Package 'invivoPKfit'

March 24, 2025

Type Package

Title Fits Toxicokinetic Models to In Vivo PK Data Sets

Version 2.0.1

Date 2025-3-24

Author John Wambaugh [aut, cre] (<https://orcid.org/0000-0002-4024-534X>),
Caroline Ring [aut] (<https://orcid.org/0000-0002-0463-1251>),
Gilberto Padilla Mercado [aut]
(<https://orcid.org/0000-0001-5423-1646>),
Chris Cook [aut]

Maintainer John Wambaugh <wambaugh.john@epa.gov>

Description Takes in vivo toxicokinetic concentration-time data and fits
parameters of 1-compartment and 2-compartment models for each
chemical. These methods are described in detail in ``Informatics for Toxicokinetics'' (submitted).

License GPL-3

Depends R (>= 4.1.0)

Imports cli, devtools, dplyr, ggplot2, glue, httk, magrittr, MASS,
Matrix, multidplyr (>= 0.1.3), numDeriv, optimx, PK, pracma,
purrr, rlang, scales, tibble, tidyr, tidyselect

Suggests cowplot, knitr, RColorBrewer, rmarkdown, stringr, testthat
(>= 3.0.0), tidyverse

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

LazyData TRUE

NeedsCompilation no

RoxygenNote 7.3.2

Repository CRAN

Date/Publication 2025-03-24 16:00:02 UTC

Contents

`+.pk` *Add a 'pkproto' object to a 'pk' object*

Description

Add a 'pkproto' object to a 'pk' object

Usage

```
## S3 method for class 'pk'
e1 + e2
```

Arguments

e1	A ‘pk‘ pbject
e2	A ‘pkproto‘ object

Details

Note that ‘e1 + e2‘ is equivalent to
“‘ ‘+‘(e1, e2) “‘

Value

The ‘pk‘ object, modified by adding the ‘pkproto‘ object

Author(s)

Caroline Ring

-.pk *Subtract a pkproto object from a pk object*

Description

Subtract a pkproto object from a pk object

Usage

```
## S3 method for class 'pk'
e1 - e2
```

Arguments

e1	A pk pbject
e2	A pkproto object

Value

The pk object, modified by adding the pkproto object

Author(s)

Caroline Ring

AAFE *AAFE()*

Description

This is the S3 method generic for AAFE()

Usage

```
AAFE(obj, ...)
```

Arguments

obj	An object.
...	Additional arguments currently not in use.

Value

A dataframe with one row for each ‘data_group‘, ‘model‘ and ‘method‘. The final column contains the AAFE of the model fitted by the corresponding method, using the data in ‘newdata‘.

See Also

[AAFE.pk()] for the method for class [pk()]

AAFE.default *Default method for AAFE()*

Description

Default method for AAFE()

Usage

```
## Default S3 method:
AAFE(obj, ...)
```

Arguments

obj	An object
...	Additional arguments currently not in use.

Value

An error, when a non-pk object is used for the first argument.

AAFE.pk *Calculate absolute average fold error (AAFE)*

Description

Calculate aboslute average fold error (AAFE)

Usage

```
## S3 method for class 'pk'
AAFE(
  obj,
  newdata = NULL,
  model = NULL,
  method = NULL,
  exclude = TRUE,
  use_scale_conc = FALSE,
  AAFE_group = NULL,
  sub_pLOQ = TRUE,
  ...
)
```

Arguments

obj	A ‘pk‘ object
newdata	Optional: A ‘data.frame‘ with new data for which to make predictions and compute AAFE. If NULL (the default), then AAFE will be computed for the data in ‘obj$data‘. ‘newdata‘ is required to contain at least the following variables: ‘Time‘, ‘Time.Units‘, ‘Dose‘, ‘Route‘, ‘Media‘, ‘Conc‘, ‘Conc_SD‘, ‘N_Subjects‘, ‘Detect‘, ‘pLOQ‘.
model	Optional: Specify one or more of the fitted models for which to make predictions and calculate AAFE. If NULL (the default), AAFE will be returned for all of the models in ‘obj$stat_model‘.
method	Optional: Specify one or more of the [optimx::optimx()] methods for which to make predictions and calculate AAFE. If NULL (the default), RMSEs will be returned for all of the models in ‘obj$optimx_settings$method‘.
exclude	Logical: ‘TRUE‘ to compute the AAFE excluding any observations in the data marked for exclusion (if there is a variable ‘exclude‘ in the data, an observation is marked for exclusion when ‘exclude ‘FALSE‘ to include all observations, regardless of exclusion status. Default ‘TRUE‘.
use_scale_conc	Possible values: ‘TRUE‘, ‘FALSE‘, or a named list with elements ‘dose_norm‘ and ‘log10_trans‘ which themselves should be either ‘TRUE‘ or ‘FALSE‘. If ‘use_scale_conc = TRUE‘, then the concentration scaling/transformations in ‘object‘ will be applied to both predicted and observed concentrations before the log-likelihood is computed. If ‘use_scale_conc = FALSE‘ (the default for

this function), then no concentration scaling or transformation will be applied before the log-likelihood is computed. If 'use_scale_conc = list(dose_norm = ..., log10_trans = ...)', then the specified dose normalization and/or log10-transformation will be applied.

`AAFE_group` Default: Chemical, Species. Determines what the data grouping that is used to calculate absolute average fold error (AAFE). Should be set to lowest number of variables that still would return unique experimental conditions. Input in the form of 'ggplot2::vars(Chemical, Species, Route, Media, Dose)'.

`sub_pLOQ` TRUE (default): Substitute all predictions below the LOQ with the LOQ before computing AAFE. FALSE: do not.

... Additional arguments. Not currently in use.

Details

Absolute average fold error (AAFE) is calculated as

$$10^{\frac{1}{N}\sum \text{abs}\left[log_{10}\left(\frac{\text{predicted}}{\text{observed}}\right)\right]}$$

Left-censored data

If the observed value is censored, and the predicted value is less than the reported LOQ, then the observed value is (temporarily) set equal to the predicted value, for an effective error of zero.

If the predicted value is less than the reported LOQ, then the user may choose whether to (temporarily) set the predicted value equal to LOQ, using argument 'sub_pLOQ').

Value

A dataframe with one row for each 'data_group', 'model' and 'method'. The final column contains the AAFE of the model fitted by the corresponding method, using the data in 'newdata'.

Author(s)

Caroline Ring

See Also

Other fit evaluation metrics: `AFE.pk()`, `AIC.pk()`, `BIC.pk()`, `logLik.pk()`, `rmse.pk()`, `rsq.pk()`

Other methods for fitted pk objects: `AFE.pk()`, `AIC.pk()`, `BIC.pk()`, `coef.pk()`, `coef_sd.pk()`, `eval_tkstats.pk()`, `get_fit.pk()`, `get_hessian.pk()`, `get_tkstats.pk()`, `logLik.pk()`, `predict.pk()`, `residuals.pk()`, `rmse.pk()`, `rsq.pk()`

add_pk *Add various 'pkproto' objects to a 'pk' object*

Description

Add various 'pkproto' objects to a 'pk' object

Usage

```
add_pk(pk_obj, pkproto_obj, objectname)
```

Arguments

pk_obj	The 'pk' object
pkproto_obj	The 'pkproto' object to be added
objectname	The name of the 'pkproto' object to be added

Value

The 'pk' object modified by the addition.

AFE *AFE()*

Description

This is the S3 method generic for AFE()

Usage

```
AFE(obj, ...)
```

Arguments

obj	An object.
...	Additional arguments currently not in use.

Value

A dataframe with one row for each 'data_group', 'model' and 'method'. The final column contains the AFE of the model fitted by the corresponding method, using the data in 'newdata'.

See Also

[AFE.pk()] for the method for class [pk()]

AFE.default *Default method for AFE()*

Description

Default method for AFE()

Usage

```
## Default S3 method:
AFE(obj, ...)
```

Arguments

obj	An object
...	Additional arguments currently not in use.

Value

An error, when a non-pk object is used for the first argument.

AFE.pk *Calculate average fold error*

Description

Calculate average fold error

Usage

```
## S3 method for class 'pk'
AFE(
  obj,
  newdata = NULL,
  model = NULL,
  method = NULL,
  exclude = TRUE,
  use_scale_conc = FALSE,
  AFE_group = NULL,
  sub_pLOQ = TRUE,
  ...
)
```

Arguments

obj	A ‘pk‘ object
newdata	Optional: A ‘data.frame‘ with new data for which to make predictions and compute AFE. If NULL (the default), then AFE will be computed for the data in ‘obj$data‘. ‘newdata‘ is required to contain at least the following variables: ‘Time‘, ‘Time.Units‘, ‘Dose‘, ‘Route‘, ‘Media‘, ‘Conc‘, ‘Conc_SD‘, ‘N_Subjects‘, ‘Detect‘, ‘pLOQ‘.
model	Optional: Specify one or more of the fitted models for which to make predictions and calculate AFE. If NULL (the default), AFE will be returned for all of the models in ‘obj$stat_model‘.
method	Optional: Specify one or more of the [optimx::optimx()] methods for which to make predictions and calculate AFE. If NULL (the default), RMSEs will be returned for all of the models in ‘obj$optimx_settings$method‘.
exclude	Logical: ‘TRUE‘ to compute the AFE excluding any observations in the data marked for exclusion (if there is a variable ‘exclude‘ in the data, an observation is marked for exclusion when ‘exclude ‘FALSE‘ to include all observations, regardless of exclusion status. Default ‘TRUE‘.
use_scale_conc	Possible values: ‘TRUE‘, ‘FALSE‘, or a named list with elements ‘dose_norm‘ and ‘log10_trans‘ which themselves should be either ‘TRUE‘ or ‘FALSE‘. If ‘use_scale_conc = TRUE‘, then the concentration scaling/transformations in ‘object‘ will be applied to both predicted and observed concentrations before the log-likelihood is computed. If ‘use_scale_conc = FALSE‘ (the default for this function), then no concentration scaling or transformation will be applied before the log-likelihood is computed. If ‘use_scale_conc = list(dose_norm = ..., log10_trans = ...)‘, then the specified dose normalization and/or log10-transformation will be applied.
AFE_group	Default: Chemical, Species. Determines what the data grouping that is used to calculate average fold error (AFE). Should be set to lowest number of variables that still would return unique experimental conditions. Input in the form of ‘ggplot2::vars(Chemical, Species, Route, Media, Dose)‘.
sub_pLOQ	TRUE (default): Substitute all predictions below the LOQ with the LOQ before computing AFE. FALSE: do not.
...	Additional arguments. Not currently in use.

Details

Average fold error is calculated as

$$10^{\frac{1}{N}\sum log_{10}\left(\frac{\text{predicted}}{\text{observed}}\right)}$$

Left-censored data

If the observed value is censored, and the predicted value is less than the reported LOQ, then the observed value is (temporarily) set equal to the predicted value, for an effective error of zero.

If the predicted value is less than the reported LOQ, then the user may choose whether to (temporarily) set the predicted value equal to LOQ, using argument ‘sub_pLOQ‘).

Value

A dataframe with one row for each ‘data_group‘, ‘model‘ and ‘method‘. The final column contains the AFE of the model fitted by the corresponding method, using the data in ‘newdata‘.

Author(s)

Caroline Ring

See Also

Other fit evaluation metrics: `AAFE.pk()`, `AIC.pk()`, `BIC.pk()`, `logLik.pk()`, `rmse.pk()`, `rsq.pk()`

Other methods for fitted pk objects: `AAFE.pk()`, `AIC.pk()`, `BIC.pk()`, `coef.pk()`, `coef_sd.pk()`, `eval_tkstats.pk()`, `get_fit.pk()`, `get_hessian.pk()`, `get_tkstats.pk()`, `logLik.pk()`, `predict.pk()`, `residuals.pk()`, `rmse.pk()`, `rsq.pk()`

AIC.pk *Akaike information criterion*

Description

Get the Akaike information criterion (AIC) for a fitted ‘pk‘ object

Usage

```
## S3 method for class 'pk'
AIC(
  object,
  newdata = NULL,
  model = NULL,
  method = NULL,
  exclude = TRUE,
  drop_obs = TRUE,
  ...,
  k = 2
)
```

Arguments

object	A ‘pk‘ object
newdata	Optional: A ‘data.frame‘ with new data for which to compute log-likelihood. If NULL (the default), then log-likelihoods will be computed for the data in ‘object$data‘. ‘newdata‘ is required to contain at least the following variables: ‘Time‘, ‘Time.Units‘, ‘Dose‘, ‘Route‘,‘Media‘, ‘Conc‘, ‘Detect‘, ‘N_Subjects‘. Before log-likelihood is calculated, ‘Time‘ will be transformed according to the transformation in ‘object$scales$time‘ and ‘Conc‘ will be transformed according to the transformation in ‘object$scales$conc‘.

model	Optional: Specify one or more of the fitted models for which to calculate log-likelihood. If NULL (the default), log-likelihoods will be returned for all of the models in 'object$stat_model'.
method	Optional: Specify one or more of the [optimx::optimx()] methods for which to make predictions and calculate AICs. If NULL (the default), log-likelihoods will be returned for all of the models in 'object$settings_optimx$method'.
exclude	Logical: 'TRUE' to compute the AIC after removing any observations in the data marked for exclusion (if there is a variable 'exclude' in the data, an observation is marked for exclusion when 'exclude status. Default 'TRUE'.
drop_obs	Logical: 'TRUE' to drop the observations column in the output of [logLik()].
...	Additional argument. Not in use.
k	Default 2. The 'k' parameter in the log-likelihood formula (see Details). Must be named if used.

Details

The AIC is calculated from the log-likelihood (LL) as follows:

$$\text{AIC} = -2\text{LL} + kn_{par}$$

where n_{par} is the number of parameters in the fitted model, and $k = 2$ for the standard AIC.

Value

A data.frame with log-likelihood values and calculated AIC using 'newdata'. There is one row for each model in 'obj''s [stat_model()] element and each [optimx::optimx()] method (specified in [settings_optimx()]).

Author(s)

Caroline Ring, Gilberto Padilla Mercado

See Also

Other fit evaluation metrics: AAFE.pk(), AFE.pk(), BIC.pk(), logLik.pk(), rmse.pk(), rsq.pk()

Other log likelihood functions: BIC.pk(), logLik.pk()

Other methods for fitted pk objects: AAFE.pk(), AFE.pk(), BIC.pk(), coef.pk(), coef_sd.pk(), eval_tkstats.pk(), get_fit.pk(), get_hessian.pk(), get_tkstats.pk(), logLik.pk(), predict.pk(), residuals.pk(), rmse.pk(), rsq.pk()

auc_1comp *Analytic AUC for 1-compartment model*

Description

Calculate area under the plasma concentration vs. time curve for the 1-compartment model, using an analytical equation (the integral of the 1-compartment model equation with respect to time).

Usage

```
auc_1comp(params, time, dose, route, medium)
```

Arguments

params	A named numeric vector of model parameter values. See Details for requirements.
time	A numeric vector of times, reflecting the time point when concentration is measured after the corresponding single bolus dose. Must be same length as ‘dose‘ and ‘iv.dose‘, or length 1.
dose	A numeric vector of doses, reflecting single bolus doses administered at time 0. Must be same length as ‘time‘ and ‘iv.dose‘, or length 1.
route	A character vector, reflecting the route of administration of each single bolus dose: ‘’oral’‘ or ‘’iv’‘. Must be same length as ‘time‘ and ‘dose‘, or length 1.
medium	A character vector reflecting the medium in which each resulting concentration is to be calculated: "blood" or "plasma". Default is "plasma". Must be same length as ‘time‘ and ‘dose‘, or length 1.

Details

Required parameters

‘params‘ must include the following named items:

kelim Elimination rate, 1/time.

Vdist Apparent volume of central compartment, volume/unit BW. Or see below for ‘Fgutabs_Vdist‘

For oral administration (if any ‘route include:

Fgutabs Oral bioavailability, unitless fraction. Or see below for ‘Fgutabs_Vdist‘

kgutabs Rate of absorption from gut, 1/time.

For oral administration, in lieu of ‘Vdist‘ and ‘Fgutabs‘, you may instead provide ‘Fgutabs_Vdist‘, the ratio of Fgutabs to Vdist (1/volume). This is an alternate parameterization for situations where ‘Fgutabs‘ and ‘Vdist‘ are not identifiable separately (i.e., when oral TK data are available, but IV data are not). If ‘Fgutabs‘ and ‘Vdist‘ are provided, they will override any value provided for ‘Fgutabs_Vdist‘.

If both oral and IV administration are specified (i.e., some 'route and some 'route 'Fgutabs' or 'Fgutabs_Vdist'. (If 'Vdist' and 'Fgutabs_Vdist' are provided, but 'Fgutabs' is not provided, then 'Fgutabs' will be calculated from 'Vdist' and 'Fgutabs_Vdist'.)

If 'any(medium 'Rblood2plasma', the ratio of chemical concentration in whole blood to the chemical concentration in blood plasma.

Value

A vector of plasma AUC values (concentration*time) corresponding to 'time'.

Author(s)

Caroline Ring, John Wambaugh

See Also

Other built-in model functions: auc_1comp_cl(), auc_2comp(), auc_flat(), cp_1comp(), cp_1comp_cl(), cp_2comp(), cp_2comp_dt(), cp_flat(), get_params_1comp(), get_params_1comp_cl(), get_params_1comp_fup(), get_params_2comp(), get_params_flat(), get_starts_1comp(), get_starts_1comp_cl(), get_starts_1comp_fup(), get_starts_2comp(), get_starts_flat(), tkstats_2comp(), transformed_params_2comp()

Other 1-compartment model functions: auc_1comp_cl(), cp_1comp(), cp_1comp_cl(), get_params_1comp(), get_params_1comp_cl(), get_params_1comp_fup(), get_starts_1comp(), get_starts_1comp_cl(), get_starts_1comp_fup()

Other model AUC functions: auc_1comp_cl(), auc_2comp(), auc_flat()

---

auc_1comp_cl *Analytic AUC for 1-compartment model with specific clearance*

Description

Calculate area under the plasma concentration vs. time curve for the 1-compartment model, using an analytical equation (the integral of the 1-compartment model equation with respect to time).

Usage

```
auc_1comp_cl(params, time, dose, route, medium)
```

Arguments

params	A named numeric vector of model parameter values. See Details for requirements.
time	A numeric vector of times, reflecting the time point when concentration is measured after the corresponding single bolus dose. Must be same length as 'dose' and 'iv.dose', or length 1.
dose	A numeric vector of doses, reflecting single bolus doses administered at time 0. Must be same length as 'time' and 'iv.dose', or length 1.

route — A character vector, reflecting the route of administration of each single bolus dose: ‘’oral’‘ or ‘’iv’‘. Must be same length as ‘time‘ and ‘dose‘, or length 1.

medium — A character vector reflecting the medium in which each resulting concentration is to be calculated: "blood" or "plasma". Default is "plasma". Must be same length as ‘time‘ and ‘dose‘, or length 1.

Details

Required parameters

‘params‘ must include the following named items:

Fup Fraction of compound unbound in plasma. Unitless.

Clint Intrinsic clearance by hepatocytes. Units: 1/hr

Q_totli Total blood flow through the liver. Units: L/h/kg body weight ^ (3/4)

Q_gfr Glomerular filtration rate, how quickly do kidneys filter. Units: L/h/kg body weight ^ (3/4)

Vdist Apparent volume of central compartment, volume/unit BW. Or see below for ‘Fgutabs_Vdist‘

For oral administration (if any ‘route include:

Fgutabs Oral bioavailability, unitless fraction. Or see below for ‘Fgutabs_Vdist‘

kgutabs Rate of absorption from gut, 1/time.

For oral administration, in lieu of ‘Vdist‘ and ‘Fgutabs‘, you may instead provide ‘Fgutabs_Vdist‘, the ratio of Fgutabs to Vdist (1/volume). This is an alternate parameterization for situations where ‘Fgutabs‘ and ‘Vdist‘ are not identifiable separately (i.e., when oral TK data are available, but IV data are not). If ‘Fgutabs‘ and ‘Vdist‘ are provided, they will override any value provided for ‘Fgutabs_Vdist‘.

If both oral and IV administration are specified (i.e., some ‘route and some ‘route ‘Fgutabs‘ or ‘Fgutabs_Vdist‘. (If ‘Vdist‘ and ‘Fgutabs_Vdist‘ are provided, but ‘Fgutabs‘ is not provided, then ‘Fgutabs‘ will be calculated from ‘Vdist‘ and ‘Fgutabs_Vdist‘.)

If ‘any(medium ‘Rblood2plasma‘, the ratio of chemical concentration in whole blood to the chemical concentration in blood plasma.

Value

A vector of plasma AUC values (concentration*time) corresponding to ‘time‘.

Author(s)

Caroline Ring, John Wambaugh

See Also

Other built-in model functions: auc_1comp(), auc_2comp(), auc_flat(), cp_1comp(), cp_1comp_cl(), cp_2comp(), cp_2comp_dt(), cp_flat(), get_params_1comp(), get_params_1comp_cl(), get_params_1comp_fup(), get_params_2comp(), get_params_flat(), get_starts_1comp(), get_starts_1comp_cl(), get_starts_1comp_fup(), get_starts_2comp(), get_starts_flat(), tkstats_2comp(), transformed_params_2comp()

Other 1-compartment model functions: auc_1comp(), cp_1comp(), cp_1comp_cl(), get_params_1comp(), get_params_1comp_cl(), get_params_1comp_fup(), get_starts_1comp(), get_starts_1comp_cl(), get_starts_1comp_fup()

Other model AUC functions: auc_1comp(), auc_2comp(), auc_flat()

---

auc_2comp *Analytical AUC for the 2-compartment model*

Description

Calculate area under the plasma concentration vs. time curve for the 1-compartment model, using an analytical equation (the integral of the 1-compartment model equation with respect to time).

Usage

```
auc_2comp(params, time, dose, route, medium = "plasma")
```

Arguments

params	A named list of parameter values including the following: • k12: Rate at which compound moves from central to peripheral compartment, 1/h. • k21: Rate at which compound moves from peripheral to central compartment, 1/h. • kelim: Elimination rate, 1/h. • V1: Apparent volume of central compartment, L/kg BW. Or see below for "Fgutabs_V1" For oral administration (route FALSE), params must also include: • Fgutabs: Oral bioavailability, unitless fraction. Or see below for "Fgutabs_V1" • kgutabs: Rate of absorption from gut, 1/h. For oral administration, in lieu of "V1" and "Fgutabs", you may instead provide "Fgutabs_V1", the ratio of Fgutabs to V1 (1/L). This is an alternate parameterization for situations where "Fgutabs" and "V1" are not identifiable separately (i.e. when oral data are available, but IV data are not). If "Fgutabs" and "V1" are provided, then "Fgutabs_V1" will not be used.
time	A numeric vector of time values, in hours
dose	A numeric vector of doses in mg/kg
route	A logical vector: TRUE for single IV bolus dose, FALSE for single oral dose
medium	A character string that determines the measured media. Default: "plasma".

Value

A vector of plasma AUC values, evaluated at each time point in time.

Author(s)

Caroline Ring, John Wambaugh

See Also

Other built-in model functions: auc_1comp(), auc_1comp_cl(), auc_flat(), cp_1comp(), cp_1comp_cl(), cp_2comp(), cp_2comp_dt(), cp_flat(), get_params_1comp(), get_params_1comp_cl(), get_params_1comp_fup(), get_params_2comp(), get_params_flat(), get_starts_1comp(), get_starts_1comp_cl(), get_starts_1comp_fup(), get_starts_2comp(), get_starts_flat(), tkstats_2comp(), transformed_params_2comp()

Other 2-compartment model functions: cp_2comp(), cp_2comp_dt(), get_params_2comp(), get_starts_2comp(), tkstats_2comp(), transformed_params_2comp()

Other model AUC functions: auc_1comp(), auc_1comp_cl(), auc_flat()

---

auc_flat *AUC for flat model*

Description

Evaluates the area under the concentration-time curve for a "flat" model

Usage

```
auc_flat(time, params, dose, route, medium)
```

Arguments

time	A numeric vector of times in hours.
params	A named list of model parameter values. See Details for requirements.
dose	A numeric vector of doses in mg/kg
route	A logical vector: TRUE for single IV bolus dose; FALSE for single oral dose. Not used, but must be present for compatibility with other model functions.
medium	A character vector reflecting the medium in which each resulting concentration is to be calculated: "blood" or "plasma". Default is "plasma". Must be same length as other arguments, or length 1.

Details

Required parameters

‘params‘ must include the following named items:

Vdist Apparent volume of central compartment, L/kg BW. Or see below for ‘Fgutabs_Vdist‘

For oral administration (if any ‘route include:

Fgutabs Oral bioavailability, unitless fraction. Or see below for ‘Fgutabs_Vdist‘

For oral administration, in lieu of 'Vdist' and 'Fgutabs', you may instead provide 'Fgutabs_Vdist', the ratio of Fgutabs to Vdist (1/L). This is an alternate parameterization for situations where 'Fgutabs' and 'Vdist' are not identifiable separately (i.e., when oral TK data are available, but IV data are not). If 'Fgutabs' and 'Vdist' are provided, they will override any value provided for 'Fgutabs_Vdist'.

If both oral and IV administration are specified (i.e., some 'route and some 'route 'Fgutabs' or 'Fgutabs_Vdist'. (If 'Vdist' and 'Fgutabs_Vdist' are provided, but 'Fgutabs' is not provided, then 'Fgutabs' will be calculated from 'Vdist' and 'Fgutabs_Vdist'.)

If 'any(medium 'Rblood2plasma', the ratio of chemical concentration in whole blood to the chemical concentration in blood plasma.

Value

A vector of plasma concentration values (mg/L) corresponding to `time`.

Author(s)

Caroline Ring, John Wambaugh, Chris Cook

See Also

Other built-in model functions: `auc_1comp()`, `auc_1comp_cl()`, `auc_2comp()`, `cp_1comp()`, `cp_1comp_cl()`, `cp_2comp()`, `cp_2comp_dt()`, `cp_flat()`, `get_params_1comp()`, `get_params_1comp_cl()`, `get_params_1comp_fup()`, `get_params_2comp()`, `get_params_flat()`, `get_starts_1comp()`, `get_starts_1comp_cl()`, `get_starts_1comp_fup()`, `get_starts_2comp()`, `get_starts_flat()`, `tkstats_2comp()`, `transformed_params_2comp()`

Other flat model functions: `cp_flat()`, `get_params_flat()`, `get_starts_flat()`

Other model AUC functions: `auc_1comp()`, `auc_1comp_cl()`, `auc_2comp()`

---

auto_units *Automatically select new time units*

Description

Given a vector of time values in original units, this function selects new time units such that, when time is rescaled to the new units, the midpoint of the time vector is as close to 10 as possible.

Usage

```
auto_units(y, from, target = 10, period_units = time_units)
```

Arguments

`y`	A numeric vector of time values
`from`	The original units of 'y'
`target`	The target value (order of magnitude) for the midpoint of rescaled time values. Default 10.
`period_units`	A list of acceptable/understood time units. See Details. Default 'time_units'.

Details

Acceptable/understood time units in ‘period_units‘

“‘ c("picoseconds", "nanoseconds", "microseconds", "milliseconds", "seconds", "minutes", "hours", "days", "weeks", "months", "years") “‘

Value

Character: Automatically-selected new time units, which will be one of ‘period_units‘.

Author(s)

Caroline Ring

BIC.pk *Bayesian information criterion*

Description

Get the Bayesian information criterion (BIC) for a fitted ‘pk‘ object

Usage

```
## S3 method for class 'pk'
BIC(object, newdata = NULL, model = NULL, method = NULL, exclude = TRUE, ...)
```

Arguments

object	A ‘pk‘ object
newdata	Optional: A ‘data.frame‘ with new data for which to compute log-likelihood. If NULL (the default), then BICs will be computed for the data in ‘obj$data‘. ‘newdata‘ is required to contain at least the following variables: ‘Time‘, ‘Time.Units‘, ‘Dose‘, ‘Route‘,‘Media‘, ‘Conc‘, ‘Detect‘, ‘N_Subjects‘. Before log-likelihood is calculated, ‘Time‘ will be transformed according to the transformation in ‘obj$scales$time‘ and ‘Conc‘ will be transformed according to the transformation in ‘obj$scales$conc‘.
model	Optional: Specify one or more of the fitted models for which to calculate BIC. If NULL (the default), log-likelihoods will be returned for all of the models in ‘obj$stat_model‘.
method	Optional: Specify one or more of the [optimx::optimx()] methods for which to calculate BICs. If NULL (the default), log-likelihoods will be returned for all of the methods in ‘obj$settings_optimx$method‘.
exclude	Logical: ‘TRUE‘ to compute the AIC after removing any observations in the data marked for exclusion (if there is a variable ‘exclude‘ in the data, an observation is marked for exclusion when ‘exclude status. Default ‘TRUE‘.
...	Additional arguments. Not in use.

Details

The BIC is calculated from the log-likelihood (LL) as follows:

$$\text{BIC} = -2\text{LL} + \log(n_{obs})n_{par}$$

where n_{par} is the number of parameters in the fitted model.

Note that the BIC is just the AIC with $k = \log(n_{obs})$.

Value

A data.frame with log-likelihood values and calculated BIC using 'newdata'. There is one row for each model in 'obj''s [stat_model()] element and each [optimx::optimx()] method (specified in [settings_optimx()]).

Author(s)

Caroline Ring, Gilberto Padilla Mercado

See Also

Other fit evaluation metrics: `AAFE.pk()`, `AFE.pk()`, `AIC.pk()`, `logLik.pk()`, `rmse.pk()`, `rsq.pk()`

Other log likelihood functions: `AIC.pk()`, `logLik.pk()`

Other methods for fitted pk objects: `AAFE.pk()`, `AFE.pk()`, `AIC.pk()`, `coef.pk()`, `coef_sd.pk()`, `eval_tkstats.pk()`, `get_fit.pk()`, `get_hessian.pk()`, `get_tkstats.pk()`, `logLik.pk()`, `predict.pk()`, `residuals.pk()`, `rmse.pk()`, `rsq.pk()`

---

calc_hessian *Calculate Hessian*

Description

Calculate Hessian matrix given parameter values and data

Usage

```
calc_hessian(
  pars_opt,
  pars_const,
  observations,
  modelfun,
  dose_norm,
  log10_trans
)
```

Arguments

pars_opt	Named numeric: A vector of parameter values for the parameters that were optimized. For example, you can get this using [coef.pk()] with ‘include_type = "optim"‘.
pars_const	Named numeric: A vector of parameter values for parameters that were held constant, not optimized (but are necessary to evaluate the model). For example, you can get this using [coef.pk()] with ‘include_type = "const"‘.
observations	The data used to fit the model. For example, you can get this using [get_data.pk()].
modelfun	The name of the function that evaluates the model (passed to [log_likelihood()]).
dose_norm	Logical: Whether to dose-normalize concentrations before evaluating log-likelihood. Passed to [log_likelihood()].
log10_trans	Logical: Whether to log10-transform concentrations before evaluating log-likelihood. Passed to [log_likelihood()].

Details

Calculate the Hessian matrix: the matrix of second derivatives of the objective function with respect to parameters, evaluated for a single set of parameter values for a single model and a single data set. Here, the objective function is the negative log-likelihood implemented in [log_likelihood()], evaluated jointly across the data that was used to fit the model. This is a workhorse function called by [get_hessian.pk()] and, indirectly, by [coef_sd.pk()]. When the number of optimized parameters is n, the respective Hessian matrix will be $n \times n$.

Value

A square numeric matrix, both dimensions the same as the length of ‘pars_opt‘. It will have row-names and column names that are the same as the names of ‘pars_opt‘.

Author(s)

Caroline Ring

---

calc_nca *Non-compartmental analysis*

Description

Do non-compartmental analysis on a single-dose set of concentration vs. time data

Usage

```
calc_nca(time, conc, detect, series_id = NULL, dose, route, method = "z", ...)
```

Arguments

time	A numeric vector of time points.
conc	A numeric vector of concentrations. If detected (above limit of detection/quantification), contains the measured value; if not detected (below LOD/LOQ), contains the LOD/LOQ.
detect	A logical vector: Whether each concentration was detected (above LOD/LOQ) or not.
series_id	Optional: A variable that can be coerced to a factor, identifying individual time series (e.g., individual replicates – individual subjects, or replicate dose groups). Default NULL, in which case each observation will be assumed to have a different series ID. In other words, a serial sampling design will be assumed, in which each observation is from a different subject.
dose	A numeric scalar: The dose for this data set.
route	A character scalar: The route of administration for this data set. Currently, only "oral" and "iv" are supported.
method	As for [PK::nca()]: the method to use for calculation of confidence intervals. Default ‘’z’‘ (this differs from the [PK::nca()] default).
...	Other arguments that will be passed to [PK::nca()] (other than ‘data‘, ‘design‘, and ‘method‘: *i.e.*, ‘n.tail‘, ‘nsample‘)

Details

This function is a wrapper around [PK::nca()] to do non-compartmental analysis, after automatically detecting the study design. It additionally calls [get_peak()] to calculate the peak concentration and time of peak concentration.

Automatic detection of study design

[PK::nca()] understands three different study designs, and requires the user to specify which one is being used.

- ‘ssd‘: Serial sampling design. Each observation is from a different subject. - ‘complete‘: Every subject was observed at every time point. - ‘batch‘: Each subject was observed at multiple time points, but not at every time point.

To automatically detect which study design is applicable, this function first sorts the data by increasing time. Then, a table of time vs. series ID is created, with 1 indicating that a measurement exists for the corresponding time point/series ID combination, and 0 indicating that a measurement does not exist. If the column sums of this table are all 1, then it is a serial sampling design, except if there is only one observation per time point, it is a complete sampling design, and if there are multiple observations for some time points and only one observation for other time points, it is a batch design. If the column sums are all equal to the number of rows of the table, then it is a complete sampling design. Otherwise, it is a batch sampling design.

Parameters estimated by NCA

- ‘AUC_infinity‘: The area under the concentration-time curve, extrapolated out to infinite time. Estimated using the trapezoidal rule, with a tail area correction calculated using the slope of the last 3 data points (by default). - ‘AUC_tlast‘: The area under the concentration-time curve, calculated at the last observed time point. Estimated using the trapezoidal rule. - ‘AUMC_infinity‘: The area

under the concentration-time first moment curve (the area under the AUC vs. time), extrapolated out to infinite time. Estimated using the trapezoidal rule, with a tail area correction calculated using the slope of the last 3 data points (by default). - ‘CLtot‘: The total clearance rate. Only calculated for ‘route == ’iv’‘. If ‘route == ’oral’‘, this is ‘NA_real_‘, and only ‘CLtot/Fgutabs‘ is calculated. - ‘CLtot/Fgutabs‘: The total clearance rate, normalized by the oral bioavailability. Only calculated for ‘route == ’oral’‘. If ‘route == ’iv’‘, this is ‘NA_real_‘, and only ‘CLtot/‘ is calculated. - ‘Cmax‘: The peak concentration. For ‘route == ’iv’‘, this is expected to be the concentration at the earliest time; for ‘route == ’oral’‘, it is not. This and ‘tmax‘ are calculated using [get_peak()], not by [PK::nca()]. - ‘halflife‘: The half-life of elimination. Only calculated for ‘route == ’iv’‘. If ‘route == ’oral’‘, this is ‘NA_real_‘, because half-life estimates are not valid for oral data. - ‘MRT‘: The mean residence time. Only calculated for ‘route == ’iv’‘. If ‘route == ’oral’‘, this is ‘NA_real_‘, and only ‘MTT‘ is calculated. - ‘MTT‘: The mean transit time (the sum of MRT and mean absorption time). Only calculated for ‘route == ’oral’‘. If ‘route == ’iv’‘, this is ‘NA_real_‘, and only ‘MRT‘ is calculated. -‘tmax‘: The time of peak concentration. For ‘route == ’iv’‘, this is expected to be the earliest time; for ‘route == ’oral’‘, it is not. This and ‘Cmax‘ are calculated using [get_peak()], not by [PK::nca()]. - ‘Vss‘: The volume of distribution at steady state (‘AUMC_infinity/AUC_infinity^2‘). If ‘route == ’oral’‘, this is ‘NA_real_‘, because ‘Vss‘ estimates are not valid for oral data.

Output

The output is a data.frame with 9 rows (one for each NCA parameter) and a number of variables equal to ‘length(method) + 3‘.

The variables are

- ‘design‘: The automatically-detected design. One of ‘ssd‘, ‘complete‘, or ‘batch‘ (or ‘NA_character_‘ if no analysis could be done). - ‘param_name‘: The name of each NCA parameter. - ‘param_value‘: The value of each NCA parameter. - ‘param_sd_[method]‘: The parameter standard error estimated by the corresponding method.

Value

A ‘data.frame‘ with 9 rows and ‘length(method) + 3‘ variables. See Details.

Author(s)

Caroline Ring

`calc_rmse` *Calculate RMSE (root mean squared error)*

Description

Calculate RMSE when observed data may be left-censored (non-detect) or may be reported in summary form (as sample mean, sample standard deviation, and sample number of subjects). Additionally, handle the situation when observed data and predictions need to be log10-transformed before RMSE is calculated.

Usage

```
calc_rmse(pred, obs, obs_sd, n_subj, detect, log10_trans = FALSE)
```

Arguments

pred	Numeric vector: Model-predicted value corresponding to each observed value. Even if 'log10_trans log-transformed. (If 'log10_trans internally to this function before calculation.)
obs	Numeric vector: Observed sample means for each observation if summary data, or observed values for each observation if non-summary data. Censored observations should *not* be NA; they should be substituted with the LOQ. Even if 'log10_trans log-transformed. (If 'log10_trans log-scale means internally to this function before calculation.)
obs_sd	Numeric vector: Observed sample SDs for each observation, if summary data. For non-summary data (individual-subject observations), the corresponding element of 'group_sd' should be set to 0. Even if 'log10_trans 'log10_trans deviations internally to this function before calculation.)
n_subj	Numeric vector: Observed sample number of subjects for each observation. For non-summary data (individual-subject observations), 'n_subj' should be set to 1.
detect	Logical vector: 'TRUE' for each observation that was detected (above LOQ); 'FALSE' for each observation that was non-detect (below LOQ).
log10_trans	Logical. FALSE (default) means that RMSE is computed for natural-scale observations and predictions – effectively, 'sqrt(mean((observed - pred)^2))'. TRUE means that observations and predictions will be log10-transformed before RMSE is calculated (see Details) – effectively 'sqrt(mean((log(observed) - log(pred))^2))'.

Details

RMSE is calculated using the following formula, to properly handle summary data:

$$\sqrt{\frac{1}{N}\sum_{i=1}^{G}((n_i-1)s_i^2+n_i\bar{y}_i^2-2n_i\bar{y}_i\mu_i+\mu_i^2)}$$

In this formula, there are G groups. (For CvTdb data, a "group" is a specific combination of chemical, species, route, medium, dose, and timepoint.) n_i is the number of subjects for group i. $\bar{y}_i$ is the sample mean for group i. s_i is the sample standard deviation for group i.μ_i is the model-predicted value for group i.

N is the grand total of subjects:

$$N=\sum_{i=1}^{G}n_i$$

For the non-summary case (N single-subject observations, with all $n_i = 1$, $s_i = 0$, and $\bar{y}_i = y_i$), this formula reduces to the familiar RMSE formula

$$\sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \mu_i)^2}$$

Left-censored data

If the observed value is censored, and the predicted value is less than the reported LOQ, then the observed value is (temporarily) set equal to the predicted value, for an effective error of zero.

If the observed value is censored, and the predicted value is greater than the reported LOQ, the the observed value is set equal to the reported LOQ.

Log10 transformation

If 'log10_trans log10-transformed before calculating the RMSE. In the case where observed values are reported in summary format, each sample mean and sample SD (reported on the natural scale, i.e. the mean and SD of natural-scale individual observations) are used to produce an estimate of the log10-scale sample mean and sample SD (i.e., the mean and SD of log10-transformed individual observations), using [convert_summary_to_log10()].

The formulas are as follows. Again, $\bar{y}_i$ is the sample mean for group i. s_i is the sample standard deviation for group i.

$$\text{log10-scale sample mean}_i = \log_{10}\left(\frac{\bar{y}_i^2}{\sqrt{\bar{y}_i^2 + s_i^2}}\right)$$

$$\text{log10-scale sample SD}_i = \sqrt{\log_{10}\left(1 + \frac{s_i^2}{\bar{y}_i^2}\right)}$$

Value

A numeric scalar: the root mean squared error (RMSE) for this set of observations and predictions.

Author(s)

Caroline Ring

`calc_rsq` *Calculate r-squared for observed vs. predicted values*

Description

Calculate the square of the Pearson correlation coefficient (r) between observed and model-predicted values

Usage

```
calc_rsq(pred, obs, obs_sd, n_subj, detect, log10_trans = FALSE)
```

Arguments

pred	Numeric vector: Model-predicted value corresponding to each observed value. Even if 'log10_trans log-transformed. (If 'log10_trans log10-transformed internally to this function before calculation.)
obs	Numeric vector: Observed sample means for summary data, or observed values for non-summary data. Censored observations should *not* be NA; they should be substituted with the LOQ. Even if 'log10_trans TRUE', these should *not* be log10-transformed. (If 'log10_trans they will be transformed to log10-scale means internally to this function before calculation.)
obs_sd	Numeric vector: Observed sample SDs for summary data. For non-summary data (individual-subject observations), the corresponding element of 'obs_sd' should be set to 0. Even if 'log10_trans these should *not* be log10-transformed. (If 'log10_trans will be transformed to log10-scale standard deviations internally to this function before calculation.)
n_subj	Numeric vector: Observed sample number of subjects for summary data. For non-summary data (individual-subject observations), 'group_n' should be set to 1.
detect	Logical: Whether each
log10_trans	Logical. FALSE (default) means that R-squared is computed for observations vs. predictions. TRUE means that R-squared is computed for log10(observations) vs. log10(predictions) (see Details).

Details

Calculate the square of the Pearson correlation coefficient (r) between observed and model-predicted values, when observed data may be left-censored (non-detect) or may be reported in summary form (as sample mean, sample standard deviation, and sample number of subjects). Additionally, handle the situation when observed data and predictions need to be log-transformed before RMSE is calculated.

r^2 is calculated according to the following formula, to properly handle observations reported in summary format:

$$r^2 = \left(\frac{\sum_{i=1}^{G} \mu_i n_i \bar{y}_i - (\bar{\mu} + \bar{y}) \sum_{i=1}^{G} n_i \mu_i + (\bar{\mu}\bar{y}) \sum_{i=1}^{G} n_i}{\sqrt{\sum_{i=1}^{G} (n_i - 1) s_i^2 + \sum_{i=1}^{G} n_i \bar{y}_i^2 - 2\bar{y} \sum_{i=1}^{G} n_i \bar{y}_i + N + \bar{y}^2} \sqrt{\sum_{i=1}^{G} n_i \mu_i^2 - 2\bar{y} \sum_{i=1}^{G} n_i \mu_i + N + \bar{y}^2}} \right)^2$$

In this formula, there are G groups (reported observations). (For CvTdb data, a "group" is a specific combination of chemical, species, route, medium, dose, and timepoint.) n_i is the number of subjects for group i. $\bar{y}_i$ is the sample mean for group i. s_i is the sample standard deviation for group i.μ_i is the model-predicted value for group i. $\bar{y}$ is the grand mean of observations:

$$\bar{y} = \frac{\sum_{i=1}^{G} n_i \bar{y}_i}{\sum_{i=1}^{G} n_i}$$

$\bar{\mu}$ is the grand mean of predictions:

$$\bar{\mu} = \frac{\sum_{i=1}^{G} n_i \mu_i}{\sum_{i=1}^{G} n_i}$$

N is the grand total of subjects:

$$N = \sum_{i=1}^{G} n_i$$

For the non-summary case (N single-subject observations, with all $n_i = 1$, $s_i = 0$, and $\bar{y}_i = y_i$), this formula reduces to the familiar formula

$$r^2 = \left(\frac{\sum_{i=1}^{N} (y_i - \bar{y})(\mu_i - \bar{\mu})}{\sqrt{\sum_{i=1}^{N} (y_i - \bar{y})^2} \sqrt{\sum_{i=1}^{N} (\mu_i - \bar{\mu})^2}} \right)^2$$

Left-censored data

If the observed value is censored, and the predicted value is less than the reported LOQ, then the observed value is (temporarily) set equal to the predicted value, for an effective error of zero.

If the observed value is censored, and the predicted value is greater than the reported LOQ, the the observed value is (temporarily) set equal to the reported LOQ, for an effective error of (LOQ - predicted).

Log-10 transformation

If 'log10 log10-transformed before calculating the RMSE. In the case where observed values are reported in summary format, each sample mean and sample SD (reported on the natural scale, i.e. the mean and SD of natural-scale individual observations) are used to produce an estimate of the log10-scale sample mean and sample SD (i.e., the mean and SD of log10-transformed individual observations), using [convert_summary_to_log10()].

The formulas are as follows. Again, $\bar{y}_i$ is the sample mean for group i. s_i is the sample standard deviation for group i.

$$\text{log10-scale sample mean}_i = \log_{10} \left(\frac{\bar{y}_i^2}{\sqrt{\bar{y}_i^2 + s_i^2}} \right)$$

$$\text{log10-scale sample SD}_i = \sqrt{\log_{10} \left(1 + \frac{s_i^2}{\bar{y}_i^2} \right)}$$

Value

A numeric scalar: the R-squared value for observations vs. predictions.

Author(s)

Caroline Ring

`calc_sds_alerts` *Calculate parameter SDs*

Description

Calculate parameter SDs using inverse Hessian

Usage

```
calc_sds_alerts(
  pars_opt,
  pars_const,
  observations,
  modelfun,
  dose_norm,
  log10_trans
)
```

Arguments

`pars_opt`	Named numeric: A vector of parameter values for the parameters that were optimized. For example, you can get this using [coef.pk()] with ‘include_type = "optim"‘.
`pars_const`	Named numeric: A vector of parameter values for parameters that were held constant, not optimized (but are necessary to evaluate the model). For example, you can get this using [coef.pk()] with ‘include_type = "const"‘.
`observations`	The data used to fit the model. For example, you can get this using [get_data.pk()].
`modelfun`	The name of the function that evaluates the model (passed to [log_likelihood()]).
`dose_norm`	Logical: Whether to dose-normalize concentrations before evaluating log-likelihood. Passed to [log_likelihood()].
`log10_trans`	Logical: Whether to log10-transform concentrations before evaluating log-likelihood. Passed to [log_likelihood()].

Details

Calculate parameter SDs using inverse Hessian approach for a single set of parameter values for a single model and a single data set.

This is a workhorse function called by [coef_sd.pk()]. If the length of this vector is n, the Hessian matrix will be $n \times n$.

The coefficient standard deviations are estimated by computing a numerical approximation to the model Hessian (the matrix of second derivatives of the model objective function with respect to each model parameter) and then attempting to invert it. This procedure yields a variance/covariance matrix for the model parameters. The square root of the diagonal elements of this matrix represent the parameter standard deviations.

A first attempt is made to invert the Hessian using [solve()] (see [hess_sd1()]). If the Hessian is singular, an attempt is made to calculate a pseudovariance matrix, following the procedure outlined in Gill & King (2004) (see [hess_sd2()]). First, the generalized inverse of the Hessian is calculated using [MASS::ginv()]. Then, a generalized Cholesky decomposition (to ensure positive-definiteness) is calculated using [Matrix::Cholesky] with argument 'perm = TRUE'. The generalized inverse is reconstructed from the generalized Cholesky factorization. The square root of the diagonal elements of this matrix represent the parameter standard deviations.

If neither of these procedures is successful, then 'NA_real_' is returned for all coefficient standard deviations. Record any error messages encountered during the process, and note which method was used to produce the final results. This is a workhorse function called by [coef_sd.pk()].

Value

A data.frame with variables 'param_name', 'param_sd', and 'sd_alert', and as many rows as the length of 'pars_opt'. 'param_name' contains the names of 'pars_opt'. 'param_sd' contains the parameter standard deviations calculated using the inverse Hessian. 'sd_alerts' is a character variable noting any errors encountered while attempting to calculate the parameter SDs.

Author(s)

Caroline Ring

References

Gill J, King G. (2004) What to Do When Your Hessian is Not Invertible: Alternatives to Model Respecification in Nonlinear Estimation. Sociological Methods & Research 33(1):54-87. DOI: 10.1177/0049124103262681

`check_method` *Check methods*

Description

Check methods for validity

Usage

```
check_method(obj, method)
```

Arguments

`obj`	A [pk()] object
`method`	A user-supplied 'character' vector of method names

Details

Helper function to ensure that a list of methods specified by the user matches the methods available in the fitted [pk()] object.

Value

'TRUE' if all 'method

Author(s)

Caroline Ring

`check_model` *Check models*

Description

Check models for validity

Usage

```
check_model(obj, model)
```

Arguments

`obj`	A [pk()] object
`model`	A user-supplied 'character' vector of model names

Details

Helper function to ensure that a list of models specified by the user matches the models available in the fitted [pk()] object.

Value

'TRUE' if all 'model

Author(s)

Caroline Ring

check_newdata *Check new data*

Description

Check new data to ensure it has the required variables and classes

Usage

```
check_newdata(newdata, olddata, req_vars, exclude = FALSE)
```

Arguments

newdata	A ‘data.frame‘ containing new data
olddata	A ‘data.frame‘ containing existing data. ‘newdata‘ variable classes will be required to match ‘olddata‘
req_vars	A ‘character‘ vector of required variable names that must appear in ‘newdata‘
exclude	Logical: Whether a variable ‘"exclude"‘ also must be present in ‘newdata‘

Details

This is a helper function to check new data to ensure it has the required variables and that those variables are of the correct classes. This is useful, for example, when making predictions from a fitted [pk()] model object on new data.

Value

‘TRUE‘, if required variables are present in ‘newdata‘, and required variables are of the same class in ‘newdata‘ and ‘olddata‘. Otherwise, this function will stop with an error.

Author(s)

Caroline Ring

check_params_1comp *Check 1-compartment model parameters*

Description

Check to make sure required parameters are present to evaluate 1-compartment model for a given route and medium

Usage

```
check_params_1comp(params, route, medium, ...)
```

Arguments

params	A named numeric vector of parameters for the 1-compartment model.
route	A character vector of routes: "iv" and/or "oral".
medium	A character vector of tissue media: "plasma" and/or "blood".
...	Additional arguments (not currently used)

Value

Character: A message. If all required parameters are present for the given media & routes, the message is "Parameters OK". If required parameters for the oral route are missing, the message is "Error: For 1-compartment oral model, missing parameters (comma-separated list of parameter names)". If required parameters for the IV route are missing, the message is "Error: For 1-compartment oral model, missing parameters (comma-separated list of parameter names)".

Author(s)

Caroline Ring

---

check_params_1comp_cl *Check 1-compartment model parameters*

Description

Check to make sure required parameters are present to evaluate 1-compartment model for a given route and medium

Usage

```
check_params_1comp_cl(params, route, medium, ...)
```

Arguments

params	A named numeric vector of parameters for the 1-compartment model.
route	A character vector of routes: "iv" and/or "oral".
medium	A character vector of tissue media: "plasma" and/or "blood".
...	Additional arguments (not currently used)

Value

Character: A message. If all required parameters are present for the given media & routes, the message is "Parameters OK". If required parameters for the oral route are missing, the message is "Error: For 1-compartment oral model, missing parameters (comma-separated list of parameter names)". If required parameters for the IV route are missing, the message is "Error: For 1-compartment oral model, missing parameters (comma-separated list of parameter names)".

Author(s)

Caroline Ring

`check_params_2comp` *Check 2-compartment model parameters*

Description

Check to make sure required parameters are present to evaluate 2-compartment model for a given route and medium

Usage

```
check_params_2comp(params, route, medium, ...)
```

Arguments

`params`	A named numeric vector of parameters for the 2-compartment model.
`route`	A character vector of routes: "iv" and/or "oral".
`medium`	A character vector of tissue media: "plasma" and/or "blood".
`...`	Additional arguments (not currently used)

Value

Character: A message. If all required parameters are present for the given media & routes, the message is "Parameters OK". If required parameters for the oral route are missing, the message is "Error: For 2-compartment oral model, missing parameters (comma-separated list of parameter names)". If required parameters for the IV route are missing, the message is "Error: For 2-compartment oral model, missing parameters (comma-separated list of parameter names)".

Author(s)

Caroline Ring

check_params_flat *Check flat model parameters*

Description

Check to make sure required parameters are present to evaluate flat model for a given route and medium

Usage

```
check_params_flat(params, route, medium, ...)
```

Arguments

params	A named numeric vector of parameters for the flat model.
route	A character vector of routes: "iv" and/or "oral".
medium	A character vector of tissue media: "plasma" and/or "blood".
...	Additional arguments (not currently used)

Value

Character: A message. If all required parameters are present for the given media & routes, the message is "Parameters OK". If required parameters for the oral route are missing, the message is "Error: For flat oral model, missing parameters (comma-separated list of parameter names)". If required parameters for the IV route are missing, the message is "Error: For flat oral model, missing parameters (comma-separated list of parameter names)".

Author(s)

Caroline Ring

check_required_status *Check required status*

Description

This is the S3 method generic.

Usage

```
check_required_status(obj, ...)
```

Arguments

obj	An object.
...	Additional arguments currently not in use.

Value

If the [pk()] object has the required status or greater, returns TRUE. If the [pk()] object has less than the required status, returns FALSE. Returned value has an attribute ‘msg‘, containing an informative message as a string.

See Also

[check_required_status.pk()] for the method for class [pk()]

`check_required_status.default`
Default method for checking required status

Description

Default method for checking required status

Usage

```
## Default S3 method:
check_required_status(obj, ...)
```

Arguments

`obj`	An object.
`...`	Additional arguments currently not in use.

Value

An error, when a non-pk object is used for the first argument.

`check_required_status.pk`
Check required status

Description

Check whether a [pk()] object has a particular required status level

Usage

```
## S3 method for class 'pk'
check_required_status(obj, required_status, ...)
```

Arguments

obj	A [pk()] object
required_status	Integer: The required status. 1 = initialized; 2 = pre-processed; 3 = pre-fitted; 4 = fitted.
...	Additional arguments. Not in use.

Details

This is a helper function to check whether a [pk()] object has the status required for certain operations. For example, status 4 (fitting complete) is required for any fit evaluation functions: [predict.pk()], [residuals.pk()], [coef.pk()], [coef_sd.pk()], [rmse.pk()], [fold_error.pk()]

Value

If the [pk()] object has the required status or greater, returns TRUE. If the [pk()] object has less than the required status, returns FALSE. Returned value has an attribute 'msg', containing an informative message as a string.

Author(s)

Caroline Ring

---

coef.pk *Get coefficients*

Description

Extract coefficients from a fitted [pk()] object

Usage

```
## S3 method for class 'pk'
coef(
  object,
  model = NULL,
  method = NULL,
  drop_sigma = FALSE,
  include_NAs = FALSE,
  include_type = "use",
  suppress.messages = NULL,
  ...
)
```

Arguments

object	A [pk()] object
model	Optional: Specify (as a ‘character‘ vector) one or more of the fitted models whose coefficients to return. If ‘NULL‘ (the default), coefficients will be returned for all of the models in ‘obj$stat_model‘.
method	Optional: Specify (as a ‘character‘ vector)one or more of the [optimx::optimx()] methods whose coefficients to return. If ‘NULL‘ (the default), coefficients will be returned for all of the models in ‘obj$settings_optimx$method‘.
drop_sigma	Logical: ‘FALSE‘ by default. Determines whether to include sigma in the output.
include_NAs	Logical: ‘FALSE‘ by default. Determines whether to include aborted fits which have NAs as coefficients.
include_type	Character: ‘"use"‘ (default) will return all parameters used in evaluating the model, including those that were held constant. ‘"optimize"‘ will return only parameters that were optimized, dropping all that were held constant. ‘"constant"‘ will return *only* parameters that were held constant (used, but not optimized). (‘"optimize"‘ and ‘"constant"‘ are useful, for example, when evaluating the Hessian of the log-likelihood function, which requires differentiating between parameters that were optimized and those that were held constant.) Any value other than ‘"use"‘, ‘"optim"‘, or ‘"const"‘ will return an error.
suppress.messages	Logical: ‘NULL‘ by default to use the setting in ‘object$settings_preprocess$suppress.messages‘. Determines whether to display messages.
...	Additional arguments currently not in use.

Details

This function extracts fitted model parameter values from a fitted [pk()] object.

Value

A data.frame with a row for each ‘data_group‘ x ‘method‘ x ‘model‘ combination in a fitted [pk()] object. When ‘drop_sigma = TRUE‘ there is also a row for each unique standard deviation hyperparameter defined by ‘error_group‘ in the fitted [pk()] object. There is a column for all parameter estimates given each model in ‘model‘. A list-column ‘coefs_vector‘ summarizes all estimated parameters into a named vector. This named vector is used in functions that call upon the model functions, such as [predict()].

Author(s)

Caroline Ring, Gilberto Padilla Mercado

See Also

Other methods for fitted pk objects: `AAFE.pk()`, `AFE.pk()`, `AIC.pk()`, `BIC.pk()`, `coef_sd.pk()`, `eval_tkstats.pk()`, `get_fit.pk()`, `get_hessian.pk()`, `get_tkstats.pk()`, `logLik.pk()`, `predict.pk()`, `residuals.pk()`, `rmse.pk()`, `rsq.pk()`

coef_sd *Coefficient standard deviations*

Description

This is the S3 method generic for ‘coef_sd‘.

Usage

```
coef_sd(obj, model, method, suppress.messages, ...)
```

Arguments

obj	An object
model	The TK model used.
method	Optimizer method used.
suppress.messages	Boolean. Whether messages will be printed.
...	Additional arguments currently not in use.

Value

A dataframe with one row for each ‘data_group‘, ‘model‘ and ‘method‘. The remaining columns include the parameters & hyperparameters as returned by [coef.pk()], as well as their calculated standard deviations.

See Also

[coef_sd.pk()] for the ‘coef_sd‘ method for class [pk()]

coef_sd.default *Coefficient standard deviation default*

Description

Coefficient standard deviation default

Usage

```
## Default S3 method:
coef_sd(obj, ...)
```

Arguments

obj	An object
...	Additional arguments currently not in use.

Value

An error, when a non-pk object is used for the first argument.

`coef_sd.pk` *Get coefficient standard deviations*

Description

Extract coefficient/parameter standard deviations from a fitted ‘pk‘ object

Usage

```
## S3 method for class 'pk'
coef_sd(obj, model = NULL, method = NULL, suppress.messages = TRUE, ...)
```

Arguments

`obj`	A [pk] object
`model`	Optional: Specify one or more of the fitted models whose coefficients to return. If NULL (the default), coefficients will be returned for all of the models in ‘obj$stat_model‘.
`method`	Optional: Specify one or more of the [optimx::optimx()] methods whose coefficients to return. If NULL (the default), coefficients will be returned for all of the models in ‘obj$settings_optimx$method‘.
`suppress.messages`	Logical. ‘TRUE‘ (the default) to suppress informative messages. ‘FALSE‘ to see them.
`...`	Additional arguments. Not in use right now.

Details

The coefficient standard deviations are estimated by computing a numerical approximation to the model Hessian (the matrix of second derivatives of the model objective function with respect to each model parameter) and then attempting to invert it. This procedure yields a variance/covariance matrix for the model parameters. The square root of the diagonal elements of this matrix represent the parameter standard deviations.

A first attempt is made to invert the Hessian using [solve()] (see [hess_sd1()]). If the Hessian is singular, an attempt is made to calculate a pseudovariance matrix, following the procedure outlined in Gill & King (2004) (see [hess_sd2()]). First, the generalized inverse of the Hessian is calculated using [MASS::ginv()]. Then, a generalized Cholesky decomposition (to ensure positive-definiteness) is calculated using [Matrix::Cholesky] with argument ‘perm = TRUE‘. The generalized inverse is reconstructed from the generalized Cholesky factorization. The square root of the diagonal elements of this matrix represent the parameter standard deviations.

If neither of these procedures is successful, then ‘NA_real_‘ is returned for all coefficient standard deviations.

Value

A dataframe with one row for each ‘data_group‘, ‘model‘ and ‘method‘. The remaining columns include the parameters & hyperparameters as returned by [coef.pk()], as well as their calculated standard deviations.

Author(s)

Caroline Ring and Gilberto Padilla Mercado

References

Gill J, King G. (2004) What to Do When Your Hessian is Not Invertible: Alternatives to Model Respecification in Nonlinear Estimation. Sociological Methods & Research 33(1):54-87. DOI: 10.1177/0049124103262681

See Also

Other methods for fitted pk objects: `AAFE.pk()`, `AFE.pk()`, `AIC.pk()`, `BIC.pk()`, `coef.pk()`, `eval_tkstats.pk()`, `get_fit.pk()`, `get_hessian.pk()`, `get_tkstats.pk()`, `logLik.pk()`, `predict.pk()`, `residuals.pk()`, `rmse.pk()`, `rsq.pk()`

---

combined_sd *Combined standard deviation*

Description

Given mean, standard deviation, and N for some set of groups, calculate the combined standard deviation. Note that the groups may not overlap.

Usage

```
combined_sd(
  group_mean,
  group_sd,
  group_n,
  unbiased = TRUE,
  na.rm = TRUE,
  log10 = FALSE
)
```

Arguments

`group_mean`	Numeric vector: Observed sample means for summary data, or observed values for non-summary data. Censored observations should *not* be NA; they should be substituted with some value at or below the corresponding LOQ (e.g. LOQ or LOQ/2). Even if ‘log10 should *not* be log10-transformed.

group_sd	Numeric vector: Observed sample SDs for summary data. For non-summary data (individual-subject observations), the corresponding element of ‘group_sd‘ should be set to 0. Even if ‘log10 should *not* be log10-transformed.
group_n	Numeric vector: Observed sample number of subjects for summary data. For non-summary data (individual-subject observations), ‘group_n‘ should be set to 1.
unbiased	Logical. If TRUE (the default), then ‘group_sd‘ is assumed to be the unbiased estimator of population standard deviation (i.e. calculated using ‘n-1‘ in the denominator – the way that ‘stats::sd()‘ calculates it), and the returned combined SD is also the unbiased estimator of the combined population SD. If FALSE, then ‘group_sd‘ is assumed to be the biased estimator (using ‘n‘ in the denominator), and the returned value is also the biased estimator of the combined population SD.
na.rm	Logical. If TRUE (default), then any groups where mean, SD, *or* N were NA will be dropped. If FALSE, they will be retained (and the result will be NA).
log10	Logical. If TRUE, the standard deviations are from log10-transformed values.

Value

Numeric: the standard deviation of the combined population (i.e. if all the groups were concatenated into one large group).

Author(s)

Caroline Ring

---

compare_models *Model comparison*

Description

This is the S3 method generic for compare_models()

Usage

```
compare_models(obj, ...)
```

Arguments

obj	An object.
...	Additional arguments currently not in use.

Value

A ‘data.frame‘ with variables - ‘model‘: The name of each model - ‘method‘: The name of each method - A variable named for ‘criterion‘ (e.g. if ‘criterion = "AIC"‘ then the result will have a variable named ‘AIC‘): The criterion value for each model/method

See Also

[compare_models.pk()] for the method for class [pk()]

`compare_models.default` *Default method for compare_models()*

Description

Default method for compare_models()

Usage

```
## Default S3 method:
compare_models(obj, ...)
```

Arguments

`obj`	an object
`...`	Additional arguments currently not in use.

Value

An error, when a non-pk object is used for the first argument.

`compare_models.pk` *Model comparison for [pk()] objects*

Description

Perform model comparison for a fitted [pk()] object.

Usage

```
## S3 method for class 'pk'
compare_models(
  obj,
  newdata = NULL,
  model = NULL,
  method = NULL,
  criterion = "AIC",
  ...
)
```

Arguments

obj	A [pk()] model object. Must be fitted, or the function will exit with an error.
newdata	Optional: A ‘data.frame‘ containing new data for which to compute the TK stats. Must contain at least variables ‘Chemical‘, ‘Species‘, ‘Route‘, ‘Media‘, ‘Dose‘, and any other variables named in ‘tk_grouping‘. Default ‘NULL‘, to use the data in ‘obj$data‘.
model	Character: One or more of the models fitted. Default ‘NULL‘ to return TK stats for all models.
method	Character: One or more of the [optimx::optimx()] methods used. Default ‘NULL‘ to return TK stats for all methods.
criterion	The name of a criterion function to use for model comparison. Default "AIC". Must be the name of a function that (as for ‘AIC‘) accepts arguments ‘obj‘, ‘newdata‘, ‘method‘ and ‘model‘ (may accept other arguments, specified in ‘...‘) and returns output as for ‘AIC‘: a named list of numeric vectors (named for each of the model names in ‘model‘), where each vector has elements named for each of the method names in ‘method‘, containing the criterion value calculated for that model fitted using that method.
...	Optional: Other arguments to ‘criterion‘ function.

Details

Models are compared according to the goodness-of-fit criterion named in "criterion", and the name of the winning model is returned.

Value

A ‘data.frame‘ with variables - ‘model‘: The name of each model - ‘method‘: The name of each method - A variable named for ‘criterion‘ (e.g. if ‘criterion = "AIC"‘ then the result will have a variable named ‘AIC‘): The criterion value for each model/method

Author(s)

Caroline Ring

---

conc_scale_use *Get concentration scalings*

Description

A helper function to get concentration scalings

Usage

```
conc_scale_use(use_scale_conc, obj)
```

Arguments

use_scale_conc	The ‘use_scale_conc‘ argument (see Details)
obj	A [pk()] object

Details

In methods applied to fitted [pk()] objects that also accept ‘newdata‘ arguments, the user may specify whether to use the concentration scaling of the fitted [pk()] object, or use a different concentration scaling. This is done by specifying an argument ‘use_scale_conc‘, which may be ‘TRUE‘ (to use the scaling from the fitted object), ‘FALSE‘ (to use no scaling), or may be a named list with elements ‘dose_norm‘ and ‘log10_trans‘ to specify scaling/transformation directly. This helper function parses the ‘use_scale_conc‘ argument.

Value

A named list with elements ‘dose_norm‘ and ‘log10_trans‘, both logical.

---

convert_summary_to_log10

Convert sample mean and SD to log10-scale

Description

Estimate log10-scale sample mean and standard deviation from natural-scale sample mean and standard deviation

Usage

```
convert_summary_to_log10(sample_mean, sample_SD)
```

Arguments

sample_mean	Numeric: one or more sample means
sample_SD	Numeric: one or more sample SDs

Details

$\bar{y}_i$ is the natural-scale sample mean for group i. s_i is the natural-scale sample standard deviation for group i.

$$\text{log10-scale sample mean}_i = \log_{10}\left(\frac{\bar{y}_i^2}{\sqrt{\bar{y}_i^2 + s_i^2}}\right)$$

$$\text{log10-scale sample SD}_i = \sqrt{\log_{10}\left(1 + \frac{s_i^2}{\bar{y}_i^2}\right)}$$

Value

A list with two named elements: "log10mean" and "log10SD", the log10-scale sample means and log10-scale sample SDs, respectively.

Author(s)

Caroline Ring

`convert_time` *Helper function to convert time units*

Description

Convert a vector of times between units

Usage

```
convert_time(x, from = "hours", to = "identity", inverse = FALSE)
```

Arguments

`x`	Numeric: one or more time values to be converted.
`from`	Character vector: ‘x‘ is currently in these units. Must be units understood by ‘lubridate::duration()‘, i.e. ‘"seconds"‘, ‘"hours"‘, ‘"days"‘, ‘"weeks"‘, ‘"months"‘, ‘"years"‘, ‘"milliseconds"‘, ‘"microseconds"‘, ‘"nanoseconds"‘, and/or ‘"picoseconds"‘. Default value is ‘"hours"‘.
`to`	Character vector: ‘x‘ will be converted to these units. Must be either ‘"auto"‘, ‘"identity"‘, or units understood by ‘lubridate::duration()‘, i.e. ‘"seconds"‘, ‘"hours"‘, ‘"days"‘, ‘"weeks"‘, ‘"months"‘, ‘"years"‘, ‘"milliseconds"‘, ‘"microseconds"‘, ‘"nanoseconds"‘, and/or ‘"picoseconds"‘. Default value is ‘"identity"‘. If ‘"identity"‘, then ‘x‘ will be returned unchanged. If ‘"auto"‘, then units will be automatically chosen that make the midpoint of ‘x‘ (or its inverse, if ‘inverse = TRUE‘) as close to an order of magnitude of 10 as possible (see [auto_units()]).
`inverse`	Logical: TRUE if ‘x‘ is in units of *inverse* time (e.g. 1/hour, 1/day); FALSE if ‘x‘ is in units of time (e.g. hours, days). Default value is FALSE.

Value

A numeric vector the same length as ‘x‘, converted from the units in ‘from‘ to the units in ‘to‘.

Author(s)

Caroline Ring, Gilberto Padilla Mercado

cp_1comp *Analytical 1-compartment model*

Description

Calculates plasma concentrations vs. time according to the analytical solution for the 1-compartment model, for single bolus doses (IV and/or oral).

Usage

```
cp_1comp(params, time, dose, route, medium = "plasma")
```

Arguments

params	A named numeric vector of model parameter values. See Details for requirements.
time	A numeric vector of times, reflecting the time point when concentration is measured after the corresponding single bolus dose. Must be same length as 'dose' and 'route', or length 1.
dose	A numeric vector of doses, reflecting single bolus doses administered at time 0. Must be same length as 'time' and 'route', or length 1.
route	A character vector, reflecting the route of administration of each single bolus dose: ''oral'' or ''iv''. Must be same length as 'time' and 'dose', or length 1.
medium	A character vector reflecting the medium in which each resulting concentration is to be calculated: "blood" or "plasma". Default is "plasma". Must be same length as 'time' and 'dose', or length 1.

Details

Required parameters

'params' must include the following named items:

kelim Elimination rate, 1/time.

Vdist Apparent volume of central compartment, volume/unit BW. Or see below for 'Fgutabs_Vdist'

For oral administration (if any 'route include:

Fgutabs Oral bioavailability, unitless fraction. Or see below for 'Fgutabs_Vdist'

kgutabs Rate of absorption from gut, 1/time.

For oral administration, in lieu of 'Vdist' and 'Fgutabs', you may instead provide 'Fgutabs_Vdist', the ratio of Fgutabs to Vdist (1/volume). This is an alternate parameterization for situations where 'Fgutabs' and 'Vdist' are not identifiable separately (i.e., when oral TK data are available, but IV data are not). If 'Fgutabs' and 'Vdist' are provided, they will override any value provided for 'Fgutabs_Vdist'.

If both oral and IV administration are specified (i.e., some 'route and some 'route 'Fgutabs' or 'Fgutabs_Vdist'. (If 'Vdist' and 'Fgutabs_Vdist' are provided, but 'Fgutabs' is not provided, then 'Fgutabs' will be calculated from 'Vdist' and 'Fgutabs_Vdist'.)

If 'any(medium 'Rblood2plasma', the ratio of chemical concentration in whole blood to the chemical concentration in blood plasma.

Value

A vector of blood or plasma concentration values corresponding to 'time'.

Author(s)

Caroline Ring, John Wambaugh

See Also

Other built-in model functions: auc_1comp(), auc_1comp_cl(), auc_2comp(), auc_flat(), cp_1comp_cl(), cp_2comp(), cp_2comp_dt(), cp_flat(), get_params_1comp(), get_params_1comp_cl(), get_params_1comp_fup(), get_params_2comp(), get_params_flat(), get_starts_1comp(), get_starts_1comp_cl(), get_starts_1comp_fup(), get_starts_2comp(), get_starts_flat(), tkstats_2comp(), transformed_params_2comp()

Other 1-compartment model functions: auc_1comp(), auc_1comp_cl(), cp_1comp_cl(), get_params_1comp(), get_params_1comp_cl(), get_params_1comp_fup(), get_starts_1comp(), get_starts_1comp_cl(), get_starts_1comp_fup()

Other model concentration functions: cp_1comp_cl(), cp_2comp(), cp_flat()

---

cp_1comp_cl *Analytical 1-compartment model with specific clearance*

Description

Calculates plasma concentrations vs. time according to the analytical solution for the 1-compartment model, for single bolus doses (IV and/or oral).

Usage

```
cp_1comp_cl(params, time, dose, route, medium = "plasma", restrictive = FALSE)
```

Arguments

params	A named numeric vector of model parameter values. See Details for requirements.
time	A numeric vector of times, reflecting the time point when concentration is measured after the corresponding single bolus dose. Must be same length as 'dose' and 'iv.dose', or length 1.
dose	A numeric vector of doses, reflecting single bolus doses administered at time 0. Must be same length as 'time' and 'iv.dose', or length 1.

route	A character vector, reflecting the route of administration of each single bolus dose: ‘’oral’‘ or ‘’iv’‘. Must be same length as ‘time‘ and ‘dose‘, or length 1.
medium	A character vector reflecting the medium in which each resulting concentration is to be calculated: "blood" or "plasma". Default is "plasma". Must be same length as ‘time‘ and ‘dose‘, or length 1.
restrictive	A logical value (TRUE or FALSE. Default: FALSE) that says whether the assumption is that the clearance is restrictive or non-restrictive

Details

Required parameters

‘params‘ must include the following named items:

Fup Fraction of compound unbound in plasma. Unitless.

Clint Intrinsic clearance by hepatocytes. Units: 1/hr

Q_totli Total blood flow through the liver. Units: L/h/kg body weight ^ (3/4)

Q_gfr Glomerular filtration rate, how quickly do kidneys filter. Units: L/h/kg body weight ^ (3/4)

Vdist Apparent volume of central compartment, volume/unit BW. Or see below for ‘Fgutabs_Vdist‘

For oral administration (if any ‘route include:

Fgutabs Oral bioavailability, unitless fraction. Or see below for ‘Fgutabs_Vdist‘

kgutabs Rate of absorption from gut, 1/time.

For oral administration, in lieu of ‘Vdist‘ and ‘Fgutabs‘, you may instead provide ‘Fgutabs_Vdist‘, the ratio of Fgutabs to Vdist (1/volume). This is an alternate parameterization for situations where ‘Fgutabs‘ and ‘Vdist‘ are not identifiable separately (i.e., when oral TK data are available, but IV data are not). If ‘Fgutabs‘ and ‘Vdist‘ are provided, they will override any value provided for ‘Fgutabs_Vdist‘.

If both oral and IV administration are specified (i.e., some ‘route and some ‘route ‘Fgutabs‘ or ‘Fgutabs_Vdist‘. (If ‘Vdist‘ and ‘Fgutabs_Vdist‘ are provided, but ‘Fgutabs‘ is not provided, then ‘Fgutabs‘ will be calculated from ‘Vdist‘ and ‘Fgutabs_Vdist‘.)

If ‘any(medium ‘Rblood2plasma‘, the ratio of chemical concentration in whole blood to the chemical concentration in blood plasma.

Value

A vector of blood or plasma concentration values corresponding to ‘time‘.

Author(s)

Caroline Ring, John Wambaugh

See Also

Other built-in model functions: auc_1comp(), auc_1comp_cl(), auc_2comp(), auc_flat(), cp_1comp(), cp_2comp(), cp_2comp_dt(), cp_flat(), get_params_1comp(), get_params_1comp_cl(), get_params_1comp_fup(), get_params_2comp(), get_params_flat(), get_starts_1comp(), get_starts_1comp_cl(), get_starts_1comp_fup(), get_starts_2comp(), get_starts_flat(), tkstats_2comp(), transformed_params_2comp()

Other 1-compartment model functions: auc_1comp(), auc_1comp_cl(), cp_1comp(), get_params_1comp(), get_params_1comp_cl(), get_params_1comp_fup(), get_starts_1comp(), get_starts_1comp_cl(), get_starts_1comp_fup()

Other model concentration functions: cp_1comp(), cp_2comp(), cp_flat()

---

cp_2comp *Analytical 2-compartment model*

Description

Calculates plasma concentration according to the analytical solution for the 2-compartment model.

Usage

```
cp_2comp(params, time, dose, route, medium = "plasma")
```

Arguments

params	A named numeric vector of parameter values. See Details for requirements.
time	A numeric vector of times, reflecting the time points when concentration is measured after the corresponding single bolus dose. Must be same length as other arguments, or length 1.
dose	A numeric vector of doses, reflecting single bolus doses administered at time 0. Must be same length as other arguments, or length 1.
route	A character vector, reflecting the route of administration of each single bolus dose: 'oral' or 'iv'. Must be same length as time and dose, or length 1.
medium	A character vector reflecting the medium in which each resulting concentration is to be calculated: "blood" or "plasma". Default is "plasma". Must be same length as other arguments, or length 1.

Value

A vector of blood or plasma concentration values (mass chemical/volume media) corresponding to each value in time

Author(s)

Caroline Ring, John Wambaugh

See Also

Other built-in model functions: auc_1comp(), auc_1comp_cl(), auc_2comp(), auc_flat(), cp_1comp(), cp_1comp_cl(), cp_2comp_dt(), cp_flat(), get_params_1comp(), get_params_1comp_cl(), get_params_1comp_fup(), get_params_2comp(), get_params_flat(), get_starts_1comp(), get_starts_1comp_cl(), get_starts_1comp_fup(), get_starts_2comp(), get_starts_flat(), tkstats_2comp(), transformed_params_2comp()

Other 2-compartment model functions: auc_2comp(), cp_2comp_dt(), get_params_2comp(), get_starts_2comp(), tkstats_2comp(), transformed_params_2comp()

Other model concentration functions: cp_1comp(), cp_1comp_cl(), cp_flat()

---

cp_2comp_dt *Time derivative of analytical 2-compartment model*

Description

Calculates the time derivative (instantaneous rate of change) of plasma concentration according to the analytical solution for the 2-compartment model.

Usage

```
cp_2comp_dt(params, time, dose, route, medium)
```

Arguments

params	A named list of parameter values including the following: **k12** Rate at which compound moves from central to peripheral compartment, 1/h. **k21** Rate at which compound moves from peripheral to central compartment, 1/h. **kelim** Elimination rate, 1/h. **V1** Apparent volume of central compartment, L/kg BW. Or see below for "Fgutabs_V1" For oral administration (‘route include: **Fgutabs** Oral bioavailability, unitless fraction. Or see below for "Fgutabs_V1" **kgutabs** Rate of absorption from gut, 1/h. For oral administration, in lieu of "V1" and "Fgutabs", you may instead provide "Fgutabs_V1", the ratio of Fgutabs to V1 (1/L). This is an alternate parameterization for situations where "Fgutabs" and "V1" are not identifiable separately (i.e. when oral data are available, but IV data are not). If "Fgutabs" and "V1" are provided, then "Fgutabs_V1" will not be used.
time	A numeric vector of times in hours, reflecting the time points when concentration is measured after the corresponding single bolus dose. Must be same length as ‘dose‘ and ‘route‘, or length 1.
dose	A numeric vector of doses in mg/kg, reflecting single bolus doses administered at time 0. Must be same length as ‘time‘ and ‘route‘, or length 1.

route	A character vector, reflecting the route of administration of each single bolus dose. Currently, only "iv" and "oral" are supported. Must be same length as ‘time‘ and ‘dose‘, or length 1.
medium	A character vector reflecting the medium in which each resulting concentration is to be calculated: "blood" or "plasma". Default is "plasma". Must be same length as ‘time‘ and ‘dose‘, or length 1.

Details

This function is used by [postprocess_data()] to determine the time of peak concentration for the 2-compartment model, by locating the point where the time derivative of concentration crosses zero.

Value

A vector of instantaneous rates of change of plasma concentration values (mg/L/time) corresponding to each value in `time`

Author(s)

Caroline Ring, John Wambaugh

See Also

Other built-in model functions: `auc_1comp()`, `auc_1comp_cl()`, `auc_2comp()`, `auc_flat()`, `cp_1comp()`, `cp_1comp_cl()`, `cp_2comp()`, `cp_flat()`, `get_params_1comp()`, `get_params_1comp_cl()`, `get_params_1comp_fup()`, `get_params_2comp()`, `get_params_flat()`, `get_starts_1comp()`, `get_starts_1comp_cl()`, `get_starts_1comp_fup()`, `get_starts_2comp()`, `get_starts_flat()`, `tkstats_2comp()`, `transformed_params_2comp()`

Other 2-compartment model functions: `auc_2comp()`, `cp_2comp()`, `get_params_2comp()`, `get_starts_2comp()`, `tkstats_2comp()`, `transformed_params_2comp()`

---

cp_flat *Flat model*

Description

Evaluates a "flat" model for concentration vs. time

Usage

```
cp_flat(params, time, dose, route, medium = "plasma")
```

Arguments

params	A named list of parameter values. See Details for requirements.
time	A numeric vector of times, reflecting the time points when concentration is measured after the corresponding single bolus dose. Must be same length as other arguments, or length 1.
dose	A numeric vector of doses, reflecting single bolus doses administered at time 0. Must be same length as other arguments, or length 1.
route	A character vector, reflecting the route of administration of each single bolus dose: ‘’oral’‘ or ‘’iv’‘. Must be same length as ‘time‘ and ‘dose‘, or length 1.
medium	A character vector reflecting the medium in which each resulting concentration is to be calculated: "blood" or "plasma". Default is "plasma". Must be same length as other arguments, or length 1.

Details

This function is used for model comparison: does a 1- or 2-compartment TK model fit the data any better than this naive "flat" model?

Required parameters

‘params‘ must include the following named items:

Vdist Apparent volume of central compartment, volume/unit BW. Or see below for ‘Fgutabs_Vdist‘

For oral administration (if any ‘route include:

Fgutabs Oral bioavailability, unitless fraction. Or see below for ‘Fgutabs_Vdist‘

For oral administration, in lieu of ‘Vdist‘ and ‘Fgutabs‘, you may instead provide ‘Fgutabs_Vdist‘, the ratio of Fgutabs to Vdist (1/volume). This is an alternate parameterization for situations where ‘Fgutabs‘ and ‘Vdist‘ are not identifiable separately (i.e., when oral TK data are available, but IV data are not). If ‘Fgutabs‘ and ‘Vdist‘ are provided, they will override any value provided for ‘Fgutabs_Vdist‘.

If both oral and IV administration are specified (i.e., some ‘route and some ‘route ‘Fgutabs‘ or ‘Fgutabs_Vdist‘. (If ‘Vdist‘ and ‘Fgutabs_Vdist‘ are provided, but ‘Fgutabs‘ is not provided, then ‘Fgutabs‘ will be calculated from ‘Vdist‘ and ‘Fgutabs_Vdist‘.)

If ‘any(medium ‘Rblood2plasma‘, the ratio of chemical concentration in whole blood to the chemical concentration in blood plasma.

Flat model equations

IV administration

$$\text{Conc} = \frac{\text{Dose}}{V_{\text{dist}}}$$

Oral administration

$$\text{Conc} = \frac{F_{\text{gutabs}}\text{Dose}}{V_{\text{dist}}}$$

Value

A vector of plasma concentration values (mass chemical/volume) corresponding to `time`.

Author(s)

Caroline Ring, John Wambaugh, Chris Cook

See Also

Other built-in model functions: `auc_1comp()`, `auc_1comp_cl()`, `auc_2comp()`, `auc_flat()`, `cp_1comp()`, `cp_1comp_cl()`, `cp_2comp()`, `cp_2comp_dt()`, `get_params_1comp()`, `get_params_1comp_cl()`, `get_params_1comp_fup()`, `get_params_2comp()`, `get_params_flat()`, `get_starts_1comp()`, `get_starts_1comp_cl()`, `get_starts_1comp_fup()`, `get_starts_2comp()`, `get_starts_flat()`, `tkstats_2comp()`, `transformed_params_2comp()`

Other flat model functions: `auc_flat()`, `get_params_flat()`, `get_starts_flat()`

Other model concentration functions: `cp_1comp()`, `cp_1comp_cl()`, `cp_2comp()`

`cvt` *CvTdb data*

Description

Concentration vs. time data from CvTdb

Usage

```
cvt
```

Format

A 'data.frame' with 13937 rows and 61 variables:

conc_time_id Unique database identifier for each CvT observation.

time_original Timepoint in original units.

time_hr Timepoint in hours.

conc_original Concentration in original units.

conc_sd_original Standard deviation of concentration in original units.

conc Concentration in normalized units.

conc_sd Standard deviation of concentration in normalized units.

fk_series_id Unique database identifier for experimental series.

analyte_name_original Original analyte name.

analyte_dtxsid DTXSID of chemical analyte.

analyte_casrn CASRN of chemical analyte.

fk_analyzed_chemical_id Unique database identifier for analyte.

fk_test_chemical_id Unique database identifier for tested chemical.

test_substance_dtxsid DTXSID of tested chemical.

analyte_name_secondary_original Alternative names for analyte.

conc_medium_normalized Standardized media names, blood or plasma.

conc_medium_original Original media names.

time_units_original Original time units.

conc_units_original Original concentration units.

conc_units_normalized Normalized concentration units.

conc_unit_norm_factor Ratio of conc/conc_original

loq Level of quantification.

loq_units Units for loq.

n_subjects_in_series Number of subjects in each series.

radiolabeled Answers whether this observation comes from a radiolabelling or isotope tracing experiment.

fk_study_id Unique database identifier for each study.

administration_route_normalized Route of exposure/administration, either oral or iv.

fk_dosed_chemical_id Unique database identifier for dosed chemical.

dose_volume Volume of dose.

dose_volume_units Units for dose_volume.

dose_vehicle If available, specifies what vehicle was used CvT experiment.

dose_duration Duration of the dose, if available.

dose_duration_units Units for dose_duration.

dose_frequency Frequency of dosing, these should all be 1 for a single bolus.

fasting_period If available, describes the fasting period for subjects.

dose_level_normalized Dose levels in normalized units.

dose_level_original Dose levels in original units.

dose_level_units_original Units for dose_level_original.

fk_subject_id Unique database identifier for each subject.

weight_kg Subject weight, in kilograms.

species Subject species.

sex Subject sex, if available.

age Subject age, if available.

age_units Units for age.

age_category Categories for age.

fk_extraction_document_id Unique database identifier for documents that were curated.

pmid Document PubMed ID.

year Year of publication.

other_study_identifier Alternative identifier for documents, used for NTP studies.

url Document URL address.

doi Document DOI.

extracted Curation level of document.

curation_set_tag A grouping tag for specific document extraction and curation efforts.

n_subjects_normalized Normalized subject number.

invivPK_dose_level_units dose_level_units used in this package, mg/kg.

invivPK_conc_units conc_units used in this package, ug/mL

invivPK_conc Concentrations normalized to ug/mL

invivPK_dose_level Dose normalized to mg/kg

invivPK_loq Level of quantification in ug/mL

invivPK_loq_units LOQ units used in this package, ug/mL.

invivPK_conc_sd Standard deviation of concentrations, in ug/mL.

Details

This is concentration vs. time data from CvTdb, most recently downloaded as of the date in ['cvt_date'].

These data have been filtered to retain only oral and intravenous administration, and only measurements in blood and plasma. They have also been filtered to retain only observations where the same chemical was both administered and measured in blood/plasma (i.e., excluding observations where a metabolite was measured).

`cvt_date` *CvTdb download date*

Description

The most recent download date of ['cvt'] data

Usage

```
cvt_date
```

Format

An object of class `Date` of length 1.

Details

A character scalar giving the date in "YYYY-MM-DD" format of the download date of the data in ['cvt'] from the CvTdb database.

data_summary *data_summary()*

Description

This is the S3 method generic for data_summary()

Usage

```
data_summary(obj, ...)
```

Arguments

obj	An object.
...	Additional arguments currently not in use.

Value

A ‘data.frame‘ with variables including all the grouping variables in ‘summary_group‘, ‘group_id‘; ‘param_name‘ (the name of the summary statistic; see Details); ‘param_value‘ (the summary statistic value); ‘param_units‘ (the units of the summary statistic, derived from the units of the data).

See Also

[data_summary.pk()] for the method for class [pk()]

data_summary.default *Default method for data_summary()*

Description

Default method for data_summary()

Usage

```
## Default S3 method:
data_summary(obj, ...)
```

Arguments

obj	An object
...	Additional arguments currently not in use.

Value

An error, when a non-pk object is used for the first argument.

`data_summary.pk` *Data summary for a 'pk' object*

Description

Calculate data summary statistics for a 'pk' object

Usage

```
## S3 method for class 'pk'
data_summary(obj, newdata = NULL, summary_group = NULL, ...)
```

Arguments

`obj`	A [pk()] model object. Must be fitted, or the function will exit with an error.
`newdata`	Optional: A 'data.frame' containing new data for which to compute the TK stats. Must contain at least variables 'Chemical', 'Species', 'Route', 'Dose', 'Conc', 'Dose.Units', 'Conc.Units', either 'Time_trans.Units' or 'Time.Units', and any other variables named in 'tk_grouping'. Default 'NULL', to use the data in 'get_data(obj)'.
`summary_group`	A list of variables provided using a 'dplyr::vars()' call. The data (either 'newdata' or 'obj$data') will be grouped according to the unique combinations of these variables. For each unique combination of these variables in the data, a set of summary statistics will be computed. The default is 'NULL', to use the same data grouping that was set in [stat_nca()] for the 'pk' object. However, you may specify a different data grouping if you wish.
`...`	Additional arguments. Not in use.

Details

Get summary statistics for data in a 'pk' object (or optionally, new data), using data groupings defined by 'get_nca_group()' for the 'pk' object (or optionally, new groupings). If you provide both 'newdata' and 'summary_group', then everything in the 'pk' object will be ignored and you will simply be doing data summary *de novo* (which may be what you want).

Summary statistics include, for each group:

- 'n_obs': the number of observations - 'n_exclude': The number of excluded observations - 'n_detect': The number of non-excluded detected observations - 'n_series_id': The number of unique series IDs - 'n_timepts': The number of unique time points - 'n_ref': The number of unique reference IDs - 'tlast': The time of the latest non-excluded observation - 'tlast_detect': The time of the latest non-excluded detected observation - 'tfirst': The time of the earliest non-excluded observation - 'tfirst_detect': The time of the earliest non-excluded detected observation

Value

A 'data.frame' with variables including all the grouping variables in 'summary_group', 'group_id'; 'param_name' (the name of the summary statistic; see Details); 'param_value' (the summary statistic value); 'param_units' (the units of the summary statistic, derived from the units of the data).

Author(s)

Caroline Ring

`dlnorm_summary` *Log-normal distribution density function for summary data*

Description

Evaluates the normal distribution density function for summary data reported as sample mean, sample SD, and sample N. Sample mean and sample SD should be on the *natural* scale. If you have log-scale sample mean and SD (i.e., the mean and SD of log-transformed observations),then use [dnorm_summary()] instead.

Usage

```
dlnorm_summary(mu, sigma, x_mean, x_sd, x_N, log = FALSE)
```

Arguments

`mu`	*Log-scale* mean of the log-normal distribution to be evaluated (*not* the sample mean). May be a numeric scalar or vector.
`sigma`	*Log-scale* standard deviation of the log-normal distribution to be evaluated (*not* the sample SD). May be a numeric scalar or vector.
`x_mean`	Sample mean (on the *natural* scale). May be a numeric scalar or vector.
`x_sd`	Sample standard deviation (on the *natural* scale). May be a numeric scalar or vector.
`x_N`	Sample number of observations. May be a numeric scalar or vector.
`log`	TRUE/FALSE: Whether to return the log of the density function (i.e., the log-likelihood). Default FALSE.

Details

‘x_mean‘, ‘x_sd‘, ‘X_N‘, ‘mu‘, and ‘sigma‘ should either be all the same size, or length 1. If they are different lengths, they will be repeated until their lengths match, with a warning.

Value

A numeric scalar or vector matching the length of the longest of ‘mu‘, ‘sigma‘, ‘x_mean‘, ‘x_sd‘, and ‘x_N‘.

Author(s)

Caroline Ring

dnorm_summary *Normal distribution density function for summary data*

Description

Evaluates the normal distribution density function for summary data reported as sample mean, sample SD, and sample N.

Usage

```
dnorm_summary(mu, sigma, x_mean, x_sd, x_N, log = FALSE)
```

Arguments

mu	Mean of the normal distribution to be evaluated (*not* the sample mean). May be a numeric scalar or vector.
sigma	Standard deviation of the normal distribution to be evaluated (*not* the sample SD). May be a numeric scalar or vector.
x_mean	Sample mean. May be a numeric scalar or vector.
x_sd	Sample standard deviation. May be a numeric scalar or vector.
x_N	Sample number of observations. May be a numeric scalar or vector.
log	TRUE/FALSE: Whether to return the log of the density function. Default FALSE (to return the density function value on the natural scale).

Details

‘x_mean‘, ‘x_sd‘, ‘X_N‘, ‘mu‘, and ‘sigma‘ should either be all the same size, or length 1. If they are different lengths, they will be repeated until their lengths match, with a warning.

Value

A numeric scalar or vector matching the length of the longest of ‘mu‘, ‘sigma‘, ‘x_mean‘, ‘x_sd‘, and ‘x_N‘.

Author(s)

Caroline Ring

do_data_info *do_data_info generic*

Description

do_data_info generic

Usage

```
do_data_info(obj, ...)
```

Arguments

obj	the pk object
...	Additional arguments currently not in use.

Value

Object of class [pk()] with an added '$data_info' list containing non-compartmental analysis results.

See Also

[do_data_info.pk()] for the 'do_data_info' method for class [pk()]

do_data_info.default *do_data_info default method*

Description

do_data_info default method

Usage

```
## Default S3 method:
do_data_info(obj, ...)
```

Arguments

obj	an object
...	Additional arguments currently not in use.

Value

An error, when a non-pk object is used for the first argument.

do_data_info.pk *calculate summary data info*

Description

Calculate summary data information, including non-compartmental analysis.

Usage

```
## S3 method for class 'pk'
do_data_info(obj, ...)
```

Arguments

obj	An object of class [pk()].
...	Additional arguments. Not currently in use.

Value

Object of class [pk()] with an added '$data_info' list containing non-compartmental analysis results.

Author(s)

Caroline Ring

do_fit *Fitting*

Description

This is the S3 generic method for 'do_fit'.

Usage

```
do_fit(obj, ...)
```

Arguments

obj	the pk object
...	Additional arguments currently not in use.

Value

The same [pk] object, with element 'fit' containing the fitted results for each model in 'stat_model'.

See Also

[do_fit.pk()] for the 'do_fit' method for class [pk()]

do_fit.default *do_fit default method*

Description

do_fit default method

Usage

```
## Default S3 method:
do_fit(obj, ...)
```

Arguments

obj	an object
...	Additional arguments currently not in use.

Value

An error, when a non-pk object is used for the first argument.

do_fit.pk *Do fitting*

Description

Fit PK model(s) for a 'pk' object

Usage

```
## S3 method for class 'pk'
do_fit(obj, n_cores = NULL, rate_names = NULL, max_multiplier = NULL, ...)
```

Arguments

obj	A [pk] object.
n_cores	Number of cores used for parallel computing.
rate_names	The names of the rate units. Leave NULL to utilize default 1/hour.
max_multiplier	Numeric value for upper prediction limit (this number multiplied by maximum concentrations in each experiment). Default set to NULL does not apply this limit.
...	Additional arguments. Not currently in use.

Details

This function estimates the parameters for each model in ‘stat_model‘ from the data, using numerical optimization implemented in [optimx::optimx()]. The optimization is done by maximizing the log-likelihood function implemented in [log_likelihood()] (technically, by minimizing the negative log-likelihood). Only the non-excluded observations are used.

Due to limitations of [optimx::optimx()], the log-likelihood function is forced to return finite values during this optimization. Impossible combinations of parameters (e.g., parameter values that produce negative predicted concentrations) should have a log-likelihood of ‘-Inf‘, but due to this limitation, they instead have a log-likelihood of ‘-Machine.doublexmax‘. This limitation means that the log-likelihood function is flat in regions of impossible parameter values. It is unlikely, but possible, that the optimizer might get "stuck" in such a flat region – report convergence, but return a "bad" set of parameter values that produces non-physical predictions.

Before trusting the results of any fit, it is recommended to check the log-likelihood using [logLik()] and the Akaike Information Criterion using [AIC()], which check the log-likelihood *without* forcing it to return finite values.

Value

The same [pk] object, with element ‘fit‘ containing the fitted results for each model in ‘stat_model‘.

Author(s)

Caroline Ring, Gilberto Padilla Mercado

`do_prefit` *Prefitting*

Description

Prefitting

Usage

```
do_prefit(obj, ...)
```

Arguments

`obj`	the pk object
`...`	Additional arguments currently not in use.

Value

The same ‘pk‘ object, but with a new element ‘prefit‘, containing the results of pre-fit calculations and checks for each model and for the error model.

See Also

[do_prefit.pk()] for the ‘do_prefit‘ method for class [pk()]

do_prefit.default *do_prefit default method*

Description

do_prefit default method

Usage

```
## Default S3 method:
do_prefit(obj, ...)
```

Arguments

obj	an object
...	Additional arguments currently not in use.

Value

An error, when a non-pk object is used for the first argument.

do_prefit.pk *Do pre-fitting*

Description

Do pre-fit calculations and checks

Usage

```
## S3 method for class 'pk'
do_prefit(obj, ...)
```

Arguments

obj	A ‘pk‘ object
...	Additional arguments. Not in use.

Details

This function does the following:

- Based on the error model in ‘stat_error_model‘ and the pre-processed data, determines the number of residual standard deviations ("sigmas") hyperparameters to be estimated. - Determines which "sigma" hyperparameter corresponds to each observation in the data. - Calculates lower/upper bounds and starting guesses for each "sigma" hyperparameter - For each model in ‘stat_model‘, calls its ‘params_fun‘, the function that, based on the data, determines whether to optimize each model parameter, and calculates lower/upper bounds and starting guesses for each model parameter to be optimized. Only non-excluded observations are passed to each model’s ‘params_fun‘.

Lower bounds for each "sigma" hyperparameter are set to ‘sqrt(.Machine$double_eps)‘.

Upper bounds for each "sigma" hyperparameter are calculated as the standard deviation of observations in the corresponding error SD group (see [combined_sd()]), with any specified transformations applied (dose-normalization and/or log10-transformation). If the combined SD is non-finite or less than the sigma lower bound, then the maximum concentration is used as an upper bound; if this still returns a non-finite value or a value less than the lower bound, then a constant value of 1000 is substituted.

The starting guess for each "sigma" hyperparameter is one-tenth of the upper bound.

Value

The same ‘pk‘ object, but with a new element ‘prefit‘, containing the results of pre-fit calculations and checks for each model and for the error model.

Author(s)

Caroline Ring

`do_preprocess` *Preprocess data generic*

Description

Preprocess data generic

Usage

```
do_preprocess(obj, ...)
```

Arguments

`obj`	the pk object.
`...`	Additional arguments currently not in use.

Value

The same ‘pk‘ object, with added elements ‘data‘ (containing the cleaned, gap-filled data) and ‘data_info‘ (containing summary information about the data, e.g. number of observations by route, media, detect/nondetect; empirical tmax, time of peak concentration for oral data; number of observations before and after empirical tmax)

See Also

[do_preprocess.pk()] for the ‘do_preprocess‘ method for class [pk()]

---

do_preprocess.default *do_preprocess default method*

Description

do_preprocess default method

Usage

```
## Default S3 method:
do_preprocess(obj, ...)
```

Arguments

obj	an object
...	Additional arguments currently not in use.

Value

An error, when a non-pk object is used for the first argument.

---

do_preprocess.pk *Do pre-processing*

Description

Pre-process data for a ‘pk‘ object

Usage

```
## S3 method for class 'pk'
do_preprocess(obj, ...)
```

Arguments

obj	A ‘pk‘ object
...	Additional arguments. Not in use currently.

Details

Data pre-processing for an object ‘obj‘ includes the following steps, in order:

- Coerce data to class ‘data.frame‘ (if it is not already) - Rename variables to harmonized "‘invivopkfit‘ aesthetic" variable names, using ‘obj$mapping‘ - Check that the data includes only routes in ‘obj$settings_preprocess$routes_keep‘ and media in ‘obj$settings_preprocess$media_keep‘ - Check that the data includes only one unit for concentration, one unit for time, and one unit for dose. - Coerce ‘Value‘, ‘Value_SD‘, ‘LOQ‘, ‘Dose‘, and ‘Time‘ to numeric, if they are not already. - Coerce ‘Species‘, ‘Route‘, and ‘Media‘ to lowercase. - Replace any negative ‘Value‘, ‘Value_SD‘, ‘Dose‘, or ‘Time‘ with ‘NA‘ - If any non-NA ‘Value‘ is currently less than its non-NA LOQ, then replace it with NA - Impute any NA ‘LOQ‘: as ‘calc_loq_factor‘ * minimum non-NA ‘Value‘ in each ‘loq_group‘ - For any cases where ‘N_Subject‘s is NA, impute ‘N_Subjects‘ = 1 - For anything with ‘N_Subjects‘ == 1, set ‘Value_SD‘ to 0 - Impute missing ‘Value_SD‘ as follows: For observations with ‘N_Subjects‘ > 1, take the minimum non-missing ‘Value_SD‘ for each ‘sd_group‘. If all SDs are missing in an ‘sd_group‘, then ‘Value_SD‘ for each observation in that group will be imputed as 0. - Mark data for exclusion according to the following criteria: - Exclude any remaining observations where both Value and LOQ are NA - For any cases where ‘N_Subjects‘ is NA, impute ‘N_Subjects‘ = 1 - Exclude any remaining observations with ‘N_Subjects‘ > 1 and ‘Value_SD‘ still NA. (This should never occur, if SD imputation is performed, but just in case.) - Exclude any observations with ‘N_Subjects‘ > 1 where reported ‘Value‘ is NA, because log-likelihood for non-detect multi-subject observations has not been implemented. - Exclude any observations with NA ‘Time‘ values - Exclude any observations with ‘Dose‘ = 0 - Apply any time transformations specified by user - Scale concentration by ‘ratio_conc_dose‘ - Apply any concentration transformations specified by the user. - If ‘Series_ID‘ is not included, then assign it as NA - Create variable ‘pLOQ‘ and set it equal to ‘LOQ‘

Value

The same ‘pk‘ object, with added elements ‘data‘ (containing the cleaned, gap-filled data) and ‘data_info‘ (containing summary information about the data, e.g. number of observations by route, media, detect/nondetect; empirical tmax, time of peak concentration for oral data; number of observations before and after empirical tmax)

Author(s)

John Wambaugh, Caroline Ring, Christopher Cook, Gilberto Padilla Mercado

---

eval_tkstats *eval_tkstats()*

Description

This is the S3 method generic for eval_tkstats()

Usage

```
eval_tkstats(obj, ...)
```

Arguments

obj	An object.
...	Additional arguments currently not in use.

Value

A ‘data.frame‘ with one row for each "winning" model in ‘model‘ from [get_winning_model()]. The ‘data.frame‘ will have the variables returned by the ‘tkstats_fun‘ for its corresponding model. (For the built-in models ‘model_flat‘, ‘model_1comp‘, and ‘model_2comp‘, these variables are ‘param_name‘ and ‘param_value‘.) Additionally, there will be a variable ‘method‘ denoting the [optimx::optimx()] method used to optimize the set of model parameters used to derive each set of TK statistics.

See Also

[eval_tkstats.pk()] for the method for class [pk()]

---

eval_tkstats.default *Default method for eval_tkstats()*

Description

Default method for eval_tkstats()

Usage

```
## Default S3 method:
eval_tkstats(obj, ...)
```

Arguments

obj	An object
...	Additional arguments currently not in use.

Value

An error, when a non-pk object is used for the first argument.

`eval_tkstats.pk` *Evaluate TK statistics*

Description

Evaluate TK statistics from a fitted model by comparing to NCA results

Usage

```
## S3 method for class 'pk'
eval_tkstats(
  obj,
  newdata = NULL,
  model = "winning",
  method = NULL,
  tk_group = NULL,
  exclude = TRUE,
  dose_norm = FALSE,
  finite_only = TRUE,
  suppress.messages = NULL,
  ...
)
```

Arguments

`obj`	A [pk()] model object. Must be fitted, or the function will exit with an error.
`newdata`	Optional: A ‘data.frame‘ containing new data for which to compute the TK stats. Must contain at least variables ‘Chemical‘, ‘Species‘, ‘Route‘, ‘Media‘, ‘Dose‘, ‘Dose.Units‘, ‘Conc.Units‘, either ‘Time_trans.Units‘ or ‘Time.Units‘, and any other variables named in ‘tk_grouping‘. Default ‘NULL‘, to use the data in ‘obj$data‘.
`model`	Character: One or more of the models fitted. Default ‘"winning"‘ to return results for only the winning model(s). Supply ‘NULL‘ to return TK stats for all models.
`method`	Character: One or more of the [optimx::optimx()] methods used. Default ‘NULL‘ to return TK stats for all methods.
`tk_group`	A list of variables provided using a ‘dplyr::vars()‘ call. The data (either ‘newdata‘ or ‘obj$data‘) will be grouped according to the unique combinations of these variables. For each unique combination of these variables in the data, a set of TK statistics will be computed. The default is ‘obj$settings_data_info$summary_group‘, to derive TK statistics for the same groups of data as non-compartmental analysis statistics. With the default, you can directly compare e.g. a model-predicted AUC_inf to the corresponding NCA-estimated AUC_inf. However, you may specify a different data grouping if you wish. Each group should have a unique combination of ‘Chemical‘, ‘Species‘, ‘Route‘, ‘Media‘, and ‘Dose‘, because the TK stats depend on these values, and it is required to have one unique set of TK stats per group.

`exclude` Logical: 'TRUE' to get the TK groupings after removing any observations in the data marked for exclusion (if there is a variable 'exclude' in the data, an observation is marked for exclusion when 'TRUE'). 'FALSE' to include all observations when getting the TK groupings, regardless of exclusion status. Default 'TRUE'.

`dose_norm` Logical: 'TRUE' (default) to dose-normalize before calculating both the NCA statistics and the fitted TK statistics (i.e. all dose-dependent statistics will be for a unit dose of 1 mg/kg, including Cmax, AUC, Css). 'FALSE' to calculate NCA and fitted TK stats separately for each dose group (you must specify 'Dose' as one of the variables in 'tk_group' for this to work). If 'dose_norm' is 'TRUE' and you also specify 'Dose' as one of the 'tk_group' variables, then the dose part of the grouping will be ignored in the output. (If 'TRUE', under the hood, this function will temporarily overwrite the 'Dose' column in its local copy of 'newdata' with 1's. This doesn't affect the data outside of this function. But it means that any values in the 'Dose' variable of 'newdata' will be ignored if 'TRUE'.)

`finite_only` Logical: 'TRUE' (default) returns only rows (observations) for which AUC is finite in both 'nca' and 'tkstats'. This also means it will by default never return instances where winning model == 'model_flat'.

`suppress.messages` Logical: whether to suppress message printing. If NULL (default), uses the setting in 'obj$settings_preprocess$suppress.messages'

`...` Additional arguments. Currently not in use.

Value

A 'data.frame' with one row for each "winning" model in 'model' from [get_winning_model()]. The 'data.frame' will have the variables returned by the 'tkstats_fun' for its corresponding model. (For the built-in models 'model_flat', 'model_1comp', and 'model_2comp', these variables are 'param_name' and 'param_value'.) Additionally, there will be a variable 'method' denoting the [optimx::optimx()] method used to optimize the set of model parameters used to derive each set of TK statistics.

Author(s)

Caroline Ring, Gilberto Padilla Mercado, John Wambaugh

See Also

Other methods for fitted pk objects: `AAFE.pk()`, `AFE.pk()`, `AIC.pk()`, `BIC.pk()`, `coef.pk()`, `coef_sd.pk()`, `get_fit.pk()`, `get_hessian.pk()`, `get_tkstats.pk()`, `logLik.pk()`, `predict.pk()`, `residuals.pk()`, `rmse.pk()`, `rsq.pk()`

`facet_data` *Facet a PK fit*

Description

Create a "faceted" [pk()] object.

Usage

```
facet_data(facets = vars(Chemical, Species), ...)
```

Arguments

`facets`	A set of variables or expressions quoted by [dplyr::vars()], defining groups of data that will each be fitted separately. These variables should appear in the *original* data (i.e., the ‘data‘ argument to [pk()]).
`...`	Additional arguments, not currently used.

Details

This function automates the process of doing PK fitting in "batch mode", when you have multiple concentration-dose-time datasets to fit, and you want to fit them all using the same set of instructions.

When you do something like

“‘ pk_cvt <- pk(cvt) + facet_data(facets = vars(chemicals_analyzed.dsstox_substance_id, subjects.species_harmonized)) “‘

Now ‘pk_cvt‘ is an object of class ‘pk_faceted‘: under the hood, a [tibble::tibble()] with one row for each group defined by a unique combination of the faceting variables, and a ‘list‘ column containing a [pk()] object corresponding to each group.

All of the [pk()] objects in the ‘list‘ column contain the same set of instructions, and they will all have the same status (*i.e.*, they are all in the same stage of the workflow at the same time). The only thing different among them is the data.

If you call a ‘pk‘ method on a ‘pk_faceted‘ object, the method will be applied in turn to the [pk()] object for each group.

If the method returns a [pk()] object (e.g. [preprocess_data.pk()], [data_info.pk()], [prefit.pk()], and [fit.pk()]), then the result for a ‘pk_faceted‘ object will be another ‘pk_faceted‘ object.

If the method returns something other than a [pk()] object (e.g. [coef.pk()], [coef_sd.pk()], [residuals.pk()], [predict.pk()], ...) then the result for a ‘pk_faceted‘ object will simply be a [tibble::tibble()] with a ‘list‘ column containing the result for each group – it won’t have class ‘pk_faceted‘.

This function is named by analogy to [ggplot2::facet_wrap()] and [ggplot2::facet_grid()]. Those functions split up a dataset into groups by one or more ‘factor‘ variables, and produce a "faceted" grid of plots for each group of data. This function does an analogous thing for a [pk()] analysis. The dataset is split into groups by the unique combinations of variables in ‘facets‘. For each group, a separate [pk()] object is created, using the instructions provided by the user. When methods like [preprocess_data()], [data_info()], [prefit()], and [fit()] are called on the resulting "faceted"

Value

An object of class ‘c("pkproto", "pk_facet_data")‘. Under the hood, a named ‘list‘ containing the arguments provided to this function. Almost always added to a [pk()] object using [‘+.pk‘].

Author(s)

Caroline Ring, Gilbert Padilla Mercado, Paul Kruse

---

fill_params_1comp *Fill parameters for 1-compartment model*

Description

Fill parameters for 1-compartment model

Usage

```
fill_params_1comp(params)
```

Arguments

params	Named numeric vector of parameters for the 1-compartment model

Value

A named numeric vector of parameters, with any 1-compartment model parameters not present in ‘params‘ filled with ‘NA_real_‘. If any two of ‘Fgutabs‘, ‘Vdist‘, and ‘Fgutabs_Vdist‘ were present in ‘params‘, the third will be imputed to agree with the other two.

Author(s)

Caroline Ring

---

fill_params_1comp_cl *Fill parameters for 1-compartment model with specific clearance*

Description

Fill parameters for 1-compartment model with specific clearance

Usage

```
fill_params_1comp_cl(params)
```

Arguments

params	Named numeric vector of parameters for the 1-compartment model

Value

A named numeric vector of parameters, with any 1-compartment model parameters not present in ‘params‘ filled with ‘NA_real_‘. If any two of ‘Fgutabs‘, ‘Vdist‘, and ‘Fgutabs_Vdist‘ were present in ‘params‘, the third will be imputed to agree with the other two.

Author(s)

Caroline Ring

`fill_params_2comp` *Fill parameters for 2-compartment model*

Description

Fill parameters for 2-compartment model

Usage

```
fill_params_2comp(params)
```

Arguments

`params`	Named numeric vector of parameters for the 2-compartment model

Value

A named numeric vector of parameters, with any 2-compartment model parameters not present in ‘params‘ filled with ‘NA_real_‘. If any two of ‘Fgutabs‘, ‘V1‘, and ‘Fgutabs_V1‘ were present in ‘params‘, the third will be imputed to agree with the other two.

Author(s)

Caroline Ring

`fill_params_flat` *Fill parameters for flat model*

Description

Fill parameters for flat model

Usage

```
fill_params_flat(params)
```

Arguments

params	Named numeric vector of parameters for the flat model

Value

A named numeric vector of parameters, with any flat model parameters not present in ‘params‘ filled with ‘NA_real_‘. If any two of ‘Fgutabs‘, ‘Vdist‘, and ‘Fgutabs_Vdist‘ were present in ‘params‘, the third will be imputed to agree with the other two.

Author(s)

Caroline Ring

---

fit_group *Fit a single group of data*

Description

Fit a single group of data

Usage

```
fit_group(
  data,
  par_DF,
  sigma_DF,
  fit_decision,
  this_model,
  settings_optimx,
  modelfun,
  dose_norm,
  log10_trans,
  max_mult,
  suppress.messages
)
```

Arguments

data	A single group of data
par_DF	par_DF for a single group of data
sigma_DF	sigma_DF for a single group of data
fit_decision	Whether the fit is able to be calculated or excluded.
this_model	Name of the ‘pk_model‘ object to fit
settings_optimx	The settings for optimization.

modelfun	Name of the model concentration function
dose_norm	TRUE or FALSE – whether to dose-normalize concentrations before evaluating log-likelihood
log10_trans	TRUE or FALSE – whether to 1og10-transform concentrations before evaluating log-likelihood
max_mult	Multiplier for maximum acceptable value. Default set to NULL and currently unused.
suppress.messages	TRUE or FALSE – whether to suppress messages or emit them

Value

An object of class ‘optimx‘ (i.e. a data.frame with fit results)

`fold_error` *Fold error*

Description

This is the S3 method generic for ‘fold_error‘.

Usage

```
fold_error(obj, ...)
```

Arguments

obj	an object
...	Additional arguments currently not in use.

Value

A data.frame with one row for each ‘data_group‘, ‘model‘ and ‘method‘. A column contains the fold errors (observed/predicted) of the model fitted by the corresponding method. These residuals are concentrations in the same units as ‘obj$data$Conc.Units‘; any concentration transformations (in ‘obj$scale$conc‘) are *not* applied.

See Also

[fold_error.pk()] for the ‘fold_error‘ method for class [pk()]

fold_error.default *fold_error default method*

Description

fold_error default method

Usage

```
## Default S3 method:
fold_error(obj, ...)
```

Arguments

obj	an object
...	Additional arguments currently not in use.

Value

An error, when a non-pk object is used for the first argument.

fold_error.pk *Fold errors*

Description

Calculate fold errors for a fitted ‘pk‘ object.

Usage

```
## S3 method for class 'pk'
fold_error(
  obj,
  newdata = NULL,
  model = NULL,
  method = NULL,
  exclude = TRUE,
  sub_pLOQ = TRUE,
  suppress.messages = NULL,
  ...
)
```

Arguments

obj	A ‘pk‘ object
newdata	Optional: A ‘data.frame‘ with new data for which to compute fold errors. If NULL (the default), then fold errors will be computed for the data in ‘obj$data‘. ‘newdata‘ is required to contain at least the following variables: ‘Time‘, ‘Dose‘, ‘Route‘, and ‘Media‘.
model	Optional: Specify one or more of the fitted models for which to make predictions and calculate fold errors. If NULL (the default), fold errors will be returned for all of the models in ‘obj$stat_model‘.
method	Optional: Specify one or more of the [optimx::optimx()] methods for which to make predictions and calculate RMSEs. If NULL (the default), fold errors will be returned for all of the models in ‘obj$settings_optimx$method‘.
exclude	Logical: ‘TRUE‘ to return ‘NA_real_‘ for any observations in the data marked for exclusion (if there is a variable ‘exclude‘ in the data, an observation is marked for exclusion when ‘exclude ‘FALSE‘ to return the prediction for each observation, regardless of exclusion. Default ‘TRUE‘.
sub_pLOQ	Logical: whether or not to include predictions below pLOQ. when TRUE, values below pLOQ will be replaced by pLOQ.
suppress.messages	Logical: whether to suppress message printing. If NULL (default), uses the setting in ‘object$settings_preprocess$suppress.messages‘
...	Additional arguments. Currently not in use.

Details

Here, fold error is defined as ‘observed/predicted‘.

Scaling and transformation of concentration variables in ‘newdata‘

This function differs from some of the other methods for a fitted [pk()] object that accept ‘newdata‘, in that there is no ‘use_scale_conc‘ argument for [fold_error.pk()]. Fold errors are always computed on the natural, un-transformed concentration scale (but note that fold error on a dose-normalized scale will be the same as fold error on a non-dose-normalized scale).

Value

A data.frame with one row for each ‘data_group‘, ‘model‘ and ‘method‘. A column contains the fold errors (observed/predicted) of the model fitted by the corresponding method. These residuals are concentrations in the same units as ‘obj$data$Conc.Units‘; any concentration transformations (in ‘obj$scale$conc‘) are *not* applied.

Author(s)

Caroline Ring

get_data *get_data()*

Description

This is the S3 method generic for get_data()

Usage

```
get_data(obj, ...)
```

Arguments

obj	An object.
...	Additional arguments currently not in use.

Value

A ‘data.frame‘: the ‘data‘ element of ‘obj‘

See Also

[get_data.pk()] for the method for class [pk()]

get_data.default *Default method for get_data()*

Description

Default method for get_data()

Usage

```
## Default S3 method:
get_data(obj, ...)
```

Arguments

obj	An object
...	Additional arguments currently not in use.

Value

An error, when a non-pk object is used for the first argument.

get_data.pk *Get data*

Description

Extract pre-processed data from a [pk()] object

Usage

```
## S3 method for class 'pk'
get_data(obj, ...)
```

Arguments

obj	A [pk()] object that has been pre-processed
...	Additional arguments. Currently not in use.

Value

A ‘data.frame‘: the ‘data‘ element of ‘obj‘

Author(s)

Caroline Ring

get_data_group *get_data_group()*

Description

This is the S3 method generic for get_data_group()

Usage

```
get_data_group(obj, ...)
```

Arguments

obj	An object.
...	Additional arguments currently not in use.

Value

An object of class ‘call‘ giving the data grouping as a ‘dplyr::vars()‘ specification

See Also

[get_data_group.pk()] for the method for class [pk()]

get_data_group.default *Default method for get_data_group()*

Description

Default method for get_data_group()

Usage

```
## Default S3 method:
get_data_group(obj, ...)
```

Arguments

obj	An object
...	Additional arguments currently not in use.

Value

An error, when a non-pk object is used for the first argument.

get_data_group.pk *Get data grouping*

Description

Get data grouping

Usage

```
## S3 method for class 'pk'
get_data_group(obj, ...)
```

Arguments

obj	An initialized ‘pk‘ object
...	Additional arguments not currently in use.

Value

An object of class ‘call‘ giving the data grouping as a ‘dplyr::vars()‘ specification

get_data_info *get_data_info()*

Description

This is the S3 method generic for get_data_info()

Usage

```
get_data_info(obj, ...)
```

Arguments

obj	An object.
...	Additional arguments currently not in use.

Value

A ‘list‘ of ‘tibble‘s: the ‘data_info‘ element of ‘obj‘

See Also

[get_data_info.pk()] for the method for class [pk()]

get_data_info.default *Default method for get_data_info()*

Description

Default method for get_data_info()

Usage

```
## Default S3 method:
get_data_info(obj, ...)
```

Arguments

obj	An object
...	Additional arguments currently not in use.

Value

An error, when a non-pk object is used for the first argument.

get_data_info.pk *Get data_info*

Description

Extract summary data information results from a [pk()] object

Usage

```
## S3 method for class 'pk'
get_data_info(obj, ...)
```

Arguments

obj	A [pk()] object that has had ‘data_info()‘ run on it
...	Additional arguments. Currently not in use.

Value

A ‘list‘ of ‘tibble‘s: the ‘data_info‘ element of ‘obj‘

Author(s)

Caroline Ring

get_data_original *get_data_original()*

Description

This is the S3 method generic for get_data_original()

Usage

```
get_data_original(obj, ...)
```

Arguments

obj	An object.
...	Additional arguments currently not in use.

Value

A ‘data.frame‘ – the ‘data_original‘ element of ‘obj‘

See Also

[get_data_original.pk()] for the method for class [pk()]

get_data_original.default *Default method for get_data_original()*

Description

Default method for get_data_original()

Usage

```
## Default S3 method:
get_data_original(obj, ...)
```

Arguments

obj	An object
...	Additional arguments currently not in use.

Value

An error, when a non-pk object is used for the first argument.

get_data_original.pk *Get data_original*

Description

Get data_original

Usage

```
## S3 method for class 'pk'
get_data_original(obj, ...)
```

Arguments

obj	A [pk()] object
...	Additional arguments. Not currently in use.

Value

A ‘data.frame‘ – the ‘data_original‘ element of ‘obj‘

Author(s)

Caroline Ring

get_data_sigma_group *get_data_sigma_group()*

Description

This is the S3 method generic for get_data_sigma_group()

Usage

```
get_data_sigma_group(obj, ...)
```

Arguments

obj	An object.
...	Additional arguments currently not in use.

Value

A 'factor' vector giving the error SD group ID for each observation, as the interaction of the factors specified in 'obj$stat_error_model$error_group'.

See Also

[get_data_sigma_group.pk()] for the method for class [pk()]

get_data_sigma_group.default *Default method for get_data_sigma_group()*

Description

Default method for get_data_sigma_group()

Usage

```
## Default S3 method:
get_data_sigma_group(obj, ...)
```

Arguments

obj	An object
...	Additional arguments currently not in use.

Value

An error, when a non-pk object is used for the first argument.

get_data_sigma_group.pk *Get data_sigma_group*

Description

Get data_sigma_group

Usage

```
## S3 method for class 'pk'
get_data_sigma_group(obj, newdata = NULL, ...)
```

Arguments

obj	A [pk()] object
newdata	Optional: A ‘data.frame‘ with new data for which to get the ‘data_sigma_group‘s. If NULL (the default), then the groups will be evaluated for the ‘obj$data‘.
...	Additional arguments. Not currently in use.

Value

A ‘factor‘ vector giving the error SD group ID for each observation, as the interaction of the factors specified in ‘obj$stat_error_model$error_group‘.

Author(s)

Caroline Ring

get_data_summary *get_data_summary()*

Description

This is the S3 method generic for get_data_summary()

Usage

```
get_data_summary(obj, ...)
```

Arguments

obj	An object.
...	Additional arguments currently not in use.

Details

'get_data_summary()' is an alias for 'data_summary()'

Value

A 'data.frame' with variables including all the grouping variables in 'summary_group', 'group_id'; 'param_name' (the name of the summary statistic; see Details); 'param_value' (the summary statistic value); 'param_units' (the units of the summary statistic, derived from the units of the data).

See Also

[data_summary.pk()] for the method for class [pk()]

---

get_data_summary.default

Default method for get_data_summary()

Description

Default method for get_data_summary()

Usage

```
## Default S3 method:
get_data_summary(obj, ...)
```

Arguments

obj	An object
...	Additional arguments currently not in use.

Value

An error, when a non-pk object is used for the first argument.

`get_elbow` *Get an elbow point*

Description

Given a set of data specified as two vectors of ‘x‘ and ‘y‘ values, find an elbow point.

Usage

```
get_elbow(x, y, ...)
```

Arguments

`x`	The ‘x‘ values from the data where an elbow point is to be found.
`y`	The ‘y‘ values from the data where an elbow point is to be found.
`...`	Optional: additional arguments that will be passed to [stats::approx()] if it is used.

Details

This is a helper function for [get_starts()] to find elbow points.

Given a set of (x,y) data points, an "elbow point" can be defined by drawing a line connecting the points for minimum and maximum x, and then finding the x value of the observation where the distance to that line is greatest.

[get_starts()] uses elbow points as a way to automate separation of concentration-time data into different kinetic phases in order to calculate starting points for fitting TK model parameters. For example, if concentration-time data are described by a two-compartment TK model, then early and late elimination phases will be separated by an elbow point. This helper function finds the elbow points. (When this function is called from [get_starts()], ‘x‘ will be a vector of time values, and ‘y‘ will be a vector of log-transformed dose-normalized concentration values.)

Value

A list with two named numeric scalar elements, ‘x‘ and ‘y‘. ‘x‘ contains the ‘x‘ value at the elbow point. ‘y‘ contains the ‘y‘ value at the elbow point. The elbow point is *not* necessarily one of the input data points; it may be interpolated.

Author(s)

Caroline Ring

get_error_group *get_error_group()*

Description

This is the S3 method generic for get_error_group()

Usage

```
get_error_group(obj, ...)
```

Arguments

obj	An object.
...	Additional arguments currently not in use.

Value

The stat_error_model error grouping

See Also

[get_error_group.pk()] for the method for class [pk()]

get_error_group.default *Default method for get_error_group()*

Description

Default method for get_error_group()

Usage

```
## Default S3 method:
get_error_group(obj, ...)
```

Arguments

obj	An object
...	Additional arguments currently not in use.

Value

An error, when a non-pk object is used for the first argument.

get_error_group.pk *Get error group*

Description

Get error group

Usage

```
## S3 method for class 'pk'
get_error_group(obj, ...)
```

Arguments

obj	A [pk()] object
...	Additional arguments. Not in use currently.

Value

The stat_error_model error grouping

Author(s)

Caroline Ring

get_fit *get_fit()*

Description

This is the S3 method generic for get_fit()

Usage

```
get_fit(obj, ...)
```

Arguments

obj	An object.
...	Additional arguments currently not in use.

Value

A named list of objects of class ‘optimx‘, named for the models in ‘model‘. As described in [optimx::optimx()] If only one model is specified, the return value will still be a list, but with only one element.

See Also

[get_fit.pk()] for the method for class [pk()]

`get_fit.default` *Default method for get_fit()*

Description

Default method for get_fit()

Usage

```
## Default S3 method:
get_fit(obj, ...)
```

Arguments

`obj`	An object
`...`	Additional arguments currently not in use.

Value

An error, when a non-pk object is used for the first argument.

`get_fit.pk` *Get fits from a 'pk' object*

Description

Get the [optimx::optimx()] output from a fitted 'pk' object

Usage

```
## S3 method for class 'pk'
get_fit(obj, model = NULL, ...)
```

Arguments

`obj`	A [pk] object.
`model`	Optional: Specify one or more of the fitted models for which to make predictions. If NULL (the default), predictions will be returned for all of the models in 'obj$stat_model'.
`...`	Additional arguments. Not in use.

Details

This function returns the object(s) returned by [optimx::optimx()] for the specified model(s) and method(s), for a fitted ‘pk‘ object. See [optimx::optimx()] for details. Briefly, an ‘optimx‘ object is a ‘data.frame‘ with one row for each method used, and variables that give the optimized values for each parameter, along with several diagnostic variables (e.g. the objective function value at the optimized parameter values; the number of function evaluations/iterations; an integer code describing convergence status). The object will have attributes ‘details‘ (providing any messages returned by the methods) and ‘npar‘ (the number of parameters optimized).

Value

A named list of objects of class ‘optimx‘, named for the models in ‘model‘. As described in [optimx::optimx()] If only one model is specified, the return value will still be a list, but with only one element.

Author(s)

Caroline Ring, Gilberto Padilla Mercado

See Also

Other methods for fitted pk objects: `AAFE.pk()`, `AFE.pk()`, `AIC.pk()`, `BIC.pk()`, `coef.pk()`, `coef_sd.pk()`, `eval_tkstats.pk()`, `get_hessian.pk()`, `get_tkstats.pk()`, `logLik.pk()`, `predict.pk()`, `residuals.pk()`, `rmse.pk()`, `rsq.pk()`

`get_hessian` *get_hessian()*

Description

This is the S3 method generic for get_hessian()

Usage

```
get_hessian(obj, ...)
```

Arguments

`obj`	An object.
`...`	Additional arguments currently not in use.

Value

A dataframe with one row for each ‘data_group‘, ‘model‘ and ‘method‘. The remaining column is a ‘list‘ column containing the Hessian for each row.

See Also

[hessian.pk()] for the method for class [pk()]

get_hessian.default *Default method for get_hessian()*

Description

Default method for get_hessian()

Usage

```
## Default S3 method:
get_hessian(obj, ...)
```

Arguments

obj	An object
...	Additional arguments currently not in use.

Value

An error, when a non-pk object is used for the first argument.

get_hessian.pk *Get Hessian matrixes*

Description

Extract Hessian matrixes from a fitted 'pk' object

Usage

```
## S3 method for class 'pk'
get_hessian(obj, model = NULL, method = NULL, suppress.messages = TRUE, ...)
```

Arguments

obj	A [pk] object
model	Optional: Specify one or more of the fitted models whose coefficients to return. If NULL (the default), coefficients will be returned for all of the models in 'obj$stat_model'.
method	Optional: Specify one or more of the [optimx::optimx()] methods whose coefficients to return. If NULL (the default), coefficients will be returned for all of the models in 'obj$settings_optimx$method'.
suppress.messages	Logical. 'TRUE' (the default) to suppress informative messages. 'FALSE' to see them.
...	Additional arguments. Not in use right now.

Details

This function computes a numerical approximation to the model Hessian for each data group and each model in a fitted ‘pk‘ object. The Hessian is the matrix of second derivatives of the model objective function with respect to each model parameter. Here, the objective function is the negative log-likelihood implemented in [log_likelihood()], evaluated jointly across the data that was used to fit the model.

Value

A dataframe with one row for each ‘data_group‘, ‘model‘ and ‘method‘. The remaining column is a ‘list‘ column containing the Hessian for each row.

Author(s)

Caroline Ring and Gilberto Padilla Mercado

References

Gill J, King G. (2004) What to Do When Your Hessian is Not Invertible: Alternatives to Model Respecification in Nonlinear Estimation. Sociological Methods & Research 33(1):54-87. DOI: 10.1177/0049124103262681

See Also

Other methods for fitted pk objects: `AAFE.pk()`, `AFE.pk()`, `AIC.pk()`, `BIC.pk()`, `coef.pk()`, `coef_sd.pk()`, `eval_tkstats.pk()`, `get_fit.pk()`, `get_tkstats.pk()`, `logLik.pk()`, `predict.pk()`, `residuals.pk()`, `rmse.pk()`, `rsq.pk()`

`get_mapping` *get_mapping()*

Description

This is the S3 method generic for get_mapping()

Usage

```
get_mapping(obj, ...)
```

Arguments

`obj`	An object.
`...`	Additional arguments currently not in use.

Value

A list of ‘quosure‘s – the ‘mapping‘ element of ‘obj‘

See Also

[get_mapping.pk()] for the method for class [pk()]

`get_mapping.default` *Default method for get_mapping()*

Description

Default method for get_mapping()

Usage

```
## Default S3 method:
get_mapping(obj, ...)
```

Arguments

`obj`	An object
`...`	Additional arguments currently not in use.

Value

An error, when a non-pk object is used for the first argument.

`get_mapping.pk` *Get mapping*

Description

Get mapping

Usage

```
## S3 method for class 'pk'
get_mapping(obj, ...)
```

Arguments

`obj`	A [pk()] object
`...`	Additional arguments. Currently not in use.

Value

A list of ‘quosure‘s – the ‘mapping‘ element of ‘obj‘

Author(s)

Caroline Ring

get_nca *get_nca()*

Description

This is the S3 method generic for get_nca()

Usage

```
get_nca(obj, ...)
```

Arguments

obj	An object.
...	Additional arguments currently not in use.

Value

A ‘data.frame‘: the ‘data‘ element of ‘obj‘

See Also

[get_nca.pk()] for the method for class [pk()]

get_nca.default *Default method for get_nca()*

Description

Default method for get_nca()

Usage

```
## Default S3 method:
get_nca(obj, ...)
```

Arguments

obj	An object
...	Additional arguments currently not in use.

Value

An error, when a non-pk object is used for the first argument.

get_nca.pk *Get NCA*

Description

Extract non-compartmental analysis results from a [pk()] object

Usage

```
## S3 method for class 'pk'
get_nca(obj, ...)
```

Arguments

obj	A [pk()] object that has had ‘data_info()‘ run on it
...	Additional arguments. Currently not used.

Value

A ‘data.frame‘: the ‘data‘ element of ‘obj‘

Author(s)

Caroline Ring, Gilberto Padilla Mercado

get_params_1comp *Get parameters for 1-compartment model*

Description

Get parameters for 1-compartment model and determine whether each is to be estimated from the data

Usage

```
get_params_1comp(
  data,
 lower_bound = ggplot2::aes(kelim = log(2)/(2 * max(Time_trans)), Vdist = 0.01, Fgutabs
  = 0, kgutabs = log(2)/(2 * max(Time_trans)), Fgutabs_Vdist = 0.01, Rblood2plasma =
    0.01),
 upper_bound = ggplot2::aes(kelim = log(2)/(0.5 * min(Time_trans[Time_trans > 0])),
  Vdist = 100, Fgutabs = 1, kgutabs = log(2)/(0.5 * min(Time_trans[Time_trans > 0])),
    Fgutabs_Vdist = 100, Rblood2plasma = 100),
 param_units = ggplot2::aes(kelim = paste0("1/", unique(Time_trans.Units)), Vdist =
  paste0("(", unique(Dose.Units), ")", "/", "(", unique(Conc.Units), ")"), Fgutabs =
```

```
    "unitless fraction", kgutabs = paste0("1/", unique(Time_trans.Units)), Fgutabs_Vdist
     = paste0("(", unique(Conc.Units), ")", "/", "(", unique(Dose.Units), ")"),
     Rblood2plasma = "unitless ratio")
)
```

Arguments

data	The data set to be fitted (e.g. the result of [preprocess_data()])
lower_bound	A mapping specified using a call to [ggplot2::aes()], giving the lower bounds for each variable, as expressions which may include variables in 'data'.
upper_bound	A mapping specified using a call to [ggplot2::aes()], giving the upper bounds for each variable, as expressions which may include variables in 'data'.
param_units	A mapping specified using a call to [ggplot2::aes()], giving the units for each variable, as expressions which may include variables in 'data'.

Details

The full set of model parameters for the 1-compartment model includes 'Vdist', 'kelim', 'kgutabs', 'Fgutabs', and 'Rblood2plasma'. Whether each one can be estimated from the data depends on what routes of administration are included in the data.

IV data, no oral data

If IV dosing data are available, but no oral dosing data are available, then only the parameters 'Vdist' and 'kelim' will be estimated from the data. The parameters 'kgutabs' and 'Fgutabs' cannot be estimated from IV data alone, and will not be used in evaluating the model.

Oral data, no IV data

If oral dosing data are available, but no IV dosing data are available, then the parameters 'kelim' and 'kgutabs' can be estimated from the data. However, the parameters 'Fgutabs' and 'Vdist' cannot be identified separately. From oral data alone, only the ratio 'Fgutabs/Vdist' can be identified. This ratio is represented by a single parameter named 'Fgutabs_Vdist'. 'Fgutabs' and 'Vdist' will not be used to evaluate the model nor be estimated from data, but 'Fgutabs_Vdist' will be estimated from data, along with 'kelim' and 'kgutabs'.

Oral data and IV data

If both oral and IV dosing data are available, then 'Vdist', 'kelim', 'kgutabs', and 'Fgutabs' will all be estimated from the data.

Blood and plasma data

If both blood and plasma data are available, then 'Rblood2plasma' will be estimated from the data.

Only one of blood or plasma data

If only one of blood or plasma data are available, then 'Rblood2plasma' will be held constant at 1, not estimated from the data.

Default lower and upper bounds for each parameter

Default lower and upper bounds for 'kelim' and 'kgutabs'

Default bounds for time constants 'kelim' and 'kgutabs' are set based on the time scale of the available data.

The lower bounds are based on the assumption that elimination and absorption are very slow compared to the time scale of the study. Specifically, the lower bounds assume that elimination and absorption half-lives are twice as long as the duration of the available study data, or ‘2*max(Time_trans)‘. Under this assumption, the corresponding elimination and absorption time constants would be ‘log(2)/(2*max(Time_trans))‘. Therefore, the default lower bounds for ‘kelim‘ and ‘kgutabs‘ are ‘log(2)/(2*max(Time_trans))‘.

Upper bounds are based on the opposite assumption: that elimination and absorption are very fast compared to the time scale of the study. Specifically, the upper bounds assume that the elimination and absorption half-lives are half as long as the time of the first observation after time 0, or ‘0.5*min(Time_trans[Time_trans>0])‘. Under this asumption, the corresponding elimination and absorption time constants would be ‘log(2)/(0.5*min(Time_trans[Time_trans>0]))‘. Therefore, the default lower bounds for ‘kelim‘ and ‘kgutabs‘ are ‘log(2)/(0.5*min(Time_trans[Time_trans>0]))‘.

Default lower and upper bounds for ‘Vdist‘

By default, the lower bound for ‘Vdist‘ is 0.01, and the upper bound for ‘Vdist‘ is 100. These values were chosen based on professional judgment.

Default lower and upper bounds for ‘Fgutabs‘

By default, the lower bound for ‘Fgutabs‘ is 0.0, and the upper bound for ‘Fgutabs‘ is 1. These are simply the bounds of the physically-meaningful range for a fraction.

Default lower and upper bounds for ‘Fgutabs_Vdist‘

By default, the lower bound for the ratio ‘Fgutabs_Vdist‘ is 0.01, and the upper bound is 100. These values were chosen based on professional judgment.

Default lower and upper bounds for ‘Rblood2plasma‘

By default, the lower bound for the blood:plasma partition coefficient ‘Rblood2plasma‘ is 0.01, and the upper bound is 100. These values were chosen based on professional judgment.

Starting values for each parameter

Starting values for each parameter (starting guesses for the numerical optimizer) are derived from the data using [get_starts_1comp()].

If the starting values returned by [get_starts_1comp()] fall outside the bounds for any parameter(s), then the starting value will be reset to a value halfway between the lower and upper bounds for that parameter.

Value

A ‘data.frame‘with the following variables: - ‘param_name‘: Character: Names of the model parameters - ‘param_units‘: Character: Units of the model parameters - ‘optimize_param‘: TRUE if each parameter is to be estimated from the data; FALSE otherwise - ‘use_param‘: TRUE if each parameter is to be used in evaluating the model; FALSE otherwise -‘lower_bounds‘: Numeric: The lower bounds for each parameter - ‘upper_bounds‘: Numeric: The upper bounds for each parameter - ‘start‘: Numeric: The starting guesses for each parameter

Author(s)

Caroline Ring

See Also

Other 1-compartment model functions: auc_1comp(), auc_1comp_cl(), cp_1comp(), cp_1comp_cl(), get_params_1comp_cl(), get_params_1comp_fup(), get_starts_1comp(), get_starts_1comp_cl(), get_starts_1comp_fup()

Other get_params functions: get_params_1comp_cl(), get_params_1comp_fup(), get_params_2comp(), get_params_flat()

Other built-in model functions: auc_1comp(), auc_1comp_cl(), auc_2comp(), auc_flat(), cp_1comp(), cp_1comp_cl(), cp_2comp(), cp_2comp_dt(), cp_flat(), get_params_1comp_cl(), get_params_1comp_fup(), get_params_2comp(), get_params_flat(), get_starts_1comp(), get_starts_1comp_cl(), get_starts_1comp_fup(), get_starts_2comp(), get_starts_flat(), tkstats_2comp(), transformed_params_2comp()

---

get_params_1comp_cl *Get parameters for 1-compartment model with clearance assumption*

Description

Get parameters for 1-compartment model and determine whether each is to be estimated from the data

Usage

```
get_params_1comp_cl(
  data,
 lower_bound = ggplot2::aes(Q_totli = NA, Q_gfr = NA, Fup = 0, Clint = 0, Vdist = 0.01,
    Fgutabs = 0, kgutabs = log(2)/(2 * max(Time_trans)), Fgutabs_Vdist = 0.01,
    Rblood2plasma = 0.01),
 upper_bound = ggplot2::aes(Q_totli = NA, Q_gfr = NA, Fup = 1, Clint = 1e+05, Vdist =
    100, Fgutabs = 1, kgutabs = log(2)/(0.5 * min(Time_trans[Time_trans > 0])),
    Fgutabs_Vdist = 100, Rblood2plasma = 100),
 param_units = ggplot2::aes(Q_totli = "L/h/kg ^3/4", Q_gfr = "L/h/kg ^3/4", Fup =
  "unitless fraction", Clint = "L/h/kg", Vdist = paste0("(", unique(Dose.Units), ")",
    "/", "(", unique(Conc.Units), ")"), Fgutabs = "unitless fraction", kgutabs =
    paste0("1/", unique(Time_trans.Units)), Fgutabs_Vdist = paste0("(",
    unique(Conc.Units), ")", "/", "(", unique(Dose.Units), ")"), Rblood2plasma =
    "unitless ratio"),
 restrictive = FALSE
)
```

Arguments

data	The data set to be fitted (e.g. the result of [preprocess_data()])
lower_bound	A mapping specified using a call to [ggplot2::aes()], giving the lower bounds for each variable, as expressions which may include variables in 'data'.
upper_bound	A mapping specified using a call to [ggplot2::aes()], giving the upper bounds for each variable, as expressions which may include variables in 'data'.

param_units	A mapping specified using a call to [ggplot2::aes()], giving the units for each variable, as expressions which may include variables in ‘data‘.
restrictive	A boolean value (Default: FALSE) that determines whether to assume restrictive clearance when setting starting values for parameters.

Details

The full set of model parameters for the 1-compartment model includes ‘Vdist‘, ‘Clint‘, ‘kgutabs‘, ‘Fgutabs‘, ‘Q_totli‘, ‘Q_gfr‘, ‘Fup‘, and ‘Rblood2plasma‘. Whether each one can be estimated from the data depends on which routes of administration are included in the data.

Constant parameters from ‘httk‘ ‘Q_totli‘ is the flow through the portal vein from the gut to the liver and is estimated in this model via [httk::tissue.data] in a species-specific manner. ‘Q_gfr‘ is the glomular filtration rate of kidneys. It is estimated via [httk::physiology.data] in a species specific manner. ‘Fup‘ and ‘Rblood2plasma‘ are both calculated in [httk::parameterize_1comp()] For each of these parameters default lower bounds are 10

IV data, no oral data

If IV dosing data are available, but no oral dosing data are available, then only the parameters ‘Vdist‘ and ‘kelim‘ will be estimated from the data. The parameters ‘kgutabs‘ and ‘Fgutabs‘ cannot be estimated from IV data alone, and will not be used in evaluating the model.

Oral data, no IV data

If oral dosing data are available, but no IV dosing data are available, then the parameters ‘kelim‘ and ‘kgutabs‘ can be estimated from the data. However, the parameters ‘Fgutabs‘ and ‘Vdist‘ cannot be identified separately. From oral data alone, only the ratio ‘Fgutabs/Vdist‘ can be identified. This ratio is represented by a single parameter named ‘Fgutabs_Vdist‘. ‘Fgutabs‘ and ‘Vdist‘ will not be used to evaluate the model nor be estimated from data, but ‘Fgutabs_Vdist‘ will be estimated from data, along with ‘kelim‘ and ‘kgutabs‘.

Oral data and IV data

If both oral and IV dosing data are available, then ‘Vdist‘, ‘kelim‘, ‘kgutabs‘, and ‘Fgutabs‘ will all be estimated from the data.

Blood and plasma data

If both blood and plasma data are available, then ‘Rblood2plasma‘ will be estimated from the data.

Only one of blood or plasma data

If only one of blood or plasma data are available, then ‘Rblood2plasma‘ will be held constant at 1, not estimated from the data.

Default lower and upper bounds for each parameter

Default lower and upper bounds for ‘kelim‘ and ‘kgutabs‘

Default bounds for time constants ‘kelim‘ and ‘kgutabs‘ are set based on the time scale of the available data.

The lower bounds are based on the assumption that elimination and absorption are very slow compared to the time scale of the study. Specifically, the lower bounds assume that elimination and absorption half-lives are twice as long as the duration of the available study data, or ‘2*max(Time_trans)‘. Under this assumption, the corresponding elimination and absorption time constants would be ‘log(2)/(2*max(Time_trans))‘. Therefore, the default lower bounds for ‘kelim‘ and ‘kgutabs‘ are ‘log(2)/(2*max(Time_trans))‘.

Upper bounds are based on the opposite assumption: that elimination and absorption are very fast compared to the time scale of the study. Specifically, the upper bounds assume that the elimination and absorption half-lives are half as long as the time of the first observation after time 0, or ‘0.5*min(Time_trans[Time_trans>0])‘. Under this asumption, the corresponding elimination and absorption time constants would be ‘log(2)/(0.5*min(Time_trans[Time_trans>0]))‘. Therefore, the default lower bounds for ‘kelim‘ and ‘kgutabs‘ are ‘log(2)/(0.5*min(Time_trans[Time_trans>0]))‘.

Default lower and upper bounds for ‘Vdist‘

By default, the lower bound for ‘Vdist‘ is 0.01, and the upper bound for ‘Vdist‘ is 100. These values were chosen based on professional judgment.

Default lower and upper bounds for ‘Fgutabs‘

By default, the lower bound for ‘Fgutabs‘ is 0.0, and the upper bound for ‘Fgutabs‘ is 1. These are simply the bounds of the physically-meaningful range for a fraction.

Default lower and upper bounds for ‘Fgutabs_Vdist‘

By default, the lower bound for the ratio ‘Fgutabs_Vdist‘ is 0.01, and the upper bound is 100. These values were chosen based on professional judgment.

Default lower and upper bounds for ‘Rblood2plasma‘

By default, the lower bound for the blood:plasma partition coefficient ‘Rblood2plasma‘ is 0.01, and the upper bound is 100. These values were chosen based on professional judgment.

Starting values for each parameter

Starting values for each parameter (starting guesses for the numerical optimizer) are derived from the data using [get_starts_1comp()].

If the starting values returned by [get_starts_1comp()] fall outside the bounds for any parameter(s), then the starting value will be reset to a value halfway between the lower and upper bounds for that parameter.

Value

A ‘data.frame‘with the following variables: - ‘param_name‘: Character: Names of the model parameters - ‘param_units‘: Character: Units of the model parameters - ‘optimize_param‘: TRUE if each parameter is to be estimated from the data; FALSE otherwise - ‘use_param‘: TRUE if each parameter is to be used in evaluating the model; FALSE otherwise -‘lower_bounds‘: Numeric: The lower bounds for each parameter - ‘upper_bounds‘: Numeric: The upper bounds for each parameter - ‘start‘: Numeric: The starting guesses for each parameter

Author(s)

Caroline Ring

See Also

Other 1-compartment model functions: `auc_1comp()`, `auc_1comp_cl()`, `cp_1comp()`, `cp_1comp_cl()`, `get_params_1comp()`, `get_params_1comp_fup()`, `get_starts_1comp()`, `get_starts_1comp_cl()`, `get_starts_1comp_fup()`

Other get_params functions: `get_params_1comp()`, `get_params_1comp_fup()`, `get_params_2comp()`, `get_params_flat()`

Other built-in model functions: auc_1comp(), auc_1comp_cl(), auc_2comp(), auc_flat(), cp_1comp(), cp_1comp_cl(), cp_2comp(), cp_2comp_dt(), cp_flat(), get_params_1comp(), get_params_1comp_fup(), get_params_2comp(), get_params_flat(), get_starts_1comp(), get_starts_1comp_cl(), get_starts_1comp_fup(), get_starts_2comp(), get_starts_flat(), tkstats_2comp(), transformed_params_2comp()

`get_params_1comp_fup` *Get parameters for 1-compartment model with clearance assumption*

Description

Get parameters for 1-compartment model and determine whether each is to be estimated from the data

Usage

```
get_params_1comp_fup(
  data,
 lower_bound = ggplot2::aes(Q_totli = NA, Q_gfr = NA, Fup = 0, Clint = 0, Vdist = 0.01,
    Fgutabs = 0, kgutabs = log(2)/(2 * max(Time_trans)), Fgutabs_Vdist = 0.01,
    Rblood2plasma = 0.01),
 upper_bound = ggplot2::aes(Q_totli = NA, Q_gfr = NA, Fup = 1, Clint = 1e+05, Vdist =
    100, Fgutabs = 1, kgutabs = log(2)/(0.5 * min(Time_trans[Time_trans > 0])),
    Fgutabs_Vdist = 100, Rblood2plasma = 100),
 param_units = ggplot2::aes(Q_totli = "L/h/kg ^3/4", Q_gfr = "L/h/kg ^3/4", Fup =
  "unitless fraction", Clint = "L/h/kg", Vdist = paste0("(", unique(Dose.Units), ")",
    "/", "(", unique(Conc.Units), ")"), Fgutabs = "unitless fraction", kgutabs =
    paste0("1/", unique(Time_trans.Units)), Fgutabs_Vdist = paste0("(",
    unique(Conc.Units), ")", "/", "(", unique(Dose.Units), ")"), Rblood2plasma =
    "unitless ratio")
)
```

Arguments

`data`	The data set to be fitted (e.g. the result of [preprocess_data()])
`lower_bound`	A mapping specified using a call to [ggplot2::aes()], giving the lower bounds for each variable, as expressions which may include variables in 'data'.
`upper_bound`	A mapping specified using a call to [ggplot2::aes()], giving the upper bounds for each variable, as expressions which may include variables in 'data'.
`param_units`	A mapping specified using a call to [ggplot2::aes()], giving the units for each variable, as expressions which may include variables in 'data'.

Details

The full set of model parameters for the 1-compartment model includes 'Vdist', 'Clint', 'kgutabs', 'Fgutabs', 'Q_totli', 'Q_gfr', 'Fup', and 'Rblood2plasma'. Whether each one can be estimated from the data depends on which routes of administration are included in the data.

Constant parameters from ‘httk‘ ‘Q_totli‘ is the flow through the portal vein from the gut to the liver and is estimated in this model via [httk::tissue.data] in a species-specific manner. ‘Q_gfr‘ is the glomular filtration rate of kidneys. It is estimated via [httk::physiology.data] in a species specific manner. ‘Fup‘ and ‘Rblood2plasma‘ are both calculated in [httk::parameterize_1comp()] For each of these parameters default lower bounds are 10

IV data, no oral data

If IV dosing data are available, but no oral dosing data are available, then only the parameters ‘Vdist‘ and ‘kelim‘ will be estimated from the data. The parameters ‘kgutabs‘ and ‘Fgutabs‘ cannot be estimated from IV data alone, and will not be used in evaluating the model.

Oral data, no IV data

If oral dosing data are available, but no IV dosing data are available, then the parameters ‘kelim‘ and ‘kgutabs‘ can be estimated from the data. However, the parameters ‘Fgutabs‘ and ‘Vdist‘ cannot be identified separately. From oral data alone, only the ratio ‘Fgutabs/Vdist‘ can be identified. This ratio is represented by a single parameter named ‘Fgutabs_Vdist‘. ‘Fgutabs‘ and ‘Vdist‘ will not be used to evaluate the model nor be estimated from data, but ‘Fgutabs_Vdist‘ will be estimated from data, along with ‘kelim‘ and ‘kgutabs‘.

Oral data and IV data

If both oral and IV dosing data are available, then ‘Vdist‘, ‘kelim‘, ‘kgutabs‘, and ‘Fgutabs‘ will all be estimated from the data.

Blood and plasma data

If both blood and plasma data are available, then ‘Rblood2plasma‘ will be estimated from the data.

Only one of blood or plasma data

If only one of blood or plasma data are available, then ‘Rblood2plasma‘ will be held constant at 1, not estimated from the data.

Default lower and upper bounds for each parameter

Default lower and upper bounds for ‘kelim‘ and ‘kgutabs‘

Default bounds for time constants ‘kelim‘ and ‘kgutabs‘ are set based on the time scale of the available data.

The lower bounds are based on the assumption that elimination and absorption are very slow compared to the time scale of the study. Specifically, the lower bounds assume that elimination and absorption half-lives are twice as long as the duration of the available study data, or ‘2*max(Time_trans)‘. Under this assumption, the corresponding elimination and absorption time constants would be ‘log(2)/(2*max(Time_trans))‘. Therefore, the default lower bounds for ‘kelim‘ and ‘kgutabs‘ are ‘log(2)/(2*max(Time_trans))‘.

Upper bounds are based on the opposite assumption: that elimination and absorption are very fast compared to the time scale of the study. Specifically, the upper bounds assume that the elimination and absorption half-lives are half as long as the time of the first observation after time 0, or ‘0.5*min(Time_trans[Time_trans>0])‘. Under this asumption, the corresponding elimination and absorption time constants would be ‘log(2)/(0.5*min(Time_trans[Time_trans>0]))‘. Therefore, the default lower bounds for ‘kelim‘ and ‘kgutabs‘ are ‘log(2)/(0.5*min(Time_trans[Time_trans>0]))‘.

Default lower and upper bounds for ‘Vdist‘

By default, the lower bound for ‘Vdist‘ is 0.01, and the upper bound for ‘Vdist‘ is 100. These values were chosen based on professional judgment.

Default lower and upper bounds for ‘Fgutabs‘

By default, the lower bound for ‘Fgutabs‘ is 0.0, and the upper bound for ‘Fgutabs‘ is 1. These are simply the bounds of the physically-meaningful range for a fraction.

Default lower and upper bounds for ‘Fgutabs_Vdist‘

By default, the lower bound for the ratio ‘Fgutabs_Vdist‘ is 0.01, and the upper bound is 100. These values were chosen based on professional judgment.

Default lower and upper bounds for ‘Rblood2plasma‘

By default, the lower bound for the blood:plasma partition coefficient ‘Rblood2plasma‘ is 0.01, and the upper bound is 100. These values were chosen based on professional judgment.

Starting values for each parameter

Starting values for each parameter (starting guesses for the numerical optimizer) are derived from the data using [get_starts_1comp()].

If the starting values returned by [get_starts_1comp()] fall outside the bounds for any parameter(s), then the starting value will be reset to a value halfway between the lower and upper bounds for that parameter.

Value

A ‘data.frame‘with the following variables: - ‘param_name‘: Character: Names of the model parameters - ‘param_units‘: Character: Units of the model parameters - ‘optimize_param‘: TRUE if each parameter is to be estimated from the data; FALSE otherwise - ‘use_param‘: TRUE if each parameter is to be used in evaluating the model; FALSE otherwise -‘lower_bounds‘: Numeric: The lower bounds for each parameter - ‘upper_bounds‘: Numeric: The upper bounds for each parameter - ‘start‘: Numeric: The starting guesses for each parameter

Author(s)

Caroline Ring, Gilberto Padilla Mercado

See Also

Other 1-compartment model functions: auc_1comp(), auc_1comp_cl(), cp_1comp(), cp_1comp_cl(), get_params_1comp(), get_params_1comp_cl(), get_starts_1comp(), get_starts_1comp_cl(), get_starts_1comp_fup()

Other get_params functions: get_params_1comp(), get_params_1comp_cl(), get_params_2comp(), get_params_flat()

Other built-in model functions: auc_1comp(), auc_1comp_cl(), auc_2comp(), auc_flat(), cp_1comp(), cp_1comp_cl(), cp_2comp(), cp_2comp_dt(), cp_flat(), get_params_1comp(), get_params_1comp_cl(), get_params_2comp(), get_params_flat(), get_starts_1comp(), get_starts_1comp_cl(), get_starts_1comp_fup(), get_starts_2comp(), get_starts_flat(), tkstats_2comp(), transformed_params_2comp()

`get_params_2comp` *Get parameters for 2-compartment model*

Description

Get parameters for 2-compartment model and determine whether each is to be estimated from the data

Usage

```
get_params_2comp(
  data,
 lower_bound = ggplot2::aes(kelim = log(2)/(2 * max(Time_trans)), k12 = log(2)/(2 *
  max(Time_trans)), k21 = log(2)/(2 * max(Time_trans)), V1 = 0.01, Fgutabs = 0, kgutabs
    = log(2)/(2 * max(Time_trans)), Fgutabs_V1 = 0.01, Rblood2plasma = 0.01),
 upper_bound = ggplot2::aes(kelim = log(2)/(0.5 * min(Time_trans[Time_trans > 0])), k12
    = log(2)/(0.5 * min(Time_trans[Time_trans > 0])), k21 = log(2)/(0.5 *
  min(Time_trans[Time_trans > 0])), V1 = 100, Fgutabs = 1, kgutabs = log(2)/(0.5 *
    min(Time_trans[Time_trans > 0])), Fgutabs_V1 = 100, Rblood2plasma = 100),
 param_units = ggplot2::aes(kelim = paste0("1/", unique(Time_trans.Units)), V1 =
  paste0("(", unique(Dose.Units), ")", "/", "(", unique(Conc.Units), ")"), k21 =
  paste0("1/", unique(Time_trans.Units)), k12 = paste0("1/", unique(Time_trans.Units)),
  Fgutabs = "unitless fraction", kgutabs = paste0("1/", unique(Time_trans.Units)),
  Fgutabs_V1 = paste0("(", unique(Conc.Units), ")", "/", "(", unique(Dose.Units), ")"),
    Rblood2plasma = "unitless ratio")
)
```

Arguments

`data`	The data set to be fitted (e.g. the result of [preprocess_data()])
`lower_bound`	A mapping specified using a call to [ggplot2::aes()], giving the lower bounds for each variable, as expressions which may include variables in 'data'.
`upper_bound`	A mapping specified using a call to [ggplot2::aes()], giving the upper bounds for each variable, as expressions which may include variables in 'data'.
`param_units`	A mapping specified using a call to [ggplot2::aes()], giving the units for each variable, as expressions which may include variables in 'data'.

Details

The full set of model parameters for the 2-compartment model includes 'V1', 'kelim', 'k12', 'k21', 'kgutabs','Fgutabs', and 'Rblood2plasma'. Whether each one can be estimated from the data depends on what routes of administration are included in the data.

IV data, no oral data

If IV dosing data are available, but no oral dosing data are available, then only the parameters 'V1', 'kelim', 'k12', and 'k21' will be estimated from the data. The parameters 'kgutabs' and 'Fgutabs' cannot be estimated from IV data alone.

Oral data, no IV data

If oral dosing data are available, but no IV dosing data are available, then the parameters ‘kelim‘, ‘k12‘, ‘k21‘, and ‘kgutabs‘ will be estimated from the data. However, the parameters ‘Fgutabs‘ and ‘V1‘ cannot be identified separately. From oral data alone, only the ratio ‘Fgutabs/V1‘ can be identified. This ratio is represented by a single parameter named ‘Fgutabs_V1‘. ‘Fgutabs‘ and ‘V1‘ will not be optimized, but ‘Fgutabs_V1‘ will be optimized, along with ‘kelim‘, ‘k12‘, ‘k21‘, and ‘kgutabs‘.

Oral data and IV data

If both oral and IV dosing data are available, then ‘V1‘, ‘kelim‘, ‘k12‘, ‘k21‘, ‘kgutabs‘, and ‘Fgutabs‘ will all be estimated from the data.

Blood and plasma data

If both blood and plasma data are available, then ‘Rblood2plasma‘ will be estimated from the data.

Only one of blood or plasma data

If only one of blood or plasma data are available, then ‘Rblood2plasma‘ will be held constant at 1, not estimated from the data.

Default lower and upper bounds for each parameter

Default lower and upper bounds for time constants ‘kelim‘, ‘kgutabs‘, ‘k12‘, and ‘k21‘.

Default bounds for time constants ‘kelim‘ and ‘kgutabs‘ are set based on the time scale of the available data.

The lower bounds are based on the assumption that elimination, absorption, and distribution are very slow compared to the time scale of the study. Specifically, the lower bounds assume thatelimination, absorption, and distribution half-lives are twice as long as the duration of the available study data, or ‘2*max(Time_trans)‘. Under this assumption, the corresponding elimination, absorption, and distribution time constants would be ‘log(2)/(2*max(Time_trans))‘. Therefore, the default lower bounds for ‘kelim‘, ‘kgutabs‘, ‘k12‘, and ‘k21‘ are ‘log(2)/(2*max(Time_trans))‘.

Upper bounds are based on the opposite assumption: that elimination, absorption, and distribution are very fast compared to the time scale of the study. Specifically, the upper bounds assume that the elimination, absorption, and distribution half-lives are half as long as the time of the first observation after time 0, or ‘0.5*min(Time_trans[Time_trans>0])‘. Under this assumption, the correspondinge-limination, absorption, and distribution time constants would be ‘log(2)/(0.5*min(Time_trans[Time_trans>0]))‘. Therefore, the default lower bounds for ‘kelim‘, ‘kgutabs‘, ‘k12‘, and ‘k21‘ are ‘log(2)/(0.5*min(Time_trans[Time_trans>0]))

Default lower and upper bounds for ‘V1‘

By default, the lower bound for ‘V1‘ is 0.01, and the upper bound for ‘V1‘ is 100. These values were chosen based on professional judgment.

Default lower and upper bounds for ‘Fgutabs‘

By default, the lower bound for ‘Fgutabs‘ is 0.0, and the upper bound for ‘Fgutabs‘ is 1. These are simply the bounds of the physically-meaningful range for a fraction.

Default lower and upper bounds for ‘Fgutabs_V1‘

By default, the lower bound for the ratio ‘Fgutabs_V1‘ is 0.01, and the upper bound is 100. These values were chosen based on professional judgment.

Default lower and upper bounds for ‘Rblood2plasma‘

By default, the lower bound for the blood:plasma partition coefficient 'Rblood2plasma' is 0.01, and the upper bound is 100. These values were chosen based on professional judgment.

Starting values for each parameter

Starting values for each parameter (starting guesses for the numerical optimizer) are derived from the data using [get_starts_2comp()].

If the starting values returned by [get_starts_2comp()] fall outside the bounds for any parameter(s), then the starting value will be reset to a value halfway between the lower and upper bounds for that parameter.

Value

A 'data.frame'with the following variables: - 'param_name': Character: Names of the model parameters - 'param_units': Character: Units of the model parameters - 'optimize_param': TRUE if each parameter is to be estimated from the data; FALSE otherwise - 'use_param': TRUE if each parameter is to be used in evaluating the model; FALSE otherwise -'lower_bounds': Numeric: The lower bounds for each parameter - 'upper_bounds': Numeric: The upper bounds for each parameter - 'start': Numeric: The starting guesses for each parameter

Author(s)

Caroline Ring

See Also

Other 2-compartment model functions: `auc_2comp()`, `cp_2comp()`, `cp_2comp_dt()`, `get_starts_2comp()`, `tkstats_2comp()`, `transformed_params_2comp()`

Other get_params functions: `get_params_1comp()`, `get_params_1comp_cl()`, `get_params_1comp_fup()`, `get_params_flat()`

Other built-in model functions: `auc_1comp()`, `auc_1comp_cl()`, `auc_2comp()`, `auc_flat()`, `cp_1comp()`, `cp_1comp_cl()`, `cp_2comp()`, `cp_2comp_dt()`, `cp_flat()`, `get_params_1comp()`, `get_params_1comp_cl()`, `get_params_1comp_fup()`, `get_params_flat()`, `get_starts_1comp()`, `get_starts_1comp_cl()`, `get_starts_1comp_fup()`, `get_starts_2comp()`, `get_starts_flat()`, `tkstats_2comp()`, `transformed_params_2comp`

`get_params_flat` *Get parameters to be optimized for flat model*

Description

The full set of model parameters for the flat model includes 'Vdist', 'Fgutabs', and 'Rblood2plasma'. Whether each one can be estimated from the data depends on what routes of administration are included in the data.

Usage

```
get_params_flat(
  data,
  lower_bound = ggplot2::aes(Vdist = 0.01, Fgutabs = 0, Fgutabs_Vdist = 0.01,
    Rblood2plasma = 0.01),
 upper_bound = ggplot2::aes(Vdist = 100, Fgutabs = 1, Fgutabs_Vdist = 100, Rblood2plasma
    = 100),
 param_units = ggplot2::aes(Vdist = paste0("(", unique(Dose.Units), ")", "/", "(",
  unique(Conc.Units), ")"), Fgutabs = "unitless fraction", Fgutabs_Vdist = paste0("(",
   unique(Conc.Units), ")", "/", "(", unique(Dose.Units), ")"), Rblood2plasma =
   "unitless ratio")
)
```

Arguments

data	The data set to be fitted (e.g. the result of [preprocess_data()])
lower_bound	A mapping specified using a call to [ggplot2::aes()], giving the lower bounds for each variable, as expressions which may include variables in 'data'.
upper_bound	A mapping specified using a call to [ggplot2::aes()], giving the upper bounds for each variable, as expressions which may include variables in 'data'.
param_units	A mapping specified using a call to [ggplot2::aes()], giving the units for each variable, as expressions which may include variables in 'data'.

Details

IV data, no oral data

If IV dosing data are available, but no oral dosing data are available, then only the parameter 'Vdist' will be estimated from the data. The parameter 'Fgutabs' cannot be estimated from IV data alone and will not be used to evaluate the model.

Oral data, no IV data

If oral dosing data are available, but no IV dosing data are available, then the parameters 'Fgutabs' and 'Vdist' cannot be identified separately. From oral data alone, only the ratio 'Fgutabs/Vdist' can be identified. This ratio is represented by a single parameter named 'Fgutabs_Vdist'. 'Fgutabs' and 'Vdist' will not be estimated nor used in model evaluation, but 'Fgutabs_Vdist' will be estimated.

Oral data and IV data

If both oral and IV dosing data are available, then 'Vdist' and 'Fgutabs' will both be estimated from the data.

Blood and plasma data

If both blood and plasma data are available, then 'Rblood2plasma' will be estimated from the data.

Only one of blood or plasma data

If only one of blood or plasma data are available, then 'Rblood2plasma' will be held constant at 1, not estimated from the data.

Default lower and upper bounds for each parameter

Default lower and upper bounds for 'Vdist'

By default, the lower bound for ‘Vdist‘ is 0.01, and the upper bound for ‘Vdist‘ is 100. These values were chosen based on professional judgment.

Default lower and upper bounds for ‘Fgutabs‘

By default, the lower bound for ‘Fgutabs‘ is 0.0, and the upper bound for ‘Fgutabs‘ is 1. These are simply the bounds of the physically-meaningful range for a fraction.

Default lower and upper bounds for ‘Fgutabs_Vdist‘

By default, the lower bound for the ratio ‘Fgutabs_Vdist‘ is 0.01, and the upper bound is 100. These values were chosen based on professional judgment.

Default lower and upper bounds for ‘Rblood2plasma‘

By default, the lower bound for the blood:plasma partition coefficient ‘Rblood2plasma‘ is 0.01, and the upper bound is 100. These values were chosen based on professional judgment.

Starting values for each parameter

Starting values for each parameter (starting guesses for the numerical optimizer) are derived from the data using [get_starts_flat()].

If the starting values returned by [get_starts_flat()] fall outside the bounds for any parameter(s), then the starting value will be reset to a value halfway between the lower and upper bounds for that parameter.

Value

A ‘data.frame‘with the following variables: - ‘param_name‘: Character: Names of the model parameters - ‘param_units‘: Character: Units of the model parameters - ‘optimize_param‘: TRUE if each parameter is to be estimated from the data; FALSE otherwise - ‘use_param‘: TRUE if each parameter is to be used in evaluating the model; FALSE otherwise -‘lower_bounds‘: Numeric: The lower bounds for each parameter - ‘upper_bounds‘: Numeric: The upper bounds for each parameter - ‘start‘: Numeric: The starting guesses for each parameter

Author(s)

Caroline Ring

See Also

Other flat model functions: `auc_flat()`, `cp_flat()`, `get_starts_flat()`

Other get_params functions: `get_params_1comp()`, `get_params_1comp_cl()`, `get_params_1comp_fup()`, `get_params_2comp()`

Other built-in model functions: `auc_1comp()`, `auc_1comp_cl()`, `auc_2comp()`, `auc_flat()`, `cp_1comp()`, `cp_1comp_cl()`, `cp_2comp()`, `cp_2comp_dt()`, `cp_flat()`, `get_params_1comp()`, `get_params_1comp_cl()`, `get_params_1comp_fup()`, `get_params_2comp()`, `get_starts_1comp()`, `get_starts_1comp_cl()`, `get_starts_1comp_fup()`, `get_starts_2comp()`, `get_starts_flat()`, `tkstats_2comp()`, `transformed_params_2comp`

`get_peak` *Find the peak of a data series*

Description

Finds x- and y-value at peak y value.

Usage

```
get_peak(x, y, ties = "median", na.rm = TRUE, ...)
```

Arguments

`x`	A numeric vector of ‘x‘ data
`y`	A numeric vector of ‘y‘ data
`ties`	As for [stats::approxfun()]: The function to apply to y-values that have the same x-value. Default ‘’median’‘. ‘’mean’‘ may also be useful.
`na.rm`	As for [stats::approxfun()]: How to handle missing values. Default ‘TRUE‘ to exclude missing values from analysis.
`...`	Optional: Additional arguments which will be passed to [stats::approx()] (other than ‘x‘, ‘y‘, and ‘xout‘).

Details

If there is more than one unique ‘x‘ value where both ‘x‘ and corresponding ‘y‘ are finite, this function calls [stats::approx()] with ‘method = ’linear’‘, then uses [base::which.max()] to locate the maximum interpolated ‘y‘-value.

If there is only one unique ‘x‘ value where both ‘x‘ and corresponding ‘y‘ are finite, this function calls [stats::approx()] with ‘method = ’constant’‘, then uses [base::which.max()] to locate the maximum interpolated ‘y‘-value.

If there are no unique ‘x‘ values where both ‘x‘ and corresponding ‘y‘ are finite, this function returns ‘NA_real_‘ for the peak ‘x‘ and ‘y‘ values.

Value

A list with two named numeric scalar components, ‘x‘ and ‘y‘, containing the x- and y-values at the peak.

Author(s)

Caroline Ring

get_prefit *get_prefit()*

Description

This is the S3 method generic for get_prefit()

Usage

```
get_prefit(obj, ...)
```

Arguments

obj	An object.
...	Additional arguments currently not in use.

Value

A list of 'data.frame's in the object's 'prefit' element.

See Also

[get_prefit.pk()] for the method for class [pk()]

get_prefit.default *Default method for get_prefit()*

Description

Default method for get_prefit()

Usage

```
## Default S3 method:
get_prefit(obj, ...)
```

Arguments

obj	An object
...	Additional arguments currently not in use.

Value

An error, when a non-pk object is used for the first argument.

get_prefit.pk *Get prefit*

Description

Extract pre-fitting results from a [pk()] object

Usage

```
## S3 method for class 'pk'
get_prefit(obj, ...)
```

Arguments

obj	A [pk()] object that has had ‘do_prefit()‘ run on it
...	Additional arguments. Currently not in use.

Value

A list of ‘data.frame‘s in the object’s ‘prefit‘ element.

Author(s)

Caroline Ring

get_scale_conc *get_scale_conc()*

Description

This is the S3 method generic for get_scale_conc()

Usage

```
get_scale_conc(obj, ...)
```

Arguments

obj	An object.
...	Additional arguments currently not in use.

Value

A ‘list‘: ‘obj$scales$conc‘

See Also

[get_scale_conc.pk()] for the method for class [pk()]

get_scale_conc.default *Default method for get_scale_conc()*

Description

Default method for get_scale_conc()

Usage

```
## Default S3 method:
get_scale_conc(obj, ...)
```

Arguments

obj	An object
...	Additional arguments currently not in use.

Value

An error, when a non-pk object is used for the first argument.

get_scale_conc.pk *Get scale_conc*

Description

Extract concentration scale/transformation instructions from a [pk()] object

Usage

```
## S3 method for class 'pk'
get_scale_conc(obj, ...)
```

Arguments

obj	A [pk()] object
...	Additional arguments not currently in use.

Value

A ‘list‘: ‘obj$scales$conc‘

Author(s)

Caroline Ring

get_scale_time *get_scale_time()*

Description

This is the S3 method generic for get_scale_time()

Usage

```
get_scale_time(obj, ...)
```

Arguments

obj	An object.
...	Additional arguments currently not in use.

Value

A ‘list‘: ‘obj$scales$time‘

See Also

[get_scale_time.pk()] for the method for class [pk()]

get_scale_time.default *Default method for get_scale_time()*

Description

Default method for get_scale_time()

Usage

```
## Default S3 method:
get_scale_time(obj, ...)
```

Arguments

obj	An object
...	Additional arguments currently not in use.

Value

An error, when a non-pk object is used for the first argument.

get_scale_time.pk *Get scale_time*

Description

Extract time scale/transformation instructions from a [pk()] object

Usage

```
## S3 method for class 'pk'
get_scale_time(obj, ...)
```

Arguments

obj	A [pk()] object
...	Additional arguments not in use.

Value

A 'list': 'obj$scales$time'

Author(s)

Caroline Ring

get_settings_data_info *get_settings_data_info()*

Description

This is the S3 method generic for get_settings_data_info()

Usage

```
get_settings_data_info(obj, ...)
```

Arguments

obj	An object.
...	Additional arguments currently not in use.

Value

A named list of the data_info settings

See Also

[get_settings_data_info.pk()] for the method for class [pk()]

---

get_settings_data_info.default

Default method for get_settings_data_info()

Description

Default method for get_settings_data_info()

Usage

```
## Default S3 method:
get_settings_data_info(obj, ...)
```

Arguments

obj	An object
...	Additional arguments currently not in use.

Value

An error, when a non-pk object is used for the first argument.

---

get_settings_data_info.pk

Get settings_data_info

Description

Get settings_data_info

Usage

```
## S3 method for class 'pk'
get_settings_data_info(obj, ...)
```

Arguments

obj	A [pk()] object
...	Additional arguments. Currently not implemented.

Value

A named list of the data_info settings

Author(s)

Caroline Ring

`get_settings_optimx` *get_settings_optimx()*

Description

This is the S3 method generic for get_settings_optimx()

Usage

```
get_settings_optimx(obj, ...)
```

Arguments

`obj`	An object.
`...`	Additional arguments currently not in use.

Value

A named list of the optimx settings

See Also

[get_settings_optimx.pk()] for the method for class [pk()]

`get_settings_optimx.default`
Default method for get_settings_optimx()

Description

Default method for get_settings_optimx()

Usage

```
## Default S3 method:
get_settings_optimx(obj, ...)
```

Arguments

`obj`	An object
`...`	Additional arguments currently not in use.

Value

An error, when a non-pk object is used for the first argument.

get_settings_optimx.pk

Get settings_optimx

Description

Get settings_optimx

Usage

```
## S3 method for class 'pk'
get_settings_optimx(obj, ...)
```

Arguments

obj	A [pk()] object
...	Additional arguents not currently in use.

Value

A named list of the optimx settings

Author(s)

Caroline Ring

get_settings_preprocess

get_settings_preprocess()

Description

This is the S3 method generic for get_settings_preprocess()

Usage

```
get_settings_preprocess(obj, ...)
```

Arguments

obj	An object.
...	Additional arguments currently not in use.

Value

A named list of the preprocessing settings

See Also

[get_settings_preprocess.pk()] for the method for class [pk()]

`get_settings_preprocess.default`
Default method for get_settings_preprocess()

Description

Default method for get_settings_preprocess()

Usage

```
## Default S3 method:
get_settings_preprocess(obj, ...)
```

Arguments

`obj`	An object
`...`	Additional arguments currently not in use.

Value

An error, when a non-pk object is used for the first argument.

`get_settings_preprocess.pk`
Get settings_preprocess

Description

Get settings_preprocess

Usage

```
## S3 method for class 'pk'
get_settings_preprocess(obj, ...)
```

Arguments

`obj`	A [pk()] object
`...`	Additional arguments. Currently not in use.

Value

A named list of the preprocessing settings

Author(s)

Caroline Ring

`get_starts_1comp` *Get starting values for 1-compartment model*

Description

Derive starting values for 1-compartment model parameters from available data

Usage

```
get_starts_1comp(data, par_DF)
```

Arguments

`data`	The data set to be fitted (e.g. the result of [preprocess_data()])
`par_DF`	A ‘data.frame‘ with the following variables (e.g., as produced by [get_params_1comp()]) - ‘param_name‘: Character: Names of the model parameters - ‘param_units‘: Character: Units of the model parameters - ‘optimize_param‘: TRUE if each parameter is to be estimated from the data; FALSE otherwise - ‘use_param‘: TRUE if each parameter is to be used in evaluating the model; FALSE otherwise -‘lower_bounds‘: Numeric: The lower bounds for each parameter - ‘upper_bounds‘: Numeric: The upper bounds for each parameter

Details

This function is called internally by [get_params_1comp()] and should generally not be called directly by the user.

The full set of model parameters for the 1-compartment model includes ‘Vdist‘, ‘kelim‘, ‘kgutabs‘, ‘Fgutabs‘, and ‘Rblood2plasma‘. Whether each one can be estimated from the data depends on what routes of administration are included in the data.

The numerical optimizer requires starting guesses for the value of each parameter to be estimated from the data. Default starting guesses are derived from the available data.

These are intended to be *very* rough starting guesses, so the algorithm here is extremely naive. This function is not itself intended to produce valid estimates for any of the model parameters, and it is highly unlikely to do so.

The derivation process is as follows.

First, data are filtered to exclude any non-detects.

Then, data are split by route of administration, into an IV data set and an oral data set. (It is possible that either IV or oral data may not be available for a chemical.)

Starting value for ‘kelim‘

If IV data exist, then only IV data are used to derive starting estimates for ‘kelim‘, even if oral data also exist.

If only oral data exist, then the oral data are used to derive a starting estimate for ‘kelim‘.

Whichever data set is used (IV or oral), the starting value for ‘kelim‘ is derived by assuming that the range of observed time values in the data set spans two elimination half-lives. This implies that the elimination half-life is equal to the midpoint of observed time values, and that the starting value for the elimination time constant ‘kelim‘ is therefore ‘log(2)‘ divided by the midpoint of observed time values.

Of course, this assumption is unlikely to be correct. However, we hope that it will yield a starting guess for ‘kelim‘ that is at least on the right order of magnitude.

Starting value for ‘Vdist‘

If IV data exist, then only IV data are used to derive a starting estimate for ‘Vdist‘.

This starting estimate is derived by assuming that the IV data obey a one-compartment model, which means that when concentrations are dose-normalized and log10-transformed and plotted against time, they will follow a straight line with slope ‘-kelim‘.

First, concentrations are dose-normalized by dividing them by their corresponding doses. Then the normalized concentrations are log10-transformed.

From all observations at the earliest observed time point in the data set (call it ‘tmin‘), the median of the dose-normalized, log10-transformed concentrations is calculated; call it ‘C_tmin‘. (The median is used, rather than the mean, in an attempt to be more robust to outliers.)

If the earliest observed time point is not at time = 0, then the dose-normalized, log10-transformed concentration at time = 0 is extrapolated by drawing a straight line with slope ‘-kelim‘ back from ‘C_tmin‘, where the value of ‘kelim‘ is the starting value derived as in the previous section.

This extrapolated concentration at time t = 0 is called ‘A_log10‘. ‘A_log10‘ represents the expected body concentration immediately after IV injection of a unit dose (under the assumption that TK obeys a one-compartment model).

Then, the volume of distribution ‘Vdist‘ is derived as ‘1/(10^A_log10)‘. In other words, ‘Vdist‘ is the volume that would be required to produce a concentration equal to ‘A_log10‘ after injecting a unit dose.

(No starting value for ‘Vdist‘ can be derived with only oral data, but none is needed, because with only oral data, ‘Vdist‘ will not be estimated from the data).

Starting value for ‘kgutabs‘

If oral data exist (whether or not IV data also exist), then the oral data are used to derive a starting value for ‘kgutabs‘.

First, concentrations are dose-normalized by dividing them by their corresponding doses. Then the normalized concentrations are log10-transformed.

The time of peak concentration (‘tmax‘), and the median (normalized, log-transformed) peak concentration (‘Cmax_log10‘), are identified using [get_peak()].

As a very rough guess,‘tmax‘ is assumed to occur at one absorption half-life. Under this assumption, ‘kgutabs‘ is equal to ‘log(2)/tmax‘, and this is taken as the starting value.

Starting value for ‘Fgutabs_Vdist‘

If any oral data exist (whether or not IV data also exist), then the oral data are used to derive a starting value for ‘Fgutabs_Vdist‘.

If the kinetics obey a one-compartment model, then if concentrations are dose-normalized, log-transformed, and plotted vs. time, then at late time points (after concentration has peaked), the concentration vs. time relationship will approach a straight line with slope '-kelim'.

If this straight line is extrapolated back to time 0, then the resulting intercept (call it 'A'), expressed on the natural scale, is equal to 'Fgutabs_Vdist * kgutabs/(kgutabs-kelim)'. See https://www.boomer.org/c/p4/c09/c0902.php .

Roughly, we approximate 'A' on the log10 scale by extrapolating back from the peak along a straight line with slope '-kelim', using the previously-derived starting value for 'kelim'. So 'log10(A) = Cmax_log10 + kelim*tmax'.

Using the previously-derived starting values for 'kgutabs' and 'kelim', then, the starting value for 'Fgutabs_Vdist' can be derived as 'A * (kgutabs-kelim)/kgutabs'.

Starting value for 'Fgutabs'

If both oral and IV data exist, then the derived starting values for 'Vdist' (from the IV data) and 'Fgutabs_Vdist' (from the oral data) are multiplied to yield a derived starting value for 'Fgutabs'.

#Starting value for 'Rblood2plasma'

The starting value for 'Rblood2plasma' is always set at a constant 1.

Value

The same 'data.frame' as 'par_DF', with an additional variable 'starts' containing the derived starting value for each parameter. If a parameter cannot be estimated from the available data, then its starting value will be 'NA_real_'

Author(s)

Caroline Ring

See Also

Other 1-compartment model functions: auc_1comp(), auc_1comp_cl(), cp_1comp(), cp_1comp_cl(), get_params_1comp(), get_params_1comp_cl(), get_params_1comp_fup(), get_starts_1comp_cl(), get_starts_1comp_fup()

Other get_starts functions: get_starts_1comp_cl(), get_starts_1comp_fup(), get_starts_2comp(), get_starts_flat()

Other built-in model functions: auc_1comp(), auc_1comp_cl(), auc_2comp(), auc_flat(), cp_1comp(), cp_1comp_cl(), cp_2comp(), cp_2comp_dt(), cp_flat(), get_params_1comp(), get_params_1comp_cl(), get_params_1comp_fup(), get_params_2comp(), get_params_flat(), get_starts_1comp_cl(), get_starts_1comp_fup(), get_starts_2comp(), get_starts_flat(), tkstats_2comp(), transformed_params_2comp

`get_starts_1comp_cl` *Get starting values for 1-compartment model with specific clearance*

Description

Derive starting values for 1-compartment model parameters from available data

Usage

```
get_starts_1comp_cl(data, par_DF, restrictive)
```

Arguments

`data`	The data set to be fitted (e.g. the result of [preprocess_data()])
`par_DF`	A 'data.frame' with the following variables (e.g., as produced by [get_params_1comp()]) - 'param_name': Character: Names of the model parameters - 'param_units': Character: Units of the model parameters - 'optimize_param': TRUE if each parameter is to be estimated from the data; FALSE otherwise - 'use_param': TRUE if each parameter is to be used in evaluating the model; FALSE otherwise -'lower_bounds': Numeric: The lower bounds for each parameter - 'upper_bounds': Numeric: The upper bounds for each parameter
`restrictive`	A boolean value determinining whether to assume restrictive or non-restrictive clearance when getting starting values.

Details

This function is called internally by [get_params_1comp()] and should generally not be called directly by the user.

The full set of model parameters for the 1-compartment model includes 'Vdist', 'kelim', 'kgutabs', 'Fgutabs', and 'Rblood2plasma'. Whether each one can be estimated from the data depends on what routes of administration are included in the data. However, in this version of the one-compartment model, we use the liver and glomerular flow rates ('Q_totli' and 'Q_gfr', respectively) provided by [httk] as well as intrisic clearance 'Clint' and fraction unbound in plasma, 'Fup'. These starting values are established and then are used to calculate the other parameters in the 1-compartment model.

The numerical optimizer requires starting guesses for the value of each parameter to be estimated from the data. Default starting guesses are derived from the available data.

These are intended to be *very* rough starting guesses, so the algorithm here is extremely naive. This function is not itself intended to produce valid estimates for any of the model parameters, and it is highly unlikely to do so.

The derivation process is as follows.

First, data are filtered to exclude any non-detects.

Then, data are split by route of administration, into an IV data set and an oral data set. (It is possible that either IV or oral data may not be available for a chemical.)

Starting value for ‘kelim‘

If IV data exist, then only IV data are used to derive starting estimates for ‘kelim‘, even if oral data also exist.

If only oral data exist, then the oral data are used to derive a starting estimate for ‘kelim‘.

Whichever data set is used (IV or oral), the starting value for ‘kelim‘ is derived by assuming that the range of observed time values in the data set spans two elimination half-lives. This implies that the elimination half-life is equal to the midpoint of observed time values, and that the starting value for the elimination time constant ‘kelim‘ is therefore ‘log(2)‘ divided by the midpoint of observed time values.

Of course, this assumption is unlikely to be correct. However, we hope that it will yield a starting guess for ‘kelim‘ that is at least on the right order of magnitude.

Starting value for ‘Vdist‘

Using a calculated value for total clearance, ‘Cl_tot‘, ‘Vdist‘ is estimated by dividing this by the estimation of ‘kelim‘.

Starting value for ‘kgutabs‘

If oral data exist (whether or not IV data also exist), then the oral data are used to derive a starting value for ‘kgutabs‘.

First, concentrations are dose-normalized by dividing them by their corresponding doses. Then the normalized concentrations are log10-transformed.

The time of peak concentration (‘tmax‘), and the median (normalized, log-transformed) peak concentration (‘Cmax_log10‘), are identified using [get_peak()].

As a very rough guess,‘tmax‘ is assumed to occur at one absorption half-life. Under this assumption, ‘kgutabs‘ is equal to ‘log(2)/tmax‘, and this is taken as the starting value.

Starting value for ‘Fgutabs_Vdist‘

If any oral data exist (whether or not IV data also exist), then the oral data are used to derive a starting value for ‘Fgutabs_Vdist‘.

If the kinetics obey a one-compartment model, then if concentrations are dose-normalized, log-transformed, and plotted vs. time, then at late time points (after concentration has peaked), the concentration vs. time relationship will approach a straight line with slope ‘-kelim‘.

If this straight line is extrapolated back to time 0, then the resulting intercept (call it ‘A‘), expressed on the natural scale, is equal to ‘Fgutabs_Vdist * kgutabs/(kgutabs-kelim)‘. See https://www.boomer.org/c/p4/c09/c0902.php .

Roughly, we approximate ‘A‘ on the log10 scale by extrapolating back from the peak along a straight line with slope ‘-kelim‘, using the previously-derived starting value for ‘kelim‘. So ‘log10(A) = Cmax_log10 + kelim*tmax‘.

Using the previously-derived starting values for ‘kgutabs‘ and ‘kelim‘, then, the starting value for ‘Fgutabs_Vdist‘ can be derived as ‘A * (kgutabs-kelim)/kgutabs‘.

Starting value for ‘Fgutabs‘

If both oral and IV data exist, then the derived starting values for ‘Vdist‘ (from the IV data) and ‘Fgutabs_Vdist‘ (from the oral data) are multiplied to yield a derived starting value for ‘Fgutabs‘.

#Starting value for ‘Rblood2plasma‘

The starting value for ‘Rblood2plasma‘ is set to the value given by [httk::parameterize_gas_pbtk()].

Value

The same ‘data.frame‘ as ‘par_DF‘, with an additional variable ‘starts‘ containing the derived starting value for each parameter. If a parameter cannot be estimated from the available data, then its starting value will be ‘NA_real_‘

Author(s)

Caroline Ring

See Also

Other 1-compartment model functions: auc_1comp(), auc_1comp_cl(), cp_1comp(), cp_1comp_cl(), get_params_1comp(), get_params_1comp_cl(), get_params_1comp_fup(), get_starts_1comp(), get_starts_1comp_fup()

Other get_starts functions: get_starts_1comp(), get_starts_1comp_fup(), get_starts_2comp(), get_starts_flat()

Other built-in model functions: auc_1comp(), auc_1comp_cl(), auc_2comp(), auc_flat(), cp_1comp(), cp_1comp_cl(), cp_2comp(), cp_2comp_dt(), cp_flat(), get_params_1comp(), get_params_1comp_cl(), get_params_1comp_fup(), get_params_2comp(), get_params_flat(), get_starts_1comp(), get_starts_1comp_fup(), get_starts_2comp(), get_starts_flat(), tkstats_2comp(), transformed_params_2comp

---

get_starts_1comp_fup *Get starting values for 1-compartment model with specific clearance*

Description

Derive starting values for 1-compartment model parameters from available data

Usage

```
get_starts_1comp_fup(data, par_DF)
```

Arguments

data	The data set to be fitted (e.g. the result of [preprocess_data()])
par_DF	A ‘data.frame‘ with the following variables (e.g., as produced by [get_params_1comp()]) - ‘param_name‘: Character: Names of the model parameters - ‘param_units‘: Character: Units of the model parameters - ‘optimize_param‘: TRUE if each parameter is to be estimated from the data; FALSE otherwise - ‘use_param‘: TRUE if each parameter is to be used in evaluating the model; FALSE otherwise -‘lower_bounds‘: Numeric: The lower bounds for each parameter - ‘upper_bounds‘: Numeric: The upper bounds for each parameter

Details

This function is called internally by [get_params_1comp()] and should generally not be called directly by the user.

The full set of model parameters for the 1-compartment model includes ‘Vdist‘, ‘kelim‘, ‘kgutabs‘, ‘Fgutabs‘, and ‘Rblood2plasma‘. Whether each one can be estimated from the data depends on what routes of administration are included in the data. However, in this version of the one-compartment model, we use the liver and glomerular flow rates (‘Q_totli‘ and ‘Q_gfr‘, respectively) provided by [httk] as well as intrisic clearance ‘Clint‘ and fraction unbound in plasma, ‘Fup‘. These starting values are established and then are used to calculate the other parameters in the 1-compartment model.

The numerical optimizer requires starting guesses for the value of each parameter to be estimated from the data. Default starting guesses are derived from the available data.

These are intended to be *very* rough starting guesses, so the algorithm here is extremely naive. This function is not itself intended to produce valid estimates for any of the model parameters, and it is highly unlikely to do so.

The derivation process is as follows.

First, data are filtered to exclude any non-detects.

Then, data are split by route of administration, into an IV data set and an oral data set. (It is possible that either IV or oral data may not be available for a chemical.)

Starting value for ‘kelim‘

If IV data exist, then only IV data are used to derive starting estimates for ‘kelim‘, even if oral data also exist.

If only oral data exist, then the oral data are used to derive a starting estimate for ‘kelim‘.

Whichever data set is used (IV or oral), the starting value for ‘kelim‘ is derived by assuming that the range of observed time values in the data set spans two elimination half-lives. This implies that the elimination half-life is equal to the midpoint of observed time values, and that the starting value for the elimination time constant ‘kelim‘ is therefore ‘log(2)‘ divided by the midpoint of observed time values.

Of course, this assumption is unlikely to be correct. However, we hope that it will yield a starting guess for ‘kelim‘ that is at least on the right order of magnitude.

Starting value for ‘Vdist‘

Using a calculated value for total clearance, ‘Cl_tot‘, ‘Vdist‘ is estimated by dividing this by the estimation of ‘kelim‘.

Starting value for ‘kgutabs‘

If oral data exist (whether or not IV data also exist), then the oral data are used to derive a starting value for ‘kgutabs‘.

First, concentrations are dose-normalized by dividing them by their corresponding doses. Then the normalized concentrations are log10-transformed.

The time of peak concentration (‘tmax‘), and the median (normalized, log-transformed) peak concentration (‘Cmax_log10‘), are identified using [get_peak()].

As a very rough guess,‘tmax‘ is assumed to occur at one absorption half-life. Under this assumption, ‘kgutabs‘ is equal to ‘log(2)/tmax‘, and this is taken as the starting value.

Starting value for ‘Fgutabs_Vdist‘

If any oral data exist (whether or not IV data also exist), then the oral data are used to derive a starting value for ‘Fgutabs_Vdist‘.

If the kinetics obey a one-compartment model, then if concentrations are dose-normalized, log-transformed, and plotted vs. time, then at late time points (after concentration has peaked), the concentration vs. time relationship will approach a straight line with slope ‘-kelim‘.

If this straight line is extrapolated back to time 0, then the resulting intercept (call it ‘A‘), expressed on the natural scale, is equal to ‘Fgutabs_Vdist * kgutabs/(kgutabs-kelim)‘. See https://www.boomer.org/c/p4/c09/c0902.php .

Roughly, we approximate ‘A‘ on the log10 scale by extrapolating back from the peak along a straight line with slope ‘-kelim‘, using the previously-derived starting value for ‘kelim‘. So ‘log10(A) = Cmax_log10 + kelim*tmax‘.

Using the previously-derived starting values for ‘kgutabs‘ and ‘kelim‘, then, the starting value for ‘Fgutabs_Vdist‘ can be derived as ‘A * (kgutabs-kelim)/kgutabs‘.

Starting value for ‘Fgutabs‘

If both oral and IV data exist, then the derived starting values for ‘Vdist‘ (from the IV data) and ‘Fgutabs_Vdist‘ (from the oral data) are multiplied to yield a derived starting value for ‘Fgutabs‘.

#Starting value for ‘Rblood2plasma‘

The starting value for ‘Rblood2plasma‘ is set to the value given by [httk::parameterize_gas_pbtk()].

Value

The same ‘data.frame‘ as ‘par_DF‘, with an additional variable ‘starts‘ containing the derived starting value for each parameter. If a parameter cannot be estimated from the available data, then its starting value will be ‘NA_real_‘

Author(s)

Caroline Ring

See Also

Other 1-compartment model functions: auc_1comp(), auc_1comp_cl(), cp_1comp(), cp_1comp_cl(), get_params_1comp(), get_params_1comp_cl(), get_params_1comp_fup(), get_starts_1comp(), get_starts_1comp_cl()

Other get_starts functions: get_starts_1comp(), get_starts_1comp_cl(), get_starts_2comp(), get_starts_flat()

Other built-in model functions: auc_1comp(), auc_1comp_cl(), auc_2comp(), auc_flat(), cp_1comp(), cp_1comp_cl(), cp_2comp(), cp_2comp_dt(), cp_flat(), get_params_1comp(), get_params_1comp_cl(), get_params_1comp_fup(), get_params_2comp(), get_params_flat(), get_starts_1comp(), get_starts_1comp_cl(), get_starts_2comp(), get_starts_flat(), tkstats_2comp(), transformed_params_2comp(

`get_starts_2comp` *Get starting values for 2-compartment model*

Description

Derive starting values for 2-compartment model parameters from available data

Usage

```
get_starts_2comp(data, par_DF)
```

Arguments

`data`	The data set to be fitted (e.g. the result of [preprocess_data()])
`par_DF`	A 'data.frame' with the following variables (e.g., as produced by [get_params_2comp()]) - 'param_name': Character: Names of the model parameters - 'param_units': Character: Units of the model parameters - 'optimize_param': TRUE if each parameter is to be estimated from the data; FALSE otherwise - 'use_param': TRUE if each parameter is to be used in evaluating the model; FALSE otherwise -'lower_bounds': Numeric: The lower bounds for each parameter - 'upper_bounds': Numeric: The upper bounds for each parameter

Details

This function is called internally by [get_params_2comp()] and should generally not be called directly by the user.

The full set of model parameters for the 2-compartment model includes 'V1', 'kelim', 'k12', 'k21', 'kgutabs', and 'Fgutabs'. Whether each one can be estimated from the data depends on what routes of administration are included in the data.

The numerical optimizer requires starting guesses for the value of each parameter to be estimated from the data. Default starting guesses are derived from the available data.

These are intended to be *very* rough starting guesses, so the algorithm here is extremely naive. This function is not itself intended to produce valid estimates for any of the model parameters, and it is highly unlikely to do so.

At present, the starting guesses for the 2-compartment model are derived in the same way as for the 1-compartment model, for the parameters that are common to both. That is, the data are assumed to obey a 1-compartment model to derive starting guesses for 'kelim', 'V1', 'kgutabs', 'Fgutabs_V1', and 'Fgutabs'.

Then, starting values for 'k12' and 'k21' are arbitrarily set to 0.1 and 0.5, respectively.

The following description of the derivation process is therefore identical to that for [get_starts_1comp()].

The derivation process

First, data are filtered to exclude any non-detects.

Then, data are split by route of administration, into an IV data set and an oral data set. (It is possible that either IV or oral data may not be available for a chemical.)

Starting value for ‘kelim‘

If IV data exist, then only IV data are used to derive starting estimates for ‘kelim‘, even if oral data also exist.

If only oral data exist, then the oral data are used to derive a starting estimate for ‘kelim‘.

Whichever data set is used (IV or oral), the starting value for ‘kelim‘ is derived by assuming that the range of observed time values in the data set spans two elimination half-lives. This implies that the elimination half-life is equal to the midpoint of observed time values, and that the starting value for the elimination time constant ‘kelim‘ is therefore ‘log(2)‘ divided by the midpoint of observed time values.

Of course, this assumption is unlikely to be correct. However, we hope that it will yield a starting guess for ‘kelim‘ that is at least on the right order of magnitude.

Starting value for ‘V1‘

If IV data exist, then only IV data are used to derive a starting estimate for ‘V1‘.

This starting estimate is derived by assuming that the IV data obey a one-compartment model, which means that when concentrations are dose-normalized and log10-transformed and plotted against time, they will follow a straight line with slope ‘-kelim‘.

First, concentrations are dose-normalized by dividing them by their corresponding doses. Then the normalized concentrations are log10-transformed.

From all observations at the earliest observed time point in the data set (call it ‘tmin‘), the median of the dose-normalized, log10-transformed concentrations is calculated; call it ‘C_tmin‘. (The median is used, rather than the mean, in an attempt to be more robust to outliers.)

If the earliest observed time point is not at time = 0, then the dose-normalized, log10-transformed concentration at time = 0 is extrapolated by drawing a straight line with slope ‘-kelim‘ back from ‘C_tmin‘, where the value of ‘kelim‘ is the starting value derived as in the previous section.

This extrapolated concentration at time t = 0 is called ‘A_log10‘. ‘A_log10‘ represents the expected body concentration immediately after IV injection of a unit dose (under the assumption that TK obeys a one-compartment model).

Then, the volume of distribution ‘V1‘ is derived as ‘1/(10^A_log10)‘. In other words, ‘V1‘ is the volume that would be required to produce a concentration equal to ‘A_log10‘ after injecting a unit dose.

(No starting value for ‘V1‘ can be derived with only oral data, but none is needed, because with only oral data, ‘V1‘ will not be estimated from the data).

Starting value for ‘kgutabs‘

If oral data exist (whether or not IV data also exist), then the oral data are used to derive a starting value for ‘kgutabs‘.

First, concentrations are dose-normalized by dividing them by their corresponding doses. Then the normalized concentrations are log10-transformed.

The time of peak concentration (‘tmax‘), and the median (normalized, log-transformed) peak concentration (‘Cmax_log10‘), are identified using [get_peak()].

As a very rough guess,‘tmax‘ is assumed to occur at one absorption half-life. Under this assumption, ‘kgutabs‘ is equal to ‘log(2)/tmax‘, and this is taken as the starting value.

Starting value for ‘Fgutabs_V1‘

If any oral data exist (whether or not IV data also exist), then the oral data are used to derive a starting value for 'Fgutabs_V1'.

If the kinetics obey a one-compartment model, then if concentrations are dose-normalized, log-transformed, and plotted vs. time, then at late time points (after concentration has peaked), the concentration vs. time relationship will approach a straight line with slope '-kelim'.

If this straight line is extrapolated back to time 0, then the resulting intercept (call it 'A'), expressed on the natural scale, is equal to 'Fgutabs_V1 * kgutabs/(kgutabs-kelim)'. See https://www.boomer.org/c/p4/c09/c0902.php .

Roughly, we approximate 'A' on the log10 scale by extrapolating back from the peak along a straight line with slope '-kelim', using the previously-derived starting value for 'kelim'. So 'log10(A) = Cmax_log10 + kelim*tmax'.

Using the previously-derived starting values for 'kgutabs' and 'kelim', then, the starting value for 'Fgutabs_V1' can be derived as 'A * (kgutabs-kelim)/kgutabs'.

Starting value for 'Fgutabs'

If both oral and IV data exist, then the derived starting values for 'V1' (from the IV data) and 'Fgutabs_V1' (from the oral data) are multiplied to yield a derived starting value for 'Fgutabs'. #Starting value for 'Rblood2plasma'

The starting value for 'Rblood2plasma' is always set at a constant 1.

Value

The same 'data.frame' as 'par_DF', with an additional variable 'starts' containing the derived starting value for each parameter. If a parameter cannot be estimated from the available data, then its starting value will be 'NA_real_'

Author(s)

Caroline Ring

See Also

Other 2-compartment model functions: `auc_2comp()`, `cp_2comp()`, `cp_2comp_dt()`, `get_params_2comp()`, `tkstats_2comp()`, `transformed_params_2comp()`

Other get_starts functions: `get_starts_1comp()`, `get_starts_1comp_cl()`, `get_starts_1comp_fup()`, `get_starts_flat()`

Other built-in model functions: `auc_1comp()`, `auc_1comp_cl()`, `auc_2comp()`, `auc_flat()`, `cp_1comp()`, `cp_1comp_cl()`, `cp_2comp()`, `cp_2comp_dt()`, `cp_flat()`, `get_params_1comp()`, `get_params_1comp_cl()`, `get_params_1comp_fup()`, `get_params_2comp()`, `get_params_flat()`, `get_starts_1comp()`, `get_starts_1comp_cl()`, `get_starts_1comp_fup()`, `get_starts_flat()`, `tkstats_2comp()`, `transformed_params_2comp()`

`get_starts_flat` *Get starting values for flat model*

Description

Derive starting values for flat model parameters from available data

Usage

```
get_starts_flat(data, par_DF)
```

Arguments

`data`	The data set to be fitted (e.g. the result of [preprocess_data()])
`par_DF`	A 'data.frame' with the following variables (e.g., as produced by [get_params_flat()]) - 'param_name': Character: Names of the model parameters - 'param_units': Character: Units of the model parameters - 'optimize_param': TRUE if each parameter is to be estimated from the data; FALSE otherwise - 'use_param': TRUE if each parameter is to be used in evaluating the model; FALSE otherwise -'lower_bounds': Numeric: The lower bounds for each parameter - 'upper_bounds': Numeric: The upper bounds for each parameter

Details

This function is called internally by [get_params_1comp()] and should generally not be called directly by the user.

The full set of model parameters for the flat model includes 'Vdist','Fgutabs', and 'Rblood2plasma'. Whether each one can be estimated from the data depends on what routes of administration are included in the data.

The numerical optimizer requires starting guesses for the value of each parameter to be estimated from the data. Default starting guesses are derived from the available data.

These are intended to be *very* rough starting guesses, so the algorithm here is extremely naive. This function is not itself intended to produce valid estimates for any of the model parameters, and it is highly unlikely to do so.

The derivation process is as follows.

First, data are filtered to exclude any non-detects.

Then, data are split by route of administration, into an IV data set and an oral data set. (It is possible that either IV or oral data may not be available for a chemical.)

If IV data exist, then only IV data are used to derive a starting estimate for 'Vdist'. Concentrations are dose-normalized (divided by their corresponding dose) and log10-transformed. The mean dose-normalized, log10-transformed concentration is calculated (call it 'Cmean_log10'). 'Vdist' starting value is then derived as '1/(10^Cmean_log10)' .

If any oral data exist (whether or not IV data also exist), then the oral data are used to derive a starting value for 'Fgutabs_Vdist'. Concentrations are dose-normalized (divided by their corresponding dose) and log10-transformed. The mean dose-normalized, log10-transformed concentration is calculated (call it 'Cmean_log10'). 'Fgutabs_Vdist' starting value is then set equal to '10^Cmean_log10' .

#Starting value for 'Rblood2plasma'

If both blood and plasma data are available, then the starting value for 'Rblood2plasma' is derived as follows.

If IV data are available for both blood and plasma, then the starting value for 'Rblood2plasma' is derived as the ratio of 'Vdist' for blood data and 'Vdist' for plasma data.

If oral data, but not IV data, are available for both blood and plasma, then the starting value for 'Rblood2plasma' is derived as the ratio of 'Fgutabs_Vdist' for plasma data and 'Fgutabs_Vdist' for blood data.

If only blood data or only plasma data are available, then the starting value for 'Rblood2plasma' is set at a constant 1.

Value

The same 'data.frame' as 'par_DF', with an additional variable 'starts' containing the derived starting value for each parameter. If a parameter cannot be estimated from the available data, then its starting value will be 'NA_real_'

Author(s)

Caroline Ring

See Also

Other flat model functions: `auc_flat()`, `cp_flat()`, `get_params_flat()`

Other get_starts functions: `get_starts_1comp()`, `get_starts_1comp_cl()`, `get_starts_1comp_fup()`, `get_starts_2comp()`

Other built-in model functions: `auc_1comp()`, `auc_1comp_cl()`, `auc_2comp()`, `auc_flat()`, `cp_1comp()`, `cp_1comp_cl()`, `cp_2comp()`, `cp_2comp_dt()`, `cp_flat()`, `get_params_1comp()`, `get_params_1comp_cl()`, `get_params_1comp_fup()`, `get_params_2comp()`, `get_params_flat()`, `get_starts_1comp()`, `get_starts_1comp_cl()`, `get_starts_1comp_fup()`, `get_starts_2comp()`, `tkstats_2comp()`, `transformed_params_2comp()`

`get_status` *Get status*

Description

This is the S3 method generic.

Usage

```
get_status(obj, ...)
```

Arguments

obj	an object
...	Additional arguments currently not in use.

Value

The status of the ‘pk‘ object as an integer.

See Also

[get_status.pk()] for the ‘get_status‘ method for class [pk()]

---

get_status.default *Default method for getting status*

Description

Default method for getting status

Usage

```
## Default S3 method:
get_status(obj, ...)
```

Arguments

obj	an object
...	Additional arguments currently not in use.

Value

An error, when a non-pk object is used for the first argument.

`get_status.pk` *Check status of a 'pk' object*

Description

Check status of a 'pk' object

Usage

```
## S3 method for class 'pk'
get_status(obj, suppress.messages = NULL, ...)
```

Arguments

`obj`	A 'pk' object
`suppress.messages`	Logical. Whether to display messages.
`...`	Additional arguments.

Details

'pk' objects have integer statuses reflecting what stage of the analysis process they are at.

1. Object has been initialized 2. Data pre-processing complete 3. Model pre-fitting complete 4 . Model fitting complete

If a 'pk' object of status 2 or greater has its instructions modified with '+', then its status will be reset to 1, indicating that any analysis results contained in the object are now outdated and all steps of the analysis need to be re-run.

This function allows the user to check the status of a 'pk' object.

A message will be printed listing the analysis steps that have been completed for this 'pk' object, and the integer status will be returned.

Value

The status of the 'pk' object as an integer.

Author(s)

Caroline Ring

get_stat_error_model *get_stat_error_model()*

Description

This is the S3 method generic for get_stat_error_model()

Usage

```
get_stat_error_model(obj, ...)
```

Arguments

obj	An object.
...	Additional arguments currently not in use.

Value

A named list of the stat_error_model settings

See Also

[get_stat_error_model.pk()] for the method for class [pk()]

get_stat_error_model.default *Default method for get_stat_error_model()*

Description

Default method for get_stat_error_model()

Usage

```
## Default S3 method:
get_stat_error_model(obj, ...)
```

Arguments

obj	An object
...	Additional arguments currently not in use.

Value

An error, when a non-pk object is used for the first argument.

get_stat_error_model.pk *Get stat_error_model*

Description

Get stat_error_model

Usage

```
## S3 method for class 'pk'
get_stat_error_model(obj, ...)
```

Arguments

obj	A [pk()] object
...	Additional arguments.

Value

A named list of the stat_error_model settings

Author(s)

Caroline Ring

get_stat_model *get_stat_model()*

Description

This is the S3 method generic for get_stat_model()

Usage

```
get_stat_model(obj, ...)
```

Arguments

obj	An object.
...	Additional arguments currently not in use.

Value

A ‘list‘ – the ‘stat_model‘ element of ‘obj‘

See Also

[get_stat_model.pk()] for the method for class [pk()]

`get_stat_model.default` *Default method for get_stat_model()*

Description

Default method for get_stat_model()

Usage

```
## Default S3 method:
get_stat_model(obj, ...)
```

Arguments

`obj`	An object
`...`	Additional arguments currently not in use.

Value

An error, when a non-pk object is used for the first argument.

`get_stat_model.pk` *Get stat_model*

Description

Get stat_model

Usage

```
## S3 method for class 'pk'
get_stat_model(obj, ...)
```

Arguments

`obj`	A [pk()] object
`...`	Additional arguments. Currently not in use.

Value

A ‘list‘ – the ‘stat_model‘ element of ‘obj‘

Author(s)

Caroline Ring

`get_tkstats` *Get TK stats*

Description

This is the S3 method generic for get_tkstats(0)

Usage

```
get_tkstats(obj, ...)
```

Arguments

`obj`	an object
`...`	Additional arguments currently not in use.

Value

A data.frame with one row for each ‘data_group‘, ‘model‘ and ‘method‘ with the variables in the ‘data.frame‘ returned by the ‘tkstats_fun‘ for its corresponding model. (For the built-in models ‘model_flat‘, ‘model_1comp‘, and ‘model_2comp‘, these variables are ‘param_name‘ and ‘param_value‘.)

See Also

[get_tkstats.pk()] for the method for class [pk()]

`get_tkstats.default` *Default method for get_tkstats()*

Description

Default method for get_tkstats()

Usage

```
## Default S3 method:
get_tkstats(obj, ...)
```

Arguments

`obj`	an object
`...`	Additional arguments currently not in use.

Value

An error, when a non-pk object is used for the first argument.

`get_tkstats.pk` *Get TK stats*

Description

Extract derived TK statistics from a fitted [pk()] model object.

Usage

```
## S3 method for class 'pk'
get_tkstats(
  obj,
  newdata = NULL,
  tk_group = NULL,
  model = NULL,
  method = NULL,
  exclude = TRUE,
  vol_unit = "L",
  dose_norm = TRUE,
  suppress.messages = NULL,
  ...
)
```

Arguments

`obj`	A [pk()] model object. Must be fitted, or the function will exit with an error.
`newdata`	Optional: A ‘data.frame‘ containing new data for which to compute the TK stats. Must contain at least variables ‘Chemical‘, ‘Species‘, ‘Route‘, ‘Media‘, ‘Dose‘, ‘Dose.Units‘, ‘Conc.Units‘, either ‘Time_trans.Units‘ or ‘Time.Units‘, and any other variables named in ‘tk_grouping‘. Default ‘NULL‘, to use the data in ‘obj$data‘.
`tk_group`	A list of variables provided using a ‘dplyr::vars()‘ call. The data (either ‘newdata‘ or ‘obj$data‘) will be grouped according to the unique combinations of these variables. For each unique combination of these variables in the data, a set of TK statistics will be computed. The default is ‘obj$settings_data_info$summary_group‘, to derive TK statistics for the same groups of data as non-compartmental analysis statistics. With the default, you can directly compare e.g. a model-predicted AUC_inf to the corresponding NCA-estimated AUC_inf. However, you may specify a different data grouping if you wish. Each group should have a unique combination of ‘Chemical‘, ‘Species‘, ‘Route‘, ‘Media‘, and ‘Dose‘, because the TK stats depend on these values, and it is required to have one unique set of TK stats per group.

model	Character: One or more of the models fitted. Default ‘NULL‘ to return TK stats for all models.
method	Character: One or more of the [optimx::optimx()] methods used. Default ‘NULL‘ to return TK stats for all methods.
exclude	Logical: ‘TRUE‘ to get the TK groupings after removing any observations in the data marked for exclusion (if there is a variable ‘exclude‘ in the data, an observation is marked for exclusion when ‘TRUE‘). ‘FALSE‘ to include all observations when getting the TK groupings, regardless of exclusion status. Default ‘TRUE‘.
vol_unit	Character: Specifies the unit of volume. Defaults to "L" for liters.
dose_norm	Logical: ‘TRUE‘ (default) specifies whether the concentrations are dose-normalized.
suppress.messages	Logical: whether to suppress message printing. If NULL (default), uses the setting in ‘obj$settings_preprocess$suppress.messages‘
...	Additional arguments not currently in use.

Details

After fitting model parameters (e.g. elimination rate, volume of distribution, absorption rate, bioavailability), it can be useful to derive summary toxicokinetic statistics such as total clearance rate, half-life, peak concentration, AUC_inf (the area under the concentration-time curve when time goes to infinity), etc.

Many of these TK statistics depend not only on chemical and species, but also on route, media (tissue), and dose. Therefore, TK stats need to be computed for a specific set of Chemical, Species, Route, Media, and Dose.

TK statistics for a defined [pk_model()] object are computed using the function named in the model’s ‘tkstats_fun‘. For the built-in models, the ‘tkstats_fun‘ functions are the following. See the documentation for the individual functions for details on what TK stats are calculated for each model, and how they are calculated. -‘model_1comp‘: [tkstats_1comp()] -‘model_2comp‘: [tkstats_2comp()] -‘model_flat‘: [tkstats_flat()]

Value

A data.frame with one row for each ‘data_group‘, ‘model‘ and ‘method‘ with the variables in the ‘data.frame‘ returned by the ‘tkstats_fun‘ for its corresponding model. (For the built-in models ‘model_flat‘, ‘model_1comp‘, and ‘model_2comp‘, these variables are ‘param_name‘ and ‘param_value‘.)

Author(s)

Caroline Ring, Gilberto Padilla Mercado

See Also

Other methods for fitted pk objects: AAFE.pk(), AFE.pk(), AIC.pk(), BIC.pk(), coef.pk(), coef_sd.pk(), eval_tkstats.pk(), get_fit.pk(), get_hessian.pk(), logLik.pk(), predict.pk(), residuals.pk(), rmse.pk(), rsq.pk()

`get_winning_model` *get_winning_model()*

Description

This is the S3 method generic for get_winning_model()

Usage

```
get_winning_model(obj, ...)
```

Arguments

`obj`	An object.
`...`	Additional arguments currently not in use.

Value

A data.frame with one row for each ‘data_group‘, ‘model‘ and ‘method‘ and The return value has attribute ‘criterion‘ giving the name of the criterion function used to compare models.

See Also

[get_winning_model.pk()] for the method for class [pk()]

`get_winning_model.default` *Default method for get_winning_model()*

Description

Default method for get_winning_model()

Usage

```
## Default S3 method:
get_winning_model(obj, ...)
```

Arguments

`obj`	An object
`...`	Additional arguments currently not in use.

Value

An error, when a non-pk object is used for the first argument.

`get_winning_model.pk` *Get winning model*

Description

Get winning model for a fitted ‘pk‘ object

Usage

```
## S3 method for class 'pk'
get_winning_model(obj, newdata = NULL, method = NULL, criterion = "AIC", ...)
```

Arguments

`obj`	A [pk()] object
`newdata`	Optional: A ‘data.frame‘ containing new data to plot. Must contain at least variables ‘Chemical‘, ‘Species‘, ‘Route‘, ‘Media‘, ‘Dose‘, ‘Time‘, ‘Time.Units‘, ‘Conc‘, ‘Detect‘, ‘Conc_SD‘. Default ‘NULL‘, to use the data in ‘obj$data‘.
`method`	Character: One or more of the [optimx::optimx()] methods used in fitting. The winning model will be determined for each of these methods. Default ‘NULL‘ to get the winning model for each method in ‘obj$settings_optimx$method‘.
`criterion`	The name of a criterion function to use for model comparison. Default "AIC". Must be the name of a function that (as for ‘AIC‘) accepts arguments ‘obj‘, ‘newdata‘, ‘method‘ and ‘model‘ (may accept other arguments, specified in ‘...‘) and returns output as for ‘AIC‘: a data.frame with a column with the same name as ‘criterion‘ that has calculated values for model comparison. The "winning" value will be the smallest value.
`...`	Optional: Other arguments to ‘criterion‘ function.

Details

Get the winning model (i.e. the model with the lowest value of the criterion specified in ‘criterion‘) for a fitted ‘pk‘ object, for a specified method, and optionally for a specified new dataset. When there are ties it will return the first encounter, where the priority is: model_1comp > model_2comp > model_flat.

Value

A data.frame with one row for each ‘data_group‘, ‘model‘ and ‘method‘ and The return value has attribute ‘criterion‘ giving the name of the criterion function used to compare models.

Author(s)

Caroline Ring, Gilberto Padilla Mercado

hess_sd1 *Inverse diagonal, method 1*

Description

Get square root of diagonal of inverse matrix, first method

Usage

```
hess_sd1(m)
```

Arguments

m	A square numeric matrix.

Details

Invert a square numeric matrix ‘m‘ of size $n \times n$ using [solve()], then take the square root of the diagonal.

Value

A numeric vector of length ‘n‘.

Author(s)

Caroline Ring

hess_sd2 *Inverse diagonal, method 2*

Description

Get square root of diagonal of inverse matrix, second method

Usage

```
hess_sd2(m)
```

Arguments

m	A square numeric matrix, $n \times n$.

Details

Following the procedure outlined in Gill & King (2004): Calculate generalized inverse of a matrix ‘m‘ using [MASS::ginv()]. Then perform a generalized Cholesky factorization of the generalized inverse using [Matrix::Cholesky()] with ‘perm = TRUE‘. Reconstruct the generalized inverse as

$$\left(m^{-1} + E\right) = P_1' L L' P_1$$

This should ensure positive semi-definiteness of the reconstruction.

Then, take the diagonal of $\left(m^{-1} + E\right)$, and take the square root.

Value

A numeric vector of length n.

Author(s)

Caroline Ring

References

Gill J, King G. (2004) What to Do When Your Hessian is Not Invertible: Alternatives to Model Respecification in Nonlinear Estimation. Sociological Methods & Research 33(1):54-87. DOI: 10.1177/0049124103262681

---

ignore_unused_imports *Ignore unused imports*

Description

Placeholder function to appease R CMD CHECK

Usage

```
ignore_unused_imports()
```

Details

This function does nothing and should be ignored by the user.

Why it exists: Whenever possible, ‘invivopkfit‘ code calls functions from other packages using the syntax ‘package::function()‘, which means that the whole package does not have to be loaded, nor does the function itself have to be loaded until it is used. The relevant packages are listed in the ‘invivofit‘ package ‘DESCRIPTION‘ file under ‘Imports‘, because they must be installed to use ‘invivopkfit‘. But because the packages are not actually loaded, this creates a (spurious) NOTE from ‘R CMD CHECK‘ about a declared Import not being used. This function is a workaround to suppress that NOTE. It does nothing except contain namespace-qualified references (not calls) to objects in the relevant packages.

Author(s)

Caroline Ring

References

https://r-pkgs.org/dependencies-in-practice.html#how-to-not-use-a-package-in-imports

---

is.pk *Check whether an object is of class 'pk'*

Description

Check whether an object is of class 'pk'

Usage

```
is.pk(obj)
```

Arguments

obj	The object whose class is to be tested

Value

TRUE if the object inherits from class 'pk', FALSE if it does not

Author(s)

Caroline Ring

---

is.pkproto *Is an object pkproto?*

Description

Is an object pkproto?

Usage

```
is.pkproto(obj)
```

Arguments

obj	An object

Value

TRUE if ‘obj‘ inherits from class ‘pkproto‘; FALSE if not

Author(s)

Caroline Ring

---

is.pk_faceted *Check whether an object is of class ‘pk_faceted‘*

Description

Check whether an object is of class ‘pk_faceted‘

Usage

```
is.pk_faceted(obj)
```

Arguments

obj	The object whose class is to be tested

Value

TRUE if the object inherits from class ‘pk‘, FALSE if it does not

Author(s)

Caroline Ring

---

is.pk_scales *Is an object class ‘pk_scales‘?*

Description

Is an object class ‘pk_scales‘?

Usage

```
is.pk_scales(obj)
```

Arguments

obj	An object

Value

TRUE if ‘obj‘ inherits from class ‘pk_scales‘; FALSE if not

Author(s)

Caroline Ring

`logLik.pk` *Log-likelihood*

Description

Extract log-likelihood(s) from a fitted ‘pk‘ object

Usage

```
## S3 method for class 'pk'
logLik(
  object,
  newdata = NULL,
  model = NULL,
  method = NULL,
  negative = FALSE,
  force_finite = FALSE,
  exclude = TRUE,
  drop_obs = TRUE,
  ...
)
```

Arguments

`object`	A ‘pk‘ object
`newdata`	Optional: A ‘data.frame‘ with new data for which to compute log-likelihood. If NULL (the default), then log-likelihoods will be computed for the data in ‘obj$data‘. ‘newdata‘ is required to contain at least the following variables: ‘Time‘, ‘Time.Units‘, ‘Dose‘, ‘Route‘,‘Media‘, ‘Conc‘, ‘Detect‘, ‘Conc_SD‘, ‘N_Subjects‘. ‘Time‘ will be transformed according to the transformation in ‘obj$scales$time‘ before making predictions. ‘Conc‘ will be transformed according to the transformation in ‘obj$scales$conc‘.
`model`	Optional: Specify one or more of the fitted models for which to calculate log-likelihood. If NULL (the default), log-likelihoods will be returned for all of the models in ‘obj$stat_model‘.
`method`	Optional: Specify one or more of the [optimx::optimx()] methods for which to calculate log-likelihoods. If NULL (the default), log-likelihoods will be returned for all of the models in ‘obj$optimx_settings$method‘.

negative	Logical: Whether to return the *negative* log-likelihood (i.e., the log-likelihood multiplied by negative 1). Default 'FALSE'.
force_finite	Logical: Whether to force return of a finite value (e.g. as required by method 'L-BFGS-B' in [optimx::optimx()]). Default FALSE. If TRUE, then if the log-likelihood works out to be non-finite, then it will be replaced with '.Machine$double.xmax'.
exclude	Logical: 'TRUE' to compute the log-likelihood excluding any observations in the data marked for exclusion (if there is a variable 'exclude' in the data, an observation is marked for exclusion when 'exclude log-likelihood, regardless of exclusion status. Default 'TRUE'.
drop_obs	Logical: 'TRUE' to drop the observations column after calculating log-likelihood.
...	Additional arguments. Not in use currently.

Details

For details on how the log-likelihood is calculated, see [log_likelihood()].

New levels in 'newdata'

The log-likelihood requires an error variance for each observation. Depending on the error model used for the model fit, each observation may have a different corresponding error variance, based on its unique combinations of levels of factor variables as specified in [stat_error_model()] when setting up the [pk()] object. (The error model specified in the 'pk' object can be viewed using [get_error_group()]).

If you are supplying new data in 'newdata', then the unique combinations of the factor levels for the new observations will be used to find the matching error hyperparameter. If the new data contains new combinations of factor levels not found in the data used to fit the model, then the following procedure will be used to calculate the log-likelihood for the observations with new levels:

If there are $i = 1, 2, ...G$ groups with unique combinations of the factor levels in the original data, then there are corresponding error standard deviations $\sigma_i = \sigma_1, \sigma_2, ..., \sigma_G$.

Each observation with a new combination of factor levels will have G differerent log-likelihoods computed, as though it were part of each of the G existing groups. Then, the average of these G log-likelihoods will be taken and assigned to the observation. In effect, each observation with a new level is treated as though it is equally likely to belong to any of the existing groups.

Scaling and transformation of concentration variables in 'newdata'

This function differs from some of the other methods for a fitted [pk()] object that accept 'newdata', in that there is no 'use_scale_conc' argument for [logLik.pk()]. You cannot specify a different scaling/transformation for concentration variables – you have to use the same scaling/transformation that was used to fit the model. This is because the log-likelihood depends on the fitted values of the error variance hyperparameters, and those are valid only for the transformation of the concentration data that was used to fit the model.

Value

A data.frame with coefficients and log-likelihood values calculated using 'newdata'. There is one row for each model in 'obj''s [stat_model()] element and each [optimx::optimx()] method (specified in [settings_optimx()]).

Author(s)

Caroline Ring, Gilberto Padilla Mercado

See Also

Other fit evaluation metrics: AAFE.pk(), AFE.pk(), AIC.pk(), BIC.pk(), rmse.pk(), rsq.pk()

Other log likelihood functions: AIC.pk(), BIC.pk()

Other methods for fitted pk objects: AAFE.pk(), AFE.pk(), AIC.pk(), BIC.pk(), coef.pk(), coef_sd.pk(), eval_tkstats.pk(), get_fit.pk(), get_hessian.pk(), get_tkstats.pk(), predict.pk(), residuals.pk(), rmse.pk(), rsq.pk()

---

log_likelihood *Log-likelihood*

Description

The log-likelihood function (probability of data given model parameters).

Usage

```
log_likelihood(
  par,
  const_params = NULL,
  data = NULL,
  data_sigma_group = NULL,
  modelfun = NULL,
  dose_norm = FALSE,
  log10_trans = FALSE,
  negative = TRUE,
  force_finite = FALSE,
  max_multiplier = NULL,
  includes_preds = FALSE,
  suppress.messages = TRUE
)
```

Arguments

par	A named list of parameters and their values that are being optimized.
const_params	Optional: A named list of parameters and their values that are being held constant.
data	A 'data.frame' of data with harmonized variable names. Required: 'Time_trans', 'Dose', 'Conc', 'Detect', 'N_Subjects', 'Conc_SD'. 'Conc' and 'Conc_SD' will be transformed according to 'dose_norm' and 'log10_trans'.
data_sigma_group	A 'factor' vector which could be a new variable in 'data', giving the error group for each row in 'data'.

`modelfun`	Character or function: The name of the function that produces concentration predictions for the model being evaluated.
`dose_norm`	Logical: Whether to dose-normalize predicted and observed concentrations before calculating likelihood.
`log10_trans`	Logical: Whether to apply a [log10()] transformation to predicted and observed concentrations before calculating likelihood.
`negative`	Logical: Whether to return the *negative* log-likelihood (i.e., the log-likelihood multiplied by negative 1). Default TRUE, to multiply the log-likelihood by negative 1 before returning it. This option is useful when treating the log-likelihood as an objective function to be *minimized* by an optimization algorithm.
`force_finite`	Logical: Whether to force return of a finite value (e.g. as required by method 'L-BFGS-B' in [optimx::optimx()]). Default FALSE. If TRUE, then if the log-likelihood works out to be non-finite, then it will be replaced with '.Machine$double.xmax'.
`max_multiplier`	Numeric, but NULL by default. Determines which multiple of the maximum concentration to use to limit possible predictions. Note: only use this as part of the fitting function.
`includes_preds`	Logical: whether 'data' includes predictions.
`suppress.messages`	Logical.

Details

The log-likelihood is formulated by assuming that residuals (transformed model-predicted concentrations minus transformed observed concentrations) are independent, and that groups of residuals obey zero-mean normal distributions where each group may have a separate error variance. Error groups are defined as unique combinations of variables in the harmonized data, by a command such as 'pk(data = ...) + stat_error_model(error_group = vars(...)'.

Log-likelihood equations

For chemical-species combination i and study j, define the following quantities.

y_{ijk} is the k^{th} observation of concentration (which may be transformed, e.g. dose-normalized and/or log10-transformed), corresponding to dose d_{ijk} and time t_{ijk}. Each observation has a corresponding LOQ, LOQ_{ijk}.

For multiple-subject observations, y_{ijk} is the k^{th} *sample mean* observed concentration for chemical-species combination i and study j, corresponding to dose d_{ijk} and time t_{ijk}. It represents the mean of n_{ijk} individual measurements. It has a corresponding sample standard deviation, s_{ijk}. In the harmonized data, s_{ijk} is contained in variable 'Conc_SD'.

$\bar{\theta}_i$ represents the vector of model parameters supplied in argument 'params' for chemical-species combination i.

μ_{ijk} is the model-predicted concentration for dose d_{ijk} and time t_{ijk}. If $f(d, t; \bar{\theta})$ is the model function evaluated at dose d and time t, with parameter vector $\bar{\theta}$, then

$$\mu_{ijk} = f\left(d_{ijk}, t_{ijk}; \bar{\theta}_i\right)$$

σ_{ij}^2 is the study- and chemical-specific residual variance. (It is a hyperparameter.)

Single-subject observations above limit of quantification (detects)

This is the normal probability density function evaluated at the observed concentration, as implemented in [stats::dnorm()].

$$LL_{ijk} = \log\left(\frac{1}{\sqrt{\sigma_{ij}2\pi}}\exp\left[\frac{-1}{2}\left(\frac{y_{ijk}-\mu_{ijk}}{\sigma_{ij}}\right)^2\right]\right)$$

Single-subject observations below limit of quantification (non-detects)

This is the normal cumulative density function evaluated at the LOQ, as implemented in [stats::pnorm()]. It is the total probability of observing a concentration anywhere between 0 and the LOQ.

$$LL_{ijk} = \log\left(\frac{1}{2}\left[1+\text{erf}\left(\frac{\text{LOQ}_{ijk}-\mu_{ijk}}{\sigma_{ij}\sqrt{2}}\right)\right]\right)$$

Multiple-subject observations above limit of quantification

This is the joint log-likelihood across the multiple subjects included in one observation, re-expressed in terms of the sample mean, sample SD, and number of subjects for that observation. It is implemented in [dnorm_summary()].

$$LL_{ijk} = n_{ijk} * \log\left[\frac{1}{\sigma_{ij}\sqrt{2\pi}} + \frac{-1}{2\sigma_{ij}^2}\left(\left(n_{ijk}-1\right)s_{ijk}^2 + n_{ijk}\left(y_{ijk}-\mu_{ijk}\right)^2\right)\right]$$

Multiple-subject observations below limit of quantification

This case is not implemented. If sample mean concentration is reported below LOQ, then it is unclear what individual observed concentrations are represented, and how they were combined to produce the summary data in terms of sample mean and sample SD. Were all individual observations below LOQ? Or were below-LOQ observations replaced with 0, LOQ/2, etc. before sample mean and sample SD were computed? If the sample mean is reported below LOQ, what LOQ is reported? Did individual observations all have the same LOQ, or is an average or median LOQ being used? It is impossible to formulate the log-likelihood without knowing the answers to these questions. Therefore, multiple-subject observations below LOQ are excluded from analysis (they are marked as excluded in [preprocess_data()]).

Joint log-likelihood for a chemical and species

The joint log-likelihood for a chemical and species is simply the sum of log-likelihoods across observations.

$$LL_i = \sum_{j=1}^{J_i}\sum_{k=1}^{K_{ij}} LL_{ijk}$$

This is the overall probability of the observed data, given the model and parameters.

Value

A log-likelihood value for the data given the parameter values in params

Author(s)

Caroline Ring, Gilberto Padilla Mercado

`mapping` *New mapping*

Description

New mapping

Usage

```
mapping(
  mapping = ggplot2::aes(Chemical = analyte_dtxsid, Chemical_Name =
   analyte_name_original, DTXSID = analyte_dtxsid, CASRN = analyte_casrn, Species =
    species, Reference = document_id, Media = conc_medium_normalized, Route =
   administration_route_normalized, Dose = dose_level_corrected, Dose.Units = "mg/kg",
   Subject_ID = subject_id, Series_ID = series_id, Study_ID = study_id, ConcTime_ID =
   conc_time_id, N_Subjects = n_subjects_in_series, Weight = weight_kg, Weight.Units =
    "kg", Time = time_hr, Time.Units = "hours",
     Value = conc, Value.Units = "mg/L",
    Value_SD = conc_sd_normalized, LOQ = loq),
  ...
)
```

Arguments

`mapping`	A [ggplot2::aes()] call that maps variable names in the original data to the harmonized 'invivoPKfit' variable names.
`...`	Additional arguments. Currently unused.

Value

An object of class 'uneval' containing the mapping – see [ggplot2::aes()] for details.

Author(s)

Caroline Ring

midpt_log10 *Log10-scaled midpoint*

Description

Log10-scaled midpoint

Usage

```
midpt_log10(x)
```

Arguments

x	A numeric vector

Value

The log10-scaled midpoint, calculated as ‘log10(mean(range(x))‘

model_1comp *1-compartment model*

Description

The ‘pk_model‘ object defining the 1-compartment model.

Usage

```
model_1comp
```

Format

An object of class `list` (inherits from `pk_model`) of length 10.

Details

A ‘pk_model‘ object: under the hood, a ‘list‘ object with named elements corresponding to the arguments of [pk_model()]. See that function documentation for the definition of each element.

See [cp_1comp()] for the function that predicts blood/plasma concentration for a bolus dose (oral or IV).

See [auc_1comp()] for the function that predicts area under the concentration-time curve.

See [tkstats_1comp()] for the function that calculates summary toxicokinetic statistics from 1-compartment model parameters.

See [params_1comp()] for the function that determines bounds and starting guesses for model parameters, based on the data.

model_1comp_cl_nonrest *1-compartment model that assumes non-restrictive clearance*

Description

The ‘pk_model‘ object defining the 1-compartment model.

Usage

```
model_1comp_cl_nonrest
```

Format

An object of class `list` (inherits from `pk_model`) of length 10.

Details

A ‘pk_model‘ object: under the hood, a ‘list‘ object with named elements corresponding to the arguments of [pk_model()]. See that function documentation for the definition of each element.

See [cp_1comp_cl()] for the function that predicts blood/plasma concentration for a bolus dose (oral or IV).

See [auc_1comp_cl()] for the function that predicts area under the concentration-time curve.

See [tkstats_1comp_cl()] for the function that calculates summary toxicokinetic statistics from 1-compartment model parameters.

See [params_1comp_cl()] for the function that determines bounds and starting guesses for model parameters, based on the data.

model_1comp_cl_rest *1-compartment model that assumes restrictive clearance*

Description

The ‘pk_model‘ object defining the 1-compartment model.

Usage

```
model_1comp_cl_rest
```

Format

An object of class `list` (inherits from `pk_model`) of length 10.

Details

A ‘pk_model‘ object: under the hood, a ‘list‘ object with named elements corresponding to the arguments of [pk_model()]. See that function documentation for the definition of each element.

See [cp_1comp_cl()] for the function that predicts blood/plasma concentration for a bolus dose (oral or IV).

See [auc_1comp_cl()] for the function that predicts area under the concentration-time curve.

See [tkstats_1comp_cl()] for the function that calculates summary toxicokinetic statistics from 1-compartment model parameters.

See [params_1comp_cl()] for the function that determines bounds and starting guesses for model parameters, based on the data.

---

model_1comp_fup — *1-compartment model that assumes restrictive clearance optimizes Fup*

Description

The ‘pk_model‘ object defining the 1-compartment model.

Usage

```
model_1comp_fup
```

Format

An object of class `list` (inherits from `pk_model`) of length 10.

Details

A ‘pk_model‘ object: under the hood, a ‘list‘ object with named elements corresponding to the arguments of [pk_model()]. See that function documentation for the definition of each element.

See [cp_1comp_cl()] for the function that predicts blood/plasma concentration for a bolus dose (oral or IV).

See [auc_1comp_cl()] for the function that predicts area under the concentration-time curve.

See [tkstats_1comp_cl()] for the function that calculates summary toxicokinetic statistics from 1-compartment model parameters.

See [params_1comp_cl()] for the function that determines bounds and starting guesses for model parameters, based on the data.

model_2comp *2-compartment model*

Description

The ‘pk_model‘ object defining the 2-compartment model.

Usage

```
model_2comp
```

Format

An object of class `list` (inherits from `pk_model`) of length 10.

Details

A ‘pk_model‘ object: under the hood, a ‘list‘ object with elements corresponding to the arguments of [pk_model()]. See that function documentation for the definition of each element.

See [cp_2comp()] for the function that predicts blood/plasma concentration for a bolus dose (oral or IV).

See [auc_2comp()] for the function that predicts area under the concentration-time curve.

See [tkstats_2comp()] for the function that calculates summary toxicokinetic statistics from 1-compartment model parameters.

See [params_2comp()] for the function that determines bounds and starting guesses for model parameters, based on the data.

model_flat *Flat model*

Description

The ‘pk_model‘ object defining the flat model.

Usage

```
model_flat
```

Format

An object of class `list` (inherits from `pk_model`) of length 10.

Details

A ‘pk_model‘ object: under the hood, a ‘list‘ object with elements corresponding to the arguments of [pk_model()]. See that function documentation for the definition of each element.

See [cp_flat()] for the function that predicts blood/plasma concentration for a bolus dose (oral or IV).

See [auc_flat()] for the function that predicts area under the concentration-time curve.

See [tkstats_flat()] for the function that calculates summary toxicokinetic statistics from 1-compartment model parameters.

See [params_flat()] for the function that determines bounds and starting guesses for model parameters, based on the data.

`nca` *nca()*

Description

This is the S3 method generic for nca()

Usage

```
nca(obj, ...)
```

Arguments

`obj`	An object.
`...`	Additional arguments currently not in use.

Value

A ‘data.frame‘ with variables including all the grouping variables in ‘nca_group‘, ‘nca_group_id‘; ‘design‘ (the auto-detected study design for this group); ‘param_name‘ (the name of the NCA parameter); ‘param_value‘ (the NCA parameter value); ‘param_sd_z‘ (standard deviation of the estimated NCA parameter value, if available); ‘param_units‘ (the units of the NCA parameter, derived from the units of the data).

See Also

[nca.pk()] for the method for class [pk()]

nca.default *Default method for nca()*

Description

Default method for nca()

Usage

```
## Default S3 method:
nca(obj, ...)
```

Arguments

obj	An object
...	Additional arguments currently not in use.

Value

An error, when a non-pk object is used for the first argument.

nca.pk *NCA for a 'pk' object*

Description

Non-compartmental analysis for a 'pk' object

Usage

```
## S3 method for class 'pk'
nca(
  obj,
  newdata = NULL,
  nca_group = NULL,
  exclude = TRUE,
  dose_norm = FALSE,
  suppress.messages = NULL,
  ...
)
```

Arguments

obj	A [pk()] model object. Must be fitted, or the function will exit with an error.
newdata	Optional: A ‘data.frame‘ containing new data for which to compute the TK stats. Must contain at least variables ‘Chemical‘, ‘Species‘, ‘Route‘, ‘Dose‘, ‘Conc‘, ‘Dose.Units‘, ‘Conc.Units‘, and ‘Time.Units‘, and any other variables named in ‘tk_grouping‘. Default ‘NULL‘, to use the data in ‘get_data(obj)‘.
nca_group	A list of variables provided using a ‘dplyr::vars()‘ call. The data (either ‘newdata‘ or ‘obj$data‘) will be grouped according to the unique combinations of these variables. For each unique combination of these variables in the data, a set of TK statistics will be computed. The default is ‘NULL‘, to use the same data grouping that was set in [stat_nca()] for the ‘pk‘ object. However, you may specify a different data grouping if you wish.
exclude	Logical: ‘TRUE‘ to group the data for NCA after removing any observations in the data marked for exclusion (if there is a variable ‘exclude‘ in the data, an observation is marked for exclusion when ‘exclude NCA, regardless of exclusion status. Default ‘TRUE‘.
dose_norm	Logical: ‘TRUE‘ to perform NCA after dose-normalizing concentrations. ‘FALSE‘ (default) to perform NCA on un-transformed concentrations.
suppress.messages	Logical: whether to suppress message printing. If NULL (default), uses the setting in ‘obj$settings_preprocess$suppress.messages‘
...	Additional arguments. Currently not in use.

Details

Perform non-compartmental analysis of data in a ‘pk‘ object (or optionally, new data), using data groupings defined by ‘get_nca_group()‘ for the ‘pk‘ object (or optionally, new groupings). If you provide both ‘newdata‘ and ‘nca_group‘, then everything in the ‘pk‘ object will be ignored and you will simply be doing NCA *de novo* (which may be what you want).

Value

A ‘data.frame‘ with variables including all the grouping variables in ‘nca_group‘, ‘nca_group_id‘; ‘design‘ (the auto-detected study design for this group); ‘param_name‘ (the name of the NCA parameter); ‘param_value‘ (the NCA parameter value); ‘param_sd_z‘ (standard deviation of the estimated NCA parameter value, if available); ‘param_units‘ (the units of the NCA parameter, derived from the units of the data).

Author(s)

Caroline Ring

pk *Create a new 'pk' object*

Description

[pk()] initializes a new 'pk' object.

Usage

```
pk(
  data = NULL,
  mapping = ggplot2::aes(Chemical = analyte_dtxsid, Chemical_Name =
   analyte_name_original, DTXSID = analyte_dtxsid, CASRN = analyte_casrn, Species =
   species, Reference = fk_extraction_document_id, Media = conc_medium_normalized, Route
   = administration_route_normalized, Dose = invivPK_dose_level, Dose.Units = "mg/kg",
    Subject_ID = fk_subject_id, Series_ID = fk_series_id, Study_ID = fk_study_id,
   ConcTime_ID = conc_time_id, N_Subjects = n_subjects_normalized, Weight = weight_kg,
    Weight.Units = "kg", Time = time_hr,
     Time.Units = "hours", Value =
   invivPK_conc, Value.Units = "mg/L", Value_SD = invivPK_conc_sd, LOQ = invivPK_loq),
  settings_preprocess_args = list(),
  settings_data_info_args = list(),
  settings_optimx_args = list(),
  scale_conc_args = list(),
  scale_time_args = list(),
  stat_model_args = list(),
  stat_error_model_args = list(),
  facet_data_args = list()
)
```

Arguments

data	A 'data.frame'. The default is an empty data frame.
mapping	A mapping set up by (ab)using [ggplot2::aes()]. Call is of form 'ggplot2::aes(new_variable = old_variable)' 'new_variable' represents the harmonized variable name that will be used within 'invivopkfit'; 'old_variable' represents the variable name in 'data'. If you want to provide a fixed/constant value for a 'new_variable' rather than taking its value from a variable in 'data', simply supply that fixed/constant value in the 'old_variable' position.
settings_preprocess_args	A list of preprocessing settings.
settings_data_info_args	A list of data_info settings.
settings_optimx_args	A list of optimx settings.

`scale_conc_args`
: A list of concentration value scaling arguments.

`scale_time_args`
: A list of time scaling arguments

`stat_model_args`
: A list of TK model arguments.

`stat_error_model_args`
: A list of error modeling arguments

`facet_data_args`
: A list of data grouping settings.

Details

[pk()] is used to construct the initial ‘pk‘ object for analysis. It is almost always followed by ‘+‘ to add steps to the workflow. For example, you could use ‘pk(my_data) + stat_model(model = ’1comp’)‘ to set up for fitting a 1-compartment model.

The ‘pk‘ object

A ‘pk‘ object consists of a set of concentration-dose-time data to be fitted, and sets of instructions for steps in the analysis workflow:

- settings for how to pre-process the data (harmonizing variable names, imputing missing data, calculating derived variables) - scalings/transformations to be applied to the data - settings for the numerical optimization algorithm to be used to fit any model - optionally: which PK model(s) should be fitted to this dataset. (You do not have to fit any PK model if you don’t want to; you can instead just set up the ‘pk‘ object with data, and do non-compartmental analysis on it.)

No data processing, model fitting, or any other analysis is done until you explicitly request it. Until then, the ‘pk‘ object remains just a set of data and instructions. This allows you to specify the instructions for each analysis step without regard for the actual order of the analysis steps, and to overwrite previous instructions, without having to re-do the model fitting each time you add or change a set of instructions. This is particularly useful if you are working in interactive mode at the R console.

Mappings

Your input data can have any variable names you like. However, internally, ‘invivopkfit‘ needs to use a set of "standard", harmonized variable names (e.g., it refers to the variable containing measured tissue concentrations as ‘Conc‘; the variable containing observed time points as ‘Time‘; and the variable containing administered doses as ‘Dose‘). In effect, ‘invivopkfit‘ needs to rename the input data, and produce a new ‘data.frame‘ that uses these internal harmonized variable names.

In order to know which variable names in the input data correspond to each of the internal harmonized variable names, we need to set up a mapping between the internal harmonized variable names and the original variable names.

The simplest, most flexible way to set up this mapping is by (ab)using a call to [ggplot2::aes()]. In the context of [ggplot2::ggplot2-package()], you would use [ggplot2::aes()] to set up mappings to ‘ggplot2‘’s "aesthetics", internal harmonized variable names which it uses for plotting: *e.g.*, ‘x‘, ‘y‘, ‘color‘, ‘size‘, ‘shape‘, and so on. In the context of [invivopkfit-package()], we are setting up mappings to ‘invivopkfit‘’s internal harmonized variable names which it uses in model fitting. These "‘invivopkfit‘ aesthetic" variables are as follows:

- ‘Chemical‘: A ‘character‘ variable containing the chemical identifier. All rows of ‘data‘ should have the same value for ‘Chemical‘. - ‘Species‘: A ‘character‘ variable containing the name of the species for which the data was measured. All rows of ‘data‘ should have the same value for ‘Species‘. - ‘Reference‘: A ‘character‘ variable containing a unique identifier for the data reference (e.g., a single publication). - ‘Subject‘: A ‘character‘ variable containing a unique identifier for the subject associated with each observation (an individual animal or group of animals). - ‘N_Subjects‘: A ‘numeric‘ variable; an integer giving the number of individual animals represented by this observation. (Some data sets report tissue concentrations for individual animals, in which case ‘N_Subjects‘ will be 1; others report average tissue concentrations for groups of multiple animals, in which case ‘N_Subjects‘ will be greater than 1.) - ‘Weight‘: A ‘numeric‘ variable giving the subject’s body weight. - ‘Weight.Units‘: A ‘character‘ variable giving the units of body weight. - ‘Route‘: A ‘character‘ variable denoting the route of administration. Either ‘po‘ (oral) or ‘iv‘ (intravenous). Other routes are not currently supported. - ‘Dose‘: A ‘numeric‘ variable giving the dose administered. - ‘Dose.Units‘: A ‘character‘ variable giving the units of the administered doses. - ‘Time‘: A ‘numeric‘ variable giving the time of tissue collection. - ‘Time.Units‘: A ‘numeric‘ variable giving the units of ‘Time‘. - ‘Media‘: A ‘character‘ variable giving the tissue that was analyzed. Either ‘blood‘ or ‘plasma‘. Other tissues are not currently supported. - ‘Value‘: A ‘numeric‘ variable giving the tissue concentration in units of mg/L. If ‘N_Subjects > 1‘, ‘Value‘ is assumed to represent the mean tissue concentration for this group of subjects. If the tissue concentration was below the limit of quantification (LOQ), this value may be ‘NA_real_‘. - ‘Value_SD‘: A ‘numeric‘ variable giving the standard deviation of the tissue concentration in units of mg/L, if available and relevant. If ‘N_Subjects > 1‘, ‘Value_SD‘ is assumed to represent the standard deviation of tissue concentrations for this group of subjects. If ‘N_Subjects == 1‘, then ‘Value_SD‘ may be ‘NA_real_‘. - ‘LOQ‘: A ‘numeric‘ variable giving the limit of quantification applicable to this tissue concentration in units of mg/L, if available. - ‘Value.Units‘: A ‘character‘ variable giving the units of ‘Value‘, ‘Value_SD‘, and ‘LOQ‘.

You may additionally include mappings to other variable names of your choice, which will appear in the ‘pk‘ object in ‘pk$data‘ after the analysis is done.

As with usual calls to [ggplot2::aes()], you should provide the variable names without quoting them. For example, use ‘ggplot2::aes(Chemical = my_chem)‘. Do * not* use ‘ggplot2::aes("Chemical" = "my_chem")‘.

Also, as with usual calls to [ggplot2::aes()], you may also specify that any of the "‘invivopkfit‘ aesthetic" variables should be mapped to a constant value, rather than to a variable in ‘data‘. For example, imagine that you don’t have a column in ‘data‘ that encodes the units of body weight, but you know that all body weights are provided in units of kilograms. You could specify ‘mapping = ggplot2::aes(Chemical = my_dtxsid, Species = my_species, Weight = my_weight, Weight.Units = "kg")‘ to map ‘Weight.Units‘ to a fixed value of "kg".

Finally, as with usual calls to [ggplot2::aes()], you may specify mappings as expressions that use variable names in ‘data‘. For example, if the species-name variable in ‘data‘ sometimes says "rat", sometimes "Rat", sometimes "RAT", you might want to harmonize the capitalization. You can do that easily by specifying ‘mapping = ggplot2::aes(Chemical = my_dtxsid, Species = tolower(my_species)‘.

The following "aesthetics" variable names are reserved for internal use (i.e., they are automatically assigned by [preprocess_data.pk()], and should *not* be included in ‘mapping‘:

- ‘Conc‘: This is assigned as the greater of ‘Value‘ and ‘LOQ‘, with NAs removed. - ‘Conc_SD‘: This is set equal to ‘Value_SD‘. - ‘Detect‘: This is a logical variable, ‘TRUE‘ if ‘Conc > LOQ‘ and ‘FALSE‘ otherwise. - ‘Conc_trans‘: This is ‘Conc‘ with all scalings and transformations applied as

specified in '+ scale_conc()'. - 'Conc_SD_trans': This is 'Conc_SD' with all scalings and transformations applied as specified in '+ scale_conc()'. - 'Conc_trans.Units': Automatically-derived from 'Conc.Units' with any scalings and transformations applied. If dose normalization is requested, then 'Dose.Units' is also used to automatically derive the resulting 'Conc_trans.Units'. For example, if both dose-normalization and [log10()] transformation are requested, and 'Conc.Units = 'mg/L'' and 'Dose.Units = 'mg/kg', then 'Conc_trans.Units = log10((mg/L)/(mg/kg))'. - 'Time_trans': This is 'Time' with any rescaling specified in '+ scale_time()'. - 'Time_trans.Units': The new units of time after any rescaling (e.g. 'hours', 'days', 'weeks',...)

If you do assign any of these reserved variable names in 'mapping', your mapping will be ignored for those reserved variable names. WARNING: If you have any variables with these reserved names in your original data, those original variables will be dropped by [preprocess_data.pk()].

The default value of 'mapping' is the following (which refers to original variable names in the built-in dataset [cvt]):

"' ggplot2::aes(Chemical = chemicals_analyzed.dsstox_substance_id, DTXSID = chemicals_analyzed.dsstox_substance_id, CASRN = chemicals_analyzed.dsstox_casrn, Species = subjects.species, Reference = as.character(ifelse(is.na(documents_ref documents_extraction.id, documents_reference.id)), Media = series.conc_medium_normalized, Route = studies.administration_route_normalized, Dose = studies.dose_level_normalized, Dose.Units = "mg/kg", Subject = subjects.id, N_Subjects = series.n_subjects_in_series, Weight = subjects.weight_kg, Weight.Units = "kg", Time = conc_time_values.time_hr, Time.Units = "hours", Value = conc_time_values.conc, Value.Units = "mg/L", LOQ = series.loq_normalized, Value_SD = conc_time_values.conc_sd_normalized) "'

Data

'Route' values should be either '"oral"' (oral bolus administration) or '"iv"' (IV bolus administration), and 'Media' values should be either '"blood"' or '"plasma"'.

If 'data' contains data for more than one 'Chemical' and 'Species', then you should use [facet_data()] to run a "faceted" analysis. A faceted analysis will group the data according to unique combinations of the faceting variables, and produce a 'pk' object for each group. The result is a [tibble::tibble()] grouped by the faceting variables, with a list column named 'pk' containing the 'pk' object for each group. This [tibble::tibble()] is an object of class 'pk_faceted'.

All methods for 'pk' objects have a corresponding version for a 'pk_faceted' object, which applies the method to each 'pk' object in turn and either returns the same 'pk_faceted' object with a modified 'pk' column (for methods that operate on a 'pk' object and return a modified version of the same 'pk' object like [preprocess_data()], [data_info()], [prefit()], [fit()]), or produces a [tibble::tibble()] grouped by the faceting variables, with a list column named after the 'pk' method containing the results of that method (for methods that operate on a 'pk' object but return something other than a modified 'pk' object, e.g. [summary.pk()], [coef.pk()], [coef_sd.pk()], [predict.pk()], [residuals.pk()], [nca.pk()]).

Value

An object of class 'pk'. The initial 'pk' object is a list with elements 'data_orig', 'data_settings', 'scales' and 'optimx_settings'. 'data_orig' is the original data set to be fitted, as supplied in the argument 'data'. 'data_settings' is a named list containing all the other input arguments: these provide settings that will be used when the data is pre-processed before fitting.

Author(s)

Caroline Ring

---

pkdataset_nheerlcleaned *Toxicokinetic data from the "Concentration vs. Time Database"*

Description

A dataset containing experimental time-course data of chemical compound concentrations in body fluids and tissues

Usage

```
pkdataset_nheerlcleaned
```

Format

A data table with 2454 rows and 19 variables:

Source

https://github.com/USEPA/CompTox-PK-CvTdb

---

pk_add *Add a ‘pkproto‘ object to a ‘pk‘ object*

Description

This is the S3 generic method.

Usage

```
pk_add(pkproto_obj, pk_obj, objectname)
```

Arguments

pkproto_obj	The ‘pkproto‘ object to be added
pk_obj	The ‘pk‘ object to which the ‘pkproto‘ object is to be added
objectname	The object name

Value

The ‘pk‘ object modified by the addition.

See Also

[pk_add.pk_scales()] for the method for adding ‘pk_scales‘ objects (from [scale_conc()] and [scale_time()]); [pk_add.pk_settings_preprocess()] for the method for adding ‘pk_settings_preprocess‘ objects (from [settings_preprocess()]); [pk_add.pk_settings_data_info()] for the method for adding ‘pk_settings_data_info‘ objects (from [settings_data_info()]); [pk_add.pk_settings_optimx()] for the method for adding ‘pk_settings_optimx‘ objects (from [settings_optimx()]); [pk_add.pk_stat_model()] for the method for adding ‘pk_stat_model‘ objects (from ‘stat_model()‘)

`pk_add.default` *Add pkproto object default method*

Description

Add pkproto object default method

Usage

```
## Default S3 method:
pk_add(pkproto_obj, pk_obj, objectname)
```

Arguments

`pkproto_obj`	The ‘pkproto‘ object to be added
`pk_obj`	The ‘pk‘ object to which the ‘pkproto‘ object is to be added
`objectname`	The object name

Value

The ‘pk‘ object modified by the addition.

`pk_add.pk_facet_data` *Add facet_data()*

Description

Add facet_data()

Usage

```
## S3 method for class 'pk_facet_data'
pk_add(pkproto_obj, pk_obj, objectname)
```

Arguments

pkproto_obj	The ‘pk_facet_data‘ object to be added.
pk_obj	The [pk()] object to which the ‘pk_facet_data‘ object will be added.
objectname	The name of the ‘pk_facet_data‘ object.

Value

The [pk()] object, with the added ‘pk_facet_data‘.

Author(s)

Caroline Ring

---

pk_add.pk_scales *Add a ‘pk_scales‘ object to a ‘pk‘ object.*

Description

Add a ‘pk_scales‘ object to a ‘pk‘ object.

Usage

```
## S3 method for class 'pk_scales'
pk_add(pkproto_obj, pk_obj, objectname)
```

Arguments

pkproto_obj	The ‘pk_scales‘ object to be added.
pk_obj	The ‘pk‘ object to which the ‘pk_scales‘ object will be added.
objectname	The name of the ‘pk_scales‘ object.

Value

The ‘pk‘ object, modified by the ‘pk_scales‘ object.

Author(s)

Caroline Ring

`pk_add.pk_settings_data_info`

Add a 'pk_settings_data_info' object.

Description

Add a 'pk_settings_data_info' object.

Usage

```
## S3 method for class 'pk_settings_data_info'
pk_add(pkproto_obj, pk_obj, objectname)
```

Arguments

`pkproto_obj`	The 'pk_settings_data_info' object to be added.
`pk_obj`	The 'pk' object to which the 'pk_settings_data_info' object will be added.
`objectname`	The name of the 'pk_settings_data_info' object.

Value

The 'pk' object, modified by the 'pk_settings_data_info' object.

Author(s)

Caroline Ring

`pk_add.pk_settings_optimx`

Add a 'pk_settings_optimx' object.

Description

Add a 'pk_settings_optimx' object.

Usage

```
## S3 method for class 'pk_settings_optimx'
pk_add(pkproto_obj, pk_obj, objectname)
```

Arguments

`pkproto_obj`	The 'pk_settings_optimx' object to be added.
`pk_obj`	The 'pk' object to which the 'pk_settings_optimx' object will be added.
`objectname`	The name of the 'pk_settings_optimx' object.

Value

The ‘pk‘ object, modified by adding the settings.

Author(s)

Caroline Ring

`pk_add.pk_settings_preprocess` *Add a ‘pk_settings_preprocess‘ object.*

Description

Add a ‘pk_settings_preprocess‘ object.

Usage

```
## S3 method for class 'pk_settings_preprocess'
pk_add(pkproto_obj, pk_obj, objectname)
```

Arguments

`pkproto_obj`	The ‘pk_settings_preprocess‘ object to be added.
`pk_obj`	The ‘pk‘ object to which the ‘pk_settings_preprocess‘ object will be added.
`objectname`	The name of the ‘pk_settings_preprocess‘ object.

Value

The ‘pk‘ object, modified by the ‘pk_settings_preprocess‘ object.

Author(s)

Caroline Ring

pk_add.pk_stat_error_model *Add a 'pk_stat_error_model' object.*

Description

Add a 'pk_stat_error_model' object.

Usage

```
## S3 method for class 'pk_stat_error_model'
pk_add(pkproto_obj, pk_obj, objectname)
```

Arguments

pkproto_obj	The 'pk_stat_error_model' object to be added.
pk_obj	The 'pk' object to which the 'pk_stat_error_model' object will be added.
objectname	The name of the 'pk_stat_error_model' object.

Value

The 'pk' object, modified by adding the 'stat_error_model'.

Author(s)

Caroline Ring

pk_add.pk_stat_model *Add a 'pk_stat_model' object.*

Description

Add a 'pk_stat_model' object.

Usage

```
## S3 method for class 'pk_stat_model'
pk_add(pkproto_obj, pk_obj, objectname)
```

Arguments

pkproto_obj	The 'pk_stat_model' object to be added.
pk_obj	The 'pk' object to which the 'pk_stat_model' object will be added.
objectname	The name of the 'pk_stat_model' object.

Value

The ‘pk‘ object, modified by adding the ‘stat_model‘.

Author(s)

Caroline Ring

`pk_add.uneval` *Add an ‘uneval‘ object*

Description

Add an object created by [ggplot2::aes()]

Usage

```
## S3 method for class 'uneval'
pk_add(pkproto_obj, pk_obj, objectname)
```

Arguments

`pkproto_obj`	The ‘uneval‘ (mapping) object to be added.
`pk_obj`	The [pk()] object to which the ‘uneval‘ object will be added.
`objectname`	The name of the ‘uneval‘ object.

Details

This function adds a new variable mapping (created by [ggplot2::aes()]), which has class ‘uneval‘, to an existing [pk()] object.

The new mapping will completely replace any existing mapping.

Value

The [pk()] object, modified by adding the new mapping.

Author(s)

Caroline Ring

pk_model *Create a new 'pk_model' object*

Description

'conc_fun' requirements

Usage

```
pk_model(
  name,
  params,
  conc_fun,
  auc_fun,
  params_fun,
  tkstats_fun,
  conc_fun_args = NULL,
  auc_fun_args = NULL,
  params_fun_args = NULL,
  tkstats_fun_args = NULL,
  ...
)
```

Arguments

name	Character: The name of the model.
params	Character vector: Parameter names of the model.
conc_fun	Character: Name of the function to predict tissue concentrations using this model. See Details for requirements.
auc_fun	Character: Name of the function to predict AUC (area under the concentration-time curve) using this model. See Details for requirements.
params_fun	Character: Name of the function that produces the 'data.frame' of parameter info for this model (see Details)
tkstats_fun	Character: Name of the function that produces a'data.frame' of derived TK statistics for this model (see Details)
conc_fun_args	A named list: any additional arguments to 'conc_fun' other than those listed in Details. Default 'NULL'.
auc_fun_args	A named list: any additional arguments to 'auc_fun' other those those listed in Details. Default 'NULL'.
params_fun_args	A named list: any additional arguments to 'params_fun' other than 'data' (see Details). Default 'NULL'.
tkstats_fun_args	A named list: any additional arguments to 'tkstats_fun' other than 'data', 'medium', 'route' (see Details). Default NULL.
...	Additional arguments (not currently implemented).

Details

‘conc_fun‘ should be a function that takes the following arguments, and returns a numeric vector of predicted tissue concentrations:

- ‘params‘: A named list of parameter values - ‘time‘: A numeric vector of time values - ‘dose‘: A numeric vector of dose values. Currently, only a single bolus dose at time 0 is supported. - ‘route‘: A character vector of the route of administration. Currently, only ‘’oral’‘ and ‘’iv’‘ are supported. - ‘medium‘: The tissue in which concentration is to be predicted. Currently, only ‘’blood’‘ and ‘’plasma’‘ are supported.

See [cp_1comp()], [cp_2comp()], [cp_flat()] for examples.

#‘auc_fun‘ requirements

‘auc_fun‘ should be a function that takes the same arguments as ‘conc_fun‘, and returns a numeric vector of predicted tissue AUCs (area under the concentration-time curve).

See [auc_1comp()], [auc_2comp()], [auc_flat()] for examples.

‘params_fun‘ requirements

‘params_fun‘ should be a function whose first argument is a ‘data.frame‘, which will be the pre-processed data using ‘invivopkfit‘ harmonized variable names. It may take additional arguments, which can be provided in ‘params_fun_args‘. The function should return a ‘data.frame‘ with the following variables:

- ‘param_name‘: Character vector, listing parameter names for the model - ‘param_units‘: Character vector, listing units of each model parameter - ‘optimize_param‘: Logical (TRUE/FALSE), whether each parameter is to be estimated given the available data - ‘use_param‘: Logical (TRUE/FALSE), whether each parameter is to be used in the model even if it is not estimated (i.e., if a parameter value is to be held constant while the others are estimated, then ‘optimize_param‘ should be FALSE but ‘use_param‘ should be TRUE) -‘lower_bound‘: Numerical. Lower bounds for each parameter. May be ‘-Inf‘ if no lower bound. If ‘optimize_param‘ or ‘use_param‘ is FALSE, thenthe corresponding ‘lower_bound‘ will be ignored (because the parameter is not being estimated from the data). - ‘upper_bound‘: Numerical. Upper bounds for each parameter. May be ‘Inf‘ if no upper bound. If ‘optimize_param‘ or ‘use_param‘ is FALSE, then the corresponding ‘upper_bound‘ will be ignored (because the parameter is not being estimated from the data). - ‘start‘: Numerical. Starting values for estimating each parameter. If ‘optimize_param‘ is FALSE and ‘use_param‘ is TRUE, then the parameter will be held constant at the corresponding value in ‘start‘. If ‘use_param‘ is FALSE, then the corresponding ‘start‘ will be ignored.

See [get_params_flat()], [get_params_1comp()], [get_params_2comp()] for examples.

‘tkstats_fun‘ requirements

‘tkstats_fun‘ should be a function which accepts a vector of model parameter values and calculates derived summary toxicokinetic statistics (e.g. total clearance, halflife, AUC, volume of distribution at steadystate).

The function must take the following named arguments:

- ‘pars‘: A named numeric vector of model parameter values. - ‘route‘: A character scalar naming a route (e.g. "oral" or "iv") - ‘medium‘: A character scalar naming a tissue medium of analysis (e.g. "blood" or "plasma") - ‘dose‘: A numeric scalar giving a dose level for which to calculate TK statistics - ‘time_unit‘: A character scalar giving the units of time - ‘conc_unit‘: A character scalar giving the units of concentration - ‘vol_unit‘: A character scalar giving the units of volume

and return a ‘data.frame‘ of derived toxicokinetic statistics, which should have the following variables:

- ‘param_name‘: A character vector giving the names of each derived TK statistic - ‘param_value‘: A character vector giving the values of each derived TK statistic - ‘param_units‘: A character vector giving the units of each derived TK statistic

It is recommended (although not required) that the function return the following statistics, using these names in the ‘param_name‘ variable:

- ‘CLtot‘: Total clearance rate (units of volume/time) - ‘CLtot/Fgutabs‘: Total clearance rate scaled by bioavailability (if oral bioavailability is available) (units of volume/time) - ‘Css‘: The steady-state concentration after a long-term daily dose of ‘dose‘ (units of concentration) - ‘halflife‘: The terminal half-life (units of time) - ‘tmax‘: The time of peak concentration (units of time) - ‘Cmax‘: The peak concentration (units of time) - ‘AUC_infinity‘: The area under the concentration-time curve, calculated at infinite time (units of concentration * time) - ‘Vdist_ss‘: The volume of distribution at steady-state (units of volume) - ‘Vdist_ss/Fgutabs‘: The volume of distribution at steady-state scaled by bioavailability (if oral bioavailability is available) (units of volume)

The recommendation to return these statistics, using these names, is intended to make it easier to compare TK statistics across models, and to compare TK statistics to the results of non-compartmental analysis. If these names are not used, then some outputs of [summary.pk()] will not be very useful. The automated comparison of TK stats from the winning model to the results of non-compartmental analysis relies on these names being present in the output of ‘tkstats_fun‘ to match the names of the statistics output from NCA; it shouldn’t crash without them, but the results won’t be very useful. And TK stats compiled across models will not be easy to compare if the models use different names for the statistics.

Value

An object of class ‘pk_model‘. Effectively, a named list containing all of the arguments provided to this function.

Author(s)

Caroline Ring

`pk_subtract` *Subtract a ‘pkproto‘ object from a ‘pk‘ object*

Description

This is the S3 generic method.

Usage

```
pk_subtract(pkproto_obj, pk_obj, objectname)
```

Arguments

pkproto_obj	The ‘pkproto‘ object to be subtracted
pk_obj	The ‘pk‘ object to which the ‘pkproto‘ object is to be subtracted
objectname	The object name

Value

The ‘pk‘ object modified by the subtraction.

See Also

[pk_subtract.pk_stat_model()] for the method for subtracting ‘pk_stat_model‘ objects (from ‘stat_model()‘)

---

pk_subtract.default *Subtract pkproto object default method*

Description

Subtract pkproto object default method

Usage

```
## Default S3 method:
pk_subtract(pkproto_obj, pk_obj, objectname)
```

Arguments

pkproto_obj	The ‘pkproto‘ object to be subtracted
pk_obj	The ‘pk‘ object to which the ‘pkproto‘ object is to be subtracted
objectname	The object name

Value

The ‘pk‘ object modified by the addition.

pk_subtract.pk_stat_model *Subtract a 'pk_stat_model' object.*

Description

Subtract a 'pk_stat_model' object.

Usage

```
## S3 method for class 'pk_stat_model'
pk_subtract(pkproto_obj, pk_obj, objectname)
```

Arguments

pkproto_obj	The 'pk_stat_model' object to be subtracted.
pk_obj	The 'pk' object from which the 'pk_stat_model' object will be subtracted.
objectname	The name of the 'pk_stat_model' object.

Value

The 'pk' object, modified by subtracting the 'stat_model'.

Author(s)

Caroline Ring

plot.pk *Plot a [pk()] object.*

Description

Plot data and model fits from a [pk()] object.

Usage

```
## S3 method for class 'pk'
plot(
  x,
  newdata = NULL,
  model = NULL,
  method = NULL,
  use_scale_conc = FALSE,
  time_trans = FALSE,
  log10_C = NULL,
```

```
  plot_data_aes = NULL,
  plot_point_aes = NULL,
  facet_fun = NULL,
  facet_fun_args = NULL,
  drop_nonDetect = FALSE,
  plot_fit_aes = NULL,
  n_interp = 10,
  fit_limits = NULL,
  print_out = FALSE,
  best_fit = FALSE,
  ...
)
```

Arguments

x	A [pk()] object. In this case ‘x‘ is used to align with generic method.
newdata	Optional: A ‘data.frame‘ containing new data to plot. Must contain at least variables ‘Chemical‘, ‘Species‘, ‘Route‘, ‘Media‘, ‘Dose‘, ‘Time‘, ‘Time.Units‘, ‘Conc‘, ‘Detect‘, ‘Conc_SD‘. Default ‘NULL‘, to use the data in ‘obj$data‘.
model	Character: One or more of the models fitted. Curve fits will be plotted for these models. Default ‘NULL‘ to plot fits for all models in ‘x$stat_model‘.
method	Character: One or more of the [optimx::optimx()] methods used. Default ‘NULL‘ to plot fits for all methods in ‘x$settings_optimx$method‘.
use_scale_conc	Possible values: ‘TRUE‘, ‘FALSE‘, or a named list with elements ‘dose_norm‘ and ‘log10_trans‘ which themselves should be either ‘TRUE‘ or ‘FALSE‘. If ‘use_scale_conc = FALSE‘ (the default for this function), then the data and fits will be plotted without any dose-normalization or log-transformation. If ‘use_scale_conc = TRUE‘ , then the concentration scaling/transformations in ‘x‘ will be applied to the y-axis (concentration axis). If ‘use_scale_conc = list(dose_norm = ..., log10_trans = ...)‘, then the specified dose normalization and/or log10-transformation will be applied to the y-axis (concentration axis) of the plots.
time_trans	Default ‘FALSE‘. Determines whether time values will be transformed.
log10_C	Default ‘NULL‘. Determines whether y-axis (concentration) should be log10 transformed. Takes ‘TRUE‘ or ‘FALSE‘ values. Otherwise it defaults to the value determined from ‘use_scale_conc‘.
plot_data_aes	Optional: Aesthetic mapping for the plot layer that visualizes the data. Default ‘NULL‘, in which case a default mapping will be used based on the value of ‘use_scale_conc‘.
plot_point_aes	Optional: Aesthetic mappings for geom_point layer that determines the fill of the points. Defaults to ‘NULL‘.
facet_fun	Default ‘"facet_grid"‘. Optional: The name of the ‘ggplot2‘ faceting function to use: [ggplot2::facet_grid()], [ggplot2::facet_wrap()], or ‘’none’‘ to do no faceting. Default ‘NULL‘, in which case a default faceting will be applied based on the value of ‘use_scale_conc‘.

facet_fun_args	A named list of arguments to the faceting function in ‘facet_fun‘ (if any). Default: “‘ list(rows = ggplot2::vars(Route), cols= ggplot2::vars(Media), scales = "free_y", labeller = "label_both") “‘
drop_nonDetect	Default ‘FALSE‘. Whether to eliminate observations below the level of quantification (LOQ).
plot_fit_aes	Optional: Aesthetic mapping for the plot layer that visualizes the fitted curves. Default ‘NULL‘, in which case a default mapping will be used based on the value of ‘use_scale_conc‘.
n_interp	For plotting: the number of time points to interpolate between each observed time point. Default 10.
fit_limits	Default ‘NULL‘. c(Upper Bound, Lower Bound). Supply a numeric vector. These values filter the predicted values for fits to not exceed 2.25x of the maximum observed concentration values for each ‘data_group‘ in the ‘pk‘ object. When there is a log10 transformation of concentration values, it limits predicted values to 1/20th of the minimum observed concentration values and 5 times the maximum value.
print_out	For plotting: whether the output of the function should be the list of plots. Default ‘FALSE‘.
best_fit	Default FALSE. Determines whether fit plot outputs only the best fit from ‘get_winning_model()‘
...	Additional arguments not in use.

Details

If the [pk()] object has not been fitted, then only the data will be plotted (because no curve fits exist).

Value

A [ggplot2::ggplot()]-class plot object.

Author(s)

Caroline Ring, Gilberto Padilla Mercado

---

predict.pk *Get predictions*

Description

Extract predictions from a fitted ‘pk‘ object.

Usage

```
## S3 method for class 'pk'
predict(
  object,
  newdata = NULL,
  model = NULL,
  method = NULL,
  type = "conc",
  exclude = TRUE,
  use_scale_conc = FALSE,
  suppress.messages = NULL,
  include_NAs = FALSE,
  ...
)
```

Arguments

object	A [pk] object.
newdata	Optional: A 'data.frame' with new data for which to make predictions. If NULL (the default), then predictions will be made for the data in 'object$data'. 'newdata' is required to contain at least the following variables: 'Time', 'Time.Units', 'Dose', 'Route', and 'Media'.
model	Optional: Specify one or more of the fitted models for which to make predictions. If NULL (the default), predictions will be returned for all of the models in 'object$stat_model'.
method	Optional: Specify one or more of the [optimx::optimx()] methods for which to make predictions. If NULL (the default), predictions will be returned for all of the models in 'object$settings_optimx$method'.
type	Either '"conc"' (the default) or '"auc"'. 'type = "conc"' predicts concentrations; 'type = "auc"' predicts area under the concentration-time curve (AUC).
exclude	Logical: 'TRUE' to return 'NA_real_' for any observations in the data marked for exclusion (if there is a variable 'exclude' in the data, an observation is marked for exclusion when 'exclude 'FALSE' to return the prediction for each observation, regardless of exclusion. Default 'TRUE'.
use_scale_conc	Possible values: 'TRUE', 'FALSE', or a named list with elements 'dose_norm' and 'log10_trans' which themselves should be either 'TRUE' or 'FALSE'. If 'use_scale_conc = TRUE', then the concentration scaling/transformations in 'object' will be applied to both predicted and observed concentrations before the log-likelihood is computed. If 'use_scale_conc = FALSE' (the default for this function), then no concentration scaling or transformation will be applied before the log-likelihood is computed. If 'use_scale_conc = list(dose_norm = ..., log10_trans = ...)', then the specified dose normalization and/or log10-transformation will be applied.
suppress.messages	Logical: whether to suppress message printing. If NULL (default), uses the setting in 'object$settings_preprocess$suppress.messages'

include_NAs Logical: 'FALSE' by default. Determines whether to include aborted fits which have NAs as coefficients.

... Additional arguments.

Value

A data.frame with one row for each 'data_group', 'model' and 'method'. Includes variable 'Conc_est' that contains the predicted concentration or AUC at that timepoint given the TK parameters for that 'model' and 'method' specified in [coefs()]. If 'use_scale_conc un-transformed concentrations in the same units as 'object$data$Conc.Units'. If 'use_scale_conc concentrations in the same units as 'object$data$Conc_trans.Units'.

Author(s)

Caroline Ring, Gilberto Padilla Mercado

See Also

Other methods for fitted pk objects: AAFE.pk(), AFE.pk(), AIC.pk(), BIC.pk(), coef.pk(), coef_sd.pk(), eval_tkstats.pk(), get_fit.pk(), get_hessian.pk(), get_tkstats.pk(), logLik.pk(), residuals.pk(), rmse.pk(), rsq.pk()

---

print.pk *Print a user-friendly version of a 'pk' object*

Description

Prints a clear summary of 'pk' object and returns it invisibly.

Usage

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

Arguments

x A pk object.

... Additional arguments. Currently not in use.

Value

Summary output

Author(s)

Gilberto Padilla Mercado

rename2_cvt *Convert invivoPKfit output table names to the CvTdb names*

Description

Convert invivoPKfit output table names to the CvTdb names

Usage

```
rename2_cvt(
  data,
  cvt_LUT = c(analyte_dtxsid = "Chemical", analyte_dtxsid = "DTXSID",
    analyte_name_original = "Chemica_Name", species = "Species",
    fk_extraction_document_id = "Reference", conc_medium_normalized = "Media",
   administration_route_normalized = "Route", invivPK_dose_level = "Dose", fk_subject_id
   = "Subject_ID", fk_series_id = "Series_ID", fk_study_id = "Study_ID", conc_time_id =
   "ConcTime_ID", invivPK_subjects_corrected = "N_Subjects", weight_kg = "Weight",
    time_hr = "Time", invivPK_conc = "Value", invivPK_conc_sd = "Value_SD",

    invivPK_loq = "LOQ")
)
```

Arguments

data	A data frame from on the invivoPKfit outputs.
cvt_LUT	A look-up table for the name conversions, can be constomized. must be a vector.

residuals.pk *Get residuals*

Description

Extract residuals from a fitted ‘pk‘ object.

Usage

```
## S3 method for class 'pk'
residuals(
  object,
  newdata = NULL,
  model = NULL,
  method = NULL,
  exclude = TRUE,
  use_scale_conc = FALSE,
  suppress.messages = NULL,
  ...
)
```

Arguments

object	A [pk] object
newdata	Optional: A 'data.frame' with new data for which to make predictions and compute residuals. If NULL (the default), then residuals will be computed for the data in 'object$data'. 'newdata' is required to contain at least the following variables: 'Time', 'Time.Units', 'Dose', 'Route', 'Media', 'Conc', 'Detect'.
model	Optional: Specify one or more of the fitted models for which to make predictions and calculate residuals. If NULL (the default), residuals will be returned for all of the models in 'object$stat_model'.
method	Optional: Specify one or more of the [optimx::optimx()] methods for which to make predictions and calculate residuals. If NULL (the default), residuals will be returned for all of the models in 'object$optimx_settings$method'.
exclude	Logical: 'TRUE' to return 'NA_real_' for any observations in the data marked for exclusion (if there is a variable 'exclude' in the data, an observation is marked for exclusion when 'exclude 'FALSE' to return the residual for each observation, regardless of exclusion. Default 'TRUE'.
use_scale_conc	Possible values: 'TRUE', 'FALSE', or a named list with elements 'dose_norm' and 'log10_trans' which themselves should be either 'TRUE' or 'FALSE'. If 'use_scale_conc = TRUE', then the concentration scaling/transformations in 'object' will be applied to both predicted and observed concentrations before the residuals are computed (i.e., the residuals will be computed on the same scale as the model was originally fitted). If 'use_scale_conc = FALSE' (the default for this function), then no concentration scaling or transformation will be applied before the residuals are computed (i.e., the residuals will be computed on natural scale concentration data). If 'use_scale_conc = list(dose_norm = ..., log10_trans = ...)', then the specified dose normalization and/or log10-transformation will be applied.
suppress.messages	Logical: whether to suppress message printing. If NULL (default), uses the setting in 'object$settings_preprocess$suppress.messages'
...	Additional arguments not currently used.

Details

Residuals are 'obs - pred' in general, where 'obs' is the observed concentration value and 'pred' is the predicted concentration value.

For non-detect observations, residual is zero if 'pred' is also below the LOQ. Otherwise, the residual is the difference between the LOQ and 'pred'.

Value

A data.frame with the final column being calculated residuals. There is one row per each [optimx::optimx()] methods (specified in [settings_optimx()]), and 'data_group'. The final column contains the residuals (observed - predicted) of the model fitted by the corresponding method. If 'use_scale_conc in the same units as 'object$data$Conc.Units'. If 'use_scale_conc the residuals are in the same units as 'object$data$Conc_trans.Units'. If 'use_scale_conc' was a named list, then the residuals are in units of 'object$data$Conc.Units' transformed as specified in 'use_scale_conc'.

Author(s)

Caroline Ring, Gilberto Padilla Mercado

See Also

Other methods for fitted pk objects: AAFE.pk(), AFE.pk(), AIC.pk(), BIC.pk(), coef.pk(), coef_sd.pk(), eval_tkstats.pk(), get_fit.pk(), get_hessian.pk(), get_tkstats.pk(), logLik.pk(), predict.pk(), rmse.pk(), rsq.pk()

---

rmse *Root mean squared error (RMSE)*

Description

This is the S3 method generic for ‘rmse‘.

Usage

```
rmse(obj, ...)
```

Arguments

obj	the pk object
...	Additional arguments currently not in use.

Value

A ‘data.frame‘ with calculated RMSE as the final column. There is one row per each model in ‘obj‘’s [stat_model()] element, i.e. each PK model that was fitted to the data, each [optimx::optimx()] methods (specified in [settings_optimx()]), ‘rmse_group‘ specified.

See Also

[rmse.pk()] for the ‘rmse‘ method for class [pk()]

rmse.default *Root mean squared error (RMSE) default method*

Description

Root mean squared error (RMSE) default method

Usage

```
## Default S3 method:
rmse(obj, ...)
```

Arguments

obj	an object
...	Additional arguments.

Value

An error, when a non-pk object is used for the first argument.

rmse.pk *Root mean squared error*

Description

Extract root mean squared error of a fitted ‘pk‘ object

Usage

```
## S3 method for class 'pk'
rmse(
  obj,
  newdata = NULL,
  model = NULL,
  method = NULL,
  exclude = TRUE,
  use_scale_conc = FALSE,
  rmse_group = NULL,
  sub_pLOQ = TRUE,
  suppress.messages = NULL,
  ...
)
```

Arguments

obj	A ‘pk‘ object
newdata	Optional: A ‘data.frame‘ with new data for which to make predictions and compute RMSEs. If NULL (the default), then RMSEs will be computed for the data in ‘obj$data‘. ‘newdata‘ is required to contain at least the following variables: ‘Time‘, ‘Time.Units‘, ‘Dose‘, ‘Route‘, ‘Media‘, ‘Conc‘, ‘Conc_SD‘, ‘N_Subjects‘, ‘Detect‘.
model	Optional: Specify one or more of the fitted models for which to make predictions and calculate RMSEs. If NULL (the default), RMSEs will be returned for all of the models in ‘obj$stat_model‘.
method	Optional: Specify one or more of the [optimx::optimx()] methods for which to make predictions and calculate RMSEs. If NULL (the default), RMSEs will be returned for all of the models in ‘obj$optimx_settings$method‘.
exclude	Logical: ‘TRUE‘ to compute the RMSE excluding any observations in the data marked for exclusion (if there is a variable ‘exclude‘ in the data, an observation is marked for exclusion when ‘exclude ‘FALSE‘ to include all observations, regardless of exclusion status. Default ‘TRUE‘.
use_scale_conc	Possible values: ‘FALSE‘ (default, ‘TRUE‘, or a named list with elements ‘dose_norm‘ and ‘log10_trans‘ which themselves should be either ‘TRUE‘ or ‘FALSE‘. If ‘use_scale_conc = FALSE‘ (the default for this function), then no concentration scaling or transformation will be applied when the RMSE is computed. If ‘use_scale_conc = TRUE, then the concentration scaling/transformations in ‘obj‘ will be applied to both predicted and observed concentrations when the RMSE is computed (see [calc_rmse()] for details).If ‘use_scale_conc = list(dose_norm = ..., log10_trans = ...)‘, then the specified dose normalization and/or log10-transformation will be applied when the RMSE is computed.
rmse_group	A list of quosures provided in the format ‘vars(...)‘ that determines the data groupings for which RMSE is calculated. Default NULL, in which case RMSE is calculated for each data group defined in the object’s ‘data_group‘ element (use [get_data_group.pk()] to access the object’s ‘data_group‘).
sub_pLOQ	TRUE (default): Substitute all predictions below the LOQ with the LOQ before computing R-squared. FALSE: do not.
suppress.messages	Logical: whether to suppress message printing. If NULL (default), uses the setting in ‘obj$settings_preprocess$suppress.messages‘
...	Additional arguments. Not currently used.

Details

Formula for RMSE

RMSE is calculated using the following formula, to properly handle summary data:

$$\sqrt{\frac{1}{N}\sum_{i=1}^{G}((n_i-1)s_i^2+n_i\bar{y}_i^2-2n_i\bar{y}_i\mu_i+\mu_i^2)}$$

In this formula, there are G observations, each of which may be for one subject or for multiple subjects.

- n_i is the number of subjects for observation i. - $\bar{y}_i$ is the sample mean concentration for observation i, with no transformations applied. - s_i is the sample standard deviation of concentrations for observation i, with no transformations applied. - μ_i is the model-predicted concentration for observation i, with no transformations applied.

N is the grand total of subjects across observations:

$$N = \sum_{i=1}^{G} n_i$$

For the non-summary case (N single-subject observations, with all $n_i = 1$, $s_i = 0$, and $\bar{y}_i = y_i$), this formula reduces to the familiar RMSE formula

$$\sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \mu_i)^2}$$

Left-censored data

If the observed value is censored, and the predicted value is less than the reported LOQ, then the predicted value is (temporarily) set equal to the LOQ, for an effective error of zero.

If the observed value is censored, and the predicted value is greater than the reported LOQ, the the observed value is treated as the reported LOQ (so that the effective error is the difference between the LOQ and the predicted value).

Log10 transformation

If 'log10_trans log10-transformed before calculating the RMSE. In the case where observed values are reported in summary format, each sample mean and sample SD (reported on the natural scale, i.e. the mean and SD of natural-scale individual observations) are used to produce an estimate of the log10-scale sample mean and sample SD (i.e., the mean and SD of log10-transformed individual observations), using [convert_summary_to_log10()].

The formulas are as follows. Again, $\bar{y}_i$ is the sample mean for group i. s_i is the sample standard deviation for group i.

$$\text{log10-scale sample mean}_i = \log_{10}\left(\frac{\bar{y}_i^2}{\sqrt{\bar{y}_i^2 + s_i^2}}\right)$$

$$\text{log10-scale sample SD}_i = \sqrt{\log_{10}\left(1 + \frac{s_i^2}{\bar{y}_i^2}\right)}$$

Value

A 'data.frame' with calculated RMSE as the final column. There is one row per each model in 'obj''s [stat_model()] element, i.e. each PK model that was fitted to the data, each [optimx::optimx()] methods (specified in [settings_optimx()]), 'rmse_group' specified.

Author(s)

Caroline Ring, Gilberto Padilla Mercado

See Also

[calc_rmse()]

Other fit evaluation metrics: `AAFE.pk()`, `AFE.pk()`, `AIC.pk()`, `BIC.pk()`, `logLik.pk()`, `rsq.pk()`

Other methods for fitted pk objects: `AAFE.pk()`, `AFE.pk()`, `AIC.pk()`, `BIC.pk()`, `coef.pk()`, `coef_sd.pk()`, `eval_tkstats.pk()`, `get_fit.pk()`, `get_hessian.pk()`, `get_tkstats.pk()`, `logLik.pk()`, `predict.pk()`, `residuals.pk()`, `rsq.pk()`

`rowwise_calc_percentages` *Helper function for calculating percentages of count data, by row*

Description

This function takes totals and calculates rowise percentages across columns Expects columns for each percentage, can specify a vector of "grouping" column names

Usage

```
rowwise_calc_percentages(data, group_cols = NULL)
```

Arguments

`data`	A data.frame that contains columns of count data and possibly columns of group names.
`group_cols`	String or numeric indices for the columns which contain grouping variables.

Value

data.frame with rowwise totals and percentages

rsq *rsq()*

Description

This is the S3 method generic for rsq()

Usage

```
rsq(obj, ...)
```

Arguments

obj	An object.
...	Additional arguments currently not in use.

Value

A dataframe with one row for each ‘data_group‘, ‘model‘ and ‘method‘. The final column contains the R-squared of the model fitted by the corresponding method, using the data in ‘newdata‘.

See Also

[rsq.pk()] for the method for class [pk()]

rsq.default *Default method for rsq()*

Description

Default method for rsq()

Usage

```
## Default S3 method:
rsq(obj, ...)
```

Arguments

obj	An object
...	Additional arguments currently not in use.

Value

An error, when a non-pk object is used for the first argument.

rsq.pk *Calculate R-squared for observed vs. predicted values*

Description

Calculate the square of the Pearson correlation coefficient (r) between observed and model-predicted values

Usage

```
## S3 method for class 'pk'
rsq(
  obj,
  newdata = NULL,
  model = NULL,
  method = NULL,
  exclude = TRUE,
  use_scale_conc = FALSE,
  rsq_group = NULL,
  sub_pLOQ = TRUE,
  suppress.messages = NULL,
  ...
)
```

Arguments

obj	A ‘pk‘ object
newdata	Optional: A ‘data.frame‘ with new data for which to make predictions and compute R-squared. If NULL (the default), then R-squared will be computed for the data in ‘obj$data‘. ‘newdata‘ is required to contain at least the following variables: ‘Time‘, ‘Time.Units‘, ‘Dose‘, ‘Route‘, ‘Media‘, ‘Conc‘, ‘Conc_SD‘, ‘N_Subjects‘, ‘Detect‘.
model	Optional: Specify one or more of the fitted models for which to make predictions and calculate R-squared. If NULL (the default), R-squared will be returned for all of the models in ‘obj$stat_model‘.
method	Optional: Specify one or more of the [optimx::optimx()] methods for which to make predictions and calculate R-squared. If NULL (the default), RMSEs will be returned for all of the models in ‘obj$optimx_settings$method‘.
exclude	Logical: ‘TRUE‘ to compute the R-squared excluding any observations in the data marked for exclusion (if there is a variable ‘exclude‘ in the data, an observation is marked for exclusion when ‘exclude ‘FALSE‘ to include all observations, regardless of exclusion status. Default ‘TRUE‘.
use_scale_conc	Possible values: ‘TRUE‘ (default), ‘FALSE‘, or a named list with elements ‘dose_norm‘ and ‘log10_trans‘ which themselves should be either ‘TRUE‘ or ‘FALSE‘. If ‘use_scale_conc = TRUE‘ (the default for this function), then the

concentration scaling/transformations in ‘obj‘ will be applied to both predicted and observed concentrations when the R-squared is computed (see [calc_rsq()] for details). If ‘use_scale_conc = FALSE‘, then no concentration scaling or transformation will be applied when the R-squared is computed. If ‘use_scale_conc = list(dose_norm = ..., log10_trans = ...)‘, then the specified dose normalization and/or log10-transformation will be applied.

`rsq_group` Default: Chemical, Species. Determines what the data grouping that is used to calculate R-squared value. Should be set to lowest number of variables that still would return unique experimental conditions. Input in the form of ‘ggplot2::vars(Chemical, Species, Route, Media, Dose)‘.

`sub_pLOQ` TRUE (default): Substitute all predictions below the LOQ with the LOQ before computing R-squared. FALSE: do not.

`suppress.messages`
Logical: whether to suppress message printing. If NULL (default), uses the setting in ‘obj$settings_preprocess$suppress.messages‘

... Additional arguments. Not currently in use.

Details

Calculate the square of the Pearson correlation coefficient (r) between observed and model-predicted values, when observed data may be left-censored (non-detect) or may be reported in summary form (as sample mean, sample standard deviation, and sample number of subjects). Additionally, handle the situation when observed data and predictions need to be log-transformed before RMSE is calculated.

r^2 is calculated according to the following formula, to properly handle multi-subject observations reported in summary format:

$$r^2 = \left(\frac{\sum_{i=1}^{G} \mu_i n_i \bar{y}_i - (\bar{\mu} + \bar{y}) \sum_{i=1}^{G} n_i \mu_i + (\bar{mu}\bar{y}) \sum_{i=1}^{G} n_i}{\sqrt{\sum_{i=1}^{G} (n_i - 1) s_i^2 + \sum_{i=1}^{G} n_i \bar{y}_i^2 - 2\bar{y} \sum_{i=1}^{G} n_i \bar{y}_i + N + \bar{y}^2} \sqrt{\sum_{i=1}^{G} n_i \mu_i^2 - 2\bar{y} \sum_{i=1}^{G} n_i \mu_i + N + \bar{y}^2}} \right)^2$$

In this formula, there are G groups (reported observations). (For CvTdb data, a "group" is a specific combination of chemical, species, route, medium, dose, and timepoint.) n_i is the number of subjects for group i. $\bar{y}_i$ is the sample mean for group i. s_i is the sample standard deviation for group i.μ_i is the model-predicted value for group i. $\bar{y}$ is the grand mean of observations:

$$\bar{y} = \frac{\sum_{i=1}^{G} n_i \bar{y}_i}{\sum_{i=1}^{G} n_i}$$

$\bar{\mu}$ is the grand mean of predictions:

$$\bar{\mu} = \frac{\sum_{i=1}^{G} n_i \mu_i}{\sum_{i=1}^{G} n_i}$$

N is the grand total of subjects:

$$N = \sum_{i=1}^{G} n_i$$

For the non-summary case (N single-subject observations, with all $n_i = 1$, $s_i = 0$, and $\bar{y}_i = y_i$), this formula reduces to the familiar formula

$$r^2 = \left(\frac{\sum_{i=1}^{N}(y_i - \bar{y})(\mu_i - \bar{\mu})}{\sqrt{\sum_{i=1}^{N}(y_i - \bar{y})^2}\sqrt{\sum_{i=1}^{N}(\mu_i - \bar{\mu})^2}} \right)^2$$

Left-censored data

If the observed value is censored, and the predicted value is less than the reported LOQ, then the observed value is (temporarily) set equal to the predicted value, for an effective error of zero.

If the observed value is censored, and the predicted value is greater than the reported LOQ, the the observed value is (temporarily) set equal to the reported LOQ, for an effective error of (LOQ - predicted).

Log10 transformation

If 'log10_trans log10(observations) vs. log10(predictions).

In the case where observed values are reported in summary format, each sample mean and sample SD (reported on the natural scale, i.e. the mean and SD of natural-scale individual observations) are used to produce an estimate of the log10-scale sample mean and sample SD (i.e., the mean and SD of log10-transformed individual observations), using [convert_summary_to_log10()].

The formulas are as follows. Again, $\bar{y}_i$ is the sample mean for group i. s_i is the sample standard deviation for group i.

$$\text{log10-scale sample mean}_i = \log_{10}\left(\frac{\bar{y}_i^2}{\sqrt{\bar{y}_i^2 + s_i^2}} \right)$$

$$\text{log10-scale sample SD}_i = \sqrt{\log_{10}\left(1 + \frac{s_i^2}{\bar{y}_i^2}\right)}$$

Value

A dataframe with one row for each 'data_group', 'model' and 'method'. The final column contains the R-squared of the model fitted by the corresponding method, using the data in 'newdata'.

Author(s)

Caroline Ring

See Also

[calc_rsq()]

Other fit evaluation metrics: AAFE.pk(), AFE.pk(), AIC.pk(), BIC.pk(), logLik.pk(), rmse.pk()

Other methods for fitted pk objects: AAFE.pk(), AFE.pk(), AIC.pk(), BIC.pk(), coef.pk(), coef_sd.pk(), eval_tkstats.pk(), get_fit.pk(), get_hessian.pk(), get_tkstats.pk(), logLik.pk(), predict.pk(), residuals.pk(), rmse.pk()

---

scale_conc *Scale concentrations*

Description

Scale concentrations

Usage

```
scale_conc(dose_norm = FALSE, log10_trans = FALSE, ...)
```

Arguments

dose_norm	Logical: Whether to normalize observed concentrations (and observed concentration standard deviations and limits of quantification) by dividing them by the corresponding dose. Default 'FALSE'.
log10_trans	Logical: Whether to apply a 'log10()' transformation to observed concentrations (and limits of quantification), after any dose normalization is applied. Default 'FALSE'.
...	Other arguments (not currently used)

Value

An object of class 'pk_scales': A named list with two elements. The first element 'name = "conc"' denotes the variable to be scaled. The second element 'value' is itself a named list of the arguments supplied to [scale_conc()]). This object is usually added to an existing [pk()] object using '+'. See [pk_add.pk_scales()].

Author(s)

Caroline Ring

scale_time *Scale times*

Description

Transform time data

Usage

```
scale_time(new_units = "identity", ...)
```

Arguments

new_units	New units to use for time. Default is '"identity"' (leave time in the original units). Another useful option is '"auto"', to automatically select new time units based on the time of the last detected observation. You may also specify any time units understood by 'lubridate::duration()', i.e., '"seconds"', '"hours"', '"days"', '"weeks"', '"months"', '"years"', '"milliseconds"', '"microseconds"', '"nanoseconds"', and/or '"picoseconds"'. You may only specify one new unit (e.g., 'new_units = c("days", "weeks")' is not valid).
...	Other arguments (not currently used)

Value

An object of class 'pk_scales': A named list with two elements 'name = "time"' (denoting the variable to be scaled) and 'value = list("new_units" = new_units, ...)' (denoting the arguments supplied to [scale_time()]). See [pk_add.pk_scales()].

settings_data_info *Data info settings*

Description

Data info settings

Usage

```
settings_data_info(
  summary_group = dplyr::vars(Chemical, Species, Reference, Route, Media, Dose),
  ...
)
```

Arguments

summary_group	A set of variables. Data will be split into groups according to unique combinations of these variables, and non-compartmental analysis will be performed separately on each group. Default ‘dplyr::vars(Chemical, Species, Reference, Route, Media, Dose)‘.
...	Other arguments (currently ignored).

Value

An object of class ‘c(pkproto, pk_settings_data_info)‘

---

settings_optimx *‘optimx‘ optimizer settings*

Description

‘optimx‘ optimizer settings

Usage

```
settings_optimx(
  method = c("bobyqa", "L-BFGS-B"),
  itnmax = 1e+06,
  hessian = FALSE,
  control = list(kkt = FALSE),
  ...
)
```

Arguments

method	The name(s) of optimization methods to be used. See [optimx::optimx()] for options. Default is ‘"bobyqa"‘.
itnmax	The maximum number of iterations; as in [optimx::optimx()].
hessian	Whether to compute the Hessian at the final set of parameters; as in [optimx::optimx()].
control	A list of control parameters for the optimizer; see [optimx::optimx()] for options and details.
...	Additional arguments not currently implemented.

Value

An object of class ‘pk_settings‘.

Author(s)

Caroline Ring

`settings_preprocess` *Data preprocessing settings*

Description

Data preprocessing settings

Usage

```
settings_preprocess(
  routes_keep = c("oral", "iv"),
  media_keep = c("blood", "plasma"),
  ratio_conc_dose = 1,
  impute_loq = TRUE,
  loq_group = dplyr::vars(Chemical, Species, Reference, Media),
  calc_loq_factor = 0.9,
  impute_sd = TRUE,
  sd_group = dplyr::vars(Chemical, Species, Reference, Media),
  suppress.messages = FALSE,
  ...
)
```

Arguments

`routes_keep`	Character: A list of routes to keep. Data will be filtered so that the harmonized variable 'Route' includes only values in 'routes_keep'. Default is 'c("oral", "iv")'.
`media_keep`	Character: A list of media to keep. Data will be filtered so that the harmonized variable name 'Media' includes only values in 'media_keep'. Default is 'c("blood", "plasma")'.
`ratio_conc_dose`	Numeric: The ratio of mass units of observed concentrations to mass units of applied doses. Default 1, to indicate the same mass units are used for both.
`impute_loq`	TRUE or FALSE: Whether to impute missing LOQ values.
`loq_group`	A list of variables, specified using a call to [dplyr::vars()]. These should be harmonized variable names. Unique combinations of these variables define groups of data. Within each group, any missing LOQ values will be imputed as the minimum detected Value in the group, multiplied by 'calc_loq_factor'. Default is 'dplyr::vars(Chemical, Species, Reference, Media)'.
`calc_loq_factor`	A numeric factor used for imputing missing LOQ. Within each group defined in 'loq_group', any missing LOQ values will be imputed as the minimum detected Value in the group, multiplied by 'calc_loq_factor'. Default 0.9.
`impute_sd`	TRUE or FALSE: Whether to impute missing SD values.

sd_group	A list of variables, specified using a call to [dplyr::vars()]. These should be harmonized variable names. Unique combinations of these variables define groups of data. Within each group, any missing SD values will be imputed as the minimum non-missing SD value in the group. Default is 'dplyr::vars(Chemical, Species, Reference, Media)'.
suppress.messages	TRUE or FALSE: Whether to suppress verbose messages. Default FALSE.
...	Any additional arguments. Currently ignored.

Value

An object of class 'pk_settings_preprocess'. This is a named list of the arguments provided to this function and their values.

Author(s)

Caroline Ring

`status_data_info` *Status ID for data summary info*

Description

An integer status that denotes [data_info()] has been completed.

Usage

```
status_data_info
```

Format

An object of class `integer` of length 1.

`status_fit` *Status ID for fitting*

Description

An integer status that denotes [fit()] has been completed.

Usage

```
status_fit
```

Format

An object of class `integer` of length 1.

status_init *Status ID for initialization*

Description

An integer status that denotes a [pk()] object has been initialized.

Usage

```
status_init
```

Format

An object of class `integer` of length 1.

status_prefit *Status ID for pre-fitting*

Description

An integer status that denotes [prefit()] has been completed.

Usage

```
status_prefit
```

Format

An object of class `integer` of length 1.

status_preprocess *Status ID for preprocessing*

Description

An integer status that denotes [preprocess_data()] has been completed.

Usage

```
status_preprocess
```

Format

An object of class `integer` of length 1.

`stat_error_model` *Error model*

Description

Define an error model.

Usage

```
stat_error_model(error_group = vars(Chemical, Species, Reference, Media), ...)
```

Arguments

`error_group`	Defined using [dplyr::vars()]: A set of harmonized variables whose unique combinations define a group with its own error variance. These variables refer to the ‘data‘ element of the ‘pk‘ object to which ‘stat_error_model()‘ is added. The variables should not be quoted. Default is ‘vars(Chemical, Species, Reference, Media)‘.
`...`	Additional arguments. Not currently used.

Details

‘stat_error_model‘ defines groupings for a fixed-effects error model. For each model in ‘stat_model‘, a single set of model parameters will be fit to ‘data‘. In order to do the fitting, the residual errors (observed concentrations - model-predicted concentrations) are assumed to obey a zero-mean normal distribution. However, in this package, the residuals are not all required to obey the *same* zero-mean normal distribution. Different groups of residuals may obey zero-mean normal distributions with different variances. ‘stat_error_model‘ defines these groups as unique combinations of the variables given in argument ‘error_group‘. For example, the default value ‘vars(Chemical, Species, Reference, Media)‘ means that for each group of observations in ‘data‘ with a unique combination of ‘Chemical‘, ‘Species‘, ‘Reference‘, and ‘Media‘, there is a separate residual error variance. For example, if there happened to be three such unique combinations, there would be three error variances.

If you want all residuals to obey the same zero-mean normal distribution (i.e., for there to be only one residual error variance), then you should provide an ‘error_group‘ that puts all the data in the same group. For example, since all data in ‘data‘ should already be for a single ‘Chemical‘ and ‘Species‘, you could provide ‘error_group = vars(Chemical, Species)‘ to put all the data in the same group.

Note that, since all data in ‘data‘ should already be for a single ‘Chemical‘ and ‘Species‘, you could leave out ‘Chemical‘ and ‘Species‘ from ‘error_group‘ and still get the same result. However, we recommend explicitly including ‘Chemical‘ and ‘Species‘. Tncluding them will make your code more explicit and transparent, and it does no harm. In addition, [invivopkfit] may be extended in the future to allow input of data with multiple chemicals or species; explicitly including ‘Chemical‘ and ‘Species‘ in your ‘error_group‘ will future-proof your code in that sense.

The error variance(s) are hyperparameters that will be estimated from the data along with the model parameters. That means there needs to be enough data to fit the model parameters plus the error

variances. For example, if you are fitting a 1-compartment model to oral and IV data measured in plasma, and using an error model with three separate error-variance groups (e.g. three different References), then you are trying to fit 4 model parameters (kelim, Vdist, Fgutabs, kgutabs) plus 3 error variances, for a total of 7 parameters. That means you need to have at least 8 data points. (When you call [prefit()], this checking is done automatically. But it is useful to be aware of this, in case you are trying to figure out why your fit was aborted due to insufficient data availability.)

Value

An object of class ‘pk_stat_error_model‘: A named list of all the arguments to ‘stat_error_model‘.

Author(s)

Caroline Ring

`stat_model` *Add PK model(s) to be fitted*

Description

Add PK model(s) to be fitted

Usage

```
stat_model(model = c("model_flat", "model_1comp", "model_2comp"), ...)
```

Arguments

`model`	A character vector: the name(s) of models to be fitted. These should be the names of objects of class ‘pk_model‘. Built-in options are [‘model_flat‘], [‘model_1comp‘], and [‘model_2comp‘]. You may add your own model by using [pk_model()].
`...`	Additional arguments not currently in use.

`subtract_pk` *Subtract various ‘pkproto‘ objects from a ‘pk‘ object*

Description

Subtract various ‘pkproto‘ objects from a ‘pk‘ object

Usage

```
subtract_pk(pk_obj, object, objectname)
```

Arguments

pk_obj	The ‘pk‘ object
object	The ‘pkproto‘ object to be subtracted
objectname	The name of the ‘pkproto‘ object to be subtracted

Value

The ‘pk‘ object modified by the subtraction.

---

summary.pk *Print summary of a ‘pk‘ object*

Description

This summary includes summary information about the data; about any data transformations applied; about the models being fitted; about the error model being applied; and any fitting results, if the ‘pk‘ object has been fitted. It also includes TK quantities calculated from the fitted model parameters, e.g. halflife; clearance; tmax; Cmax; AUC; Css.

Usage

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

Arguments

object	A [pk] object.
...	Additional arguments. Currently not in use.

Value

A list of ‘data.frame‘s consisting of a summary table of fitting options and results.

Author(s)

Caroline Ring

time_conversions *Time conversion table*

Description

A ‘data.frame‘ that has the converted units from "time_units"

Usage

```
time_conversions
```

Format

An object of class `data.frame` with 121 rows and 3 columns.

time_units *Allowable time units*

Description

A ‘character‘ vector of allowable units for time variables.

Usage

```
time_units
```

Format

An object of class `character` of length 11.

Details

These are the time units understood by [lubridate::period()] and [lubridate::duration()].

`tkstats_1comp` *Toxicokinetic statistics for 1-compartment model*

Description

Calculate predicted toxicokinetic statistics for a 1-compartment model.

Usage

```
tkstats_1comp(pars, route, medium, dose, time_unit, conc_unit, vol_unit, ...)
```

Arguments

`pars`	A named vector of model parameters (e.g. from [coef.pk()]).
`route`	Character: The route for which to compute TK stats. Currently only "oral" and "iv" are supported.
`medium`	Character: the media (tissue) for which to compute TK stats. Currently only "blood" and "plasma" are supported.
`dose`	Numeric: A dose for which to calculate TK stats.
`time_unit`	Character: the units of time used for the parameters ‘par‘. For example, if ‘par["kelim"]‘ is in units of 1/weeks, then ‘time_unit = "weeks"‘. If ‘par["kelim"]‘ is in units of 1/hours, then ‘time_unit = "hours"‘. This is used to calculate the steady-state plasma/blood concentration for long-term daily dosing of 1 mg/kg/day.
`conc_unit`	Character: The units of concentration.
`vol_unit`	Character: The units of dose.
`...`	Additional arguments not currently in use.

Details

Statistics computed

Total clearance

$$\mathrm{CL}_{tot} = k_{elim} + V_{dist}$$

Steady-state plasma concentration for long-term daily dose of 1 mg/kg/day

The dosing interval $\tau = \frac{1}{\text{day}}$ will be converted to the same units as k_{elim}.

To convert to steady-state *blood* concentration, multiply by the blood-to-plasma ratio.

Oral route

$$C_{ss} = \frac{F_{gutabs} V_{dist}}{k_{elim} \tau}$$

Intravenous route

$$C_{ss} = \frac{1}{24 * \text{CL}_{tot}}$$

Half-life of elimination

$$\text{Halflife} = \frac{\log(2)}{k_{elim}}$$

Time of peak concentration

For oral route:

$$\frac{\log\left(\frac{k_{gutabs}}{k_{elim}}\right)}{k_{gutabs} - k_{elim}}$$

For intravenous route, time of peak concentration is always 0.

Peak concentration

Evaluate [cp_1comp()] at the time of peak concentration.

AUC evaluated at infinite time

Evaluate [auc_1comp()] at time = 'Inf'.

AUC evaluated at the time of the last observation

Evaluate [auc_1comp()] at time = 'tlast'.

Value

A 'data.frame' with two variables: - 'param_name' = 'c("CLtot", "CLtot/Fgutabs", "Css", "halflife", "tmax", "Cmax", "AUC_infinity")' - 'param_value' = The corresponding values for each statistic (which may be NA if that statistic could not be computed).

Author(s)

John Wambaugh, Caroline Ring

`tkstats_1comp_cl` *Toxicokinetic statistics for 1-compartment model with specific clearance*

Description

Calculate predicted toxicokinetic statistics for a 1-compartment model. This does use the parameters for the 'model_1comp_cl' that are taken from [httk] estimates.

Usage

```
tkstats_1comp_cl(
  pars,
  route,
  medium,
  dose,
  time_unit,
  conc_unit,
  vol_unit,
  ...
)
```

Arguments

pars	A named vector of model parameters (e.g. from [coef.pk()]).
route	Character: The route for which to compute TK stats. Currently only "oral" and "iv" are supported.
medium	Character: the media (tissue) for which to compute TK stats. Currently only "blood" and "plasma" are supported.
dose	Numeric: A dose for which to calculate TK stats.
time_unit	Character: the units of time used for the parameters ‘par‘. For example, if ‘par["kelim"]‘ is in units of 1/weeks, then ‘time_unit = "weeks"‘. If ‘par["kelim"]‘ is in units of 1/hours, then ‘time_unit = "hours"‘. This is used to calculate the steady-state plasma/blood concentration for long-term daily dosing of 1 mg/kg/day.
conc_unit	Character: The units of concentration.
vol_unit	Character: The units of dose.
...	Additional arguments not currently in use.

Details

Statistics computed

Total clearance

$$\mathrm{CL}_{tot} = k_{elim} + V_{dist}$$

Steady-state plasma concentration for long-term daily dose of 1 mg/kg/day

The dosing interval $\tau = \frac{1}{\mathrm{day}}$ will be converted to the same units as k_{elim}.

To convert to steady-state *blood* concentration, multiply by the blood-to-plasma ratio.

Oral route

$$C_{ss} = \frac{F_{gutabs} V_{dist}}{k_{elim} \tau}$$

Intravenous route

$$C_{ss} = \frac{1}{24 * \text{CL}_{tot}}$$

Half-life of elimination

$$\text{Halflife} = \frac{\log(2)}{k_{elim}}$$

Time of peak concentration

For oral route:

$$\frac{\log\left(\frac{k_{gutabs}}{k_{elim}}\right)}{k_{gutabs} - k_{elim}}$$

For intravenous route, time of peak concentration is always 0.

Peak concentration

Evaluate [cp_1comp_cl()] at the time of peak concentration.

AUC evaluated at infinite time

Evaluate [auc_1comp_cl()] at time = 'Inf'.

AUC evaluated at the time of the last observation

Evaluate [auc_1comp_cl()] at time = 'tlast'.

Value

A 'data.frame' with two variables: - 'param_name' = 'c("CLtot", "CLtot/Fgutabs", "Css", "halflife", "tmax", "Cmax", "AUC_infinity")' - 'param_value' = The corresponding values for each statistic (which may be NA if that statistic could not be computed).

Author(s)

John Wambaugh, Caroline Ring

`tkstats_2comp` *Toxicokinetic statistics for 1-compartment model*

Description

Calculate predicted toxicokinetic statistics for a 1-compartment model.

Usage

```
tkstats_2comp(pars, route, medium, dose, time_unit, conc_unit, vol_unit, ...)
```

Arguments

pars	A named numeric vector of model parameters (e.g. from [coef.pk()]).
route	Character: The route for which to compute TK stats. Currently only "oral" and "iv" are supported.
medium	Character: the media (tissue) for which to compute TK stats.. Currently only "blood" and "plasma" are supported.
dose	Numeric: A dose for which to calculate TK stats.
time_unit	Character: the units of time used for the parameters ‘par‘. For example, if ‘par["kelim"]‘ is in units of 1/weeks, then ‘time_unit = "weeks"‘. If ‘par["kelim"]‘ is in units of 1/hours, then ‘time_unit = "hours"‘. This is used to calculate the steady-state plasma/blood concentration for long-term daily dosing of 1 mg/kg/day.
conc_unit	Character: The units of concentration.
vol_unit	Character: The units of dose.
...	Additional arguments not currently in use.

Details

Statistics computed

Total clearance

$$\mathrm{CL}_{tot} = k_{elim} + V_1$$

Steady-state plasma concentration for long-term daily dose of 1 mg/kg/day

To convert to steady-state *blood* concentration, multiply by the blood-to-plasma ratio.

The dosing interval $\tau = \frac{1}{\mathrm{day}}$ will be converted to the same units as k_{elim}.

Oral route

$$C_{ss} = \frac{F_{gutabs} V_1}{k_{elim} \tau}$$

Intravenous route

$$C_{ss} = \frac{1}{\mathrm{CL}_{tot} \tau}$$

Half-life of elimination

$$\mathrm{Halflife} = \frac{\log(2)}{k_{elim}}$$

Time of peak concentration

For oral route:

$$\frac{\log\left(\frac{k_{gutabs}}{k_{elim}}\right)}{k_{gutabs} - k_{elim}}$$

For intravenous route, time of peak concentration is always 0.

Peak concentration

Evaluate [cp_1comp()] at the time of peak concentration.

AUC evaluated at infinite time

Evaluate [auc_1comp()] at time = ‘Inf‘.

AUC evaluated at the time of the last observation

Evaluate [auc_1comp()] at time = ‘tlast‘.

Value

A ‘data.frame‘ with two variables: - ‘param_name‘ = ‘c("CLtot", "CLtot/Fgutabs", "Css", "halflife", "tmax", "Cmax", "AUC_infinity", "A", "B", "alpha", "beta", "Vbeta", "Vbeta_Fgutabs", "Vss", "Vss_Fgutabs")‘ - ‘param_value‘ = The corresponding values for each statistic (which may be NA if that statistic could not be computed; e.g. all of the ‘"x_Fgutabs"‘ parameters can only be computed if ‘route = "oral"‘).

Author(s)

John Wambaugh, Caroline Ring

See Also

Other built-in model functions: auc_1comp(), auc_1comp_cl(), auc_2comp(), auc_flat(), cp_1comp(), cp_1comp_cl(), cp_2comp(), cp_2comp_dt(), cp_flat(), get_params_1comp(), get_params_1comp_cl(), get_params_1comp_fup(), get_params_2comp(), get_params_flat(), get_starts_1comp(), get_starts_1comp_cl(), get_starts_1comp_fup(), get_starts_2comp(), get_starts_flat(), transformed_params_2comp()

Other 2-compartment model functions: auc_2comp(), cp_2comp(), cp_2comp_dt(), get_params_2comp(), get_starts_2comp(), transformed_params_2comp()

---

tkstats_flat *TK stats for flat model*

Description

TK stats for flat model

Usage

```
tkstats_flat(pars, route, medium, dose, time_unit, conc_unit, vol_unit, ...)
```

Arguments

pars	A named numeric vector of model parameters (e.g. from [coef.pk()]).
route	Character: The route for which to compute TK stats. Currently only "oral" and "iv" are supported.
medium	Character: the media (tissue) for which to compute TK stats. Currently only "blood" and "plasma" are supported.
dose	Numeric: A dose for which to calculate TK stats.
time_unit	Character: the units of time.
conc_unit	Character: The units of concentration.
vol_unit	Character: The units of dose.
...	Additional arguments not currently in use.

Value

A ‘data.frame‘ with two variables: - ‘param_name‘ = ‘c("CLtot", "CLtot/Fgutabs", "Css_1mgkgday", "halflife", "Cmax", "AUC_infinity")‘ - ‘param_value‘ = The corresponding values for each statistic (which may be NA if that statistic could not be computed).

Author(s)

John Wambaugh, Caroline Ring

---

transformed_params_2comp

Transformed parameters for 2-compartment model

Description

Transformed parameters for 2-compartment model

Usage

```
transformed_params_2comp(params, ...)
```

Arguments

params	A named numeric vector of parameters for the 2-compartment model. Any missing parameters will be filled with ‘NA_real_‘.
...	Additional arguments (not currently used).

Value

A named numeric vector of transformed parameters with elements "alpha", "beta", "A_iv_unit", "B_iv_unit", "A_oral_unit", "B_oral_unit".

Author(s)

Caroline Ring, Gilberto Padilla Mercado, John Wambaugh

See Also

Other built-in model functions: auc_1comp(), auc_1comp_cl(), auc_2comp(), auc_flat(), cp_1comp(), cp_1comp_cl(), cp_2comp(), cp_2comp_dt(), cp_flat(), get_params_1comp(), get_params_1comp_cl(), get_params_1comp_fup(), get_params_2comp(), get_params_flat(), get_starts_1comp(), get_starts_1comp_cl(), get_starts_1comp_fup(), get_starts_2comp(), get_starts_flat(), tkstats_2comp()

Other 2-compartment model functions: auc_2comp(), cp_2comp(), cp_2comp_dt(), get_params_2comp(), get_starts_2comp(), tkstats_2comp()

---

twofold_test *twofold_test()*

Description

This is the S3 method generic for twofold_test()

Usage

```
twofold_test(obj, ...)
```

Arguments

obj	An object.
...	Additional arguments currently not in use.

Value

A list of data frames.

See Also

[twofold_test.pk()] for the method for class [pk()]

twofold_test.default *Default method for twofold_test()*

Description

Default method for twofold_test()

Usage

```
## Default S3 method:
twofold_test(obj, ...)
```

Arguments

obj	An object
...	Additional arguments currently not in use.

Value

An error, when a non-pk object is used for the first argument.

twofold_test.pk *Evaluate whether data and predictions are within two-fold of mean or concentration, respectively*

Description

At each timepoint across CvT experimental data, there are three ways that data may be presented. These can be found as either: - multiple individual observations - single individual observation - summarized group of observations (mean concentration and standard deviation)

Usage

```
## S3 method for class 'pk'
twofold_test(
  obj,
  sub_pLOQ = TRUE,
  suppress.messages = NULL,
  model = NULL,
  method = NULL,
  ...
)
```

Arguments

obj	A pk object.
sub_pLOQ	TRUE (default): Substitute all predictions below the LOQ with the LOQ before computing fold errors. FALSE: do not. Only used if 'obj' has been fitted and predictions are possible.
suppress.messages	Logical: whether to suppress message printing. If NULL (default), uses the setting in 'obj$settings_preprocess$suppress.messages'.
model	Optional: Specify one or more of the fitted models for which to make predictions. If NULL (the default), predictions will be returned for all of the models in 'object$stat_model'.
method	Optional: Specify one or more of the [optimx::optimx()] methods for which to make predictions. If NULL (the default), predictions will be returned for all of the models in 'object$settings_optimx$method'.
...	Additional arguments. Currently unused.

Details

For the purposes of this calculations we largely divide the data into two groups, those with individual observations, where N_Subjects == 1, and the summarized group of observations.

First this creates mean-normalized concentrations for individual data. Then it summarizes data (individual & summarized) by 'mean' and 'sd'. It tests whether predictions are within two-fold of mean, in the latter case whether the 95

Furthermore if 'pk' object 'status == 5' then it calculates the model error by evaluating _prediction/concentration_ at each timepoint for all data. Each test is done for data from individual subject observations and for all data by summarizing the observations.

Only non-excluded detects are included in this analysis.

Value

A list of data frames.

Author(s)

Gilberto Padilla Mercado

Index