 10. This argument maps to the yaxis type value in the layout object in plotly. Use the plotly `plotly::schema()` command in the console and navigate to layout > layoutAttributes > yaxis > type.

Example: `log = T`

grid — Controls grid display. This argument maps to the xaxis and yaxis layout objects in plotly. Use the plotly `plotly::schema()` command in the console and navigate to layout > layoutAttributes > xaxis or yaxis > gridcolor or gridwidth. It is a Boolean operator. If `FALSE`, no grid is plotted. If `TRUE`, the default color *grey* and width 1 will be plotted at major tick marks. If a list, color and width can be customized.

Examples:

- `grid = F`
- `grid = list(gridcolor = "black", gridwidth = 2)`

xlab — Value for x axis label. This argument maps to the the xaxis title element of the layout object in plotly. It can simply be a character vector of length 1 that specifies the name of the axis, or it can be a list for greater control. Use the plotly `plotly::schema()` command in the console and navigate to layout > layoutAttributes > xaxis > title to see the ways to customize this axis label. In addition to the plotly attributes, a custom Pmetrics attribute `bold` may be included as a list element, either on its own or within the font list. The default for `bold` is `TRUE`.

Examples:

- `xlab = "Time (h)"`
- `xlab = list(text = "Time", bold = F, font = list(color = "red", family = "Arial", size = 10))`
- `xlab = list(text = "Time", font = list(bold = T))`

If missing, will default to the name of the x variable in the formula.

ylab — Value for y axis label. This argument maps to the the yaxis title element of the layout object in plotly. See `xlab` for details. If `xlab` is specified as a list with formatting, and `ylab` is simply a character label, then the formatting for the `xlab` will be applied to the `ylab`. To format `ylab` independently, specify a formatting list as for `xlab`.

If missing, will default to the name of the y variable in the formula.

`title` Plot title. This argument maps to the the title layout object in plotly. It can simply be a character vector of length 1 that specifies the name of the plot title, or it can be a list for greater control. Use the plotly `plotly::schema()` command in the console and navigate to layout > layoutAttributes > title to see other ways to customize the title using lists as additional arguments. In addition to the plotly attributes, a custom Pmetrics attribute `bold` may be included as a list element. The default for `bold` is `TRUE`.

Examples:

- `title = "Observed vs. Predicted"`
- `title = list(text = "Raw Data", font = list(color = "red", family = "Arial", size = 10, bold = T))`

Default is to have no title.

`stats` Add the statistics from linear regression to the plot. If `FALSE`, will be suppressed. Default is `TRUE` which results in default format of `list(x= 0.8, y = 0.1, bold = F, font = list(color = "black", family = "Arial", size = 14))`. The coordinates are relative to the plot with lower left = (0,0), upper right = (1,1). This argument maps to `plotly::add_text()`.

`xlim` Limits of the x axis as a vector. This argument maps to the the xaxis range in the layout object in plotly. Use the plotly `plotly::schema()` command in the console and navigate to layout > layoutAttributes > xaxis > range.

Example: `xlim = c(0,1)`

`ylim` Limits of the y axis as a vector. This argument maps to the the yaxis range in the layout object in plotly. Use the plotly `plotly::schema()` command in the console and navigate to layout > layoutAttributes > yaxis > range.

Example: `ylim = c(0,100)`

`print` If `TRUE`, will print the plotly object and return it. If `FALSE`, will only return the plotly object.

`...` Other attributes which can be passed to the layout > xaxis/yaxis in a plotly plot to further control formatting. Note that `log`, `xlab`, `ylab`, `xlim`, and `ylim` are all controlled by the layout object, but are provided throughout Pmetrics plotly function arguments as shortcuts that map to layout elements. Therefore, the dots argument should be used to specify other aspects of the x axis, y axis, or both. See `plotly::schema()` layout > layoutAttributes > xaxis/yaxis for options. To add to single axis, name it as a list. If attributes are specified without an enclosing xaxis or yaxis list, they will be applied to both axes.

Examples:

- `NPex$data$plot(xaxis = list(tickcolor = "black", tickfont = list(family = "Arial", size = 14, color = "black"))) #applies to x axis only`
- `NPex$data$plot(linecolor = "red", ticks = "inside") #applies to both axes`

.

Details

This method will plot any two columns, specified using a formula, of a PMcov object, which contains covariate and Bayesian posterior parameter information for each subject. Specifiying any two

variables that do not include time will result in a scatter plot with optional regression and reference lines. If time is included as the x variable, the y variable will be plotted vs. time, aggregated by subject. This can be useful to see time varying parameters, although a formula within formula approach may be required, e.g. `$plot(I(cl_0*wt**0.75) ~ time)` in order to see the change in cl over time according to the change in wt over time, even though cl_0 is constant for a given subject.

Value

Plots the object.

Author(s)

Michael Neely

See Also

PM_cov, PM_result, schema

Other PMplots: `plot.PM_cycle()`, `plot.PM_data()`, `plot.PM_final()`, `plot.PM_model()`, `plot.PM_op()`, `plot.PM_opt()`, `plot.PM_pop()`, `plot.PM_post()`, `plot.PM_pta()`, `plot.PM_sim()`, `plot.PM_valid()`

Examples

```
#'
## Not run:
NPex$cov$plot(V ~ wt)
NPex$cov$plot(Ke ~ wt, line = list(lm = TRUE, ref = FALSE, loess = FALSE))
NPex$cov$plot(Ke ~ wt, line = list(loess = list(ci = 0.9, color = "green")))
NPex$cov$plot(V ~ time, marker = list(color = "blue"))
NPex$cov$plot(V ~ wt,
  line = list(lm = TRUE, loess = FALSE),
  stats = list(x = 0.5, y = 0.2, font = list(size = 7, color = "blue"))
)

## End(Not run)
```

---

plot.PM_cycle *Plot Cycle Information*

Description

[Stable]

Plot PM_cycle objects. These objects are created by as part of a PM_result object when PM_load is run.

Usage

```
## S3 method for class 'PM_cycle'
plot(
  x,
  line = TRUE,
  marker = TRUE,
  colors,
  linetypes,
  omit,
  grid = TRUE,
  xlab,
  ylab,
  print = TRUE,
  ...
)
```

Arguments

x	The name of a PM_cycle object, e.g. NPex$cycle.
line	Controls characteristics of lines. This argument maps to plotly line attributes. TRUE will plot default lines. FALSE will suppress lines. If a list, can control many line characteristics, including overriding defaults. Use the plotly plotly::schema() command in the console and navigate to traces > scatter > attributes > line to see all the ways the line can be formatted. Most common will be: • color Line color. • dash Plotting character. See plotly::schema(), traces > scatter > attributes > line > dash > values. • width Thickness in points. Example: line = list(color = "red", dash = "longdash", width = 2) Default = list(color = "dodgerblue", width = 2, dash = "solid"). **Note** The width will apply to all plots, but color and dash will only apply to the first three plots (log-likelihood, AIC, gamma/lambda). Use colors and linetypes below to control the appearance of the line traces for the normalized plots, because each of those traces is mapped to a parameter.
marker	Controls the plotting symbol for observations. This argument maps to the plotly marker object. It can be boolean or a list. TRUE will plot the marker with default characteristics. FALSE will suppress marker plotting. If a list, can control many marker characteristics, including overriding defaults. Use the plotly plotly::schema() command in the console and navigate to traces > scatter > attributes > marker to see all the ways the marker can be formatted. Most common will be: • color Marker color. • symbol Plotting character. See plotly::schema(), traces > scatter > attributes > marker > symbol > values. • size Character size in points.

- opacity Ranging between 0 (fully transparent) to 1 (fully opaque).
- line A list of additional attributes governing the outline for filled shapes, most commonly color and width.

Example: marker = list(color = "red", symbol = "triangle", opacity = 0.8, line = list(color = "black", width = 2)) Here, the observation controlled is the value of a given trace at a specific cycle number. Default = list(symbol = "circle", color = "dodgerblue", size = 4). **Note** the marker color for the normalized parameter value plots will be controlled by the colors parameter below, but size and symbol will apply to all plots.

colors to use for normalized parameter value line traces. This can be a palette or a vector of colors. For accepted palette names see RColorBrewer::brewer.pal.info. Examples include "BrBG", or "Set2". An example vector could be c("red", "green", "blue"). It is not necessary to specify the same number of colors parameters to be plotted, as colors will be interpolated to generate the correct number. The default when color is not specified is the "Set2" palette.

linetypes to use for normalized parameter value line traces. See plotly::schema(), traces > scatter > attributes > line > dash > values. An example vector could be c("solid", "dash", "longdash"). It is not necessary to specify the same number of linetype parameters to be plotted, as they will be recycled to generate the correct number. The default when linetypes is not specified is "solid".

omit Decimal between 0 and 1 specifying the proportion of "burn-in" cycles to omit from the plots. If missing, the first 20% will be omitted.

grid Controls grid display. This argument maps to the xaxis and yaxis layout objects in plotly. Use the plotly plotly::schema() command in the console and navigate to layout > layoutAttributes > xaxis or yaxis > gridcolor or gridwidth. It is a Boolean operator. If FALSE, no grid is plotted. If TRUE, the default color *grey* and width 1 will be plotted at major tick marks. If a list, color and width can be customized.

Examples:

- grid = F
- grid = list(gridcolor = "black", gridwidth = 2)

xlab Controls the formatting of the x-axis label. The text is fixed by the function and cannot be altered. Use the plotly plotly::schema() command in the console and navigate to layout > layoutAttributes > xaxis > title to see the ways to customize this axis label. In addition to the plotly attributes, a custom Pmetrics attribute bold may be included as a list element, either on its own or within the font list. The default for bold is TRUE.

Examples:

- xlab = list(bold = F, font = list(color = "red", family = "Arial", size = 10))
- xlab = list(font = list(bold = T))

ylab Format for y-axis label. This argument maps to the the yaxis title element of the layout object in plotly. See xlab for details. If xlab is specified as a list with formatting, then the formatting for the xlab will be applied to the ylab. To format ylab independently, specify a formatting list as for xlab.

print	If TRUE, will print the plotly object and return it. If FALSE, will only return the plotly object.
...	Additional R plotting parameters.

Value

Plots a panel with the following windows: -2 times the log-likelihood at each cycle, gamma/lambda at each cycle; Akaike Information Criterion at each cyle and Bayesian (Schwartz) Information Criterion at each cycle, the mean parameter values at each cycle (normalized to starting values); the normalized standard deviation of the population distribution for each parameter at each cycle; and the normalized median parameter values at each cycle.

Author(s)

Michael Neely

See Also

PM_result, schema

Other PMplots: plot.PM_cov(), plot.PM_data(), plot.PM_final(), plot.PM_model(), plot.PM_op(), plot.PM_opt(), plot.PM_pop(), plot.PM_post(), plot.PM_pta(), plot.PM_sim(), plot.PM_valid()

Examples

```
## Not run:
NPex$cycle$plot()
NPex$cycle$plot(omit = 0, marker = list(symbol = "cross"), line = list(width = 1))
NPex$cycle$plot(
  linetypes = "dash", colors = "Blues", marker = list(size = 1),
  line = list(width = 3)
)
NPex$cycle$plot(
  grid = FALSE,
  xlab = list(bold = FALSE, font = list(color = "red", family = "Arial", size = 10))
)

## End(Not run)
```

---

plot.PM_cycle_data *Plot PM_cycle_data objects*

Description

[Stable] Plots the raw data (class: PM_cycle_data) from a PM_cycle object in the same way as plotting a PM_cycle object. Both use plot.PM_cycle.

Usage

```
## S3 method for class 'PM_cycle_data'
plot(x, ...)
```

Arguments

x	A PM_cycle_data object
...	Additional arguments passed to plot.PM_cycle

Examples

```
# There is no example we can think of to filter or otherwise process a PM_cycle object,
# but we provide this function for completeness.
NPex$cycle$data |> plot()
```

---

plot.PM_data *Plot PM_data Time-Output Data*

Description

[Stable]

Plots *PM_data* objects

Usage

```
## S3 method for class 'PM_data'
plot(
  x,
  include = NULL,
  exclude = NULL,
  line = list(join = TRUE, pred = FALSE),
  marker = TRUE,
  group = NULL,
  group_names = NULL,
  mult = 1,
  outeq = 1,
  out_names = NULL,
  block = 1,
  tad = FALSE,
  overlay = TRUE,
  legend,
  log = FALSE,
  grid = FALSE,
  xlab = "Time",
  ylab = "Output",
  title = "",
```

```
  xlim,
  ylim,
  print = TRUE,
  ...
)
```

Arguments

x	The name of an PM_data data object or loaded as a field in a PM_result object
include	A vector of subject IDs to include in the plot, e.g. c(1:3,5,15)
exclude	A vector of subject IDs to exclude in the plot, e.g. c(4,6:14,16:20)
line	Controls characteristics of lines as for all plotly plots. Here line is a list of two elements: • join Can either be a boolean or a list. If set to TRUE or a list of plotly line attributes, it will generate line segments joining observations. If set to FALSE, no segments will be generated. **Note:**The color of the joining line is the same as the marker color for that line. To avoid confusion, the line color cannot be changed. Change the marker color instead. The default values for the other elements of the join list, both of which can be overriden are: - width Width of the segments, default 1. - dash See plotly::schema(), traces > scatter > attributes > line > dash > values. Default is "solid". Example: line = list(join = list(dash = "longdash", width = 2)) • pred Default is FALSE, which means that predictions will not be included in the plot. To include predictions, supply one of the following: • If plotting data contained in a PM_result, use "pop" or "post" to include population or posterior predictions. ** Example 1: run1 <- PM_load(1); run1$data$plot(line = • If plotting data not contained in a PM_result, you may add the name of a population PM_pop or posterior PM_post prediction object in a PM_result object. This might be useful if you want to see how the predictions from one population match the raw data from another. ** Example 2: dat <- PM_data$new("new.csv"); dat To format the predictions, supply pred as a list, with the prediction object first, followed by named options to control the prediction plot: • icen Chooses the median or mean of each subject's Bayesian posterior parameter distribution. Default is "median", but could be "mean". • As for join, the color of the pred line is fixed to the same color as the marker. Other parameters to pass to plotly to control line characteristics that join the predictions are width, and dash. Continuing Example 1 above: pred = list("post", icen = "mean", width = 2). Default formats are the same as for the join argument, since normally one would not plot both lines joining observations and prediction lines, i.e., typical use would be line = list(join = F, pred = "post").
marker	Formats the symbols plotting observations. Controls the plotting symbol for observations. This argument maps to the plotly marker object. It can be boolean or a list. TRUE will plot the marker with default characteristics. FALSE will suppress marker plotting. If a list, can control many marker characteristics,

including overriding defaults. Use the plotly `plotly::schema()` command in the console and navigate to traces > scatter > attributes > marker to see all the ways the marker can be formatted. Most common will be:

- `color` Marker color.
- `symbol` Plotting character. See `plotly::schema()`, traces > scatter > attributes > marker > symbol > values.
- `size` Character size in points.
- `opacity` Ranging between 0 (fully transparent) to 1 (fully opaque).
- `line` A list of additional attributes governing the outline for filled shapes, most commonly color and width.

Example: `marker = list(color = "red", symbol = "triangle", opacity = 0.8, line = list(color = "black", width = 2))` . When a `group` and/or multiple `outeq` are specified, the `$color` element should be a palette or a vector of colors. For accepted palette names see `RColorBrewer::brewer.pal.info`. Examples include "BrBG", or "Set2". An example vector could be `c("red", "green", "blue")`. It is not necessary to specify the same number of colors as groups within `group`, as colors will be interpolated to generate the correct number. The default when `group` is specified is the "Set1" palette. When there are groups, the `$color` element for join or pred lines will be set to the same as `marker$color`.

`group` Character vector naming one column in `x` to **group** by, e.g. "id" or a covariate like "gender"

`group_names` A character vector of names to label the **groups** if `legend = TRUE`. This vector must be the same length as the number of groups within `group`. If missing, the vector will be generated from the unique values in `group`. Example: `c("Male", "Female")` if `group = "gender"` and "gender" is a covariate in the data.

`mult` Multiplication factor for y axis, e.g. to convert mg/L to ng/mL

`outeq` Which output equation to plot. Default is 1. Default is 1, but can be multiple if present in the data, e.g. `1:2` or `c(1, 3)`. In the case of multiple outputs, `group_colors` will be used to color the lines and markers.

`out_names` Character vector of names to label the outputs if `legend = TRUE`. These can be combined with `group_names`. The number must match the number of outputs in `outeq`. If missing, the default is "Output 1", "Output 2", etc.

`block` Which block to plot, where blocks are defined by dose reset events (EVID = 4) in the data. Default is 1, but can be multiple if present in the data, as for `outeq`.

`tad` Boolean operator to use time after dose rather than time after start. Default is `FALSE`.

`overlay` Operator to overlay all time concentration profiles in a single plot. The default is `TRUE`. If `FALSE`, will trellisplot subjects one at a time. Can also be specified as a vector with number of rows and columns, e.g. `c(3, 2)` for 3 rows and 2 columns of subject splots to include in each trellis.

`legend` Controls display of legend. This argument maps to the plotly layout showlegend and legend arguments. It is either a boolean operator (most common) or a list of parameters to be supplied to plotly. See `plotly::schema()` > layout > layoutAttributes > legend and showlegend for more details on the available options

for formatting. If legend is supplied as a list, the plotly layout > layoutAttributes > showlegend value will be set to TRUE automatically.

Examples:

- legend = T
- legend = list(orientation = "h", font = list(color = "blue"))

Default is FALSE unless groups are specified with colorabove.

log — Boolean operator to plot the y axis in log base 10. This argument maps to the yaxis type value in the layout object in plotly. Use the plotly plotly::schema() command in the console and navigate to layout > layoutAttributes > yaxis > type.

Example: log = T

grid — Controls grid display. This argument maps to the xaxis and yaxis layout objects in plotly. Use the plotly plotly::schema() command in the console and navigate to layout > layoutAttributes > xaxis or yaxis > gridcolor or gridwidth. It is a Boolean operator. If FALSE, no grid is plotted. If TRUE, the default color *grey* and width 1 will be plotted at major tick marks. If a list, color and width can be customized.

Examples:

- grid = F
- grid = list(gridcolor = "black", gridwidth = 2)

xlab — Value for x axis label. This argument maps to the the xaxis title element of the layout object in plotly. It can simply be a character vector of length 1 that specifies the name of the axis, or it can be a list for greater control. Use the plotly plotly::schema() command in the console and navigate to layout > layoutAttributes > xaxis > title to see the ways to customize this axis label. In addition to the plotly attributes, a custom Pmetrics attribute bold may be included as a list element, either on its own or within the font list. The default for bold is TRUE.

Examples:

- xlab = "Time (h)"
- xlab = list(text = "Time", bold = F, font = list(color = "red", family = "Arial", size = 10))
- xlab = list(text = "Time", font = list(bold = T))

Default is "Time".

ylab — Value for y axis label. This argument maps to the the yaxis title element of the layout object in plotly. See xlab for details. If xlab is specified as a list with formatting, and ylab is simply a character label, then the formatting for the xlab will be applied to the ylab. To format ylab independently, specify a formatting list as for xlab.

Default is "Output".

title — Plot title. This argument maps to the the title layout object in plotly. It can simply be a character vector of length 1 that specifies the name of the plot title, or it can be a list for greater control. Use the plotly plotly::schema() command in the console and navigate to layout > layoutAttributes > title to see other ways

to customize the title using lists as additional arguments. In addition to the plotly attributes, a custom Pmetrics attribute `bold` may be included as a list element. The default for `bold` is `TRUE`.

Examples:

- `title = "Observed vs. Predicted"`
- `title = list(text = "Raw Data", font = list(color = "red", family = "Arial", size = 10, bold = T))`

Default is to have no title.

`xlim` Limits of the x axis as a vector. This argument maps to the the xaxis range in the layout object in plotly. Use the plotly `plotly::schema()` command in the console and navigate to layout > layoutAttributes > xaxis > range.

Example: `xlim = c(0,1)`

`ylim` Limits of the y axis as a vector. This argument maps to the the yaxis range in the layout object in plotly. Use the plotly `plotly::schema()` command in the console and navigate to layout > layoutAttributes > yaxis > range.

Example: `ylim = c(0,100)`

`print` If `TRUE`, will print the plotly object and return it. If `FALSE`, will only return the plotly object.

`...` Other attributes which can be passed to the layout > xaxis/yaxis in a plotly plot to further control formatting. Note that `log`, `xlab`, `ylab`, `xlim`, and `ylim` are all controlled by the layout object, but are provided throughout Pmetrics plotly function arguments as shortcuts that map to layout elements. Therefore, the dots argument should be used to specify other aspects of the x axis, y axis, or both. See `plotly::schema()` layout > layoutAttributes > xaxis/yaxis for options. To add to single axis, name it as a list. If attributes are specified without an enclosing xaxis or yaxis list, they will be applied to both axes.

Examples:

- `NPex$data$plot(xaxis = list(tickcolor = "black", tickfont = list(family = "Arial", size = 14, color = "black"))) #applies to x axis only`
- `NPex$data$plot(linecolor = "red", ticks = "inside") #applies to both axes`

.

Details

This function will plot raw and fitted time and concentration data with a variety of options. By default markers are included and have the following plotly properties: `list(symbol = "circle", color = "red", size = 10, opacity = 0.5, line = list(color = "black", width = 1))`. Markers can be joined by lines, default is `FALSE`. If chosen to be `TRUE`, the joining lines will have the following properties: `list(color = "dodgerblue", width = 1, dash = "solid"`. The grid and legend are omitted by default.

Value

Plots the object.

Author(s)

Michael Neely

See Also

PM_data, PM_result

Other PMplots: plot.PM_cov(), plot.PM_cycle(), plot.PM_final(), plot.PM_model(), plot.PM_op(), plot.PM_opt(), plot.PM_pop(), plot.PM_post(), plot.PM_pta(), plot.PM_sim(), plot.PM_valid()

Examples

```
## Not run:
# basic spaghetti plot
dataEx$plot()
# format line and marker
dataEx$plot(
  marker = list(color = "blue", symbol = "square", size = 12, opacity = 0.4),
  line = list(join = list(color = "orange"))
)
# include predictions with default format and suppress joining lines
dataEx$plot(
  line = list(join = FALSE, pred = NPex$post),
  xlim = c(119, 146)
)
# customize prediction lines
dataEx$plot(
  line = list(
    pred = list(NPex$post, color = "slategrey", dash = "dash"),
    join = FALSE
  )
)

## End(Not run)
```

---

plot.PM_data_data *Plot method for PM_data data frames*

Description

[Stable] Allows plotting of altered PM_data objects.

Usage

```
## S3 method for class 'PM_data_data'
plot(x, ...)
```

Arguments

x	A data frame in the format of the standard_data field of a PM_data object
...	Additional arguments passed to plot.PM_data

Details

This is useful if you want to modify the data in a PM_data object, e.g. to filter the data, but still want to use the plotting capabilities of plot.PM_data. See plot.PM_data for details on how to format the plot.

Value

A plot of the data.

Author(s)

Michael Neely

See Also

PM_data, plot.PM_data

Examples

```
## Not run:
# filter then plot the standard_data data frame from a PM_data object
dataEx$standard_data |>
  filter(gender == 0) |>
  plot()

## End(Not run)
```

---

plot.PM_final *Plot Pmetrics Final Cycle Parameter Value Distributions*

Description

[Stable]

Plot R6 PM_final objects loaded as a field in the PM_result object, e.g. PM_result$final.

Usage

```
## S3 method for class 'PM_final'
plot(
  x,
  formula = NULL,
  line,
  marker = TRUE,
  standardize,
  legend,
  log,
  grid = TRUE,
  xlab,
  ylab,
  zlab,
  title,
  xlim,
  ylim,
  print = TRUE,
  ...
)
```

Arguments

x	The name of an PM_final data object as a field in a PM_result R6 object, e.g `PM_result$final`.
formula	An optional formula of the form `y ~ x`, where `y` and `x` are two model parameters to plot in a 3-dimensional bivariate plot. See details.
line	Controls characteristics of lines. This argument maps to plotly line attributes. `TRUE` will plot default lines. `FALSE` will suppress lines. If a list, can control many line characteristics, including overriding defaults. Use the plotly `plotly::schema()` command in the console and navigate to traces > scatter > attributes > line to see all the ways the line can be formatted. Most common will be: • `color` Line color. • `dash` Plotting character. See `plotly::schema()`, traces > scatter > attributes > line > dash > values. • `width` Thickness in points. Example: `line = list(color = "red", dash = "longdash", width = 2)` The `line` argument is used to format: • the density line drawn over an NPAG PM_final object. Default is `FALSE`, which means no density line will be drawn. Use `TRUE` to draw the default line, which is black, solid, and of width 1, or specify as a list to control these elements, e.g. `line = list(color = "red", width = 2, dash = "dash")` • the drop lines from point to lower surface when a `formula` is specified to generate a bivariate plot from an NPAG PM_final object. In this case, default is `line = TRUE`. The default format is black, dashed, and of width 1.

- the lines drawing the normal distribution of parameter values from an IT2B PM_Final object. Again, here the default is `line = TRUE`, and the format is black, solid, of width 1. See density. Ignored for IT2B output.

`marker` See details for which objects the `marker` argument controls. This argument maps to the plotly marker object. It can be boolean or a list. `TRUE` will plot the marker with default characteristics. `FALSE` will suppress marker plotting. If a list, can control many marker characteristics, including overriding defaults. Use the plotly `plotly::schema()` command in the console and navigate to traces > scatter > attributes > marker to see all the ways the marker can be formatted. Most common will be:

- `color` Fill color for NPAG bars, marker color for bivariate NPAG plots. Ignored for IT2B plots.
- `symbol` Plotting character. See `plotly::schema()`, traces > scatter > attributes > marker > symbol > values. Only relevant for bivariate NPAG plots.
- `size` Character size in points. Only relevant for bivariate NPAG plots.
- `opacity` Bar fill color for univariate NPAG plots or marker opacity for bivariate NPAG plots. Ignored for IT2B plots. Ranges between 0 (fully transparent) to 1 (fully opaque).
- `line` A list of additional attributes governing the outline for bars in univariate NPAG plots or markers in bivariate NPAG plots. Ignored for IT2B plots.
  - `color` Outline color. Default is "black".
  - `width` Outline width. Default is 1. Example: `marker = list(color = "red", symbol = "triangle", opacity = 0.8, line = list(color = "black", width = 2))`

`standardize` Standardize the normal parameter distribution plots from IT2B to the same scale x-axis. Ignored for NPAG output.

`legend` Ignored for this plot.

`log` Boolean operator to plot the y axis in log base 10. This argument maps to the yaxis type value in the layout object in plotly. Use the plotly `plotly::schema()` command in the console and navigate to layout > layoutAttributes > yaxis > type.

Example: `log = T`

`grid` Controls grid display. This argument maps to the xaxis and yaxis layout objects in plotly. Use the plotly `plotly::schema()` command in the console and navigate to layout > layoutAttributes > xaxis or yaxis > gridcolor or gridwidth. It is a Boolean operator. If `FALSE`, no grid is plotted. If `TRUE`, the default color *grey* and width 1 will be plotted at major tick marks. If a list, color and width can be customized.

Examples:

- `grid = F`
- `grid = list(gridcolor = "black", gridwidth = 2)`

`xlab` Value for x axis label. This argument maps to the the xaxis title element of the layout object in plotly. It can simply be a character vector of length 1 that

specifies the name of the axis, or it can be a list for greater control. Use the plotly `plotly::schema()` command in the console and navigate to layout > layoutAttributes > xaxis > title to see the ways to customize this axis label. In addition to the plotly attributes, a custom Pmetrics attribute `bold` may be included as a list element, either on its own or within the font list. The default for `bold` is `TRUE`.

Examples:

- `xlab = "Time (h)"`
- `xlab = list(text = "Time", bold = F, font = list(color = "red", family = "Arial", size = 10))`
- `xlab = list(text = "Time", font = list(bold = T))`

Default is the name of the plotted x-variable. Formatting can be controlled, but the text is recommended to not be changed.

`ylab` Value for y axis label. This argument maps to the the yaxis title element of the layout object in plotly. See `xlab` for details. If `xlab` is specified as a list with formatting, and `ylab` is simply a character label, then the formatting for the `xlab` will be applied to the `ylab`. To format `ylab` independently, specify a formatting list as for `xlab`.

Default is "Probability" for univariate plots, and the name of the plotted y-variable for bivariate plots. Formatting can be controlled, but the text is recommended to not be changed.

`zlab` Only for bivariate plots. Default is "Probability". Can be a list to control formatting or default text, as for `xlab` and `zlab`.

`title` Plot title. This argument maps to the the title layout object in plotly. It can simply be a character vector of length 1 that specifies the name of the plot title, or it can be a list for greater control. Use the plotly `plotly::schema()` command in the console and navigate to layout > layoutAttributes > title to see other ways to customize the title using lists as additional arguments. In addition to the plotly attributes, a custom Pmetrics attribute `bold` may be included as a list element. The default for `bold` is `TRUE`.

Examples:

- `title = "Observed vs. Predicted"`
- `title = list(text = "Raw Data", font = list(color = "red", family = "Arial", size = 10, bold = T))`

Default is to have no title on plots.

`xlim` Limits of the x axis as a vector. This argument maps to the the xaxis range in the layout object in plotly. Use the plotly `plotly::schema()` command in the console and navigate to layout > layoutAttributes > xaxis > range.

Example: `xlim = c(0,1)`

`ylim` Limits of the y axis as a vector. This argument maps to the the yaxis range in the layout object in plotly. Use the plotly `plotly::schema()` command in the console and navigate to layout > layoutAttributes > yaxis > range.

Example: `ylim = c(0,100)`

`print` If `TRUE`, will print the plotly object and return it. If `FALSE`, will only return the plotly object.

... Other attributes which can be passed to the layout > xaxis/yaxis in a plotly plot to further control formatting. Note that `log`, `xlab`, `ylab`, `xlim`, and `ylim` are all controlled by the layout object, but are provided throughout Pmetrics plotly function arguments as shortcuts that map to layout elements. Therefore, the dots argument should be used to specify other aspects of the x axis, y axis, or both. See `plotly::schema()` layout > layoutAttributes > xaxis/yaxis for options. To add to single axis, name it as a list. If attributes are specified without an enclosing xaxis or yaxis list, they will be applied to both axes.

Examples:

- `NPex$data$plot(xaxis = list(tickcolor = "black", tickfont = list(family = "Arial", size = 14, color = "black"))) #applies to x axis only`
- `NPex$data$plot(linecolor = "red", ticks = "inside") #applies to both axes`

.

Details

This is a function usually called by the `$plot()` method for PM_final objects within a PM_result to generate a plot the parameter value probability densities after completion of a model fitting. The function can be called directly on a PM_final object.

If `formula` is omitted, this will generate a marginal plot for each parameter. For NPAG data, this will be a histogram of marginal values for each parameter and the associated probability of that value. For IT2B, this will be a series of normal distributions with mean and standard deviation equal to the mean and standard deviation of each parameter marginal distribution.

On the other hand, if `formula` is specified as two parameters, e.g. CL~V, this will generate a bivariate plot. For NPAG data, it will be support point with size proportional to the probability of each point. For IT2B, it will be an elliptical distribution of a bivariate normal distribution centered at the mean of each plotted variable and surrounding quantiles of the bivariate distribution plotted in decreasing shades of grey. Mulitvariate normal density code is borrowed from the mvtnorm package.

Value

Plots the object.

Author(s)

Michael Neely

See Also

PM_final, schema

Other PMplots: `plot.PM_cov()`, `plot.PM_cycle()`, `plot.PM_data()`, `plot.PM_model()`, `plot.PM_op()`, `plot.PM_opt()`, `plot.PM_pop()`, `plot.PM_post()`, `plot.PM_pta()`, `plot.PM_sim()`, `plot.PM_valid()`

Examples

```
## Not run:
# NPAG
NPex$final$plot()
NPex$final$plot(line = TRUE)
NPex$final$plot(Ke ~ V)
# IT2B
ITex$final$plot()
ITex$final$plot(Ke ~ V)

## End(Not run)
```

---

plot.PM_final_data *Plot PM_final_data objects*

Description

[Stable] Plots the raw data (`class: PM_final_data`) from a PM_final object in the same way as plotting a PM_final object. Both use plot.PM_final.

Usage

```
## S3 method for class 'PM_final_data'
plot(x, ...)
```

Arguments

x	A 'PM_final_data" object
...	Additional arguments passed to plot.PM_final

Examples

```
NPex$final$data |> plot()
```

---

plot.PM_model *Plot PM_model objects*

Description

[Stable]

Plots a PM_model based on differential equations using ggplot.

Usage

```
## S3 method for class 'PM_model'
plot(x, marker = TRUE, line = TRUE, explicit, implicit, print = TRUE, ...)
```

Arguments

x	The name of an PM_model object.
marker	Controls the characteristics of the compartments (nodes). It can be boolean or a list. TRUE will plot the compartments with default characteristics. FALSE will suppress compartment plotting. If a list, can control some marker characteristics, including overriding defaults. These include: • color Marker color (default: dodgerblue). • opacity Ranging between 0 (fully transparent) to 1 (fully opaque). Default is 0.5. • size Relative size of boxes, ranging from 0 to 1. Default is 0.25. • line A list of additional attributes governing the outline for filled shapes, most commonly color (default: black) and width (default: 0.5). Example: marker = list(color = "red", opacity = 0.8, line = list(color = "black", width = 1))
line	Controls characteristics of arrows (edges). TRUE will plot default lines. FALSE will suppress lines. If a list, can control some line characteristics, including overriding defaults. These include: • color Line color (default: black) • width Thickness in points (default: 1). Example: line = list(color = "red", width = 2)
explicit	A data frame or tibble containing two columns named from and to to add additional connecting arrows to the plot indicating transfer between compartments. For each row, the from column contains the compartment number of the arrow origin, and the to column contains the compartment number of the arrow destination. Use 0 to indicate a destination to the external sink. e.g., explicit = data.frame(from = 3, to = 0)
implicit	Similar to explicit, used to add dashed connecting arrows to the plot indicating implicit transfer between compartments. For each row, the from column contains the compartment number of the arrow origin, and the to column contains the compartment number of the arrow destination. Use 0 to indicate a destination to the external sink. e.g., implicit = data.frame(from = 2, to = 4)
print	If TRUE, will print the object and return it. If FALSE, will only return the object.
...	Not used.

Details

This accepts a PM_model object and creates a network plot where nodes are compartments and edges are arrows connecting compartments.

Value

A plot object of the model.

Author(s)

Markus Hovd, Julian Otalvaro, Michael Neely

See Also

PM_model, ggplot2::ggplot()

Other PMplots: plot.PM_cov(), plot.PM_cycle(), plot.PM_data(), plot.PM_final(), plot.PM_op(), plot.PM_opt(), plot.PM_pop(), plot.PM_post(), plot.PM_pta(), plot.PM_sim(), plot.PM_valid()

Examples

```
## Not run:
NPex$model$plot()

## End(Not run)
```

---

plot.PM_op *Plot Pmetrics Observed vs. Predicted Objects*

Description

[Stable]

Plot PM_op objects

Usage

```
## S3 method for class 'PM_op'
plot(
  x,
  line = list(lm = NULL, loess = NULL, ref = NULL),
  marker = TRUE,
  resid = FALSE,
  icen = "median",
  pred.type = "post",
  outeq = 1,
  block,
  include,
  exclude,
  mult = 1,
  legend,
  log = FALSE,
  grid = TRUE,
  xlab,
  ylab,
  title,
  stats = TRUE,
```

```
  print = TRUE,
  xlim,
  ylim,
  ...
)
```

Arguments

x	The name of a PM_op data object and loaded with PM_load as a field in a PM_result, e.g. `PM_result$op`.
line	Controls characteristics of lines. Unlike some other Pmetrics plots, for plot.PM_op, `line` is a list of three elements: • `lm` If set to `TRUE` (default) or a list of plotly line attributes, will generate a linear regression of the form obs ~ pred. Line attributes will control the appearance of the regression line and the confidence interval around the line. If set to `FALSE`, no linear regression will be generated. The default values for the elements of the `lm` list, all of which can be overriden are: – `ci` Confidence interval around the regression, default 0.95. – `color` Color of the regression line and the confidence area around the line, but at opacity = 0.2. Default is "dodgerblue". – `width` Width of the regression line, default 1. – `dash` See `plotly::schema()`, traces > scatter > attributes > line > dash > values. Default is "solid". Example: `line = list(lm = list(color = "red", dash = "longdash", width = 2))` • `loess` If set to `TRUE` (default is `FALSE`) or a list of plotly line attributes, will generate a loess regression of the form obs ~ pred. The list elements and default values in the `loess` list are the same as for `lm` except the default style is "dash". Example: `line = list(lm = FALSE, loess = TRUE)` • `ref` If set to `TRUE` (default) or a list of plotly line attributes, will generate a reference line with slope = 1 and intercept = 0. The default values for the elements of the `ref` list are: – `color` "grey". – `width` 1. – `dash` "dot". Note that there is no *ci* argument for the *ref* list. Example: `line = list(lm = FALSE, loess = TRUE, ref = list(color = "lightgrey"))` If the `line` argument is missing, it will be set to `line = list(lm = TRUE, loess = FALSE, ref = TRUE)`, i.e. there will be a linear regression with reference line, but no loess regression. However, if `resid = T`, the default will become `line = list(lm = FALSE, loess = TRUE, ref = TRUE)`, i.e., loess regression with reference line, but no linear regression.
marker	Controls the plotting symbol for observations. This argument maps to the plotly marker object. It can be boolean or a list. `TRUE` will plot the marker with default characteristics. `FALSE` will suppress marker plotting. If a list, can control many marker characteristics, including overriding defaults. Use the plotly `plotly::schema()` command in the console and navigate to traces > scatter

> attributes > marker to see all the ways the marker can be formatted. Most common will be:

- `color` Marker color.
- `symbol` Plotting character. See `plotly::schema()`, traces > scatter > attributes > marker > symbol > values.
- `size` Character size in points.
- `opacity` Ranging between 0 (fully transparent) to 1 (fully opaque).
- `line` A list of additional attributes governing the outline for filled shapes, most commonly color and width.

Example: `marker = list(color = "red", symbol = "triangle", opacity = 0.8, line = list(color = "black", width = 2))` Default is `marker = list(color = orange, shape = "circle", size = 10, opacity = 0.5, line = list(color = black, width = 1))`. The color of any BLQ points is set to a color 90 degrees different on the color wheel from the color of the other points using opposite_color to ensure good contrast. The symbol for BLQ points is fixed to "triangle-down", and the opacity is fixed to 1. Size is the same as for other points.

`resid` Boolean operator to generate a plot of weighted prediction error vs. time, a plot of weighted prediction error vs. prediction. Prediction error is pred - obs. By default a loess regression will indicate deviation from zero prediction error.

`icen` Can be either "median" for the predictions based on medians of`pred.type` parameter value distributions, or "mean". Default is "median".

`pred.type` Either 'post' for a posterior object or 'pop' for a population object. Default is 'post'.

`outeq` Which output equation to plot. Default is 1.

`block` Which block to plot, where blocks are defined by dose reset events (EVID = 4) in the data. Default is missing, which results in all blocks included.

`include` A vector of subject IDs to include in the plot, e.g. c(1:3,5,15)

`exclude` A vector of subject IDs to exclude in the plot, e.g. c(4,6:14,16:20)

`mult` Multiplication factor for y axis, e.g. to convert mg/L to ng/mL

`legend` Ignored for this plot.

`log` Boolean operator to plot the y axis in log base 10. This argument maps to the yaxis type value in the layout object in plotly. Use the plotly `plotly::schema()` command in the console and navigate to layout > layoutAttributes > yaxis > type.

Example: `log = T`

`grid` Controls grid display. This argument maps to the xaxis and yaxis layout objects in plotly. Use the plotly `plotly::schema()` command in the console and navigate to layout > layoutAttributes > xaxis or yaxis > gridcolor or gridwidth. It is a Boolean operator. If `FALSE`, no grid is plotted. If `TRUE`, the default color *grey* and width 1 will be plotted at major tick marks. If a list, color and width can be customized.

Examples:

- grid = F
- grid = list(gridcolor = "black", gridwidth = 2)

xlab — Value for x axis label. This argument maps to the the xaxis title element of the layout object in plotly. It can simply be a character vector of length 1 that specifies the name of the axis, or it can be a list for greater control. Use the plotly plotly::schema() command in the console and navigate to layout > layoutAttributes > xaxis > title to see the ways to customize this axis label. In addition to the plotly attributes, a custom Pmetrics attribute bold may be included as a list element, either on its own or within the font list. The default for bold is TRUE.

Examples:

- xlab = "Time (h)"
- xlab = list(text = "Time", bold = F, font = list(color = "red", family = "Arial", size = 10))
- xlab = list(text = "Time", font = list(bold = T))

If missing, will default to "Predicted" for plots when resid = F and either "Time" or "Predicted" for residual plots.

ylab — Value for y axis label. This argument maps to the the yaxis title element of the layout object in plotly. See xlab for details. If xlab is specified as a list with formatting, and ylab is simply a character label, then the formatting for the xlab will be applied to the ylab. To format ylab independently, specify a formatting list as for xlab.

If missing, will default to "Observed" for plots when resid = F and either "Individual weighted residuals" or "Population weighted residuals" for residual plots, depending on the value of pred.type.

title — Plot title. This argument maps to the the title layout object in plotly. It can simply be a character vector of length 1 that specifies the name of the plot title, or it can be a list for greater control. Use the plotly plotly::schema() command in the console and navigate to layout > layoutAttributes > title to see other ways to customize the title using lists as additional arguments. In addition to the plotly attributes, a custom Pmetrics attribute bold may be included as a list element. The default for bold is TRUE.

Examples:

- title = "Observed vs. Predicted"
- title = list(text = "Raw Data", font = list(color = "red", family = "Arial", size = 10, bold = T))

Default is to have no title.

stats — Add the statistics from linear regression to the plot. If FALSE, will be suppressed. Default is TRUE which results in default format of list(x= 0.8, y = 0.1, font = list(color = "black", family = "Arial", size = 14, bold = FALSE)). The coordinates are relative to the plot with lower left = (0,0), upper right = (1,1). This argument maps to plotly::add_text().

print — If TRUE, will print the plotly object and return it. If FALSE, will only return the plotly object.

`xlim` Limits of the x axis as a vector. This argument maps to the the xaxis range in the layout object in plotly. Use the plotly `plotly::schema()` command in the console and navigate to layout > layoutAttributes > xaxis > range.

Example: `xlim = c(0,1)`

`ylim` Limits of the y axis as a vector. This argument maps to the the yaxis range in the layout object in plotly. Use the plotly `plotly::schema()` command in the console and navigate to layout > layoutAttributes > yaxis > range.

Example: `ylim = c(0,100)`

`...` Other attributes which can be passed to the layout > xaxis/yaxis in a plotly plot to further control formatting. Note that `log`, `xlab`, `ylab`, `xlim`, and `ylim` are all controlled by the layout object, but are provided throughout Pmetrics plotly function arguments as shortcuts that map to layout elements. Therefore, the dots argument should be used to specify other aspects of the x axis, y axis, or both. See `plotly::schema()` layout > layoutAttributes > xaxis/yaxis for options. To add to single axis, name it as a list. If attributes are specified without an enclosing xaxis or yaxis list, they will be applied to both axes.

Examples:

- `NPex$data$plot(xaxis = list(tickcolor = "black", tickfont = list(family = "Arial", size = 14, color = "black"))) #applies to x axis only`
- `NPex$data$plot(linecolor = "red", ticks = "inside") #applies to both axes`

.

Details

This is a function usually called by the `$plot()` method for PM_op objects within a PM_result to generate a plot of Observed vs. Predicted observations. The function can be called directly on a PM_op object. The default is to generate an observed vs. predicted plot of predictions based on the median of the Bayesian posterior distributions for each subject. Missing observations are excluded. Observations reported as BLQ (below the limit of quantification) are indicated as downward triangles, and colored differently from other observations. They are plotted at the reported LOQ on the observed axis and the predicted value on the predicted axis. They are not included in any regression lines or statistics.

Clicking on a point in the plot will highlight all points from that subject. The color of the highlight is 180 degrees different on the color wheel from the color of the other points (opposite_color), ensuring good contrast. Clicking on the plot background will remove the highlighting.

Value

Plots the object.

Author(s)

Michael Neely

See Also

PM_result, PM_op, schema

Other PMplots: plot.PM_cov(), plot.PM_cycle(), plot.PM_data(), plot.PM_final(), plot.PM_model(), plot.PM_opt(), plot.PM_pop(), plot.PM_post(), plot.PM_pta(), plot.PM_sim(), plot.PM_valid()

Examples

```
## Not run:
NPex$op$plot()
NPex$op$plot(pred.type = "pop")
NPex$op$plot(line = list(lm = TRUE, ref = TRUE, loess = FALSE))
NPex$op$plot(line = list(loess = list(ci = 0.9, color = "green")))
NPex$op$plot(marker = list(color = "blue"))
NPex$op$plot(resid = TRUE)
NPex$op$plot(stats = list(x = 0.5, y = 0.2, font = list(size = 7, color = "blue")))

## End(Not run)
```

---

plot.PM_op_data *Plot PM_op_data objects*

Description

[Stable] Plots the raw data (`class: PM_op_data`) from a PM_op object in the same way as plotting a PM_op object. Both use plot.PM_op.

Usage

```
## S3 method for class 'PM_op_data'
plot(x, ...)
```

Arguments

x	A 'PM_op_data" object
...	Additional arguments passed to plot.PM_op

Examples

```
## Not run:
NPex$op$data |>
  dplyr::filter(pred > 5) |>
  dplyr::filter(pred < 10) |>
  plot()

## End(Not run)
```

`plot.PM_opt` *Plot Pmetrics Multiple-Model Optimal Sampling Objects*

Description

[Stable]

Plots PM_opt objects

Usage

```
## S3 method for class 'PM_opt'
plot(x, line = list(probs = NA), times = T, print = TRUE, ...)
```

Arguments

`x`	A PM_opt object
`line`	Passed to plot.PM_sim with default as `list(probs = NA)`.
`times`	Format the vertical lines for optimal times. Default is dashed red line. r template("line")‘
`print`	If `TRUE`, will print the plotly object and return it. If `FALSE`, will only return the plotly object.
`...`	Other parameters to pass to plot.PM_sim.
`probs`	Default is NA. See plot.PM_sim for details.

Details

Simulated observations are plotted on the y-axis vs. time on the x.axis. Optimal sampling times are indicated as vertical lines. Defaults for optimal sample times are red, dash, width 2. Defaults for the `line` format are as for plot.PM_sim.

Value

Plots the simulation profiles with MM optimal times indicated as vertical lines.

Author(s)

Michael Neely

See Also

plot.PM_sim

Other PMplots: `plot.PM_cov()`, `plot.PM_cycle()`, `plot.PM_data()`, `plot.PM_final()`, `plot.PM_model()`, `plot.PM_op()`, `plot.PM_pop()`, `plot.PM_post()`, `plot.PM_pta()`, `plot.PM_sim()`, `plot.PM_valid()`

`plot.PM_pop` *Plot PM_pop Prediction Data*

Description

[Stable]

Plots *PM_pop* objects

Usage

```
## S3 method for class 'PM_pop'
plot(
  x,
  include = NULL,
  exclude = NULL,
  line = list(join = TRUE),
  marker = FALSE,
  out_names = NULL,
  mult = 1,
  icen = "median",
  outeq = 1,
  block = 1,
  overlay = TRUE,
  legend = FALSE,
  log = FALSE,
  grid = FALSE,
  xlab = "Time",
  ylab = "Output",
  title = "",
  xlim,
  ylim,
  print = TRUE,
  ...
)
```

Arguments

`x`	The name of a PM_pop object, e.g. `NPex$pop.` in a PM_result object
`include`	A vector of subject IDs to include in the plot, e.g. c(1:3,5,15)
`exclude`	A vector of subject IDs to exclude in the plot, e.g. c(4,6:14,16:20)
`line`	Controls characteristics of lines as for all plotly plots. It can either be a boolean or a list. If set to `TRUE` or a list of plotly line attributes, it will generate line segments joining observations. If set to `FALSE`, no segments will be generated. Line colors are fixed to match marker colors for group displays and cannot be overridden. Other customizable line properties are: - `width` Width of the segments, default 1. - `dash` See `plotly::schema()`, traces > scatter > attributes

> line > dash > values. Default is "solid". - `join` Set to `TRUE` (default) to join markers with lines, or `FALSE` for markers only. Example: `line = list(width = 2, dash = "longdash", join = FALSE)`

`marker` Formats the symbols plotting observations. Controls the plotting symbol for observations. This argument maps to the plotly marker object. It can be boolean or a list. `TRUE` will plot the marker with default characteristics. `FALSE` will suppress marker plotting. If a list, can control many marker characteristics, including overriding defaults. Use the plotly `plotly::schema()` command in the console and navigate to traces > scatter > attributes > marker to see all the ways the marker can be formatted. Most common will be:

- `color` Marker color.
- `symbol` Plotting character. See `plotly::schema()`, traces > scatter > attributes > marker > symbol > values.
- `size` Character size in points.
- `opacity` Ranging between 0 (fully transparent) to 1 (fully opaque).
- `line` A list of additional attributes governing the outline for filled shapes, most commonly color and width.

Example: `marker = list(color = "red", symbol = "triangle", opacity = 0.8, line = list(color = "black", width = 2))`

Marker colors control group display colors. When multiple output equations are displayed (via `outeq`), `marker$color` can be a palette name from `RColorBrewer::brewer.pal.info` or a vector of colors. Line colors are automatically matched to marker colors.

`out_names` A character vector of names to label the output equations if `legend = TRUE`. Must be at least as long as the maximum value in `outeq`. Example: `c("Conc A", "Conc B")` if `outeq = c(1, 2)`.

`mult` Multiplication factor for y axis, e.g. to convert mg/L to ng/mL

`icen` Can be `"median"` for predictions based on the median of the parameter value distributions, or `"mean"`. Default is `"median"`. Only a single value is accepted; `outeq` is the sole grouping dimension.

`outeq` Which output equation to plot. Default is 1. Default is 1, but can be multiple if present in the data, e.g. `1:2` or `c(1, 3)`.

`block` Which block to plot, where blocks are defined by dose reset events (EVID = 4) in the data. Default is 1, but can be multiple if present in the data, as for `outeq`.

`overlay` Operator to overlay all time prediction profiles in a single plot. The default is `TRUE`. If `FALSE`, will trellisplot subjects one at a time. Can also be specified as a vector with number of rows and columns, e.g. `c(3, 2)` for 3 rows and 2 columns of subject splots to include in each trellis.

`legend` Not used for this plot.

`log` Boolean operator to plot the y axis in log base 10. This argument maps to the yaxis type value in the layout object in plotly. Use the plotly `plotly::schema()` command in the console and navigate to layout > layoutAttributes > yaxis > type.

Example: `log = T`

`grid` Controls grid display. This argument maps to the xaxis and yaxis layout objects in plotly. Use the plotly `plotly::schema()` command in the console and navigate to layout > layoutAttributes > xaxis or yaxis > gridcolor or gridwidth. It is a Boolean operator. If `FALSE`, no grid is plotted. If `TRUE`, the default color *grey* and width 1 will be plotted at major tick marks. If a list, color and width can be customized.

Examples:

- `grid = F`
- `grid = list(gridcolor = "black", gridwidth = 2)`

`xlab` Value for x axis label. This argument maps to the the xaxis title element of the layout object in plotly. It can simply be a character vector of length 1 that specifies the name of the axis, or it can be a list for greater control. Use the plotly `plotly::schema()` command in the console and navigate to layout > layoutAttributes > xaxis > title to see the ways to customize this axis label. In addition to the plotly attributes, a custom Pmetrics attribute `bold` may be included as a list element, either on its own or within the font list. The default for `bold` is `TRUE`.

Examples:

- `xlab = "Time (h)"`
- `xlab = list(text = "Time", bold = F, font = list(color = "red", family = "Arial", size = 10))`
- `xlab = list(text = "Time", font = list(bold = T))`

Default is "Time".

`ylab` Value for y axis label. This argument maps to the the yaxis title element of the layout object in plotly. See `xlab` for details. If `xlab` is specified as a list with formatting, and `ylab` is simply a character label, then the formatting for the `xlab` will be applied to the `ylab`. To format `ylab` independently, specify a formatting list as for `xlab`.

Default is "Output".

`title` Plot title. This argument maps to the the title layout object in plotly. It can simply be a character vector of length 1 that specifies the name of the plot title, or it can be a list for greater control. Use the plotly `plotly::schema()` command in the console and navigate to layout > layoutAttributes > title to see other ways to customize the title using lists as additional arguments. In addition to the plotly attributes, a custom Pmetrics attribute `bold` may be included as a list element. The default for `bold` is `TRUE`.

Examples:

- `title = "Observed vs. Predicted"`
- `title = list(text = "Raw Data", font = list(color = "red", family = "Arial", size = 10, bold = T))`

Default is to have no title.

`xlim` Limits of the x axis as a vector. This argument maps to the the xaxis range in the layout object in plotly. Use the plotly `plotly::schema()` command in the console and navigate to layout > layoutAttributes > xaxis > range.

Example: `xlim = c(0,1)`

ylim — Limits of the y axis as a vector. This argument maps to the the yaxis range in the layout object in plotly. Use the plotly `plotly::schema()` command in the console and navigate to layout > layoutAttributes > yaxis > range.

Example: `ylim = c(0,100)`

print — If `TRUE`, will print the plotly object and return it. If `FALSE`, will only return the plotly object.

... — Other attributes which can be passed to the layout > xaxis/yaxis in a plotly plot to further control formatting. Note that `log`, `xlab`, `ylab`, `xlim`, and `ylim` are all controlled by the layout object, but are provided throughout Pmetrics plotly function arguments as shortcuts that map to layout elements. Therefore, the dots argument should be used to specify other aspects of the x axis, y axis, or both. See `plotly::schema()` layout > layoutAttributes > xaxis/yaxis for options. To add to single axis, name it as a list. If attributes are specified without an enclosing xaxis or yaxis list, they will be applied to both axes.

Examples:

- `NPex$data$plot(xaxis = list(tickcolor = "black", tickfont = list(family = "Arial", size = 14, color = "black"))) #applies to x axis only`
- `NPex$data$plot(linecolor = "red", ticks = "inside") #applies to both axes`

.

Details

This is a function usually called by the `$plot()` method for PM_pop objects within a PM_result to generate the plot. However, the function can be called directly on a PM_pop object. This function will plot time and population predictions with a variety of options. By default markers are included and have the following plotly properties: `list(symbol = "circle", color = "red", size = 10, opacity = 0.5, line = list(color = "black", width = 1))`. Markers are omitted by default. If enabled, markers can be joined by lines, default is `line = list(join = TRUE)`. If joined, the joining lines will have the following properties: `list(color = "dodgerblue", width = 1, dash = "solid"`. The grid and legend are omitted by default.

Value

Plots the object.

Author(s)

Michael Neely

See Also

PM_pop, PM_result

Other PMplots: `plot.PM_cov()`, `plot.PM_cycle()`, `plot.PM_data()`, `plot.PM_final()`, `plot.PM_model()`, `plot.PM_op()`, `plot.PM_opt()`, `plot.PM_post()`, `plot.PM_pta()`, `plot.PM_sim()`, `plot.PM_valid()`

Examples

```
## Not run:
# basic spaghetti plot
NPex$pop$plot()
# format line and marker
NPex$pop$plot(
  marker = list(color = "blue", symbol = "square", size = 12, opacity = 0.4),
  line = list(color = "orange")
)

## End(Not run)
```

`plot.PM_post` *Plot PM_post Prediction Data*

Description

[Stable]

Plots PM_post objects

Usage

```
## S3 method for class 'PM_post'
plot(
  x,
  include = NULL,
  exclude = NULL,
  line = list(join = TRUE),
  marker = FALSE,
  out_names = NULL,
  mult = 1,
  icen = "median",
  outeq = 1,
  block = 1,
  overlay = TRUE,
  legend = FALSE,
  log = FALSE,
  grid = FALSE,
  xlab = "Time",
  ylab = "Output",
  title = "",
  print = TRUE,
  xlim,
  ylim,
  ...
)
```

Arguments

x	The name of a PM_post object, e.g. NPex$post. in a PM_result object
include	A vector of subject IDs to include in the plot, e.g. c(1:3,5,15)
exclude	A vector of subject IDs to exclude in the plot, e.g. c(4,6:14,16:20)
line	Controls characteristics of lines as for all plotly plots. It can either be a boolean or a list. If set to TRUE or a list of plotly line attributes, it will generate line segments joining observations. If set to FALSE, no segments will be generated. Line colors are fixed to match marker colors for group displays and cannot be overridden. Other customizable line properties are: - width Width of the segments, default 1. - dash See plotly::schema(), traces > scatter > attributes > line > dash > values. Default is "solid". - join Set to TRUE (default) to join markers with lines, or FALSE for markers only. Example: line = list(width = 2, dash = "longdash", join = FALSE)
marker	Formats the symbols plotting observations. Controls the plotting symbol for observations. This argument maps to the plotly marker object. It can be boolean or a list. TRUE will plot the marker with default characteristics. FALSE will suppress marker plotting. If a list, can control many marker characteristics, including overriding defaults. Use the plotly plotly::schema() command in the console and navigate to traces > scatter > attributes > marker to see all the ways the marker can be formatted. Most common will be: • color Marker color. • symbol Plotting character. See plotly::schema(), traces > scatter > attributes > marker > symbol > values. • size Character size in points. • opacity Ranging between 0 (fully transparent) to 1 (fully opaque). • line A list of additional attributes governing the outline for filled shapes, most commonly color and width. Example: marker = list(color = "red", symbol = "triangle", opacity = 0.8, line = list(color = "black", width = 2)) Marker colors control group display colors. When multiple output equations are displayed (via outeq), marker$color can be a palette name from RColorBrewer::brewer.pal.info or a vector of colors. Line colors are automatically matched to marker colors.
out_names	A character vector of names to label the output equations if legend = TRUE. Must be at least as long as the maximum value in outeq. Example: c("Conc A", "Conc B") if outeq = c(1, 2).
mult	Multiplication factor for y axis, e.g. to convert mg/L to ng/mL
icen	Can be "median" for predictions based on the median of the parameter value distributions, or "mean". Default is "median". Only a single value is accepted; outeq is the sole grouping dimension.
outeq	Which output equation to plot. Default is 1. Default is 1, but can be multiple if present in the data, e.g. 1:2 or c(1, 3).
block	Which block to plot, where blocks are defined by dose reset events (EVID = 4) in the data. Default is 1, but can be multiple if present in the data, as for outeq.

`overlay` Operator to overlay all time prediction profiles in a single plot. The default is `TRUE`. If `FALSE`, will trellisplot subjects one at a time. Can also be specified as a vector with number of rows and columns, e.g. `c(3, 2)` for 3 rows and 2 columns of subject splots to include in each trellis.

`legend` Not used for this plot.

`log` Boolean operator to plot the y axis in log base 10. This argument maps to the yaxis type value in the layout object in plotly. Use the plotly `plotly::schema()` command in the console and navigate to layout > layoutAttributes > yaxis > type.

Example: `log = T`

`grid` Controls grid display. This argument maps to the xaxis and yaxis layout objects in plotly. Use the plotly `plotly::schema()` command in the console and navigate to layout > layoutAttributes > xaxis or yaxis > gridcolor or gridwidth. It is a Boolean operator. If `FALSE`, no grid is plotted. If `TRUE`, the default color *grey* and width 1 will be plotted at major tick marks. If a list, color and width can be customized.

Examples:

- `grid = F`
- `grid = list(gridcolor = "black", gridwidth = 2)`

`xlab` Value for x axis label. This argument maps to the the xaxis title element of the layout object in plotly. It can simply be a character vector of length 1 that specifies the name of the axis, or it can be a list for greater control. Use the plotly `plotly::schema()` command in the console and navigate to layout > layoutAttributes > xaxis > title to see the ways to customize this axis label. In addition to the plotly attributes, a custom Pmetrics attribute `bold` may be included as a list element, either on its own or within the font list. The default for `bold` is `TRUE`.

Examples:

- `xlab = "Time (h)"`
- `xlab = list(text = "Time", bold = F, font = list(color = "red", family = "Arial", size = 10))`
- `xlab = list(text = "Time", font = list(bold = T))`

Default is "Time".

`ylab` Value for y axis label. This argument maps to the the yaxis title element of the layout object in plotly. See `xlab` for details. If `xlab` is specified as a list with formatting, and `ylab` is simply a character label, then the formatting for the `xlab` will be applied to the `ylab`. To format `ylab` independently, specify a formatting list as for `xlab`.

Default is "Output".

`title` Plot title. This argument maps to the the title layout object in plotly. It can simply be a character vector of length 1 that specifies the name of the plot title, or it can be a list for greater control. Use the plotly `plotly::schema()` command in the console and navigate to layout > layoutAttributes > title to see other ways to customize the title using lists as additional arguments. In addition to the plotly

attributes, a custom Pmetrics attribute `bold` may be included as a list element. The default for `bold` is `TRUE`.

Examples:

- `title = "Observed vs. Predicted"`
- `title = list(text = "Raw Data", font = list(color = "red", family = "Arial", size = 10, bold = T))`

Default is to have no title.

`print` If `TRUE`, will print the plotly object and return it. If `FALSE`, will only return the plotly object.

`xlim` Limits of the x axis as a vector. This argument maps to the the xaxis range in the layout object in plotly. Use the plotly `plotly::schema()` command in the console and navigate to layout > layoutAttributes > xaxis > range.

Example: `xlim = c(0,1)`

`ylim` Limits of the y axis as a vector. This argument maps to the the yaxis range in the layout object in plotly. Use the plotly `plotly::schema()` command in the console and navigate to layout > layoutAttributes > yaxis > range.

Example: `ylim = c(0,100)`

`...` Other attributes which can be passed to the layout > xaxis/yaxis in a plotly plot to further control formatting. Note that `log`, `xlab`, `ylab`, `xlim`, and `ylim` are all controlled by the layout object, but are provided throughout Pmetrics plotly function arguments as shortcuts that map to layout elements. Therefore, the dots argument should be used to specify other aspects of the x axis, y axis, or both. See `plotly::schema()` layout > layoutAttributes > xaxis/yaxis for options. To add to single axis, name it as a list. If attributes are specified without an enclosing xaxis or yaxis list, they will be applied to both axes.

Examples:

- `NPex$data$plot(xaxis = list(tickcolor = "black", tickfont = list(family = "Arial", size = 14, color = "black"))) #applies to x axis only`
- `NPex$data$plot(linecolor = "red", ticks = "inside") #applies to both axes`

.

Details

This is a function usually called by the `$plot()` method for PM_post objects within a PM_result to generate the plot. However, the function can be called directly on a PM_post object. This function will plot time and posterior predictions with a variety of options. Markers are omitted by default. When enabled, markers have default plotly properties: `list(symbol = "circle", color = "red", size = 10, opacity = 0.5, line = list(color = "black", width = 1))`. Markers can be joined by lines, with default `line = list(join = TRUE)`. When joined, line colors are automatically matched to marker colors. The joining lines have the following default properties: `list(color = "dodgerblue", width = 1, dash = "solid")`. The grid and legend are omitted by default.

Value

Plots the object.

Author(s)

Michael Neely

See Also

PM_post, PM_result

Other PMplots: plot.PM_cov(), plot.PM_cycle(), plot.PM_data(), plot.PM_final(), plot.PM_model(), plot.PM_op(), plot.PM_opt(), plot.PM_pop(), plot.PM_pta(), plot.PM_sim(), plot.PM_valid()

Examples

```
## Not run:
# basic spaghetti plot
NPex$post$plot()
# format line and marker
NPex$post$plot(
  marker = list(color = "blue", symbol = "square", size = 12, opacity = 0.4),
  line = list(color = "orange")
)

## End(Not run)
```

---

plot.PM_pta *Plot PM_pta Percent Target Attainment objects*

Description

[Stable]

This function will plot the percent target attainment for associated with simulations.

Usage

```
## S3 method for class 'PM_pta'
plot(
  x,
  at = "intersect",
  include,
  exclude,
  type = "pta",
  mult = 1,
  outeq = 1,
  line = TRUE,
  marker = TRUE,
  ci = 0.9,
  legend = TRUE,
  log = FALSE,
  grid = TRUE,
```

```
  xlab,
  ylab,
  title,
  xlim,
  ylim,
  print = TRUE,
  ...
)
```

Arguments

x	The name of an *PM_pta* data object
at	Which object in the *PM_pta* result list to plot. By default "intersect" if an intersection is present due to creation of the object with multiple target types, or 1 if no intersection is present, which means only 1 target type was selected. If "intersect" is present in the object, the default can be overridden with a number to plot one of the individual PTAs, e.g. `at = 2` to plot the second PTA rather than the intersection of all the PTAs.
include	A vector of subject IDs to include in the plot, e.g. c(1:3,5,15)
exclude	A vector of subject IDs to exclude in the plot, e.g. c(4,6:14,16:20)
type	Character vector controlling type of plot. Default is "pta", which plots proportion with success on the y-axis and target on the x-axis. The other choice is "pdi", which plots the median pdi (pharmacodynamic index), e.g. AUC/MIC, on the y-axis, and target on the x-axis.
mult	Multiplication factor for y axis, e.g. to convert mg/L to ng/mL
outeq	Which output equation to plot. Default is 1.
line	Controls characteristics of lines. This argument maps to the plotly line object. It can be boolean or a list. `TRUE` will plot the line with default characteristics for each simulated regimen. `FALSE` will suppress line plotting. If a list, it functions a little differently than other Pmetrics plotly functions. Rather than controlling individual line characteristics, for this plot, the `line` argument should be a list of the options for group based plotting, where each group corresponds to a simulated regimen. The possible elements of the `line` list should be exactly named: • color Maps to the plot_ly `colors` argument to override default colors applied to the lines for each regimen. This can be a named palette, which can be obtained with `RColorBrewer::display.brewer.all()` or a vector of hexadecimal color names. One way to ensure reliable color palettes is to use the ColorBrewer site. Choosing the number of data classes to correspond to regimens, and qualitative data results in a distinct palette. Easiest importing into R is to copy/paste the Export of JavaScript on the ColorBrewer website. The default is "Set1". Palettes with fewer colors than regimens will be recycled. A color can also be a character vector of color names, recycled as needed. For example, a print-friendly choice is `line = list(color = "black")`. • width Maps to the plot_ly `width` argument to override default widths applied to the lines for each regimen. All lines will have the same width. The default value is 2.

- dash Maps to the plot_ly `linetypes` argument to override default styles applied to the lines for each regimen. If numeric, will map to `lty` par values. It can also be a character vector of dash names as listed in plot_ly. Example: `line = list(color = "Blues", width = 1, dash = 2)`, which will result in dotted lines (dash = 2) all with width 1 but in different shades of blue.

`marker` Controls the plotting symbol. This argument maps to the plotly marker object. It can be boolean or a list. `TRUE` will plot the profiles with default characteristics for each simulated regimen. `FALSE` will suppress line plotting. If a list, it functions a little differently than other Pmetrics plotly functions. Rather than controlling individual marker characteristics, for this plot, the `marker` argument should be a list of the options for group based plotting, where each group corresponds to a simulated regimen. The possible elements of the `marker` list should be exactly named:

- color Default marker color is the same as the line color. If line color is specified, marker color does not need to also be specified. Even if line plotting is suppressed with `line = F`, the default color value of "Set1" will be applied to markers, unless specified, e.g. `marker = list(color = "Blues")`.
- symbol Maps to the plot_ly `symbols` argument to override default symbols applied to the markers for each regimen. If only one value is supplied for this, it will be recycled for each regimen, i.e. all will have the same symbol. See `plotly::schema()`, traces > scatter > attributes > marker > symbol > values for options.
- size Maps to the plot_ly `size` argument to override default size applied to the markers for each regimen. All markers will have the same size. The default value is 12.

`ci` Confidence interval around curves on `type = "pdi"` plot, on scale of 0 to 1. Default is 0.9.

`legend` Controls display of legend. This argument maps to the plotly layout showlegend and legend arguments. It is either a boolean operator (most common) or a list of parameters to be supplied to plotly. See `plotly::schema()` > layout > layoutAttributes > legend and showlegend for more details on the available options for formatting. If legend is supplied as a list, the plotly layout > layoutAttributes > showlegend value will be set to `TRUE` automatically.

Examples:

- `legend = T`
- `legend = list(orientation = "h", font = list(color = "blue"))`

Default will be the labeled regimen names as an argument when creating a PM_pta object, or if missing, "Regimen 1, Regimen 2,...Regimen n", where *n* is the number of regimens in the PM_pta object.

`log` Boolean operator to plot the x axis in log base 10. This argument maps to the xaxis type value in the layout object in plotly. Many other Pmetrics plots map this argument to the y axis, but for PTA plots, the x axis is the more common axis to log transform. Use the plotly `plotly::schema()` command in the console and navigate to layout > layoutAttributes > xaxis > type. Example: `log = T`

`grid` Controls grid display. This argument maps to the xaxis and yaxis layout objects in plotly. Use the plotly `plotly::schema()` command in the console and navigate to layout > layoutAttributes > xaxis or yaxis > gridcolor or gridwidth. It is a Boolean operator. If `FALSE`, no grid is plotted. If `TRUE`, the default color *grey* and width 1 will be plotted at major tick marks. If a list, color and width can be customized.

Examples:

- `grid = F`
- `grid = list(gridcolor = "black", gridwidth = 2)`

`xlab` Value for x axis label. This argument maps to the the xaxis title element of the layout object in plotly. It can simply be a character vector of length 1 that specifies the name of the axis, or it can be a list for greater control. Use the plotly `plotly::schema()` command in the console and navigate to layout > layoutAttributes > xaxis > title to see the ways to customize this axis label. In addition to the plotly attributes, a custom Pmetrics attribute `bold` may be included as a list element, either on its own or within the font list. The default for `bold` is `TRUE`.

Examples:

- `xlab = "Time (h)"`
- `xlab = list(text = "Time", bold = F, font = list(color = "red", family = "Arial", size = 10))`
- `xlab = list(text = "Time", font = list(bold = T))`

Default is "Target" when targets are discrete, and "Regimen" when targets are sampled.

`ylab` Value for y axis label. This argument maps to the the yaxis title element of the layout object in plotly. See `xlab` for details. If `xlab` is specified as a list with formatting, and `ylab` is simply a character label, then the formatting for the `xlab` will be applied to the `ylab`. To format `ylab` independently, specify a formatting list as for `xlab`.

Default is "Proportion with success" for plot `type = "pta"` and "Pharmacodynamic Index" for plot `type = "pdi"`.

`title` Plot title. This argument maps to the the title layout object in plotly. It can simply be a character vector of length 1 that specifies the name of the plot title, or it can be a list for greater control. Use the plotly `plotly::schema()` command in the console and navigate to layout > layoutAttributes > title to see other ways to customize the title using lists as additional arguments. In addition to the plotly attributes, a custom Pmetrics attribute `bold` may be included as a list element. The default for `bold` is `TRUE`.

Examples:

- `title = "Observed vs. Predicted"`
- `title = list(text = "Raw Data", font = list(color = "red", family = "Arial", size = 10, bold = T))`

Default is to have no title.

xlim	Limits of the x axis as a vector. This argument maps to the the xaxis range in the layout object in plotly. Use the plotly `plotly::schema()` command in the console and navigate to layout > layoutAttributes > xaxis > range. Example: `xlim = c(0,1)`
ylim	Limits of the y axis as a vector. This argument maps to the the yaxis range in the layout object in plotly. Use the plotly `plotly::schema()` command in the console and navigate to layout > layoutAttributes > yaxis > range. Example: `ylim = c(0,100)`
print	If `TRUE`, will print the plotly object and return it. If `FALSE`, will only return the plotly object.
...	Other attributes which can be passed to the layout > xaxis/yaxis in a plotly plot to further control formatting. Note that `log`, `xlab`, `ylab`, `xlim`, and `ylim` are all controlled by the layout object, but are provided throughout Pmetrics plotly function arguments as shortcuts that map to layout elements. Therefore, the dots argument should be used to specify other aspects of the x axis, y axis, or both. See `plotly::schema()` layout > layoutAttributes > xaxis/yaxis for options. To add to single axis, name it as a list. If attributes are specified without an enclosing xaxis or yaxis list, they will be applied to both axes. Examples: • `NPex$data$plot(xaxis = list(tickcolor = "black", tickfont = list(family = "Arial", size = 14, color = "black"))) #applies to x axis only` • `NPex$data$plot(linecolor = "red", ticks = "inside") #applies to both axes` .

Details

PM_pta objects are made with the `$pta` method for PM_sim or with `PM_pta$new()`.

Value

Plots the object.

Author(s)

Michael Neely

See Also

Other PMplots: `plot.PM_cov()`, `plot.PM_cycle()`, `plot.PM_data()`, `plot.PM_final()`, `plot.PM_model()`, `plot.PM_op()`, `plot.PM_opt()`, `plot.PM_pop()`, `plot.PM_post()`, `plot.PM_sim()`, `plot.PM_valid()`

Examples

```
## Not run:
pta1 <- simEx$pta(
  simlabels <- c("600 mg daily", "1200 mg daily", "300 mg bid", "600 mg bid"),
  targets = c(0.25, 0.5, 1, 2, 4, 8, 16, 32), target.type = "time",
```

```
  success = 0.6, start = 120, end = 144
)
pta1$summary()
pta1$plot()

## End(Not run)
```

`plot.PM_sim` *Plot Pmetrics Simulation Objects*

Description

[Stable]

Plots *PM_sim* objects with the option to perform a visual and numerical predictive check

Usage

```
## S3 method for class 'PM_sim'
plot(
  x,
  include,
  exclude,
  mult = 1,
  ci = 0.95,
  binSize = 0,
  outeq = 1,
  line = TRUE,
  marker = FALSE,
  obs,
  quiet = FALSE,
  legend = FALSE,
  log = TRUE,
  grid = FALSE,
  xlab,
  ylab,
  title,
  xlim,
  ylim,
  print = TRUE,
  ...
)
```

Arguments

`x`	The name of an *PM_sim* data object generated by PM_sim
`include`	A vector of subject IDs to include in the plot, e.g. c(1:3,5,15) .
`exclude`	A vector of subject IDs to exclude in the plot, e.g. c(4,6:14,16:20) .

`mult` Multiplication factor for y axis, e.g. to convert mg/L to ng/mL

`ci` Width of confidence interval bands around simulated quantiles, from 0 to 1. If 0, or *nsim*<100, will not plot. Default is 0.95, i.e. 95th percentile with tails of 2.5 percent above and below excluded.

`binSize` Width of binning interval for simulated concentrations, in time units, e.g. hours. A `binSize` of 0.5 will pull all simulated concentrations +/- 0.5 hours into the same time. This is useful for plotting PMsim objects made during make_valid. The default is 0, i.e. no binning. If an `obs` object is provided, it will be binned similarly.

`outeq` Which output equation to plot. Default is 1.

`line` Controls the appearance of lines. It can be specified in several ways.

- Default is `TRUE` which results in simulated profiles summarized as quantiles, with default values of 0.05, 0.25, 0.5, 0.75, and 0.95. The default format will be applied, which is solid black lines of width 1. Numerical predictive checking will be calculated if observations are also included (see *obs* below).
- `FALSE` results in no lines plotted and the plot will be blank.
- `NA` Quantile summaries will be suppressed, but lines joining simulated outputs will be plotted in default format as above. In other words, all profiles will be plotted, not just the quantiles. Numerical predictive checking will be suppressed.
- List of quantiles and formats to plot with the following elements:
  - `probs` Vector of quantiles to include. If missing, will be set to defaults above, i.e., `c(0.05, 0.5, and 0.95)` Example: `line = list(probs = c(0.25, 0.5, 0.75))`.
  - `color` Vector of color names whose order corresponds to `probs`. If shorter than `probs`, will be recycled. Default is "dodgerblue", but if median is present (`prob = 0.5`), that line will be "red". Examples: `line = list(color = "red")` or `line = list(color = c("red", "blue"))`.
  - `fill` Fill color between quantile lines. Can be specified in several ways:
    - `FALSE` (the default) will not fill between lines.
    - `TRUE` will fill between lines with a default color of "dodgerblue", opacity 0.2.
    - A list with the following elements:
      - `color` Fill color name. Default is "dodgerblue", e.g., `fill = list(color = "red")`.
      - `opacity` Fill opacity. Default is 0.2 e.g., `fill = list(opacity = 0.3)`.
      - `probs` Vector of paired quantiles to fill between. Default is the minimum and maximum quantile specified in `probs`, or `fill = list(probs = c(0.05, 0.95))` if not specified. Including `probs` in fill which are not in `probs` above will result in an error.
  - `width` Vector of widths in pixels, as for `color`. Default is 1. Example: `line = list(width = 2)`.

- dash Vector of dash types, as for color. Default is "solid". See plotly::schema(), traces > scatter > attributes > line > dash > values. Example: line = list(dash = "dashdot").

marker — Controls the plotting symbol for observations. This argument maps to the plotly marker object. It can be boolean or a list. TRUE will plot the marker with default characteristics. FALSE will suppress marker plotting. If a list, can control many marker characteristics, including overriding defaults. Use the plotly plotly::schema() command in the console and navigate to traces > scatter > attributes > marker to see all the ways the marker can be formatted. Most common will be:

- color Marker color.
- symbol Plotting character. See plotly::schema(), traces > scatter > attributes > marker > symbol > values.
- size Character size in points.
- opacity Ranging between 0 (fully transparent) to 1 (fully opaque).
- line A list of additional attributes governing the outline for filled shapes, most commonly color and width.

Example: marker = list(color = "red", symbol = "triangle", opacity = 0.8, line = list(color = "black", width = 2)) Formatting will only be applied to observations if included via the obs argument.

obs — The name of a PM_result data object or the PM_op field in the PM_result object, all generated by PM_load. For example, if run1 <- PM_load(1) and sim1 is a PM_sim object, then sim1$plot(obs = run1) or sim1$plot(obs = run1$op). If specified, the observations will be overlaid upon the simulation plot enabling a visual predictive check. In this case, a list object will be returned with two items: $npc containing the quantiles and probability that the observations are below each quantile (binomial test); and $simsum, the times of each observation and the value of the simulated quantile with upper and lower confidence intervals at that time. Additionally, the number of observations beyond the 5th and 95th percentiles will be reported and the binomial test P-value if this number is different than the expected 10% value.

quiet — If TRUE, suppresses the message about simulation report generation, defaults to FALSE.

legend — Controls display of legend. This argument maps to the plotly layout showlegend and legend arguments. It is either a boolean operator (most common) or a list of parameters to be supplied to plotly. See plotly::schema() > layout > layoutAttributes > legend and showlegend for more details on the available options for formatting. If legend is supplied as a list, the plotly layout > layoutAttributes > showlegend value will be set to TRUE automatically.

Examples:

- legend = T
- legend = list(orientation = "h", font = list(color = "blue"))

Default is FALSE

log — Boolean operator to plot the y axis in log base 10. This argument maps to the yaxis type value in the layout object in plotly. Use the plotly plotly::schema()

command in the console and navigate to layout > layoutAttributes > yaxis > type.

Example: `log = T` Default is `TRUE`.

`grid` Controls grid display. This argument maps to the xaxis and yaxis layout objects in plotly. Use the plotly `plotly::schema()` command in the console and navigate to layout > layoutAttributes > xaxis or yaxis > gridcolor or gridwidth. It is a Boolean operator. If `FALSE`, no grid is plotted. If `TRUE`, the default color *grey* and width 1 will be plotted at major tick marks. If a list, color and width can be customized.

Examples:

- `grid = F`
- `grid = list(gridcolor = "black", gridwidth = 2)`

Default is `FALSE`

`xlab` Value for x axis label. This argument maps to the the xaxis title element of the layout object in plotly. It can simply be a character vector of length 1 that specifies the name of the axis, or it can be a list for greater control. Use the plotly `plotly::schema()` command in the console and navigate to layout > layoutAttributes > xaxis > title to see the ways to customize this axis label. In addition to the plotly attributes, a custom Pmetrics attribute `bold` may be included as a list element, either on its own or within the font list. The default for `bold` is `TRUE`.

Examples:

- `xlab = "Time (h)"`
- `xlab = list(text = "Time", bold = F, font = list(color = "red", family = "Arial", size = 10))`
- `xlab = list(text = "Time", font = list(bold = T))`

Default is "Time".

`ylab` Value for y axis label. This argument maps to the the yaxis title element of the layout object in plotly. See `xlab` for details. If `xlab` is specified as a list with formatting, and `ylab` is simply a character label, then the formatting for the `xlab` will be applied to the `ylab`. To format `ylab` independently, specify a formatting list as for `xlab`.

Default is "Output".

`title` Plot title. This argument maps to the the title layout object in plotly. It can simply be a character vector of length 1 that specifies the name of the plot title, or it can be a list for greater control. Use the plotly `plotly::schema()` command in the console and navigate to layout > layoutAttributes > title to see other ways to customize the title using lists as additional arguments. In addition to the plotly attributes, a custom Pmetrics attribute `bold` may be included as a list element. The default for `bold` is `TRUE`.

Examples:

- `title = "Observed vs. Predicted"`
- `title = list(text = "Raw Data", font = list(color = "red", family = "Arial", size = 10, bold = T))`

Default is to have no title.

`xlim` Limits of the x axis as a vector. This argument maps to the the xaxis range in the layout object in plotly. Use the plotly `plotly::schema()` command in the console and navigate to layout > layoutAttributes > xaxis > range.

Example: `xlim = c(0,1)`

`ylim` Limits of the y axis as a vector. This argument maps to the the yaxis range in the layout object in plotly. Use the plotly `plotly::schema()` command in the console and navigate to layout > layoutAttributes > yaxis > range.

Example: `ylim = c(0,100)`

`print` If `TRUE`, will print the plotly object and return it. If `FALSE`, will only return the plotly object.

`...` Other attributes which can be passed to the layout > xaxis/yaxis in a plotly plot to further control formatting. Note that `log`, `xlab`, `ylab`, `xlim`, and `ylim` are all controlled by the layout object, but are provided throughout Pmetrics plotly function arguments as shortcuts that map to layout elements. Therefore, the dots argument should be used to specify other aspects of the x axis, y axis, or both. See `plotly::schema()` layout > layoutAttributes > xaxis/yaxis for options. To add to single axis, name it as a list. If attributes are specified without an enclosing xaxis or yaxis list, they will be applied to both axes.

Examples:

- `NPex$data$plot(xaxis = list(tickcolor = "black", tickfont = list(family = "Arial", size = 14, color = "black"))) #applies to x axis only`
- `NPex$data$plot(linecolor = "red", ticks = "inside") #applies to both axes`

.

Details

Simulated observations are plotted as quantiles on the y-axis vs. time on the x.axis. If measured observations are included, a visual and numerical predictive check will be performed. The default plot is to omit markers, but if the marker argument is set to `TRUE`, the resulting marker will have the following plotly properties: `list(symbol = "circle-open", color = "black", size = 8)`. By default a grid is omitted. The legend is also omitted by default, but if included, clicking on a quantile item in the legend will hide it in the plot, and double clicking will hide all other quantiles.

Value

Plots the simulation object. If `obs` is included, a list will be returned with the folowing items:

- *npc* A dataframe with three columns: quantile, prop_less, pval. ** *quantile* are those specified by the `probs` argument to the plot call ** *prop_less* are the proportion of simulated observations at all times less than the quantile ** *pval* is the P-value of the difference in the prop.less and quantile by the beta-binomial test.
- *simsum* A dataframe with the quantile concentration at each simulated time, with lower and upper confidence intervals
- *obs* A data frame similar to a the `$data` field of a PM_op object with the addition of the quantile for each observation

Author(s)

Michael Neely

See Also

PM_sim, plot_ly, schema

Other PMplots: plot.PM_cov(), plot.PM_cycle(), plot.PM_data(), plot.PM_final(), plot.PM_model(), plot.PM_op(), plot.PM_opt(), plot.PM_pop(), plot.PM_post(), plot.PM_pta(), plot.PM_valid()

Examples

```
## Not run:
simEx$plot()
simEx$plot(log = FALSE, line = list(color = "orange"))

## End(Not run)
```

---

plot.PM_valid *Plot Pmetrics Validation Objects*

Description

[Stable] Usually called by the $plot method for PM_valid objects, which are in turn typically added to a PM_result object by the $validate method. For example:

```
NPex$validate(limits = c(0, 3)) # creates a PM_valid object and adds it to the $valid field of NPex
 NPex$valid$plot(type = "vpc", tad = TRUE, log = TRUE) # now we can plot it
```

Plot PM_valid objects.

Usage

```
## S3 method for class 'PM_valid'
plot(
  x,
  type = "vpc",
  tad = FALSE,
  outeq = 1,
  line = TRUE,
  marker = TRUE,
  legend = FALSE,
  log = FALSE,
  grid = TRUE,
  xlab,
  ylab,
  title,
  xlim,
```

```
  ylim,
  print = TRUE,
  ...
)
```

Arguments

x	The name of an *PM_valid* data object, which is usually called by the `$validate` method for PM_result objects.
type	Default is "vpc" for a visual predictive check, but could be "pcvpc" for a prediction-corrected visual predictive check, or "npde" for a normalized prediction distribution error analysis/plot. Choosing npde will call npde::plot.NpdeObject. To modify this plot, supply argmuents as a named list: `npde = (...)`. Available arguments are in the user manual for the npde package.
tad	Boolean operator to use time after dose rather than time after start. Default is `FALSE`.
outeq	Which output equation to plot. Default is 1.
line	A list of three elements `$upper`, `$mid`, and `$lower`, each of which controls characteristics of corresponding quantiles. The arguments to each of these list elements map to several plotly attributes. Each can be a boolean value or a list. `TRUE` will plot default characteristics. `FALSE` will suppress quantile plots. The elements of the list for each argument are as follows: • `value` The quantile value. Default for lower is 0.025, mid is 0.5, and upper is 0.975. • `color` The color for both the 95%CI region around simulated quantile vs. time, and the color of the line for the observation quantile vs. time. Default for lower and upper is "dodgerblue" and for mid it is "red". • `dash` The style of the obervation quantile line. See `plotly::schema()`, traces > scatter > attributes > line > dash > values. Default for lower and upper is "dash" and for mid it is "solid". • `width` Default is 1 for lower, mid, and upper. • `opacity` The opacity of the 95%CI region around simulated quantile vs. time. Default is 0.4 for lower, mid and upper, but can range between 0 (fully transparent) to 1 (fully opaque). Example: `line = list(upper = list(value = 0.9, color = "red", dash = "longdash", opacity = 0.5, width = 2))`
marker	Controls the plotting symbol for observations. This argument maps to the plotly marker object. It can be boolean or a list. `TRUE` will plot the marker with default characteristics. `FALSE` will suppress marker plotting. If a list, can control many marker characteristics, including overriding defaults. Use the plotly `plotly::schema()` command in the console and navigate to traces > scatter > attributes > marker to see all the ways the marker can be formatted. Most common will be: • `color` Marker color. • `symbol` Plotting character. See `plotly::schema()`, traces > scatter > attributes > marker > symbol > values.

- `size` Character size in points.
- `opacity` Ranging between 0 (fully transparent) to 1 (fully opaque).
- `line` A list of additional attributes governing the outline for filled shapes, most commonly color and width.

Example: `marker = list(color = "red", symbol = "triangle", opacity = 0.8, line = list(color = "black", width = 2))`

`legend` Controls display of legend. This argument maps to the plotly layout showlegend and legend arguments. It is either a boolean operator (most common) or a list of parameters to be supplied to plotly. See `plotly::schema()` > layout > layoutAttributes > legend and showlegend for more details on the available options for formatting. If legend is supplied as a list, the plotly layout > layoutAttributes > showlegend value will be set to `TRUE` automatically.

Examples:

- `legend = T`
- `legend = list(orientation = "h", font = list(color = "blue"))`

Default is `FALSE`

`log` Boolean operator to plot the y axis in log base 10. This argument maps to the yaxis type value in the layout object in plotly. Use the plotly `plotly::schema()` command in the console and navigate to layout > layoutAttributes > yaxis > type.

Example: `log = T`

`grid` Controls grid display. This argument maps to the xaxis and yaxis layout objects in plotly. Use the plotly `plotly::schema()` command in the console and navigate to layout > layoutAttributes > xaxis or yaxis > gridcolor or gridwidth. It is a Boolean operator. If `FALSE`, no grid is plotted. If `TRUE`, the default color *grey* and width 1 will be plotted at major tick marks. If a list, color and width can be customized.

Examples:

- `grid = F`
- `grid = list(gridcolor = "black", gridwidth = 2)`

`xlab` Value for x axis label. This argument maps to the the xaxis title element of the layout object in plotly. It can simply be a character vector of length 1 that specifies the name of the axis, or it can be a list for greater control. Use the plotly `plotly::schema()` command in the console and navigate to layout > layoutAttributes > xaxis > title to see the ways to customize this axis label. In addition to the plotly attributes, a custom Pmetrics attribute `bold` may be included as a list element, either on its own or within the font list. The default for `bold` is `TRUE`.

Examples:

- `xlab = "Time (h)"`
- `xlab = list(text = "Time", bold = F, font = list(color = "red", family = "Arial", size = 10))`
- `xlab = list(text = "Time", font = list(bold = T))`

Default is "Time" or "Time after dose" if `tad = TRUE`.

ylab
: Value for y axis label. This argument maps to the the yaxis title element of the layout object in plotly. See xlab for details. If xlab is specified as a list with formatting, and ylab is simply a character label, then the formatting for the xlab will be applied to the ylab. To format ylab independently, specify a formatting list as for xlab.

  Default is "Output".

title
: Plot title. This argument maps to the the title layout object in plotly. It can simply be a character vector of length 1 that specifies the name of the plot title, or it can be a list for greater control. Use the plotly plotly::schema() command in the console and navigate to layout > layoutAttributes > title to see other ways to customize the title using lists as additional arguments. In addition to the plotly attributes, a custom Pmetrics attribute bold may be included as a list element. The default for bold is TRUE.

  Examples:

  - title = "Observed vs. Predicted"
  - title = list(text = "Raw Data", font = list(color = "red", family = "Arial", size = 10, bold = T))

  Default is to have no title.

xlim
: Limits of the x axis as a vector. This argument maps to the the xaxis range in the layout object in plotly. Use the plotly plotly::schema() command in the console and navigate to layout > layoutAttributes > xaxis > range.

  Example: xlim = c(0,1)

ylim
: Limits of the y axis as a vector. This argument maps to the the yaxis range in the layout object in plotly. Use the plotly plotly::schema() command in the console and navigate to layout > layoutAttributes > yaxis > range.

  Example: ylim = c(0,100)

print
: If TRUE, will print the plotly object and return it. If FALSE, will only return the plotly object.

...
: If type is not "npde", the following apply. Other attributes which can be passed to the layout > xaxis/yaxis in a plotly plot to further control formatting. Note that log, xlab, ylab, xlim, and ylim are all controlled by the layout object, but are provided throughout Pmetrics plotly function arguments as shortcuts that map to layout elements. Therefore, the dots argument should be used to specify other aspects of the x axis, y axis, or both. See plotly::schema() layout > layoutAttributes > xaxis/yaxis for options. To add to single axis, name it as a list. If attributes are specified without an enclosing xaxis or yaxis list, they will be applied to both axes.

  Examples:

  - NPex$data$plot(xaxis = list(tickcolor = "black", tickfont = list(family = "Arial", size = 14, color = "black"))) #applies to x axis only
  - NPex$data$plot(linecolor = "red", ticks = "inside") #applies to both axes

  . . However, if type is "npde", to modify the appearance of the plot, supply a list of options, npde = list(...). See the documentation for the type argument above.

Details

Generates a plot of outputs (typically concentrations) on the y axis and time on the x axis. If `tad` was set to `TRUE` when make_valid was called, then time may be either absolute (default) or time after dose, controlled by the `tad` argument to this plot function. The following items are included in the plot:

- Observed outputs. These may be either as measured for `type = "vpc"` or prediction corrected for `type = "pcvpc"`. Format of the observations is controlled by the `marker` argument. The default is `list(color = "black", symbol = "circle-open", size = 8)`.
- Quantiles vs. time for observations. These are plotted by default as dashed blue lines for the 2.5th and 97.5th percentiles and a solid red line for the median. Formatting and the value for each quantile can be controlled with the `upper`, `mid`, and `lower` arguments.
- 95% CI around the same quantiles of combined simulations from each subject. The values and formatting for these quantile CIs are the same as for the observations, and also controlled with the `upper`, `mid`, and `lower` arguments.

Good vpc/pcvpc plots are considered to be those where the quantile lines for the oberservations lie within the 95%CI quantile regions for simulations, indicated that the model is "centered" on the data and faithfully captures the variability in the data. For an npde plot, one expects to see approximately normally distributed normalized prediction errors.

Value

Plots and returns the plotly object

Author(s)

Michael Neely

See Also

make_valid

Other PMplots: `plot.PM_cov()`, `plot.PM_cycle()`, `plot.PM_data()`, `plot.PM_final()`, `plot.PM_model()`, `plot.PM_op()`, `plot.PM_opt()`, `plot.PM_pop()`, `plot.PM_post()`, `plot.PM_pta()`, `plot.PM_sim()`

Examples

```
## Not run:
# VPC
NPex$valid$plot()

# pcVPC
NPex$valid$plot(type = "pcvpc")

# modify median line and marker
NPex$valid$plot(
  line = list(mid = list(color = "orange", dash = "dashdot")),
  marker = list(
    color = "blue", size = 12, symbol = "diamond",
    line = list(color = "navy")
```

```
  )
)

## End(Not run)
```

`plot.PMvalid` *Plot Pmetrics Validation Objects*

Description

[Superseded]

This is largely now a legacy plotting function, with a variety of options. It has been superseded by plot.PM_valid.

Usage

```
## S3 method for class 'PMvalid'
plot(
  x,
  type = "vpc",
  tad = FALSE,
  icen = "median",
  outeq = 1,
  lower = 0.025,
  upper = 0.975,
  log = FALSE,
  pch.obs = 1,
  col.obs = "black",
  cex.obs = 1,
  data_theme = "color",
  plot_theme = theme_grey(),
  col.obs.ci = "blue",
  col.obs.med = "red",
  col.sim.ci = "dodgerblue",
  col.sim.med = "lightpink",
  axis.x = NULL,
  axis.y = NULL,
  ...
)
```

Arguments

`x`	The name of an *PMvalid* data object generated by `make_valid`.
`type`	Default is "vpc" for a visual prective check, but could be "pcvpc" for a prediction-corrected visual predictive check.

tad	Plot using time after dose if TRUE. Default is FALSE which plots using standard relative time. This will be the only option if tad was not set to TRUE when making the PMvalid object.
icen	Can be either "median" for the predictions based on medians of the population parameter value distributions, or "mean". Default is "median".
outeq	The number of the output equation to simulate/test. Default is 1.
lower	The lower quantile displayed for the observed and simulated profiles. Default is 0.025.
upper	The upper quantile displayed for the observed and simulated profiles. Default is 0.975.
log	Boolean operator to plot in semilog space. The default is FALSE.
pch.obs	Control the plotting character used for observations. Default is 1, i.e. an open circle. See points for other values of pch.
col.obs	Color for observations. Default is black.
cex.obs	Size for observatins. Default is 1.
data_theme	Default is "color", but could be "grey" or "gray".
plot_theme	Default is theme_grey() but could be any complete ggplot2 theme, e.g. ggplot2::theme_minimal().
col.obs.ci	Color of the observation confidence interval (set by lower and upper). Default is blue.
col.obs.med	Color of the observation median. Default is red.
col.sim.ci	Color of the simulation confidence interval (set by lower and upper). Default is dodgerblue.
col.sim.med	Color of the simulation median. Default is lightpink.
axis.x	List of $name and $limits. Default name is "Time".
axis.y	List of $name and $limits. Default name is "Observation".
...	Not currently used

Value

Plots the object using ggplot2.

Author(s)

Michael Neely

See Also

make_valid, plot, par, points

plotlygg *Convert a plotly object to ggplot*

Description

[Stable] Converts a plotly object to a ggplot object. It is the inverse of `plotly::ggplotly()`.

Usage

```
plotlygg(p, print = TRUE)
```

Arguments

`p`	A plotly object to convert.
`print`	If `TRUE` (the default), will print the ggplot object and invisibly return it.

Details

This function extracts the data and layout from a plotly object and constructs a ggplot object with the same data. It supports various trace types including scatter, bar, and line traces.

Value

A ggplot object.

PM_build *Build Pmetrics*

Description

[Deprecated]

Building Pmetrics is no longer required. Models are now written in the pharmsol DSL and compiled just-in-time by the bundled Rust backend, so there is no separate compilation step and no Rust toolchain is needed.

This function is retained so that existing scripts continue to run; it simply informs the user that no action is necessary.

Usage

```
PM_build()
```

Value

Invisibly returns `NULL`.

Author(s)

Michael Neely and Julian Otalvaro

`PM_compare` *Compare runs*

Description

```
r lifecycle::badge("stable")
```

Compare convergence, -2*log likelihood, AIC/BIC, bias, imprecision, and regression statistics of population and posterior predictions. Additionally, compare distributions of support points between models (see details)

Usage

```
PM_compare(..., icen = "median", outeq = 1, plot = FALSE)
```

Arguments

`...`	PM_result objects to compare. See details.
`icen`	Can be either "median" for the predictions based on medians of `pred.type` parameter value distributions, or "mean". Default is "median".
`outeq`	Number of the output equation to compare; default is 1.
`plot`	Boolean indicating whether to generate and open the comparison report; default is FALSE

Details

Objects can be specified separated by commas, e.g. `PM_compare(run1, run2, run3)`. P-values are based on comparison using the nearest neighbors approach if all models are non-parametrics. Models may only be compared on parameters that are included in the first model. The P-value is the comparison between each model and the first model in the list. Missing P-values are when a model has no parameter names in common with the first model, and for the first model compared to itself. Significant P-values indicate that the null hypothesis should be rejected, i.e. the joint distributions between the two compared models for that parameter are significantly different.

Value

A highlighted table comparing the selected models with the following columns. In each metric column, the best value is highlighted in red. In the final best column, the red highlighting applies to the model with the most "best" metrics. For bias, imprecision, and regression intercept, the best value is the one closest to zero. For regression slope and R-squared, the best value is the one closest to 1. For -2*LL, AIC, and BIC, the best value is the lowest.

- **run** The run number of the data
- **nvar** Number of random parameters in the model

- **converged** Boolean value if convergence occurred.
- **-2*ll** Final cycle -2*Log-likelihood
- One of the following, depending on the option set in setPMoptions:
  - **aic** Final cycle Akaike Information Criterion OR
  - **bic** Final cycle Bayesian (Schwartz) Information Criterion
- **popBias** Bias, calculated by the method set in setPMoptions, of the predictions based on `icen` population parameters
- **popImp** Imprecision, calculated by the method set in setPMoptions, of the predictions based on `icen` population parameters
- **postBias** Bias, calculated by the method set in setPMoptions, of the predictions based on `icen` posterior parameters
- **postImp** Imprecision, calculated by the method set in setPMoptions, of the predictions based on `icen` posterior parameters
- **popInt** Intercept of observed vs. population predicted values regression
- **postInt** Intercept of observed vs. posterior predicted values regression
- **popSl** Slope of observed vs. population predicted values regression
- **postSl** Slope of observed vs. posterior predicted values regression
- **popR2** R-squared of observed vs. population predicted values regression
- **postR2** R-squared of observed vs. posterior predicted values regression
- **pval** P-value for each model compared to the first. See details.
- **best** Number of times each model was the best (lowest) in the above bias/imprecision, likelihood, and regression metrics.

Author(s)

Michael Neely

See Also

PM_load

PM_cov *Contains covariate data*

Description

[Stable]

Contains a data frame with subject-specific covariate data output.

Details

The PM_cov object is both a data field within a PM_result, and itself an R6 object comprising data fields and associated methods suitable for analysis and plotting relationships between covariates and posterior parameters, covariates over time, or parameter values over time.

Because PM_cov objects are automatically added to the PM_result at the end of a successful run, it is generally not necessary for users to generate PM_cov objects themselves.

The results are contained in the `$data` field, and it is this field which is passed to the `$plot` and `$summary` methods. You can use this `$data` field for custom manipulations, although usually this is better done on the `summary` object, e.g., `run1$cov$summary() |> select(africa, gender) |> table()`. If you are unfamiliar with the base R `|>` pipe function, type `help("|>")` into the R console for documentation and examples.

This output of this function is suitable for exploration of covariate- parameter, covariate-time, or parameter-time relationships.

Public fields

`data` A data frame with the following columns

- id Subject identification
- time Times of covariate observations
- covnames... Columns with each covariate observations in the dataset for each subject and `time`
- parnames... Columns with each parameter in the model and the `icen` summary for each subject, replicated as necessary for covariate observation times and duplicated for Bayesian parameter means and medians
- icen The type of summarized Bayesian posterior individual parameter values: mean or median

Methods

Public methods:

- `PM_cov$new()`
- `PM_cov$step()`
- `PM_cov$summary()`
- `PM_cov$plot()`
- `PM_cov$print()`
- `PM_cov$clone()`

`PM_cov$new()`: Create new object populated with covariate-parameter information

Usage:

```
PM_cov$new(PMdata = NULL, path = ".", ...)
```

Arguments:

`PMdata` include Saved, parsed output of prior run, used when source files are not available. .

`path` include Path to the folder containing the raw results of the run. Default is the current working directory. .

... Not currently used.

Details: Creation of new `PM_cov` object is automatic and not generally necessary for the user to do.

`PM_cov$step()`: Stepwise linear regression of covariates and Bayesian posterior parameter values

Usage:

```
PM_cov$step(...)
```

Arguments:

... Arguments passed to PM_step

Details: See PM_step.

`PM_cov$summary()`: Summary method

Usage:

```
PM_cov$summary(...)
```

Arguments:

... Arguments passed to summary.PM_cov

Details: See summary.PM_cov.

`PM_cov$plot()`: Plot method

Usage:

```
PM_cov$plot(...)
```

Arguments:

... Arguments passed to plot.PM_cov

Details: See plot.PM_cov.

`PM_cov$print()`: Print method

Usage:

```
PM_cov$print(...)
```

Arguments:

... Arguments passed to print

Details: Print method for PM_cov

`PM_cov$clone()`: The objects of this class are cloneable with this method.

Usage:

```
PM_cov$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Michael Neely, Julian Otalvaro

See Also

plot.PM_cov, PM_step

PM_cycle *Pmetrics Run Cycle Information*

Description

[Stable]

Contains the cycle information after a run.

Details

The PM_cycle object is both a data field within a PM_result, and itself an R6 object comprising data fields and associated methods suitable for analysis and plotting of cycle information generated during the run.

Because PM_cycle objects are automatically added to the PM_result at the end of a successful run, it is generally not necessary for users to generate PM_cycle objects themselves.

The main results are contained in the `$data` field, and it is this field which is passed to the `$plot` and `$summary` methods. You can use this `$data` field for custom manipulations, e.g. `last <- run1$cycle$data$aic |> tail(1)`. This will report the last cycle aic. If you are unfamiliar with the base R `|>` pipe function, type `help("|>")` into the R console for documentation and examples.

To provide a more traditional experience in R, the `$data` field is also separated by list items into the other data fields within the R6 object, e.g. `mean` or `gamlam`. This allows you to access them in an S3 way, e.g. `run1$cycle$mean` if `run1` is a PM_result object.

Public fields

`data` A list with the following elements, which can also be extracted by name. e.e. `run1$cycle$objective`, which is equivalent to `run1$cycle$data$objective`. **names** Vector of names of the random parameters **objective** A tibble of -2*Log-likelihood, AIC and BIC at each cycle **gamlam** A tibble of cycle number and gamma or lambda at each cycle for each output equation **mean** A tibble of cycle number and the mean of each random parameter at each cycle, normalized to initial mean **median** A tibble of cycle number and the median of each random parameter at each cycle, normalized to initial median **sd** A tibble of cycle number and the standard deviation of each random parameter at each cycle, normalized to initial standard deviation **status** Status of the last cycle: "Converged", "Maximum cycles reached", or "Posterior".

Active bindings

`names` Vector of names of the random parameters

`objective` A tibble of -2*Log-likelihood, AIC and BIC at each cycle

`gamlam` A tibble of cycle number and gamma or lambda at each cycle for each output equation

`mean` A tibble of cycle number and the mean of each random parameter at each cycle, normalized to initial mean

`median` A tibble of cycle number and the median of each random parameter at each cycle, normalized to initial median

`sd` A tibble of cycle number and the standard deviation of each random parameter at each cycle, normalized to initial standard deviation

`status` Status of the last cycle: "Converged", "Maximum cycles reached", or "Posterior"

Methods

Public methods:

- `PM_cycle$new()`
- `PM_cycle$plot()`
- `PM_cycle$summary()`
- `PM_cycle$print()`
- `PM_cycle$clone()`

`PM_cycle$new()`: Create new object populated with cycle information

Usage:

```
PM_cycle$new(PMdata = NULL, path = ".", ...)
```

Arguments:

`PMdata` include Saved, parsed output of prior run, used when source files are not available. .

`path` include Path to the folder containing the raw results of the run. Default is the current working directory. .

`...` Not currently used.

Details: Creation of new `PM_cycle` object is automatic and not generally necessary for the user to do.

`PM_cycle$plot()`: Plot method

Usage:

```
PM_cycle$plot(...)
```

Arguments:

`...` Arguments passed to plot.PM_cycle

Details: See plot.PM_cycle.

`PM_cycle$summary()`: Summary method

Usage:

```
PM_cycle$summary(...)
```

Arguments:

`...` Arguments passed to summary.PM_cycle

Details: See summary.PM_cycle.

`PM_cycle$print()`: Print method

Usage:

```
PM_cycle$print()
```

Details: Prints the last cycle summary information.

`PM_cycle$clone()`: The objects of this class are cloneable with this method.

Usage:

```
PM_cycle$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Michael Neely, Julian Otalvaro

`PM_data` *Defines the PM_data object*

Description

[Stable]

PM_data R6 objects containing raw, standardized and valid data, and methods to process the data

Details

PM_data objects are passed to the `$fit` method of compiled PM_model objects to initiate a population analysis. The object is created by reading a delimited file in the current working directory. The data will be transformed into the standard format which is the same for all engines, with a report of any assumptions that were necessary to standardize the data. PMcheck is called on the standard data to evaluate for errors. If dates and times are converted to relative decimal times in the standard data, automatic detection of the correct format will be attempted using `lubridate::parse_date_time()`. In the case of failure due to an unusual format, use the 'dt' argument to specify the correct format in your data. In the case of successful automatic detection, the format used will be included in the standardization report generated upon creation of a new *PM_data* object. Check carefully to make sure the correct format was chosen. Note that if your clock times did not include seconds, they were appended as ":00" to the end of each time and will appear that way in the copy of the original data.

There are a number of methods defined for a PM_data object, including to write the standard data back to a file for future use, to summarize and to plot the object, to conduct a non-compartmental analysis on the raw data using make_NCA, to calculate an AUC using make_AUC, and to add event rows, which is particularly useful for making simulation templates on the fly.

Public fields

`data` Data frame containing the data to be modeled

`standard_data` Data frame containing standardized version of the data

`pop` The `$data` field from a PM_pop object. This makes it easy to add population predictions to a raw data plot. This field will be `NULL` until the PM_data object is added to the PM_result after a run. As examples:

- `dat <- PM_data$new("data.csv")`. Here, `dat$pop` will be `NULL`.
- `run1 <- PM_load(1)`. Here, `run1$data$pop` will be the same as `run1$pop$data`.

`post` The `$data` field from a PM_post object. See details in the pop argument above.

Methods

Public methods:

- PM_data$new()
- PM_data$save()
- PM_data$auc()
- PM_data$nca()
- PM_data$plot()
- PM_data$print()
- PM_data$summary()
- PM_data$addEvent()
- PM_data$clone()

PM_data$new(): Create new data object

Usage:

```
PM_data$new(data = NULL, dt = NULL, quiet = FALSE, validate = TRUE, ...)
```

Arguments:

`data` A quoted name of a file with full path if not in the working directory, an unquoted name of a data frame in the current R environment, or a PM_data object, which will rebuild it.

`dt` Pmetrics will try a variety of date/time formats. If all 16 of them fail, use this parameter to specify the correct format as a character vector whose first element is date format and second is time. Use the following abbreviations:

- Y = 4 digit year
- y = 2 digit year
- m = decimal month (1, 2, ..., 12)
- d = decimal day (1, 2, ..., 31)
- H = hours (0-23)
- M = minutes (0-59) Example: `format = c("myd", "mh")`. Not one of the tried combinations! Always check to make sure that dates/times were parsed correctly and the relative times in the `PM_data$standard_data` field look correct. Other date/time formats are possible. See `lubridate::parse_date_time()` for these.

`quiet` Quietly validate. Default is `FALSE`.

`validate` Check for errors. Default is `TRUE`. Strongly recommended.

`...` Other arguments (not currently used).

Details: Creation of a new PM_data objects from a file or a data frame. Data will be standardized and checked automatically to a fully specified, valid data object.

PM_data$save(): Save data to file

Usage:

```
PM_data$save(file_name, ...)
```

Arguments:

`file_name` A quoted name of the file to create with full path if not in the working directory.

`...` Arguments passed to PMwriteMatrix

Details: Saves a delimited file (e.g. comma-separated) from the `standard_data` field

`PM_data$auc()`: Calculate AUC

Usage:

```
PM_data$auc(...)
```

Arguments:

`...` Arguments passed to make_AUC.

Details: See make_AUC.

`PM_data$nca()`: Perform non-compartmental analysis

Usage:

```
PM_data$nca(...)
```

Arguments:

`...` Arguments passed to make_NCA.

Details: See make_NCA.

`PM_data$plot()`: Plot method

Usage:

```
PM_data$plot(...)
```

Arguments:

`...` Arguments passed to plot.PM_data

Details: See plot.PM_data.

`PM_data$print()`: Print method

Usage:

```
PM_data$print(standard = F, viewer = T, ...)
```

Arguments:

`standard` Display the standardized data if `TRUE`. Default is `FALSE`.

`viewer` Display the Viewer if `TRUE`. Default is `TRUE`.

`...` Other arguments to print.data.frame. Only passed if `viewer = FALSE`.

Details: Displays the PM_data object in a variety of ways.

`PM_data$summary()`: Summary method

Usage:

```
PM_data$summary(...)
```

Arguments:

`...` Arguments passed to summary.PM_data.

Details: See summary.PM_data.

`PM_data$addEvent()`: Add events to PM_data object

Usage:

```
PM_data$addEvent(..., dt = NULL, quiet = FALSE, validate = FALSE)
```

Arguments:

`...` Column names and values.

`dt` Pmetrics will try a variety of date/time formats. If all 16 of them fail, use this parameter to specify the correct format as a character vector whose first element is date format and second is time. Use the following abbreviations:

- Y = 4 digit year
- y = 2 digit year
- m = decimal month (1, 2, ..., 12)
- d = decimal day (1, 2, ..., 31)
- H = hours (0-23)
- M = minutes (0-59) Example: `format = c("myd", "mh")`. Not one of the tried combinations! Always check to make sure that dates/times were parsed correctly and the relative times in the `PM_data$standard_data` field look correct. Other date/time formats are possible. See `lubridate::parse_date_time()` for these.

`quiet` Quietly validate. Default is `FALSE`.

`validate` Validate the new row or not. Default is `FALSE` as a new row added to a blank will result in a one-row data object, which is invalid. Also, only one event type (dose or observation) should be added at a time, so if the new object contains only doses while building, this would cause an error. You should set `validate = TRUE` for the final addition.

Details: Add lines to a PM_data object by supplying named columns and values. `ID` is always required. `Time` is handled differently depending on the sequence of `addEvent` calls (see **Chaining** below).

- It is required for the first call to `addEvent` and should be 0. For example: For example: `dat <- PM_data$new()$addEvent(id = 1, time = 0, dose = 100, addl = 5, ii = 24)`
- For subsequent calls to `addEvent` with specific times it should be included. For example: `dat <- PM_data$new()$addEvent(id = 1, time = 0, dose = 100, addl = 5, ii = 24)$addEvent(id = 1, time = 144, out = -1)` Here, because `out` wasn't in the original call *and* the next call contains a value for `time`, an `out` value of -1 will be added at time 144 and `out` will be set to `NA` for all the previous rows.
- In contrast, the behavior is different if you omit `time` when your data object already has rows. In this case the arguments in the call to `addEvent` (without a value for `time`) will add those arguments as columns in the prior data with the specified value or *replace* values in those columns if they already exist. Be sure this is what you want. For example, building on the prior example: `dat$addEvent(id = 1, dur = 0.5)`. Note that we can chain to the previously created `dat` object. Here, a duration of 0.5 hours will be added to every previous row in `dat` to create the new `dat` object, but no new row is added since there is no `time` associated with it.

Adding covariates is supported, but since valid subject records in Pmetrics with covariates must contain non-missing values at time 0, covariates should be included with the first call to `$addEvent()`.

As we have seen in the examples above, `ADDL` and `II` are supported.

Chaining Multiple `$addEvent()` calls can be chained with `PM_data$new()` to create a blank data object and then add rows. This can be particularly useful for creating simulation templates. See the example.

Examples:

```
PM_data$new()$addEvent(id = 1, time = 0, dose = 100, addl = 4, ii = 12,
out = NA, wt = 75)$addEvent(id = 1, time = 60, out = -1)
```

`PM_data$clone()`: The objects of this class are cloneable with this method.

Usage:

```
PM_data$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `PM_data$addEvent()`
## ------------------------------------------------

## Not run:
PM_data$new()$addEvent(id = 1, time = 0, dose = 100, addl = 4, ii = 12,
out = NA, wt = 75)$addEvent(id = 1, time = 60, out = -1)

## End(Not run)
```

`PM_final` *Final Cycle Population Values*

Description

[Stable]

Contains final cycle information from run.

Details

The PM_final object is both a data field within a PM_result, and itself an R6 object comprising data fields and associated methods suitable for analysis and plotting of final cycle parameters.

Because PM_final objects are automatically added to the PM_result at the end of a successful run, it is generally not necessary for users to generate PM_final objects themselves.

The main results are contained in the `$data` field, and it is this field which is passed to the `$plot` and `$summary` methods. You can use this `$data` field for custom manipulations, e.g. `probs <- run1$final$data$popPoints |> select(prob)`. This will select the probabilities of the support points. If you are unfamiliar with the base R `|>` pipe function, type `help("|>")` into the R console for documentation and examples.

To provide a more traditional experience in R, the `$data` field is also separated by list items into the other data fields within the R6 object, e.g. `popMean` or `nsub`. This allows you to access them in an S3 way, e.g. `run1$final$popMean` if `run1` is a PM_result object.

Public fields

`data` A list with the following elements, which can also be extracted by name.

- **popPoints** (NPAG only) Data frame of the final cycle joint population density of grid points with column names equal to the name of each random parameter plus *prob* for the associated probability of that point
- **popMean** The final cycle mean for each random parameter distribution
- **popSD** The final cycle standard deviation for each random parameter distribution
- **popCV** The final cycle coefficient of variation (SD/Mean) for each random parameter distribution
- **popVar** The final cycle variance for each random parameter distribution
- **popCov** The final cycle random parameter covariance matrix
- **popCor** The final cycle random parameter correlation matrix
- **popMed** The final cycle median values for each random parameter, i.e. those that have unknown mean and unknown variance, both of which are fitted during the run
- **postPoints** (NPAG only) Data frame of posterior population points for each of the first 100 subject, with columns id, point, parameters and probability. The first column is the subject, the second column has the population point number, followed by the values for the parameters in that point and the probability.
- **postMean** A *nsub* x *npar* data frame containing the means of the posterior distributions for each parameter.
- **postSD** A *nsub* x *npar* data frame containing the SDs of the posterior distributions for each parameter.
- **postVar** A *nsub* x *npar* data frame containing the variances of the posterior distributions for each parameter.
- **postCov** NPAG only: An list of length *nsub*, each element with an *npar* x *npar* data frame that contains the posterior parameter value covariances for that subject.
- **postCor** NPAG only: An list of length *nsub*, each element with an *npar* x *npar* data frame that contains the posterior parameter value correlations for that subject.
- **postMed** A *nsub* x *npar* data frame containing the medians of the posterior distributions for each parameter.
- **shrinkage** A data frame with the shrinkage for each parameter.
- **gridpts** (NPAG only) Initial number of support points
- **nsub** Number of subjects
- **ab** Tibble/data frame of boundaries for random parameter values with columns: name, lower, upper.

Active bindings

`popPoints` (NPAG only) Data frame of the final cycle joint population density of grid points with column names equal to the name of each random parameter plus *prob* for the associated probability of that point

`popMean` The final cycle mean for each random parameter distribution

`popSD` The final cycle standard deviation for each random parameter distribution

`popCV` The final cycle coefficient of variation (SD/Mean) for each random parameter distribution

`popVar` The final cycle variance for each random parameter distribution

`popCov` The final cycle random parameter covariance matrix

`popCor` The final cycle random parameter correlation matrix

`popMed` The final cycle median values for each random parameter, i.e. those that have unknown mean and unknown variance, both of which are fitted during the run

`postPoints` (NPAG only) Data frame of posterior population points for each of the first 100 subject, with columns id, point, parameters and probability. The first column is the subject, the second column has the population point number, followed by the values for the parameters in that point and the probability.

`postMean` A *nsub* x *npar* data frame containing the means of the posterior distributions for each parameter.

`postSD` A *nsub* x *npar* data frame containing the SDs of the posterior distributions for each parameter.

`postVar` A *nsub* x *npar* data frame containing the variances of the posterior distributions for each parameter.

`postCov` NPAG only: An list of length *nsub*, each element with an *npar* x *npar* data frame that contains the posterior parameter value covariances for that subject.

`postCor` NPAG only: An list of length *nsub*, each element with an *npar* x *npar* data frame that contains the posterior parameter value correlations for that subject.

`postMed` A *nsub* x *npar* data frame containing the medians of the posterior distributions for each parameter.*

`shrinkage` A data frame with the shrinkage for each parameter. The total population variance for a parameter is comprised of variance(EBE) plus average variance(EBD), where each subject's EBE is the Empirical Bayes Estimate or mean posterior value for the parameter. EBD is the Empirical Bayes Distribution, or the full Bayesian posterior parameter value distribution for each subject.

The typical definition of η shrinkage is $[1 - \frac{SD(\eta)}{\omega}]$ or $[1 - \frac{var(\eta)}{\omega^2}]$, where η is the EBE and ω^2 is the population variance of η.

In parametric modeling approaches η is the interindividual variability around the typical (mean) value of the parameter in the population, usually referred to as θ. In nonparametric approaches, there is no assumption of normality, so η simply becomes each subject's mean parameter value estimate.

Here is how Pmetrics derives and then calculates shrinkage for a given parameter.

$$popVar = var(EBE) + mean(var(EBD))$$

$$1 = \frac{var(EBE)}{popVar} + \frac{mean(var(EBD)}{popVar}$$

$$1 - \frac{var(EBE)}{popVar} = \frac{mean(var(EBD))}{popVar}$$

$$shrinkage = \frac{mean(var(EBD))}{popVar}$$

Shrinkage is therefore a fraction between 0 and 1. If Bayesian posterior distributions are wide for a given parameter and $mean(var(EBD))$ is high due to sparse or uninformative

sampling, then most of the population variance is due to this variance and shrinkage is high, i.e., individual posterior estimates (EBE) shrink towards the population mean. Be aware, however, that a Bayesian posterior parameter value distribution for a given subject who is sparsely sampled may also be a single support point with no variance. Therefore EBD under nonparametric assumptions is not always large with uninformative sampling. This means that shrinkage is not as readily interpretable in nonparametric population modeling.

An alternative is to consider the number of support points relative to the number of subjects. Highly informed, distinct subjects will result in the maximum possible number of support points, *N*, which is the same as the number of subjects. In contrast, badly undersampled subjects can result in only one support point. There is no formal criterion for this statistic, but it can be used in combination with shrinkage to assess the information content of the data.

`gridpts` (NPAG only) Initial number of support points

`nsub` Number of subjects

`ab` Matrix of boundaries for random parameter values

Methods

Public methods:

- `PM_final$new()`
- `PM_final$plot()`
- `PM_final$summary()`
- `PM_final$clone()`

`PM_final$new()`: Create new object populated with final cycle information

Usage:

```
PM_final$new(PMdata = NULL, path = ".", ...)
```

Arguments:

`PMdata` include Saved, parsed output of prior run, used when source files are not available. .

`path` include Path to the folder containing the raw results of the run. Default is the current working directory. .

`...` Not currently used.

Details: Creation of new `PM_final` object is automatic and not generally necessary for the user to do.

`PM_final$plot()`: Plot method

Usage:

```
PM_final$plot(...)
```

Arguments:

`...` Arguments passed to plot.PM_final

Details: See plot.PM_final.

`PM_final$summary()`: Summary method

Usage:

```
PM_final$summary(...)
```

Arguments:

... Arguments passed to summary.PM_final

Details: See summary.PM_final.

`PM_final$clone()`: The objects of this class are cloneable with this method.

Usage:

```
PM_final$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Michael Neely, Julian Otalvaro

`PM_help` *Get Help and Report Issues*

Description

[Stable]

This function displays system information useful for debugging and provides a link to the GitHub repository for reporting bugs or requesting help.

Usage

```
PM_help(copy = TRUE)
```

Arguments

`copy`	Logical. If `TRUE`, copies the system information to clipboard. Default is `TRUE`.

Details

When you encounter bugs or need help, this function collects relevant system information that will help maintainers diagnose the issue. The information includes OS version, R version, RStudio version (if applicable), package version, Rust and Cargo versions (if available).

Value

Invisibly returns a list containing the system information.

Examples

```
## Not run:
# Display help information
PM_help()

# Copy information to clipboard
PM_help(copy = TRUE)

## End(Not run)
```

PM_load *Load Pmetrics NPAG or IT2B output*

Description

[Stable] Loads all the data from a prior Pmetrics run.

Usage

```
PM_load(run, path = ".", file = "PMout.Rdata")
```

Arguments

run	The numerical value of the folder in path containing run results
path	include Path to the folder containing the raw results of the run. Default is the current working directory. .
file	Default is "PMout.Rdata", which is created after a Pmetrics run, but it could also be the name of an .Rdata file created by running the $save method for a PM_result object.

Details

A combination of run, path, and file are used to locate the results.

- If run is provided, it is assumed that the results are in a subfolder /outputs of the folder named by run within the path folder.
- If run is missing, the results are assumed to be in the path folder.
- The file name is the name of the Rdata file containing the results. Default is "PMout.Rdata", which is created by Pmetrics after a run.
- If both run and path are missing, the current working directory is used for path.
- If both run and file are missing, the current working directory is used for path and "PMout.Rdata" is used for file.
- If both path and file are missing, the current working directory is used for path and run is required.
- If all three are missing, the current working directory is used for path and "PMout.Rdata" is used for file.

Value

An R6 PM_result.

Author(s)

Michael Neely and Julian Otalvaro

See Also

PM_final, PM_cycle, PM_op, PM_cov, PM_pop, PM_post

Examples

```
## Not run:
run1 <- PM_load(1)
# loads from ./1/outputs/PMout.Rdata, where "." is the current working directory

run2 <- PM_load(2, path = "Pmetrics/MyRuns")
# loads from Pmetrics/MyRuns/2/outputs/PMout.Rdata

run3 <- PM_load(path = "Pmetrics/MyRuns/3", file = "MyResults.Rdata")
# loads from Pmetrics/MyRuns/3/MyResults.Rdata

run4 <- PM_load(file = "Pmetrics/MyRuns/4/outputs/PMout.Rdata")
# loads from Pmetrics/MyRuns/4/outputs/PMout.Rdata

run5 <- PM_load()
# loads from ./PMout.Rdata

## End(Not run)
```

PM_manual *Open user and function manuals.* **[Stable]**

Description

Opens the Pmetrics User reference online

Usage

```
PM_manual()
```

Details

Help for Pmetrics.

PM_model *Defines the PM_model class*

Description

[Stable]

PM_model objects contain the variables, covariates, equations and error models necessary to run a population analysis.

Details

PM_model objects are one of two fundamental objects in Pmetrics, along with `PM_data()` objects. Defining a PM_model allows for fitting it to the data via the `$fit()` method to conduct a population analysis, i.e. estimating the probability distribution of model equation paramter values in the population. The PM_model object is created using the a model building app (coming soon), by defining a list directly in R, or by reading a model text file. When reading a model text file, the list code is generated and copied to the clipboard for pasting in to scripts. Model files will be deprecated in future versions of Pmetrics.

Use `PM_model$from_dsl()` to create a model directly from pharmsol DSL text. The DSL defines the model structure. The `pri` and `err` arguments provide the Pmetrics parameter ranges and error models.

Create a model from DSL

```
PM_model$from_dsl(
  dsl,
  pri,
  err,
  solver = NULL,
  quiet = FALSE
)
```

`dsl` is one character string containing the model source. Route and output labels declared in the DSL are the labels Pmetrics expects in the data's `INPUT` and `OUTEQ` columns. Purely numeric identifiers in the data are canonicalised to the `input_n` / `outeq_n` form pharmsol requires, so `INPUT = 1` matches a route declared as `input_1`.

Some notes on the example at the end of this help page:

- It's a complete example of a three compartment model with delayed absorption.
- We show the method of defining the model first and embedding the `PM_model$new()` within a `dontrun` block to avoid automatic compilation.
- Since this model can also be solved analytically with algebra, we could have used `eqn = function(){three_comp_bolus}`.

Public fields

`model_list` A list containing the model components built by translating the original arguments into Rust

`arg_list` A list containing the original arguments passed to the model

`binary_path` Deprecated. Retained as `NULL` for backward compatibility; models are no longer compiled to a binary. See the `dsl` field.

`dsl` The model definition rendered as pharmsol DSL source, used by the Rust backend to JIT-compile the model at run time.

Methods

Public methods:

- `PM_model$new()`
- `PM_model$print()`
- `PM_model$plot()`
- `PM_model$fit()`
- `PM_model$map()`
- `PM_model$sim()`
- `PM_model$compile()`
- `PM_model$save()`
- `PM_model$copy()`
- `PM_model$clone()`

`PM_model$new()`: This is the method to create a new `PM_model` object.

The first argument allows creation of a model from a variety of pre-existing sources, and if used, all the subsequent arguments will be ignored. If a model is defined on the fly, the arguments form the building blocks. Blocks are of two types:

- **Lists** define *primary parameters*, *covariates*, and *error models*. These portions of the model have specific and defined creator functions and no additional R code is permissible. They take this form:

  ```
  block_name = list(
    var1 = creator(),
    var2 = creator()
  )
  ```

  Note the comma separating the creator functions, "`c(`" to open the vector and "`)`" to close the vector. Names are case-insensitive and are converted to lowercase for Rust.

- **Functions** define the other parts of the model, including *secondary (global) equations*, *model equations* (e.g. ODEs), *lag time*, *bioavailability*, *initial conditions*, and *outputs*. These parts of the model are defined as R functions without arguments, but whose body contains any permissible R code.

  ```
  block_name = function() {

   # any valid R code
   # can use primary or secondary parameters and covariates
  ```

```
  # lines are not separated by commas

}
```

Note the absence of arguments between the "()", the opening curly brace "{" to start the function body and the closing curly brace "}" to end the body. Again, all R code will be converted to lowercase prior to translation into Rust.

Important: All models must have `pri`, `eqn`, `out`, and `err` blocks.

Usage:

```
PM_model$new(
  x = NULL,
  pri = NULL,
  cov = NULL,
  sec = NULL,
  eqn = NULL,
  lag = NULL,
  fa = NULL,
  ini = NULL,
  out = NULL,
  err = NULL,
  solver = NULL,
  ...,
  .dsl = NULL
)
```

Arguments:

`x` An optional argument, but if specified, all the subsequent arguments will be ignored. `x` creates a `PM_model` from existing appropriate input, which can be one of the following:

- Quoted name of a model text file in the working directory which will be read and passed to Rust engine. **Note:** Model text files are being deprecated in future versions of Pmetrics.
- List that defines the model directly in R. This will be in the same format as if all the subsequent arguments were used. For example:

```
mod_list <- list(
 pri = list(...),
 eqn = function(){...},
 out = function(){...},
 err = c(...)
)
mod <- PM_model$new(mod_list)
```

- `PM_model` object, which will simply rebuild it, e.g. carrying on the prior example: `PM_model$new(mod)`

See the user manual `PM_manual()` for more help on directly defining models in R.

`pri` The first of the arguments used if `x` is not specified. This is a named list of primary parameters, which are the model parameters that are estimated in the population analysis. They are specified by one of two creator functions: `ab()` or `msd()`. For example,

```
pri = list(
  Ke = ab(0, 5),
  V = msd(100, 10)
)
```

The ab() creator specifies the initial range [a, b] of the parameter, while the msd() creator specifies the initial mean and standard deviation of the parameter.

cov A list whose names are some or all of the covariates in the data file. Unlike prior versions of Pmetrics, as of 3.0.0, they do not have to be listed in the same order as in the data file, and they do not need to be all present. **Only those covariates you wish to use in model equations or analyze for relationships to model parameters need to be declared here.** Values for each element in the covariate vector are the interp() creator function to declare how each covariate is interpolated between entries in the data. The default argument for interp() is "lm" which means that values will be linearly interpolated between entries, like the R linear model function stats::lm(). The alternative is "none", which holds the covariate value the same as the previous entry until it changes, i.e., a carry-forward strategy. For example:

```
cov = list(
  wt = interp(), # will be linear by default
  visit = interp("none")
)
```

Note that wt = interp() is equivalent to wt = interp("lm"), since "lm" is the default.

sec A function defining the secondary (global) equations in the model. Values are not estimated for these equations but they are available to every other block in the model. For example:

```
sec = function() {
  V = V0 * (wt/70)
}
```

Note that the function must be defined with no arguments between the parentheses, and the body **must be in R syntax**. Any number of lines and R code, e.g. if - else statements, etc. are permissible.

eqn A function defining the model equations. The function must have no arguments. The body of the function may contain three kinds of equations, written in R syntax.

- **Implicit equations** referenced by calling the name of a Pmetrics model library object detailed in model_lib(). The Pmetrics model library contains a number of template models solved analytically (algebraically) and may include user-defined models. For example, to use a two-compartment model with intavenous input:

  ```
  eqn = function(){
   two_comp_iv
  }
  ```

  Required parameters for library models are listed for each model and must be included in model blocks exactly as named. For example, in a one-compartment model Ke is a required parameter. Thus, if Ke is a function of a covariate called "crcl", here is a code snippet illustrating the inclusion of the required parameter.

  ```
  mod <- PM_model$new(
    pri = list(
      ke0 = ab(0, 5),
      v = ab(0, 100)
    ),
    cov = list(
      crcl = interp()
    ),
    eqn = function(){
  ```

```
    one_comp
    ke = ke0 * crcl # the required parameter, ke, is defined
  },
  ... # more model blocks, including out, err
)
```

- **Explicit equations** are ordinary differential equations that directly define a model. Use the following notation in equations:
  - `dx[i]` for the change in amount with respect to time (i.e., dx/dt), where `i` is the compartment number,
  - `x[i]` for the compartment amount, where `i` is the compartment number.
  - `rateiv[j]` for the infusion rate of input `j`, where `j` is the input number in the data corresponding to doses for that input.
  - Bolus doses are indicated by `DUR = 0` for dose events in the data. Currently only one bolus input is allowed, which goes into compartment 1 and is not modifiable. It does not appear in the differential equations.

    For example,

    ```
    eqn = function() {
     dx[1] = -ka * x[1]
     dx[2] = rateiv[1] + ka * x[1] - ke * x[2]
    }
    ```
- **Additional equations** in R code can be defined in this block, which are similar to the `sec` block, but will only be available within the `eqn` block as opposed to global availability when defined in `sec`. They can be added to either

`lag` A function defining the lag time (delayed absorption) for the bolus input. The function must have no arguments, and the equations must be defined in R syntax The equations must be defined in the form of `lag[i] = par`, where `lag[i]` is the lag for drug (input) `i` and `par` is the lag parameter used in the `pri` block.

For example, if `antacid` is a covariate in the data file, and `lag1` is a primary parameter, this code could be used to model delayed absorption if an antacid is present.

```
lag = function() {
  lag[1] = if(antacid == 1) lag1 else 0
}
```

As for `eqn`, additional equations in R code can be defined in this block, but will only be available within the `lag` block.

`fa` A function defining the bioavailability (fraction absorbed) equations, similar to `lag`.

Example:

```
fa = function() {
  fa[1] = if(antacid == 1) fa1 else 1
}
```

As for `eqn`, additional equations in R code can be defined in this block, but will only be available within the `fa` block.

`ini` A function defining the initial conditions for a compartment in the model. Structure is similar to `lag` and `fa`.

Example:

```
ini = function() {
  x[2] = init2 * V
}
```

This sets the initial amount of drug in compartment 2 to the value of a covariate `init2` multiplied by the volume of the compartment, `V`, assuming `V` is either a primary parameter or defined in the `sec` block.

As for `eqn`, additional equations in R code can be defined in this block, but will only be available within the `ini` block.

`out` A function defining the output equations, which are the predictions from the model. The function must have no arguments, and the equations for predictions must be defined in R syntax.

Use the following notation in equations:

- `y[i]` for the predicted value, where `i` is the output equation number, typically corresponding to an observation with `outeq = i` in the data, but not always (see **Note** below).
- `x[j]` for the compartment amount, where `j` is the compartment number.

As with all function blocks, secondary equations are permitted, but will be specific to the `out` block.

For example,

```
out = function() {
  V = V0 * wt # only needed if not included in sec block
  y[1] = x[1]/V
  #Vp and Vm must be defined in pri or sec blocks
  y[2] = x[2]/Vp
  y[3] = x[3]/Vm
}
```

This assumes `V`, `Vp`, and `Vm` are either primary parameters or defined in the `sec` block.

Note that as of Pmetrics 3.0.0, you can have more output equations than values for `outeq` in the data. This is not possible with prior versions of Pmetrics. Outputs without corresponding observations are not used in the fitting, but do generate predictions. For example, this snippet is part of a model that calculates AUC:

```
eqn = function(){
  dx[1] = -ka * x[1]
  dx[2] = rateiv[1] + ka * x[1] - ke * x[2]
  dx[3] = x[2] - x[3]
  dx[4] = x[1] / v
},
out = function(){
  y[1] = x[1]/v
  y[2] = x[4]
},
err = list(
 proportional(2, c(0.1, 0.15, 0, 0))
)
```

If the data only contain observations for `y[1]`, i.e. the concentration of drug in the plasma compartment with `outeq = 1`, the model will use that information to optimize the parameter values, but will also generate predictions for `y[2]`, which is the AUC of the drug in compartment 1, even though there is no `outeq = 2` in the data. There is only one `err` equation since there is only one source of observations: plasma concentration. AUC (`y[2]`) is not fitted to any observations; it is a calculation based on the model state, given the optimized parameter values. It's not required, but shown here for illustrative purposes.

`err` An unammed vector of error models for each of the output equations with observations, i.e. those that have an `outeq` number associated with them in the data. Each error model is defined by the `proportional()` creator or the `additive()` creator, relative to the observation error. For example, if there are three output equations corresponding to three sources of observations in the data, the error models could be defined as:

```
err = list(
  proportional(2, c(0.1, 0.15, 0, 0)),
  proportional(3, c(0.05, 0.1, 0, 0), outeq = 2),
  additive(1, c(0.2, 0.25, 0, 0), outeq = 3)
)
```

This defines the first two output equations to have proportional error with initial values of 2 and 3, respectively, and the third output equation to have additive error with initial value of 1. Each output is measured by a different assay with different error characteristics. Note that you must specify which outeq corresponds to the values, with a default `outeq = 1`, which is not required if there is only one output equation.

`solver` Optional ODE solver for user-defined ODE models. Supported values are "BDF", "TRBDF2", "ESDIRK34", and "TSIT45". This is ignored for analytical library models.

`...` Not currently used.

`.dsl` Internal DSL source used by `PM_model$from_dsl()`.

`PM_model$print()`: Print the model summary.

Usage:

```
PM_model$print(...)
```

Arguments:

`...` Not used.

Details: This method prints a summary of the model.

`PM_model$plot()`: Plot the model.

Usage:

```
PM_model$plot(...)
```

Arguments:

`...` Additional arguments passed to the plot function.

Details: This method plots the model using the `plot.PM_model()` function.

`PM_model$fit()`: This is the main method to run a population analysis.

Usage:

```
PM_model$fit(
  data = NULL,
  path = ".",
  run = NULL,
  include = NULL,
  exclude = NULL,
  cycles = 100,
  prior = "sobol",
  points = 100,
```

```
  idelta = 0.1,
  tad = 0,
  seed = 23,
  overwrite = FALSE,
  algorithm = "NPAG",
  report = getPMoptions("report_template"),
  quiet = FALSE
)
```

Arguments:

`data` Either the name of a PM_data object in memory or the quoted filename (with or without a path) of a Pmetrics data file. If the path is not specified, the file is assumed to be in the current working directory, unless the `path` argument below is also specified as a global option for the fit. The file will be used to create a PM_data object on the fly. However, if created on the fly, this object will not be available to other methods or other instances of `$fit()`.

`path` Optional full path or relative path from current working directory to the folder where `data` and `model` are located if specified as filenames without their own paths, and where the output will be saved. Default is the current working directory.

`run` Specify the run number of the output folder. Default if missing is the next available number.

`include` Vector of subject id values in the data file to include in the analysis. The default (missing) is all.

`exclude` A vector of subject IDs to exclude in the analysis, e.g. `c(4,6:14,16:20)`

`cycles` Number of cycles to run. Default is 100.

`prior` The distribution for the initial support points, which can be one of several options.

- The default is "sobol", which is a semi-random distribution. This is the distribution typically used when fitting a new model to the data. An example of this is on our website.

The following all specify non-random, informative prior distributions. They are useful for either continuing a previous run which did not converge or for fitting a model to new data, whether to simply calculate Bayesian posteriors with `cycles = 0` or to revise the model to a new covergence with the new data.

- The name of a suitable PM_result object from a prior run loaded with PM_load. This starts from the non-uniform, informative distribution obtained at the end of a prior NPAG run. Example: `run1 <- PM_load(1); fit1$run(prior = run1)`.
- A character string with the filename of a csv file containing a prior distribution with format as for 'theta.csv' in the output folder of a prior run: column headers are parameter names, and rows are the support point values. A final column with probabilities for each support point is not necessary, but if present will be ignored, as these probabilities are calculated by the engine. Note that the parameter names must match the names of the primary variables in the model. Example: `fit1$run(prior = "mytheta.csv")`.
- The number of a previous run with `theta.csv` in the output folder which will be read as for the filename option above. Example: `fit1$run(prior = 2)`.
- A data frame obtained from reading an approriate file, such that the data frame is in the required format described in the filename option above. Example: `mytheta <- read_csv("mytheta.csv"); fit1$`

`points` The number of initial support points if one of the semi-random, uniform distributions are selected in the `prior` argument above. Default is 100. The initial points are spread through the hyperspace defined by the random parameter ranges and begin the search for

the optimal parameter value distribution (support points) in the population. If there are fewer than 2 points per unit range for any parameter, Pmetrics will suggest the minimum number of points that should be tried. The greater the initial number of points, the less chance of missing the globally maximally likely parameter value distribution, but the slower the run.

`idelta` How often to generate posterior predictions in units of time. Default is 0.1, which means a prediction is generated every 0.1 hours (6 minutes) if the unit of time is hours. Predictions are made at this interval until the time of the last event (dose or observation) or until `tad` if that value is greater than the time of the last dose or observation in the data.

`tad` Length of time after the last dose event to add additional predictions at frequency `idelta`. Default is 0, which means no additional predictions beyond the last dose, assuming the dose is the last event. . If the last observation in the data is after `tad`, then a prediction will be generated at time = `tad` after the last dose

`seed` Seed used if `prior = "sobol"`. Ignored otherwise.

`overwrite` Boolean operator to overwrite existing run result folders. Default is `FALSE`.

`algorithm` The algorithm to use for the run. Default is "NPAG" for the **N**on-**P**arametric **A**daptive **G**rid. Alternatives: "NPOD".

`report` If missing, the default Pmetrics report mode specified by getPMoptions is used. Otherwise, choose `"app"`, `"plotly"`, `"ggplot"`, or `"none"`.

`quiet` Boolean operator to suppress messages during the run. Default is `FALSE`.

`intern` Run NPAG in the R console without a batch script. Default is TRUE.

Details: As of Pmetrics 3.0.0, models contain compiled code to fit the model equations to the data, optimizing the parameter value probability distributions in the population to maximize their likelihood, or more precisely, minimize the objective function, which is -2*log-likelihood. The `$fit()` method is the means of running that compiled code to conduct to fitting procedure. At a minimum, it requires a PM_data object, which can be created with `PM_data$new()`. There are a number of additional arguments to control the fitting procedure, such as the number of cycles to run, the initial number of support points, and the algorithm to use, among others.
The `$fit()` method is the descendant of the legacy NPrun function, which is maintained as a wrapper to `$fit()` for backwards compatibility.

Returns: A successful run will result in creation of a new folder in the working directory with the results inside the folder.

`PM_model$map()`**:** Calculate posteriors from a fitted model. #' @details This method calculates posteriors from a compiled model. It is not necessary to have run the model first, but it is necessary to have an informative prior distribution. This prior will typically be the result of a previous run, but may also be a file containing support points, with each column named as a parameter in the model plus a final column for probability. Each row contains values for the parameters and the associated probability for those parameter values. The file can be saved as a csv file.

To calculate the posteriors, `map()` calls the `fit()` method with the `cycles` argument set to 0 and the `algorithm` argument set to "POSTPROB". If `data` are not provided as an argument to `map()`, the model's `data` field is used instead. If `data` is provided, it must be a PM_data object or the pathname of a file which can be loaded as a PM_data object.

Usage:

```
PM_model$map(...)
```

Arguments:

`...` Arguments passed to the `fit` method. Note that the `cycles` argument is set to 0, and the `algorithm` argument is set to "POSTPROB" automatically.

`PM_model$sim()`: Simulate data from the model using a set of parameter values.

Usage:

```
PM_model$sim(data, theta, quiet = FALSE)
```

Arguments:

`data` A PM_data object containing the dosing and observation information.

`theta` A matrix of parameter values to use for the simulation. The `theta` matrix should have the same number of columns as the number of primary parameters in the model. Each row of `theta` represents a different set of parameter values.

`quiet` Logical, if TRUE, suppresses messages during simulation.

Details: This method simulates output from the model using a set of parameter values provided in the `theta` matrix and the template for dosing/observations in the `data` object.

`PM_model$compile()`: Render the model to pharmsol DSL source.

Usage:

```
PM_model$compile(quiet = FALSE)
```

Arguments:

`quiet` Logical, if TRUE, suppresses messages.

Details: This method generates the pharmsol DSL representation of the model and stores it in the `dsl` field. The Rust backend JIT-compiles this source at run time, so no Rust toolchain or separate compilation step is required. The method name is retained for backward compatibility.

`PM_model$save()`: Save model to file (deprecated).

Usage:

```
PM_model$save()
```

Details: This method is deprecated. Existing or manually created model files may be read with `PM_model$new(filename)`, but including model code in scripts is preferred, as this makes models used in runs transparent and more easily edited. Use the `PM_model$copy()` method instead to copy the model code to the clipboard and paste into scripts.

`PM_model$copy()`: Copy model code to clipboard.

Usage:

```
PM_model$copy()
```

Details: This method copies the R code to create the model to the clipboard. This is useful for saving the model code in a script.

`PM_model$clone()`: The objects of this class are cloneable with this method.

Usage:

```
PM_model$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Michael Neely

Examples

```
mod_list <- list(
  pri = list(
    CL = ab(10, 200),
    V0 = ab(0, 100),
    ka = ab(0, 3),
    k23 = ab(0, 5),
    k32 = ab(0, 5),
    lag1 = ab(0, 2)
  ),
  cov = list(
    wt = interp()
  ),
  sec = function() {
    V <- V0 * (wt / 70)
    ke <- CL / V # define here to make eqn simpler
  },
  eqn = function() {
    dx[1] <- -ka * x[1]
    dx[2] <- rateiv[1] + ka * x[1] - (ke + k23) * x[2] + k32 * x[3]
    dx[3] <- k23 * x[2] - k32 * x[3]
    dx[4] <- x[1] / V
  },
  lag = function() {
    tlag[1] <- lag1
  },
  out = function() {
    y[1] <- x[1] / V
    y[2] <- x[4] # AUC, not fitted to any data, not required
  },
  err = list(
    proportional(2, c(0.1, 0.15, 0, 0)) # only applies to y[1]
  )
)

## Not run:
mod <- PM_model$new(mod_list)

## End(Not run)
```

PM_op *Observed vs. predicted data*

Description

[Stable]

Contains observed vs. predicted data after a run, typically a field in a PM_result

Details

The PM_op object is both a data field within a PM_result, and itself an R6 object comprising data fields and associated methods suitable for analysis and plotting of observed vs. population or individual predicted outputs.

Because PM_op objects are automatically added to the PM_result at the end of a successful run, it is generally not necessary for users to generate PM_op objects themselves.

The main results are contained in the `$data` field, and it is this field which is passed to the `$plot` and `$summary` methods. You can use this `$data` field for custom manipulations, e.g. `trough <- run1$op$data |> filter(time == 24)`. If you are unfamiliar with the base R `|>` pipe function, type `help("|>")` into the R console for documentation and examples.

To provide a more traditional experience in R, the `$data` field is also separated by columns into the other data fields within the R6 object, e.g. `id` or `time`. This allows you to access them in an S3 way, e.g. `run1$op$time` if `run1` is a PM_result object.

Public fields

`id` subject identification

`time` observation time in relative units, usually hours

`obs` observation

`cens` censoring information: "none" for observed, "bloq" for below limit of quantification, "aloq" for above limit of quantification

`pred` prediction

`pred.type` Population predictions based on Bayesian prior parameter value distribution, or individual predictions based on Bayesian posterior parameter value distributions

`icen` Predictions based on mean or median of Bayesian `pred.type`parameter values

`outeq` output equation number

`block` dosing block number for each subject, as defined by dose resets (evid=4).

`obsSD` standard deviation of the observation based on the assay error polynomial

`d` prediction error, `pred` - `obs`

`ds` squared prediction error

`wd` weighted prediction error, which is the prediction error divided by the `obsSD`

`wds` weighted squared prediction error

`data` A data frame of class **PM_op_data** combining all the above fields as its columns

Methods

Public methods:

- `PM_op$new()`
- `PM_op$plot()`
- `PM_op$summary()`
- `PM_op$auc()`
- `PM_op$clone()`

`PM_op$new()`: Create new object populated with observed vs. predicted data

Usage:

```
PM_op$new(PMdata = NULL, path = ".", ...)
```

Arguments:

`PMdata` include Saved, parsed output of prior run, used when source files are not available. .

`path` include Path to the folder containing the raw results of the run. Default is the current working directory. .

`...` Not currently used.

Details: Creation of new `PM_op` object is automatic at the end of a run and not generally necessary for the user to do.

`PM_op$plot()`: Plot method

Usage:

```
PM_op$plot(...)
```

Arguments:

`...` Arguments passed to plot.PM_op

Details: See plot.PM_op.

`PM_op$summary()`: Summary method

Usage:

```
PM_op$summary(...)
```

Arguments:

`...` Arguments passed to summary.PM_op

Details: See summary.PM_op.

`PM_op$auc()`: Calculate AUC

Usage:

```
PM_op$auc(...)
```

Arguments:

`...` Arguments passed to make_AUC

`data` The object to use for AUC calculation

Details: See make_AUC

`PM_op$clone()`: The objects of this class are cloneable with this method.

Usage:

```
PM_op$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Michael Neely, Julian Otalvaro

`PM_opt` *Optimal Sample Times*

Description

[Stable]

Contains optimal sampling times for a given model and dosage regimen.

Details

This object contains the methods to create and results from optimal sampling algorithms. Currently the only option multiple-model optimization. This algorithm calculates the requested number of sample times where the concentration time profiles are the most separated, thereby minimizing the risk of choosing the incorrect Bayesian posterior for an individual. Future updates will add D-optimal sampling times.

Public fields

`sampleTime` The optimal sample times, based on requested `type` argument to `$new` creation method.

`bayesRisk` Only present for MM-optimal sampling. The Bayesian risk of mis-classifying a subject based on the sample times. This is more useful for comparisons between sampling strategies, with minimization the goal.

`simdata` A PM_sim object with the simulated profiles

`mmInt` A list with the `mmInt` values, `NULL` if `mmInt` argument to `$new` is missing.

Methods

Public methods:

- `PM_opt$new()`
- `PM_opt$plot()`
- `PM_opt$print()`
- `PM_opt$clone()`

`PM_opt$new()`**: [Stable]**

Determine optimal sample times which are the most informative about the model parameters.

Usage:

```
PM_opt$new(
  poppar,
  model,
  data,
  nsamp = 1,
```

```
  weight = list(none = 1),
  predInt = 0.5,
  mmInt,
  algorithm = "mm",
  outeq = 1,
  ...
)
```

Arguments:

`poppar` There are several choices for the population parameters.

- A PM_sim object, in which case the simulated data will be used to calculate optimal sampling times. Here `model` and `data` are not required, as they will be extracted from the PM_sim object.
- A PM_result loaded with PM_load, in which case the `$final` field will be used, e.g. `run1 <- PM_load(1)` and `poppar = run1`.
- A PM_final object, typically as a field in a PM_result, e.g., `poppar = run1$final`.
- A data frame of support points in rows, where the columns contain the parameter values for each support point. The first row has the parameter names. Compatible `model` and `data` objects must be specified to simulate the output profiles from each support point.
- A list containing three items in this order, but of any name: vector of weights, vector of mean parameter values, and a covariance matrix. If only one distribution is to be specified the `weights` vector should be of length 1 and contain a 1. If multiple distributions are to be sampled, the `weights` vector should be of length equal to the number of distributions and its values should sum to 1, e.g. `c(0.25,0.05,0.7)`. The means matrix may be a vector for a single distribution, or a matrix with `length(weights)` rows and number of columns equal to the number of parameters. Compatible `model` and `data` objects must be specified to simulate the output profiles from the support points generated by sampling from the specified distributions.

`model` A PM_model object, required or optional depending on the `poppar` argument.

- When `poppar` is a PM_sim object, `model` is ignored.
- When `poppar` is a PM_result object, `model` is optional. If missing, the model from the `$model` field of the PM_result will be used. If specified, `model` must be compatible with the primary support point parameter names and values in `poppar`.
- When `poppar` is a PM_final object, `model` is mandatory and must be compatible with the primary support point parameter names and values in `poppar`.
- When `poppar` is a data frame of support points, `model` is mandatory and must be compatible with the parameter names and values in the data frame.
- When `poppar` is a list of weights, means, and covariance, `model` is mandatory and must be compatible with the parameter names and values.

`data` A PM_data object, required or optional depending on the `poppar` argument.

- When `poppar` is a PM_sim object, `data` is ignored.
- When `poppar` is a PM_result object, `data` is optional. If missing, the data from the `$data` field of the PM_result will be used. If specified, `data` must be compatible with the model that is either in the PM_result or as specified by `model`.
- When `poppar` is a PM_final object, `data` is mandatory and must be compatible with the model that is specified by `model`.

- When poppar is a data frame of support points, data is mandatory and must be compatible with the model that is specified by model.
- When poppar is a list of weights, means, and covariance, data is mandatory and must be compatible with the model that is specified by model.

For any choice of data, the value for outputs can be coded as any number(s) other than -99. The number(s) will be replaced in the simulator output with the simulated values. Good practice is to use -1 for out values to be simulated.

nsamp The number of MM-optimal sample times to compute; default is 1, but can be any number. Values >4 will take an exponentially longer time.

weight List whose names indicate the type of weighting, and values indicate the relative weight. Values should sum to 1. Names can be any of the following:

- **none** The default. MMopt times will be chosen to maximally discriminate all responses at all times.
- **AUC** MMopt times will be chosen to maximally discriminate AUC, regardless of the shape of the response profile.
- **max** MMopt times will be chosen to maximally discriminate maximum, regardless of the shape of the response profile.
- **min** MMopt times will be chosen to maximally discriminate minimum, regardless of the shape of the response profile.

Any combination of AUC, max, and min can be chosen. If "none" is specified, other weight types will be ignored and the relative value will be set to 1. For example,`list(auc = 0.5,max = 0.5)` or `list(auc = 0.2, min = 0.8)`. The default is `list(none = 1)`.

predInt The interval in fractional hours for simulated predicted outputs at times other than those specified in the template data. The default is 0.5, which means there will be simulated outputs every 30 minutes from time 0 up to the maximal time in the template file. You may also specify predInt as a vector of 3 values, e.g. `c(1,4,1)`, similar to the R command seq, where the first value is the start time, the second is the stop time, and the third is the step value. Outputs for times specified in the template file will also be simulated. To simulate outputs *only* at the output times in the template data (i.e. EVID=0 events), use `predInt = 0`. Note that the maximum number of predictions total is 594, so the interval must be sufficiently large to accommodate this for a given number of output equations and total time to simulate over. If predInt is set so that this cap is exceeded, predictions will be truncated.

mmInt Specify the time intervals from which MMopt times can be selected. These should only include simulated times specified by predInt.

algorithm Optimal sampling algorithm. Currently not modifiable and the only option is "mm".

outeq Output equation to optimize

... Other parameters to pass to PM_sim$new(). Most are not necessary, but `usePost = TRUE` can be used to calculate individual MMopt times. In this case, the number of posterior distributions contained in `poppar$final$postPoints` needs to match the number of subjects in data. You can also pass include and exclude to limit the subjects used in data. This will work whether usePost is TRUE or FALSE. Note that the following arguments to PM_sim$new cannot be modified.

- nsim is zero
- outname is "MMsim"
- combine is TRUE

`clean` Boolean parameter to specify whether temporary files made in the course of the simulation run should be deleted. Defaults to `TRUE`. This is primarily used for debugging.

Details: Currently, the only option is the multiple-model optimization algorithm.

`PM_opt$plot()`: Plot method

Usage:

```
PM_opt$plot(...)
```

Arguments:

`...` Arguments passed to plot.PM_opt

Details: See plot.PM_opt.

`PM_opt$print()`: Print method

Usage:

```
PM_opt$print()
```

Returns: Prints the optimal sampling times and Bayes Risk.

`PM_opt$clone()`: The objects of this class are cloneable with this method.

Usage:

```
PM_opt$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Michael Neely

References

Bayard, David S. & and Neely, Michael. (2017). Experiment Design for Nonparametric Models Based on Minimizing Bayes Risk: Application to Voriconazole. *Journal of Pharmacokinetics and Pharmacodynamics*, **44**(2): 95–111. https://doi.org/10.1007/s10928-016-9498-5.

PM_pop *Population predictions at short intervals*

Description

[Stable]

Contains the population predictions at short intervals specified as an argument to the `$fit` method of PM_model. Default is every 12 minutes.

Details

#' The PM_pop object is both a data field within a PM_result, and itself an R6 object comprising data fields and associated methods suitable for analysis and plotting of population predictions generated during the run.

Because PM_pop objects are automatically added to the PM_result at the end of a successful run, it is generally not necessary for users to generate PM_pop objects themselves.

The main results are contained in the `$data` field, and it is this field which is passed to the `$plot` and `$summary` methods. data frame with population predicted outputs for all subjects.

To provide a more traditional experience in R, the data frame is separated by columns into fields, e.g. `id` or `time`. This allows you to access them in an S3 way, e.g. `run1$pop$time` if `run1` is a `PM_result` object.

However, if you wish to manipulate the entire data frame, use the `data` field, e.g. `trough <- run1$pop$data |> filter(time == 24)`. If you are unfamiliar with the base R `|>` pipe function, type `help("|>")` into the R console for documentation and examples.

Public fields

`data` A data frame with the following columns:

- **id** Subject id
- **time** Time of predictions in decimal hours
- **icen** Prediction based on mean or median of Bayesian posterior parameter distribution
- **outeq** Output equation number
- **pred** Predicted output for each outeq
- **block** Observation blocks within subjects as defined by *EVID=4* dosing events

Methods

Public methods:

- `PM_pop$new()`
- `PM_pop$plot()`
- `PM_pop$summary()`
- `PM_pop$auc()`
- `PM_pop$clone()`

`PM_pop$new()`**:** Create new object populated with population predicted data at regular, frequent intervals

Usage:

```
PM_pop$new(PMdata = NULL, path = ".", ...)
```

Arguments:

`PMdata` include Saved, parsed output of prior run, used when source files are not available. .

`path` include Path to the folder containing the raw results of the run. Default is the current working directory. .

`...` Not currently used.

Details: Creation of new `PM_pop` object is automatic and not generally necessary for the user to do.

`PM_pop$plot()`: Plot method

Usage:

```
PM_pop$plot(...)
```

Arguments:

`...` Arguments passed to plot.PM_pop

Details: See plot.PM_pop.

`PM_pop$summary()`: Summary method

Usage:

```
PM_pop$summary(...)
```

Arguments:

`...` Arguments passed to summary.PM_pop

Details: See summary.PM_pop.

`PM_pop$auc()`: Calculate AUC

Usage:

```
PM_pop$auc(...)
```

Arguments:

`...` Arguments passed to make_AUC

`data` The object to use for AUC calculation

Details: See make_AUC

`PM_pop$clone()`: The objects of this class are cloneable with this method.

Usage:

```
PM_pop$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Michael Neely, Julian Otalvaro

PM_post *Individual Bayesian posterior predictions at short intervals*

Description

[Stable]

Contains the Bayesian posterior predictions at short intervals specified as an argument to the `$fit` method of PM_model. Default is every 12 minutes.

Details

#' The PM_post object is both a data field within a PM_result, and itself an R6 object comprising data fields and associated methods suitable for analysis and plotting of posterior predictions generated during the run.

Because PM_post objects are automatically added to the PM_result at the end of a successful run, it is generally not necessary for users to generate PM_post objects themselves.

The main results are contained in the `$data` field, and it is this field which is passed to the `$plot` and `$summary` methods. data frame with population predicted outputs for all subjects.

To provide a more traditional experience in R, the data frame is separated by columns into fields, e.g. `id` or `time`. This allows you to access them in an S3 way, e.g. `run1$post$time` if `run1` is a `PM_result` object.

However, if you wish to manipulate the entire data frame, use the `data` field, e.g. `trough <- run1$post$data |> filter(time == 24)`. If you are unfamiliar with the base R `|>` pipe function, type `help("|>")` into the R console for documentation and examples.

Public fields

`data` A data frame with the following columns:

- **id** Subject id
- **time** Time of predictions in decimal hours
- **icen** Prediction based on mean or median of Bayesian posterior parameter distribution
- **outeq** Output equation number
- **pred** Predicted output for each outeq
- **block** Observation blocks within subjects as defined by *EVID=4* dosing events

Methods

Public methods:

- `PM_post$new()`
- `PM_post$plot()`
- `PM_post$summary()`
- `PM_post$auc()`
- `PM_post$clone()`

`PM_post$new()`: Create new object populated with Bayesian posterior predicted data at regular, frequent intervals

Usage:

```
PM_post$new(PMdata = NULL, path = ".", ...)
```

Arguments:

`PMdata` include Saved, parsed output of prior run, used when source files are not available. .

`path` include Path to the folder containing the raw results of the run. Default is the current working directory. .

`...` Not currently used.

Details: Creation of new `PM_post` object is automatic and not generally necessary for the user to do.

`PM_post$plot()`: Plot method

Usage:

```
PM_post$plot(...)
```

Arguments:

`...` Arguments passed to plot.PM_pop

Details: See plot.PM_pop.

`PM_post$summary()`: Summary method

Usage:

```
PM_post$summary(...)
```

Arguments:

`...` Arguments passed to summary.PM_pop

Details: See summary.PM_post.

`PM_post$auc()`: Calculate AUC

Usage:

```
PM_post$auc(...)
```

Arguments:

`...` Arguments passed to make_AUC

`data` The object to use for AUC calculation

Details: See make_AUC

`PM_post$clone()`: The objects of this class are cloneable with this method.

Usage:

```
PM_post$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Michael Neely, Julian Otalvaro

PM_report *Generate a report*

Description

[Stable]

Generates a report for a completed run

Usage

```
PM_report(x, template, path, show = TRUE, quiet = TRUE)
```

Arguments

x	A PM_result object obtained from PM_load.
template	Report mode. One of "app", "plotly", "ggplot", or "none". If omitted, uses the report_template Pmetrics option. An invalid value falls back to that saved default.
path	Output directory for HTML reports.
show	Controls whether the report should be opened automatically, defaults to TRUE.
quiet	Suppress output from HTML report rendering.

Value

Invisibly returns 1 on success, 0 when reporting is disabled, and -1 when no report can be generated.

Author(s)

Markus Hovd, Julian Otalvaro, and Michael Neely

See Also

PM_load

PM_step *Stepwise covariate-parameter regressions*

Description

[Superseded]

This function is largely superseded as it is accessed through the $step methods for PM_result and PM_cov objects. There is rarely a need to call it directly any longer.

Usage

```
PM_step(x, icen = "median", direction = "both")
```

Arguments

x	A PMcov object which is the $data field of a PM_cov object
icen	A character vector to summarize covariate values. Default is "median", but can also be "mean".
direction	The direction for covariate elmination can be "backward", "forward", or "both" (default).

Details

It will perform stepwise linear regressions on a PM_cov object. Every covariate in the model will be tested in a stepwise linear regression for their relationships to each parameter in the model. Bayesian posterior parameters and individual covariates are used.

Value

A matrix with covariates in the rows and parameters in the columns. Values for the matrix are the multi-variate P-values. A value of NA indicates that the variable was not retained in the final model.

Author(s)

Michael Neely

See Also

stats::step()

PM_tree *Create a new Pmetrics folder tree*

Description

[Stable]

Sets up a directory tree for a new Pmetrics project

Usage

```
PM_tree(project = "NewProject", path = getwd())

PMtree(...)
```

Arguments

`project`	A character string of a new project name, e.g. "DrugX"
`path`	The full path to the root folder for the new project. Default is the current working directory.
`...`	Arguments passed to PM_tree.

Details

This function will create a new project folder tree with appropriate subfolders and a skeleton R script. You can edit the folder names and add new ones as you wish.

Value

A new folder with the name in `project` and the following subfolders:

- **Rscript** The folder containing a skeleton *Analysis.R* script for the project.
- **Runs** The folder for all Pmetrics runs, which will be sequentially numbered.
- **Sim** The folder for all simulations related to the project.
- **src** The folder for source data files in their original format, to preserve integrity and for audit purposes.

Author(s)

Michael Neely

See Also

PM_manual

Examples

```
## Not run:
PM_tree("DrugX")

## End(Not run)
```

`PM_tutorial` *Pmetrics tutorial*

Description

[Stable]

Introductory tutorial to Pmetrics

Usage

```
PM_tutorial()
```

Details

This function will create a *Learn* folder in the current working directory or one you specify. The folder will contain all files and subfolders necessary to conduct the tutorial found at https://lapkb.github.io/PM_tutorial/. After the Learn folder is created, open the *Rscript/Learn.R* script to begin the tutorial.

PM_valid *Pmetrics validation object*

Description

[Stable]

Contains results of internal validation by simulation to permit generation of visual predictive checks (VPCs), prediction corrected visual predictive checks, (pcVPCs), normalized prediction distribution errors (NPDE), and numerical predictive checks. This is typically a field in a PM_result

Details

The PM_valid object is both a data field within a PM_result, and itself an R6 object comprising data fields and associated methods suitable for analysis and plotting of observed vs. population or individual predicted outputs.

Because PM_valid objects are automatically added to the PM_result by calling the `$validate()` method of a PM_result after a successful run, it is generally not necessary for users to generate PM_valid objects themselves.

Public fields

`simdata` Simulated data created in the validation process

`timeBinMedian` Median times for cluster bins

`tadBinMedian` Median times after previous doses for cluster bins

`opDF` Observed-predicted data frame

`npde` Data for Normalized Prediction Distribution Error

`npde_tad` Data for Normalized Prediction Distribution Error using Time After Dose if available

`npde_stats` Captured NPDE console summaries for later replay

Methods

Public methods:

- `PM_valid$new()`
- `PM_valid$show_npde_stats()`
- `PM_valid$summary()`
- `PM_valid$plot()`
- `PM_valid$clone()`

`PM_valid$new()`: **[Stable]**

This function will create an object suitable for plotting visual predictive checks (VPCs) and prediction-corrected visual predictive checks (pcVPCs).

Usage:

```
PM_valid$new(result, tad = FALSE, binCov, doseC, timeC, tadC, limits, ...)
```

Arguments:

`result` The result of a prior run, usually supplied by calling the `$validate()` method of a PM_result at the end of a run, or later loaded with PM_load.

`tad` Boolean operator to use time after dose rather than time after start. Default is `FALSE`.

`binCov` A character vector of the names of covariates which are included in the model, i.e. in the model equations and which need to be binned. For example `binCov='wt'` if "wt" is included in a model equation like V=V0*wt, or `binCov=c( 'wt', 'crcl')` if both "wt" and "crcl" are included in model equations.

`doseC` An integer with the number of dose/covariate bins to cluster, if known from a previous run of this function. Including this value will skip the clustering portion for doses/covariates.

`timeC` An integer with the number of observation time bins to cluster, if known from a previous run of this function. Including this value will skip the clustering portion for observation times.

`tadC` An integer with the number of time after dose bins to cluster, if known from a previous run of this function. Including this value will skip the clustering portion for time after dose. This argument will be ignored if `tad=FALSE`.

`limits` Limits on simulated parameters. See PM_sim.

`...` Other parameters to be passed to PM_sim, especially `limits`.

Details: The function will guide the user through appropriate clustering of doses, covariates and sample times for prediction correction using the methods of Bergstrand et al (2011). *NOTE:* Including `tad` is only valid if steady state conditions exist for each patient. This means that dosing is stable and regular for each patient, without changes in amount or timing, and that sampling occurs after the average concentrations are the same from dose to dose. Otherwise observations are *NOT* superimposable and `tad` should *NOT* be used, i.e. should be set to `FALSE`.

Returns: An R6 object of class `PM_valid`, which is a list with the following.

- simdata The combined, simulated files for all subjects using the population mean values and each subject as a template. This object will be automatically saved to the run, to be loaded with PM_load next time.
- timeBinMedian A data frame with the median times for each cluster bin.
- tadBinMedian A data frame with the median time after dose (tad) for each cluster bin. This will be `NA` if `tad = FALSE`.
- opDF A data frame with observations, predicitons, and bin-corrected predictions for each subject.
- ndpe An object with results of normalized distrubition of prediction errors analysis.
- npde_tad NPDE with time after dose rather than absolute time, if `tad = TRUE`
- npde_stats Captured NPDE console summary text, replayable with `$show_npde_stats()`.

Examples:

```
valid <- NPex$validate(limits = c(0, 3))
```

`PM_valid$show_npde_stats()`: Replay captured NPDE summary statistics.

Usage:

```
PM_valid$show_npde_stats(outeq = 1, tad = FALSE)
```

Arguments:

`outeq` Output equation number to replay.

`tad` Logical; replay TAD-based NPDE stats.

`PM_valid$summary()`: Summarize a PM_valid object. Reports NPDE distribution statistics and a Numerical Predictive Check (NPC) for each output equation. TAD-based results are included automatically when TAD validation was performed.

Usage:

```
PM_valid$summary(probs = c(0.05, 0.5, 0.95))
```

Arguments:

`probs` Numeric vector of quantile cut-points for the NPC. Default is `c(0.05, 0.5, 0.95)`.

`PM_valid$plot()`: Plot method. Calls plot.PM_valid.

Usage:

```
PM_valid$plot(...)
```

Arguments:

`...` Arguments to pass to [plot.PM_valid].

`PM_valid$clone()`: The objects of this class are cloneable with this method.

Usage:

```
PM_valid$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Michael Neely

See Also

PM_result

Examples

```
## ------------------------------------------------
## Method `PM_valid$new()`
## ------------------------------------------------

## Not run:
valid <- NPex$validate(limits = c(0, 3))

## End(Not run)
```

PMcheck *Check Pmetrics Inputs for Errors*

Description

[Superseded]

This function is largely superseded as it is called automatically when data are initialized as a PM_data object. It can still be called independently of this route and will check for data errors.

Usage

```
PMcheck(data, path = ".", fix = FALSE, quiet = FALSE)
```

Arguments

data	The name of a Pmetrics .csv matrix file in the current working directory, the full path to one not in the current working directory, or a data.frame containing the output of a previous PMreadMatrix command.
path	The path of the data if originally a file
fix	Boolean operator; if TRUE, Pmetrics will attempt to fix errors in the data file. Default is FALSE.
quiet	Boolean operator to suppress printed output. Default is false.

Details

It will check the data for errors which would cause the analysis to fail. Note that as of Pmetrics Version 2, this function is called automatically when a new PM_data object is created, and users generally no longer need to call the function directly. In PM_data$new(), the data object is first standardized to contain all required columns, since only "ID", "TIME", "DOSE" and "OUT" are required at minimum, and then checked with PMcheck.

If calling PMcheck directly, either a filename or a Pmetrics data object in memory are accepted as data. Because there is no standardization with direct calls, in this case the format of the .csv matrix file is fairly rigid. It must have the following features. Text is case-sensitive.

- A header in row 1 with the appropriate version, currently "POPDATA DEC_11"
  - Column headers in row 2. These headers are: #ID, EVID, TIME, DUR, DOSE, ADDL, II, INPUT, OUT, CENS, OUTEQ, C0, C1, C2, C3.
- No cell should be empty. It should either contain a value or "." as a placeholder.
  - Columns after C3 are interpreted as covariates.
  - All subject records must begin with TIME=0.
- All dose events (EVID=1) must have entries in ID, EVID, TIME, DUR, DOSE and INPUT. ADDL and II are optional, but if ADDL is not 0 or missing, then II is mandatory.
- All observation events (EVID=0) must have entries in ID, EVID, TIME, OUT, OUTEQ. If an observation is missing, use -99; otherwise use a "." as a placeholder in cells that are not required (e.g. INPUT for an observation event).

- If covariates are present in the data, there must be an entry for every covariate at time 0 for each subject.
- All covariates must be numeric.
- All times within a subject ID must be monotonically increasing.
- All subject IDs must be contiguous.
- All rows must have EVID and TIME values.
- All columns must be numeric except ID which may be alpha-numeric.
- All subjects must have at least one observation, which could be missing, i.e. -99.
- Cells which are not needed (e.g. dose on an observation event, EVID=0), should contain ".".

To use this function, see the example below.

After running PMcheck and looking at the errors in the *errors.xlsx* file, you can fix the errors manually directly in the *errors.xlsx* file and resave it as a .csv file. Alternatively, you could then try to fix the problem(s) with `mdata2 <- PMcheck(mdata,fix=T)`. Note that we are now returning a PM_matrix data object called mdata2 (hopefully cleaned of errors) rather than the PMerr object returned when `fix=FALSE`. Pmetrics handles each of the errors in the following ways.

- If the columns are simply out of order, they will be reordered. If some are missing, the fix must be done by the user, i.e. manually.
- All id and covariate values are truncated to 11 characters.
- Missing observations are set to -99 (not ".").
- Incomplete dose records are flagged for the user to fix manually.
- Incomplete observation records are flagged for the user to fix manually.
- Subjects without an EVID=1 as first event are flagged for the user to fix manually.
- Subjects with TIME != 0 as first event have dummy dose=0 events inserted at time 0.
- Subjects with a missing covariate at time 0 are flagged for the user to fix manually.
- Non-numeric covariates are converted to numeric (via `factor()`).
- Non-ordered times are sorted within a subject if there are no EVID=4 events; otherwise the user must fix manually.
- Non-contiguous subject ID rows are combined and sorted if there are no EVID=4 events; otherwise the user must fix manually.
- Rows missing an EVID are assigned a value of 0 if DOSE is missing, 1 otherwise.
- Rows missing a TIME value are flagged for the user to fix manually.
- Cells with malformed NA values are attempted to be fixed.
- Columns that are non-numeric which must be numeric are flagged for the user to fix manually. These are all columns except ID. Covariate columns are fixed separately (see above).
- Dose events with censoring will be set to uncensored, with a warning to the user.

Value

If `fix=TRUE`, then PMcheck returns

- The original data if no errors are found, or
- A PMmatrix data object which has been cleaned of errors as much as possible, displaying a report on the console.

If `fix=FALSE`, then PMcheck creates a file in the working directory called "errors.xlsx". This file can be opened by Microsoft Excel or any other program that is capable of reading .xlsx files. This file contains highlighted areas that are erroneous, with clarifying comments. You can correct the errors in the file and then re-save as a .csv file.

When `fix=FALSE`, the function also returns a list of objects of class *PMerr*. Each object is itself a list whose first object (`$msg`) is a character vector with "OK" plus a brief description if there is no error, or the error. The second object (`$results`) is a vector of the row numbers that contain that error.

- colorder The first 14 columns must be named id, evid, time, dur, dose, addl, ii, input, out, outeq, c0, c1, c2, and c3 in that order.
- maxcharCol All column names should be less than or equal to 11 characters.
- maxcharID All id values should be less than or equal to 11 characters.
- missEVID Ensure that all rows have an EVID value.
- missTIME Ensure that all rows have a TIME value.
- doseDur Make sure all dose records are complete, i.e. contain a duration.
- doseDose Make sure all dose records are complete, i.e. contain a dose.
- doseInput Make sure all dose records are complete, i.e. contain an input identifier.
- obsOut Make sure all observation records are complete, i.e. contain an output.
- obsOuteq Make sure all observation records are complete, i.e. contain an outeq identifier.
- T0 Make sure each subject's first time=0.
- covT0 Make sure that there is an non-missing entry for each covariate at time=0 for each subject.
- timeOrder Ensure that all times within a subject ID are monotonically increasing.
- contigID Ensure that all subject IDs are contiguous.
- nonNum Ensure that all columns except ID, CENS, INPUT, and OUTEQ are numeric.
- noObs Ensure that all subjects have at least one observation, which could be missing, i.e. -99.
- mal_NA Ensure that all NA values are ".", not ". ", " .", "..", or other malformations.

Author(s)

Michael Neely and Patrick Nolain

See Also

PMwriteMatrix, PMreadMatrix

Examples

```
## Not run:
err <- PMcheck(badData)
# look at the errors.xlsx file in the working directory
# try to automatically fix what can be fixed
goodData <- PMcheck(badCSV, fix = T)
PMcheck(goodData)
# you have to fix manually problems which require data entry

## End(Not run)
```

PMmatrixRelTime *Convert Absolute Dates and Times to Relative Hours*

Description

[Superseded]

Convert dates/times to relative times. This function is largely superseded as it is called automatically when data are initialized as a PM_data object. There is rarely a need to call it directly any longer.

Usage

```
PMmatrixRelTime(
  data,
  idCol = "id",
  dateCol = "date",
  timeCol = "time",
  evidCol = "evid",
  format,
  split = F
)
```

Arguments

data	The name of an R data object.
idCol	A character vector with the name of the id column in data or the number of the id column, default is “id”
dateCol	A character vector with the name of the date column in data or the number of the date column, default is “date”
timeCol	A character vector with the name of the time column in data or the number of the time column, default is “time”
evidCol	A character vector with the name of the event id column in data or the number of the evid column, default is “evid”

format	Format of the date and time columns; default is m/d/y and h:m:s, as specified in the chron::chron function. Note the separators in each case (/ for dates and : for times). For dates, *m* is months in digits and can be one or two digits; *d* is the day of the month, again as one or two digits; *y* is the year in 2 or 4 digits. For times, all values can be one or two digits, but time is in 24-hour format, and *s* is required to avoid ambiguity.
split	If *true*, PMmatrixRelTime will split every id into id.block, where block is defined by a dose reset, or evid=4, e.g. id 1.1, 1.2, 1.3, 2.1, 3.1, 3.2.

Details

PMmatrixRelTime will convert absolute dates and times in a dataset into relative hours, suitable for Pmetrics analysis. Additionally, the user has the option to split subjects into pseudosubjects every time a dose reset (evid=4) is encountered.

Value

Returns a dataframe with columns *id, evid, relTime*. If split=T all evid values that were previously 4 will be converted to 1.

Author(s)

Michael Neely

See Also

PMreadMatrix

PMpatch *Download and install Pmetrics patches*

Description

Download and install Pmetrics patches from LAPK website

Usage

```
PMpatch()
```

Value

A Pmetrics patch which will be installed via source

Author(s)

Michael Neely

PMreadMatrix *Read a Pmetrics data file*

Description

[Superseded]

Reads a Pmetrics .csv matrix input file into R. This function is largely superseded as the function is called automatically when data are intialized as a PM_data object with `PM_data$new()`. There is rarely need to call `PMreadMatrix()` directly any longer.

Usage

```
PMreadMatrix(
  file,
  sep = getPMoptions("sep"),
  dec = getPMoptions("dec"),
  quiet = FALSE,
  ...
)
```

Arguments

`file`	The name of the file to be loaded, including the full path if not in the current working directory (check with getwd).
`sep`	Delimiter between columns, which is a comma by default, but can be changed with setPMoptions.
`dec`	Decimal separator, which is a period by default, but can be changed with setPMoptions.
`quiet`	Default is `FALSE`. If `TRUE`, there will be no report to the console on the contents of file.
`...`	Other parameters to be passed to `readr::read_delim()`

Details

As of Pmetrics version 2, the structure of a valid .csv file relaxed. Minimal required columns are id, time, dose, and out. This function is now included as part of the PM_data R6 object to create new `PM_data` objects. Users should rarely have a need to call `PMreadMatrix` as a standalone function unless they continue to use Pmetrics in its legacy mode (versions < 2.0). Note that support for legacy Pmetrics will eventually wither as the package evolves.

There are a number of other options for columns in the data input. Details can be found in the documentation.

Value

PMreadMatrix returns a data frame of class "PMmatrix". If the file is successfully read and quiet=F, the column headers will be reported to the console as a validation check. Note that this function converts the column headers in the file from upper to lowercase for convenient referencing in R.

Author(s)

Michael Neely

See Also

PMwriteMatrix, PMcheck, and plot.PM_data

PMwriteMatrix *Write a Pmetrics .csv Matrix File*

Description

[Superseded]

This function is largely superseded as the function is accessed with the $save() method for PM_data objects. There is rarely a need to call it directly. It is the companion function to PMreadMatrix. It will write an appropriate R data object to a formatted .csv file.

Usage

```
PMwriteMatrix(
  data,
  filename,
  override = FALSE,
  version = "DEC_11",
  header = FALSE
)
```

Arguments

data	Must be a data.frame with appropriate structure (see PMcheck).
filename	Name of file to create.
override	Boolean operator to write even if errors are detected. Default is FALSE.
version	Which matrix data format version to write. Default is the current version.
header	Is there a header row? Default is FALSE as this was the legacy format.

Details

PMwriteMatrix will first run PMcheck to determine if there are any errors in the structure of data. If the error check fails, the file will not be written and a message will be printed on the console.

Value

Returns the error report (see PMcheck for details).

Author(s)

Michael Neely

See Also

PM_data, PMcheck, PMreadMatrix

Examples

```
## Not run:
# write to the current directory
NPex$data$save("data.csv")

## End(Not run)
```

PMwrk2csv *Convert Old .wrk Files to .csv Matrix File*

Description

[Stable]

Convert old-style, USC*PACK single drug working copy files into a PM_data object and write a .csv file to the current working directory.

Usage

```
PMwrk2csv(prefix, ext = NULL, nsub)
```

Arguments

prefix	The alphabetic prefix of the working copy files to be converted, as a character vector.
ext	The extension of the working copy files files, if it exists. Does not have to be specified.
nsub	The number of subjects, or working copy files to read.

Details

This function will determine if the working copy files are old and convert them. New, multi-drug working copy files will be ignored. IDs will be suffixed with .1 to .9 for <10 subjects, .01 to .99 for <100 subjects and .001 to .999 for <1000 subjects, as needed to ensure unique ID numbers.

Value

A new file will be created with the name equal to `prefix` and an extension of "csv".

Author(s)

Michael Neely

---

print.PM_compare *Print method for PM_compare objects*

Description

`r lifecycle::badge("stable")` Prints a summary of the PM_compare object.

Usage

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

Arguments

x	A PM_compare object.
...	Additional arguments (not used).

---

print.PMerr *Print Data Errors*

Description

[Stable]

Print the errors in a Pmetrics data file or PM_data object.

Usage

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

Arguments

x	A PMerr object made by validation in PM_data.
...	Not used.

Value

A printed object.

Author(s)

Michael Neely

See Also

PM_data

`print.PMnpc` *Print NPC*

Description

[Stable]

Print a Pmetrics NPC (Numerical Predictive Check) object, made by plot.PM_sim.

Usage

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

Arguments

`x`	A PMnpc object made by plot.PM_sim.
`...`	Other parameters which are not necessary.

Value

A printed object.

Author(s)

Michael Neely

See Also

plot.PM_sim

`print.summary.PM_cycle` *Print Summary of Cycle Statistics*

Description

[Stable]

Print a Pmetrics Cycle Summary Object

Usage

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

Arguments

`x`	An object made by summary.PM_cycle.
`...`	Not used.

Details

Print a summary a summary.PM_cycle object made by summary.PM_cycle. Users do not normally need to call this function directly, as it will be the default method to display the object.

Value

A printed object.

Author(s)

Michael Neely

See Also

summary.PM_cycle

Examples

```
#'
## Not run:
NPex$cycle$summary()

## End(Not run)
```

`print.summary.PM_data` *Print Summary of Pmetrics Data*

Description

[Stable]

Usage

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

Arguments

`x`	An object made by summary.PM_data.
`...`	Not used.

Details

Print the summary of PM_data object.

Summarize the raw data used for a Pmetrics run.

Value

A printed object

Author(s)

Michael Neely

See Also

summary.PM_data

Examples

```
## Not run:
dataEx$summary()

## End(Not run)
```

print.summary.PM_final

Print Summary of Parameter Values and Credibility Intervals

Description

[Stable]

Print a Pmetrics Final Summary Object

Usage

```
## S3 method for class 'summary.PM_final'
print(x, digits = 3, ...)
```

Arguments

x	A summary.PM_final object made by summary.PM_final.
digits	Integer, used for number of digits to print.
...	Not used.

Details

Print a summary of parameter medians and MAWD, with point estimates and credibilty intervals from an object made by summary.PM_final. Users do not normally need to call this function directly, as it will be the default method to display the object.

Value

A printed object.

Author(s)

Michael Neely

See Also

summary.PM_final

Examples

```
## Not run:
NPex$final$summary

## End(Not run)
```

`print.summary.PM_op` *Print Summary of Observations and Predictions*

Description

[Stable]

Print a Pmetrics Observed vs. Predicted Summary Object

Usage

```
## S3 method for class 'summary.PM_op'
print(x, digits = getPMoptions("digits"), embed = FALSE, ...)
```

Arguments

`x`	An object made by summary.PM_op.
`digits`	Integer, used for number of digits to print.
`embed`	If `TRUE`, will embed the summary in the R Markdown document. Default is `FALSE`.
`...`	Not used.

Details

Print a summary of observations, predictions and errors in a summary.PM_op object made by summary.PM_op. Users do not normally need to call this function directly, as it will be the default method to display the object.

Value

A printed object.

Author(s)

Michael Neely

See Also

summary.PM_op

Examples

```
## Not run:
NPex$op$summary()

## End(Not run)
```

`print.summary.PM_sim` *Print Summary of Simulations*

Description

[Stable]

Print a Pmetrics Observed vs. Predicted Summary Object

Usage

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

Arguments

`x`	An object made by summary.PM_sim.
`...`	Not used.

Details

Print a summary of simulations made by summary.PM_sim. Users do not normally need to call this print function directly, as it will be the default method to display the object.

Value

A printed object.

Author(s)

Michael Neely

See Also

summary.PM_sim

Examples

```
## Not run:
simEx$summary()

## End(Not run)
```

proportional *Proportional error model*

Description

[Stable]

Create an proportional (gamma) error model

Usage

```
proportional(initial, coeff, fixed = FALSE, outeq = 1)
```

Arguments

initial	Initial value for gamma
coeff	Vector of coefficients defining assay error polynomial
fixed	Estimate if FALSE (default).
outeq	The output equation this error model applies to, given either as the number used in the model's out block (e.g. outeq = 2 for Y[2]) or as an output label declared in a DSL model. Defaults to 1.

qgrowth *Extract CDC pediatric growth charts*

Description

[Stable]

Will extract height, weight, and BMI for boys, girls or both for a given range of ages in months and percentiles. This can be useful for simulations in Pmetrics.

Usage

```
qgrowth(sex = "B", agemos = (seq(0, 18) * 12), percentile = 50)
```

Arguments

sex	A single quoted character: "M" for males, "F" for females, or "B" for both. Default is "B".
agemos	A vector of ages in months to return. The default is seq(0,18)*12, i.e. 0 to 216 months in increments of 12, which is 1 to 18 years. Values do not have to be multiples of 12.
percentile	An integer vector of the percentile for each age/sex to return. Default is 50, but could be, for example c(5, 10, 25, 50, 75, 90, 95).

Value

A dataframe with columns

- agemos Age in months
- ageyrs Age in years
- wt Weight in kilograms
- ht Height or length in centimeters
- bmi Body mass index kg/m^2
- sex The selected sex(es)
- percentile The selected percentile(s)

Author(s)

Michael Neely

---

rgba_to_rgb *Convert RGBA to RGB*

Description

[Stable] Converts a CSS `rgba()` string to an RGB color string.

Usage

```
rgba_to_rgb(rgba_str, alpha = NULL)
```

Arguments

`rgba_str`	A string in the format `rgba(r, g, b, a)` where r, g, and b are integers between 0 and 255, and a is a float between 0 and 1.
`alpha`	Optional numeric value to replace the alpha channel in the rgba string.

setPMoptions *Set Pmetrics User Options*

Description

[Stable]

Set user options for Pmetrics

Usage

```
setPMoptions(launch.app = TRUE)
```

Arguments

launch.app	Launch the app to set options. Default TRUE.

Details

When you call this function with the default launch.app = TRUE, it will start a Shiny app to set options for the Pmetrics package. Also, when the Pmetrics package is first loaded with library(Pmetrics), this function will be called with launch.app = TRUE to read saved options from a *PMoptions.json* file stored in a folder outside of the Pmetrics package, so that your options will persist when Pmetrics is updated.

Value

The user preferences file will be updated. This will persist from session to session and if stored in the external location, through Pmetrics versions.

Author(s)

Michael Neely

simEx *Example simulator output*

Description

Example simulator output

Usage

```
simEx
```

Format

R6 PM_sim

Details

This is an R6 Pmetrics PM_sim object created by created by running `$sim()` on a PM_result object, e.g. `NPex$sim(include = 1:4, limits = NA, nsim = 100)`.

Author(s)

Michael Neely

---

ss.PK *Sample size calculations for Phase 1 PK study design*

Description

[Stable]

This function calculates sample size based on a desired standard error of the mean, to a specified confidence, for a given mean and standard deviation.

Usage

```
ss.PK(n, mean, sd, precision, ci = 0.95)
```

Arguments

`n`	Sample size. This value can be missing if sample size is desired, or specified to calculate the maximum sd for given `mean`, `precision`, and 'ci.
`mean`	Mean prameter value. User value is mandatory.
`sd`	Standard deviation of parameter values. If present, the function will return `n`. If missing and `n`' is specified, will return the maximum sd as detailed above.
`precision`	Desired width of the standard error of the mean (SEM). Default is 0.2, i.e. 20% or 10% below and 10% above the mean. If missing, and `mean`, `sd` and `n` are specified, `precision` will be calculated.
`ci`	Confidence for the desired width of the SEM. Default is 0.95.

Details

The formula is `n = qnorm((1+ci)/2)**2 * sd**2 / (precision*mean)**2`

Value

The missing argument: `n`, `sd` or `precision`.

Author(s)

Michael Neely

`sub_plot` *Display multiple plotly plots*

Description

[Stable]

Wrapper around `plotly::subplot()`.

Usage

```
sub_plot(
  ...,
  nrows = 1,
  widths = NULL,
  heights = NULL,
  margin = 0.02,
  titles = NULL,
  shareX = FALSE,
  shareY = FALSE,
  titleX = shareX,
  titleY = shareY,
  which_layout = "merge",
  print = TRUE
)
```

Arguments

`...`	One of the following • any number of plotly objects • a list of plotly objects. Note that unlike `plotly::subplot()`, ggplots and tibbles cannot be passed to this function.
`nrows`	number of rows for laying out plots in a grid-like structure. Default is 1.
`widths, heights`	Vector of relative column widths or heights on a 0-1 scale. By default all columns have an equal relative width/height, i.e. `c(0.5, 0.5)` for two columns, `rep(0.25, 4)` for 4 columns.
`margin`	either a single value or a vector of four values (all between 0 and 1), e.g. `c(0.05, 0.05, 0.05, 0.1)` If four values are provided, the first is used as the left margin, the second is used as the right margin, the third is used as the top margin, and the fourth is used as the bottom margin. If a single value is provided, it will be used as all four margins.

titles	Include titles on individual subplots? Default is NULL. If specified as vector of length 2, will be rendered as x and y values relative to each plot. For example, title = c(0,1) plots the titles in the upper left corner of each subplot and title = c(1,0) renders the titles in the lower right corner. Title text and formatting will be grabbed from each individual plot. To modify these characteristics, modify the code that generated the individual plot.
shareX, shareY	Should the x- or y- axis be shared amongst the subplots?
titleX, titleY	Should x- or y- axis titles be retained?
which_layout	Adopt the layout of which plot? If the default value of "merge" is used, layout options found later in the sequence of plots will override options found earlier in the sequence. This argument also accepts a numeric vector specifying which plots to consider when merging.
print	If TRUE, will print the plotly object and return it. If FALSE, will only return the plotly object.

Details

This function addresses the deficiency with the native plotly method of combining multiple plots that prevents individualized titling of subplots. The function has identical arguments to plotly::subplot() with the addition of a titles argument. In addition to subplot, the behavior of this function is two-fold:

- Fetch the titles (text and formatting) from each included plot
- Include the titles with placement per the titles argument

Value

A plot and plotly object combining all the plots in ..., which can be further modified.

Author(s)

Michael Neely

See Also

plotly::subplot()

Examples

```
## Not run:
plot1 <- NPex$op$plot(title = "Posterior")
plot2 <- NPex$op$plot(pred.type = "pop", title = "Population")
sub_plot(plot1, plot2, titles = c(0, 0.95), nrows = 2)

## End(Not run)
```

summary.PM_cov *Summarize Covariates and Bayesian Posterior Parameter Values*

Description

[Stable]

Summarize a Pmetrics Covariate object

Usage

```
## S3 method for class 'PM_cov'
summary(object, icen = "median", ...)
```

Arguments

object	A PM_cov object
icen	Summary function for covariates with time dependent values and posterior parameters. Default is "median", but can specify "mean".
...	Not used.

Details

This is a function usually called by the `$summary()` method for PM_cov objects with a PM_result to summarize covariates and Bayesian posterior parameter values for each subject. The function can be called directly on a PM_cov object. See examples.Summarize .

Value

A data frame with the summary of the PM_cov object for each subject's covariates and Bayesian posterior parameter values.

Author(s)

Michael Neely

See Also

PM_cov

Examples

```
## Not run:
NPex$cov$summary() # preferred
summary(NPex$cov) # alternative
NPex$cov$summary(icen = "mean") # use mean as summary

## End(Not run)
```

`summary.PM_cycle` *Summarize Final Cycle Statistics*

Description

[Stable]

Summarize a Pmetrics Cycle object

Usage

```
## S3 method for class 'PM_cycle'
summary(object, cycle = NULL, digits = 3, ...)
```

Arguments

`object`	A PM_cycle object
`cycle`	Cycle number to summarize. Default is last cycle.
`digits`	Number of digits to round to. Default is 3.
`...`	Not used.

Details

This is a function usually called by the `$summary()` method for PM_cycle objects within a PM_result to summarize final cycle statistics. The function can be called directly on a PM_cycle object. See examples.

Value

A list of class *summary.PM_cycle* whose elements are the last cycle values for the following fields in a PM_cycle object.

- **cycle** Maximum cycle number
- **ll** Log likelihood
- **aic** Akaike Information Criterion
- **bic** Bayesian Information Criterion
- **gamlam** Value of gamma or lambda for each output equation
- **mean** Normalized mean parameter values compared to initial value
- **sd** Normalized standard deviation of parameter values compared to initial value
- **median** Normalized median parameter values compared to initial value

Author(s)

Michael Neely

See Also

PM_cycle

Examples

```
#'
## Not run:
NPex$cycle$summary() # preferred
summary(NPex$cycle) # alternative

## End(Not run)
```

`summary.PM_cycle_data` *Summarize PM_cycle_data objects*

Description

[Stable] Summarizes the raw data (`class: PM_cycle_data`) from a PM_cycle object in the same way as summarizing a PM_cycle object. Both use summary.PM_cycle.

Usage

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

Arguments

`object`	A `PM_cycle_data` object
`...`	Additional arguments passed to summary.PM_cycle

Examples

```
# There is no example we can think of to filter or otherwise process a PM_cycle object,
# but we provide this function for completeness.
NPex$cycle$data |> summary()
# all the below are the same
# summary(NPex$cycle$data)
# summary(NPex$cycle)
# NPex$cycle$summary()
```

`summary.PM_data` *Summarize PM_data objects*

Description

[Stable]

Summarize the raw data used for a Pmetrics run.

Usage

```
## S3 method for class 'PM_data'
summary(object, formula, FUN, include, exclude, ...)
```

Arguments

`object`	A PM_data object.
`formula`	Optional formula for specifying custom summaries. See aggregate and formula for details on how to specify formulae in R. If, for example, the data contain a covariate for weight named 'wt', then to summarize the mean dose in mg/kg per subject specify `formula = dose/wt ~ id` and `FUN = mean`.
`FUN`	The summary function to apply to formula, if specified. This is not quoted, and usual choices will be mean, median, max, or min.
`include`	A vector of subject IDs to include in the summary, e.g. `c(1:3,5,15)`
`exclude`	A vector of subject IDs to exclude in the summary, e.g. `c(4,6:14,16:20)`
`...`	Additional arguments to `FUN`, e.g. `na.rm = TRUE`

Value

A list of class *summary.PM_data* with the following items:

- **nsub** Number of subjects
- **ndrug** Number of drug inputs
- **numeqt** Number of outputs
- **nobsXouteq** Number of observations by outeq
- **missObsXouteq** Number of missing observations by outeq
- **loqObsXouteq** Number of observations coded as below the limit of quantification by outeq
- **ncov** Number of covariates
- **covnames** Covariate names
- **ndoseXid** Number of doses per input per subject
- **nobsXid** Number of observations per outeq per subject
- **doseXid** Doses per input per subject
- **obsXid** Observations per outeq per subject
- **formula** Results of including formula

Author(s)

Michael Neely

See Also

aggregate

`summary.PM_final` *Summary Statistics for Final Cycle*

Description

[Stable]

Generates summary statistics of final population model parameters.

Usage

```
## S3 method for class 'PM_final'
summary(object, lower = 0.025, upper = 0.975, file = NULL, ...)
```

Arguments

`object`	The PM_final object made after an NPAG or IT2B run
`lower`	Desired lower confidence interval boundary. Default is 0.025. Ignored for IT2B objects.
`upper`	Desired upper confidence interval boundary. Default is 0.975. Ignored for IT2B objects.
`file`	Filename to save the summary. Include path if necessary.
`...`	Not used.

Details

#' This is a function usually called by the `$summary()` method for PM_final objects within a PM_result. The function can be called directly on a PM_final object. For NPAG runs, this function will generate weighted medians as central tendencies of the population points with a 95% confidence interval (95% CI) around the median, and the median absolute weighted deviation (MAWD) from the median as a measure of the variance, with its 95% CI. These estimates correspond to weighted mean, 95% CI of the mean, variance, and 95% CI of the variance, respectively, for a sample from a normal distribution.

To estimate these non-parametric summaries, the function uses a Monte Carlo simulation approach, creating 1000 x npoint samples with replacement from the weighted marginal distribution of each parameter, where npoint is the number of support points in the model. As an example, if there are 100 support points, npoint = 100, and for Ka, there will be 1000 sets of 100 samples drawn from the weighted marginal distribution of the values for Ka. For each of the 1,000 sets of npoint values, the median and MAWD are calculated, with MAWD equal to the median absolute difference between

each point and the median of that set. The output is npoint estimates of the weighted median and npoint estimates of the MAWD for each parameter, from which the median, 2.5th, and 97.5th percentiles can be found as point estimates and 95% confidence interval limits, respectively, of both the weighted median and MAWD.

For IT2B runs, the function will return the mean and variance of each parameter, and the standard errors of these terms, using

$$SE_mean = SD/\sqrt{(nsub)}$$

$$SE_var = var * \sqrt{(2/(nsub-1))}$$

.

Value

The output is a data frame. For NPAG this has 4 columns:

- **value** The value of the summary statistic
- **par** The name of the parameter
- **type** Either *WtMed* for weighted median, or *MAWD* for MAWD (see details)
- **percentile** Requested `lower`, 0.5 (median), and `upper` quantiles For IT2B this has 6 columns:
- **mean** Parameter mean value
- **se.mean** Standard error of the mean
- **cv.mean** Error of the mean divided by mean
- **var** Variance of the parameter values
- **se.var** Standard error of the variance
- **summary** Name of the summary statistic

Author(s)

Michael Neely

See Also

PM_final

Examples

```
## Not run:
NPex$final$summary() # preferred
ITex$final$summary()
summary(NPex$final) # alternate

## End(Not run)
```

summary.PM_final_data *Summarize PM_final_data objects*

Description

[Stable] Summarizes the raw data (`class: PM_final_data`) from a PM_final object in the same way as summarizing a PM_final object. Both use summary.PM_final.

Usage

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

Arguments

`object`	A `PM_final_data` object
`...`	Additional arguments passed to summary.PM_final

Examples

```
NPex$final$data |> summary()
```

summary.PM_op *Summarize Observations and Predictions*

Description

[Stable]

Summarize a Pmetrics Observed vs. Predicted object

Usage

```
## S3 method for class 'PM_op'
summary(
  object,
  digits = max(3, getOption("digits") - 3),
  pred.type = "post",
  icen = "median",
  outeq = 1,
  ...
)
```

Arguments

`object`	A PM_op object
`digits`	Integer, used for number of digits to print.
`pred.type`	Either 'post' for a posterior object or 'pop' for a population object. Default is 'post'.
`icen`	Can be either "median" for the predictions based on medians of `pred.type` parameter value distributions, or "mean". Default is "median".
`outeq`	Output equation number. Default is 1.
`...`	Not used.

Details

This is a function usually called by the `$summary()` method for PM_op objects within a PM_result to summarize observations, predictions and errors. The function can be called directly on a PM_op object. See examples.

Value

A list with three elements of class *summary.PM_op*.

- sumstat A data frame with the minimum, first quartile, median, third quartile, maximum, mean and standard deviation for times, observations and predictions in `x`.
- pe A named vector with mean prediction error (mpe), the mean weighted prediction error (mwpe), the percent mean weighted prediction error (percent_mwpe), the mean squared prediction error (mspe), root mean sqaured error (rmse), percent root mean squared error (percent_rmse), the mean weighted squared prediction error (mwspe), the bias-adjusted mean squared prediction error (bamspe), the bias- adjusted mean weighted squared prediction error (bamwspe), the percent root mean bias- adjusted weighted squared prediction error (percent_rmbawspe). The percent_mwpe is bias and the percent_rmbawspe is imprecision on plots of PM_op objects.
- wtd.t A list of 6 elements based on a t test that the weighted mean prediction bias is different than zero
- estimate: the weighted mean of the prediction bias for each observation
- se: the standard error of the estimate
- conf.int: the 95% confidence interval of the mean
- statistic: the t statistic of the standardized difference between mean and zero
- df: degrees of freedom equal to number of observations minus one
- p.value: the probability that the weighted mean is different than zero

Author(s)

Michael Neely

See Also

PM_op

Examples

```
## Not run:
NPex$op$summary() # preferred
summary(NPex$op) # alternative

## End(Not run)
```

---

summary.PM_op_data *Summarize PM_op_data objects*

Description

[Stable] Summarizes the raw data (`class: PM_op_data`) from a PM_op object in the same way as summarizing a PM_op object. Both use summary.PM_op.

Usage

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

Arguments

`object`	A `PM_op_data` object
`...`	Additional arguments passed to summary.PM_op

Examples

```
## Not run:
NPex$op$data |>
  dplyr::filter(pred > 5) |>
  dplyr::filter(pred < 10) |>
  summary()

## End(Not run)
```

---

summary.PM_pop *Summarize Observations and Predictions*

Description

[Stable]

Summarize a Pmetrics Observed vs. Predicted object

Usage

```
## S3 method for class 'PM_pop'
summary(
  object,
  digits = max(3, getOption("digits") - 3),
  icen = "median",
  outeq = 1,
  ...
)
```

Arguments

object	A PM_pop object
digits	Integer, used for number of digits to print.
icen	Can be either "median" for the predictions based on medians of the population parameter value distributions, or "mean". Default is "median".
outeq	Output equation number. Default is 1.
...	Not used.

Details

This is a function usually called by the `$summary()` method for PM_op objects within a PM_result to summarize observations, predictions and errors. The function can be called directly on a PM_op object. See examples.

Value

A data frame with the minimum, first quartile, median, third quartile, maximum, mean and standard deviation for times and predictions in `x`.

Author(s)

Michael Neely

See Also

PM_pop

Examples

```
## Not run:
NPex$pop$summary() # preferred
summary(NPex$pop) # alternative

## End(Not run)
```

summary.PM_post *Summarize Observations and Predictions*

Description

[Stable]

Summarize a Pmetrics Observed vs. Predicted object

Usage

```
## S3 method for class 'PM_post'
summary(
  object,
  digits = max(3, getOption("digits") - 3),
  icen = "median",
  outeq = 1,
  ...
)
```

Arguments

object	A PM_post object
digits	Integer, used for number of digits to print.
icen	Can be either "median" for the predictions based on medians of the posterior parameter value distributions, or "mean". Default is "median".
outeq	Output equation number. Default is 1.
...	Not used.

Details

This is a function usually called by the `$summary()` method for PM_op objects within a PM_result to summarize observations, predictions and errors. The function can be called directly on a PM_op object. See examples.

Value

A data frame with the minimum, first quartile, median, third quartile, maximum, mean and standard deviation for times and predictions in `x`.

Author(s)

Michael Neely

See Also

PM_post

Examples

```
## Not run:
NPex$post$summary() # preferred
summary(NPex$post) # alternative

## End(Not run)
```

`summary.PM_pta` *Summarize Percent Target Attainment*

Description

[Stable]

Summarize a Pmetrics Percent Target Attainment Object

Usage

```
## S3 method for class 'PM_pta'
summary(object, at = "intersect", ci = 0.95, ...)
```

Arguments

`object`	A PM_pta object
`at`	Which object in the PM_pta result list to summarize. By default "intersect" if an intersection is present due to creation of the object with multiple target types, or 1 if no intersection is present, which means only 1 target type was selected. If "intersect" is present in the object, the default can be overridden with a number to summarize one of the individual PTAs, e.g. `at = 2` to summarize the second PTA rather than the intersection of all the PTAs.
`ci`	Width of the interval for pharmacodynamic index reporting. Default is 0.95, i.e. 2.5th to 97.5th percentile.
`...`	Not used.

Details

Summarize Pharmacodynamic Index (PDI) statistics and success proportions in a PM_pta object. The PDI is the metric calculated by the target type and target, e.g. AUC/Target, or %time>target. Since a PDI cannot be calculated for intersections, summarizing the intersection object only provides the success proportion per simulation/target.

Value

A tibble with the following columns (only the first five if `at = "intersect"`):

- **reg_num** is the number of the simulation regimen
- **label** is the simulation label, for reference

- **target** is the target for the row, if targets are discrete, not used for simulated targets
- **type** is the target type for the row, e.g. "auc", "time", "-min", etc.
- **prop_success** is the proportion of simulated profiles that met the success definition
- **median** is the median parmacodynamic index (PDI), i.e. the proportion or ratio depending on the target type
- **lower** is the lower bound of the interval defined by `ci`
- **upper** is the upper bound of the interval defined by `ci`
- **mean** is the mean of the PDI
- **sd** is the standard deviation of the PDI
- **min** is the minimum PDI
- **max** is the maximum PDI

Author(s)

Michael Neely

See Also

PM_pta

Examples

```
## Not run:
ptaEx$summary()

## End(Not run)
```

`summary.PM_sim` *Summarize Pmetrics Simulation Objects*

Description

[Stable]

Summarize simulation

Usage

```
## S3 method for class 'PM_sim'
summary(
  object,
  include,
  exclude,
  field = "obs",
  group = NULL,
  statistics = c("mean", "sd", "median", "min", "max"),
  digits = getPMoptions("digits"),
  ...
)
```

Arguments

`object`	The simulation object to summarize
`include`	A vector of subject IDs to include in the plot, e.g. c(1:3,5,15) .
`exclude`	A vector of subject IDs to exclude in the plot, e.g. c(4,6:14,16:20) .
`field`	Quoted character value, one of • **obs** for simulated observations (the default) • **amt** for simulated amounts in each compartment • **parValues** for simulated parameter values
`group`	Optional quoted values to group by, e.g. `group = "outeq"` or `group = c("id", "outeq")`.
`statistics`	The summary statistics to report. Default is `c("mean", "sd", "median", "min", "max")`, but can be any subset of these and can also include specific quantiles, e.g. `c(25, "median", 75)`.
`digits`	Integer, used for number of digits to print. Default is value set with `setPMoptions()`.
`...`	Not used.

Details

Summarizes simulated observations, compartment amounts or parameter values. Can be ungrouped (i.e. the entire simulation), average values grouped by simulated id, outeq, or both, as well as summary statistics for each individual. Default statistics reported are mean, sd, median, min, max, but could include individual quantiles. See `statistics` below.

Value

If `group` is omitted, a data frame with rows for each data element except ID, and columns labeled according to the selected `statistics`. If `group` is specified, return will be a list with named elements according to the selected `statistics`, each containing a data frame with the summaries for each group in `group`.

Author(s)

Michael Neely

See Also

PM_sim

Examples

```
## Not run:
simEx$summary() # preferred
summary(simEx) # alternative
simEx$summary(include = 2, field = "amt", group = "comp") # group amounts by compartment

## End(Not run)
```

summary.PM_valid *Summary Method for PM_valid Objects*

Description

[Stable] Prints NPDE distribution statistics and a Numerical Predictive Check (NPC) for each output equation in the PM_valid object. TAD-based results are included automatically when TAD validation was performed.

Usage

```
## S3 method for class 'PM_valid'
summary(object, probs = c(0.05, 0.5, 0.95), ...)
```

Arguments

`object`	A PM_valid object.
`probs`	Numeric vector of quantile cut-points for the NPC. Default is `c(0.05, 0.5, 0.95)`.
`...`	Ignored; present for S3 compatibility.

three_comp_bolus *Three-compartment bolus model template*

Description

Create a `PM_model` object for the `three_comp_bolus` model template.

Usage

```
three_comp_bolus()
```

Value

A `PM_model` object.

three_comp_bolus_cl *Three-compartment bolus clearance model template*

Description

Create a PM_model object for the three_comp_bolus_cl model template.

Usage

```
three_comp_bolus_cl()
```

Value

A PM_model object.

three_comp_iv *Three-compartment IV model template*

Description

Create a PM_model object for the three_comp_iv model template.

Usage

```
three_comp_iv()
```

Value

A PM_model object.

three_comp_iv_cl *Three-compartment IV clearance model template*

Description

Create a PM_model object for the three_comp_iv_cl model template.

Usage

```
three_comp_iv_cl()
```

Value

A PM_model object.

two_comp_bolus *Two-compartment bolus model template*

Description

Create a PM_model object for the two_comp_bolus model template.

Usage

```
two_comp_bolus()
```

Value

A PM_model object.

two_comp_bolus_cl *Two-compartment bolus clearance model template*

Description

Create a PM_model object for the two_comp_bolus_cl model template.

Usage

```
two_comp_bolus_cl()
```

Value

A PM_model object.

two_comp_iv *Two-compartment IV model template*

Description

Create a PM_model object for the two_comp_iv model template.

Usage

```
two_comp_iv()
```

Value

A PM_model object.

two_comp_iv_cl *Two-compartment IV clearance model template*

Description

Create a PM_model object for the two_comp_iv_cl model template.

Usage

```
two_comp_iv_cl()
```

Value

A PM_model object.

zBMI *Extract CDC pediatric BMI z-scores*

Description

[Stable]

Will extract BMI z-scores based on a single age in months and sex. Overweight is a z-score of >1.04, and obese is a z-score > 1.64. Calculations are based on CDC formulae

For a z-score > 3, which indicates an extreme BMI, consider using the modified z-score and percentile.

Usage

```
zBMI(agemos, sex, bmi, wt, ht, data = "CDC")
```

Arguments

agemos	The age in months. Should be between 24 and 240.5.
sex	A single quoted character: "M" for males, "F" for females. Default is "M".
bmi	The individual's BMI. If specified, wt and ht are not necessary and will be ignored.
wt	The individual's weight in kg as an alternative to specifying BMI. Will be ignored if BMI is specified.
ht	The individual's height in centimeters. Required if wt is specified and BMI is not. Ignored if BMI is specified.
data	Source data for calculations. Default is "CDC" which uses the cdc_bmi dataset. The alternative is "NHANES", which uses the ger_bmi dataset.

Value

A list with objects calculated for `agemos` and `sex`.

- z Z-score
- mod_z Modified Z-score for extreme BMI
- per BMI percentile
- mod_per Modified BMI percentile

Author(s)

Michael Neely

Examples

```
## Not run:
zBMI(agemos = 36, bmi = 15)

## End(Not run)
```

Index

∗ **PMplots**
  plot.PM_cov, 40
  plot.PM_cycle, 44
  plot.PM_data, 48
  plot.PM_final, 54
  plot.PM_model, 59
  plot.PM_op, 61
  plot.PM_opt, 67
  plot.PM_pop, 68
  plot.PM_post, 72
  plot.PM_pta, 76
  plot.PM_sim, 81
  plot.PM_valid, 86
∗ **datasets**
  badData, 14
  cdc_bmi, 14
  dataEx, 17
  ger_bmi, 19
  growth, 24
  locales, 25
  mic1, 32
  modEx, 33
  NPex, 37
  simEx, 155

ab, 5
ab(), *113, 114*
ab_line, 5, *9*
abline, *5*
add_renal, 6
add_shapes, *6*, 9, *11*
add_smooth, 10
additive, 12
additive(), *117*
aggregate, *162, 163*
all_is_numeric, 12
apps, 13

badData, 14
base::file.show(), *18*

cdc_bmi, 14, *20*, *176*
check_updates, 15
click_plot, 15
cor2cov, 16

dataEx, 17
density, *56*
downloadR, 17

export_plotly, 18

factor(), *139*
formula, *162*

ger_bmi, *15*, 19, *176*
getCov, 20
getDefaultColors, 21
getFixedColNum, 22
getPalettes, 22
getPMoptions, 23, *119*
getwd, *143*
ggplot2::ggplot(), *61*
ggplot2::theme_minimal(), *92*
growth, 24

interp, 24
interp(), *114*

latestR, 25
latestR(), *17*
lit_sim *(*apps*)*, 13
locales, 25
lubridate::parse_date_time(), *100, 101*, *103*

make_AUC, 26, *100*, *102*, *123*, *129*, *131*
make_NCA, 28, *29*, *100*, *102*
make_valid, *82*, *90–92*
makeErrorPoly, 30
makePTAtarget, 31
makePTAtarget(), *32*

max, *162*
mean, *162*
median, *162*
mic1, 32
min, *162*
model_lib, 33
model_lib(), *114*
modEx, 33
msd, 34
msd(), *113, 114*
mtsknn.eq, 34

NM2PM, 35
npde::plot.NpdeObject, *87*
NPex, 37
NPrun, *119*

one_comp_bolus, 38
one_comp_bolus_cl, 38
one_comp_iv, 38
one_comp_iv(), *33*
one_comp_iv_cl, 39
opposite_color, 39, *63*, *65*

par, *31*, *78*, *92*
plot, *92*
plot.default, *31*
plot.PM_cov, 40, *97*
plot.PM_cov(), *47*, *53*, *58*, *61*, *66, 67*, *71*, *76*, *80*, *86*, *90*
plot.PM_cycle, 44, *47, 48*, *99*
plot.PM_cycle(), *44*, *53*, *58*, *61*, *66, 67*, *71*, *76*, *80*, *86*, *90*
plot.PM_cycle_data, 47
plot.PM_data, 48, *54*, *102*, *144*
plot.PM_data(), *44*, *47*, *58*, *61*, *66, 67*, *71*, *76*, *80*, *86*, *90*
plot.PM_data_data, 53
plot.PM_final, 54, *59*, *107*
plot.PM_final(), *44*, *47*, *53*, *61*, *66, 67*, *71*, *76*, *80*, *86*, *90*
plot.PM_final_data, 59
plot.PM_model, 59
plot.PM_model(), *44*, *47*, *53*, *58*, *66, 67*, *71*, *76*, *80*, *86*, *90*, *117*
plot.PM_op, *40*, 61, *66*, *123*
plot.PM_op(), *44*, *47*, *53*, *58*, *61*, *67*, *71*, *76*, *80*, *86*, *90*
plot.PM_op_data, 66
plot.PM_opt, 67, *127*
plot.PM_opt(), *44*, *47*, *53*, *58*, *61*, *66*, *71*, *76*, *80*, *86*, *90*
plot.PM_pop, 68, *129*, *131*
plot.PM_pop(), *44*, *47*, *53*, *58*, *61*, *66, 67*, *76*, *80*, *86*, *90*
plot.PM_post, 72
plot.PM_post(), *44*, *47*, *53*, *58*, *61*, *66, 67*, *71*, *80*, *86*, *90*
plot.PM_pta, 76
plot.PM_pta(), *44*, *47*, *53*, *58*, *61*, *66, 67*, *71*, *76*, *86*, *90*
plot.PM_sim, *67*, 81, *147*
plot.PM_sim(), *44*, *47*, *53*, *58*, *61*, *66, 67*, *71*, *76*, *80*, *90*
plot.PM_valid, 86, *91*, *137*
plot.PM_valid(), *44*, *47*, *53*, *58*, *61*, *66, 67*, *71*, *76*, *80*, *86*
plot.PMvalid, 91
plot_ly, *77, 78*, *86*
plotly::ggplotly(), *93*
plotly::layout(), *9*
plotly::save_image(), *18, 19*
plotly::subplot(), *157, 158*
plotlygg, 93
PM_build, 93
PM_compare, 94
PM_cov, *40*, *44*, 95, *96, 97*, *110*, *132, 133*, *159*
PM_cycle, *45*, *47*, 98, *98*, *110*, *160, 161*
PM_data, *6, 7*, *14*, *17*, *20*, *53, 54*, 100, *100, 101*, *118–120*, *125*, *138*, *141*, *143–147*, *149*, *162*
PM_data(), *111*
PM_Final, *56*
PM_final, *54, 55*, *58, 59*, 104, *104*, *110*, *125*, *163–165*
PM_help, 108
PM_load, *28*, *40*, *44*, *62*, *83*, *95*, 109, *118*, *125*, *132*, *136*
PM_load(), *37*
PM_manual, 110, *134*
PM_manual(), *113*
PM_model, *34*, *59–61*, *100*, 111, *125*, *127*, *130*
PM_op, *26, 27*, *62*, *65, 66*, *85*, *110*, 121, *122*, *166–169*
PM_opt, *67*, 124
pm_plot *(*apps*)*, 13
PM_pop, *26*, *49*, *68*, *71*, *100*, *110*, 127, *128*, *168*

PM_post, *26*, *49*, *72, 73*, *75, 76*, *100*, *110*, 130, *130*, *169*
PM_pta, *31, 32*, *78*, *80*, *170, 171*
PM_report, 132
PM_result, *27, 28*, *37*, *40*, *44*, *47*, *49*, *53–55*, *58*, *62*, *65, 66*, *68*, *71*, *73*, *75, 76*, *83*, *86, 87*, *94*, *96*, *98*, *100*, *104*, *109, 110*, *118*, *122*, *125*, *128*, *130*, *132*, *135–137*, *156*, *159, 160*, *163*, *166*, *168, 169*
PM_sim, *26, 27*, *31*, *80, 81*, *86*, *124–126*, *136*, *156*, *172*
PM_step, *97*, 132
PM_tree, 133, *134*
PM_tutorial, 134
PM_valid, *86*, 135, *135*, *137*, *173*
PMcheck, *36, 37*, *100*, 138, *140*, *144, 145*
PMmatrixRelTime, 141
PMpatch, 142
PMreadMatrix, *138*, *140*, *142*, 143, *144, 145*
PMtree *(*PM_tree*)*, 133
PMwriteMatrix, *37*, *101*, *140*, 144, *144*
PMwrk2csv, *37*, 145
points, *92*
print, *97*
print.data.frame, *102*
print.PM_compare, 146
print.PMerr, 146
print.PMnpc, 147
print.summary.PM_cycle, 148
print.summary.PM_data, 149
print.summary.PM_final, 150
print.summary.PM_op, 151
print.summary.PM_sim, 152
proportional, 153
proportional(), *117*

qgrowth, 153
qgrowth(), *24*

read.csv(), *32*
readr::read_csv(), *32*
readr::read_delim(), *143*
rgba_to_rgb, 154

schema, *44*, *47*, *58*, *66*, *86*
seq, *126*
setPMoptions, *11*, *23*, *95*, *143*, 155
simEx, 155
ss.PK, 156
stats::lm(), *114*
stats::step(), *133*
sub_plot, 157
summary.PM_cov, *97*, 159
summary.PM_cycle, *99*, *148*, 160, *161*
summary.PM_cycle_data, 161
summary.PM_data, *102*, *149*, 162
summary.PM_final, *108*, *150*, 163, *165*
summary.PM_final_data, 165
summary.PM_op, *123*, *151*, 165, *167*
summary.PM_op_data, 167
summary.PM_pop, *129*, *131*, 167
summary.PM_post, *131*, 169
summary.PM_pta, 170
summary.PM_sim, *152*, 171
summary.PM_valid, 173

three_comp_bolus, 173
three_comp_bolus_cl, 174
three_comp_iv, 174
three_comp_iv_cl, 174
two_comp_bolus, 175
two_comp_bolus_cl, 175
two_comp_iv, 175
two_comp_iv_cl, 176

zBMI, 176