

Package: PKNCA (via r-universe)

October 9, 2026

Type Package

Title Perform Pharmacokinetic Non-Compartmental Analysis

Version 0.12.1.9002

Depends R (>= 4.1)

Imports checkmate, dplyr (>= 1.1.0), digest, nlme, purrr, rlang,
stats, tidyr, tibble, utils, lifecycle, lubridate, withr

Suggests cdisccdata (>= 0.2.0), covr, cowplot, emmeans, ggplot2,
ggtribble, knitr, labeling, lme4, lmerTest, pander, rmarkdown,
spelling, testthat (>= 3.0.0), tools, units

Description Compute standard Non-Compartmental Analysis (NCA)
parameters for typical pharmacokinetic analyses and summarize
them.

License AGPL-3

URL <https://pknca.humanpredictions.com/>,
<https://github.com/humanpred/pknca>

BugReports <https://github.com/humanpred/pknca/issues>

NeedsCompilation no

VignetteBuilder knitr

Roxygen list(markdown = TRUE)

Config/testthat/edition 3

Config/testthat/parallel true

Config/testthat/start-first pk.calc.all, half.life,
update.PKNCAresults, exclude_nca, class-PKNCAresults, impute,
check.intervals, bioequivalence-assess

Encoding UTF-8

Language en-US

Config/roxygen2/version 8.1.0

Config/pak/sysreqs libicu-dev

Repository <https://humanpred.r-universe.dev>

Date/Publication 2026-10-09 13:34:00 UTC

RemoteUrl <https://github.com/humanpred/pknca>

RemoteRef HEAD

RemoteSha bdde0560b03aa6c6b6a29e0c5a4a51abc3665269

Contents

add.interval.col	6
addProvenance	10
adj.r.squared	11
any_sparse_dense_in_interval	12
as.data.frame.PKNCAresults	12
as_PKNCAconc	14
as_sparse_pk	15
assert_aucmethod	16
assert_conc	16
assert_dosetau	17
assert_impute_method	18
assert_intervals	19
assert_intervaltime_single	19
assert_lambdaz	20
assert_number_between	20
assert_numeric_between	21
assert_param_name	22
assert_PKNCAdata	23
assert_unit_col	23
auc_integrate	24
be_assess	25
be_compare	28
be_dataset	30
be_design	32
be_expand_limits	33
be_extract_param	34
be_fit_model_single	35
be_fit_models	36
be_regulator	37
be_table	39
be_within_var	40
business.mean	41
check.conversion	42
check.interval.specification	43
checkProvenance	44
choose.auc.intervals	44
choose_interval_method	47
clean.conc.blq	48
clean.conc.na	49
cov_holder	50

defunct	51
ensure_column_unit_exists	51
exclude	52
exclude_nca	53
exclude_nca_by_param	56
filter.PKNCAresults	57
filter_interval	59
find.dose.regimen	59
find.tau	62
findOperator	64
fit_half_life	65
fit_half_life_tobit	65
fit_half_life_tobit_LL	66
formula.PKNCAconc	67
geomean	67
get.best.model	68
get.interval.cols	69
get.parameter.deps	69
get_halflife_curve	70
get_halflife_fit	71
get_halflife_points	72
get_impute_column	73
get_impute_method	74
getAttributeColumn	75
getColumnValueOrNot	75
getDataName.PKNCAconc	76
getDepVar	77
getGroups.PKNCAconc	77
getIndepVar	79
group_by.PKNCAresults	79
group_vars.PKNCAconc	80
inner_join.PKNCAresults	81
interp.extrap.conc	85
interp_extrap_conc_method	88
interval_add_impute	89
interval_add_param	90
interval_add_secondary	91
is_sparse_pk.PKNCAconc	93
model.frame.PKNCAconc	94
mutate.PKNCAresults	94
normalize	95
normalize_by_col	96
normalize_exclude	96
parse_formula_to_cols	97
pk.business	97
pk.calc.ae	98
pk.calc.aucabove	99
pk.calc.aucinf.obs_sparse	99

pk.calc.aucivlast_sparse	101
pk.calc.auclast_sparse	103
pk.calc.aucpext	104
pk.calc.auxc	105
pk.calc.auxcint	108
pk.calc.auxciv	112
pk.calc.c0	115
pk.calc.cav	117
pk.calc.ceoi	117
pk.calc.cl	118
pk.calc.clast.obs	119
pk.calc.clr	119
pk.calc.cmax	120
pk.calc.count_conc	121
pk.calc.ctrough	122
pk.calc.deg.fluc	122
pk.calc.dn	123
pk.calc.erint	124
pk.calc.erlst	124
pk.calc.emax	125
pk.calc.ertlst	126
pk.calc.ertmax	127
pk.calc.f	128
pk.calc.fe	128
pk.calc.half.life	129
pk.calc.kel	132
pk.calc.mrt	133
pk.calc.ptr	134
pk.calc.ratio	135
pk.calc.sparse_auc	136
pk.calc.sparse_aumc	137
pk.calc.thalf.eff	138
pk.calc.time_above	139
pk.calc.tlag	140
pk.calc.tlast	140
pk.calc.tmax	141
pk.calc.tmin	142
pk.calc.totdose	143
pk.calc.volpk	143
pk.calc.vz	144
pk.nca	145
pk.nca.interval	146
pk.nca.intervals	148
pk.tss	149
pk.tss.data.prep	150
pk.tss.monoexponential	151
pk.tss.monoexponential.individual	152
pk.tss.monoexponential.population	152

pk.tss.stepwise.linear	153
pk_nca_result_to_df	154
PKNCA	155
PKNCA.choose.option	156
PKNCA.options	157
PKNCA.options.describe	158
PKNCA.set.summary	158
pknca_cdisc_codes	160
pknca_check_parameter_classification	161
pknca_concept	161
pknca_concepts	162
pknca_difftime_to_unit	164
pknca_exclude_rules	165
pknca_find_units_param	166
PKNCA_impute_fun_list	167
PKNCA_impute_method	167
pknca_impute_methods	169
pknca_interval_table	171
pknca_match_route	173
pknca_missing_samples	174
PKNCA_options_defaults	175
pknca_parameter_table	176
pknca_presets	177
pknca_ref	177
pknca_unit_conversion	178
pknca_unit_reconcile_factor	179
pknca_units_add_paren	179
pknca_units_conversion_factor	180
pknca_units_table	180
PKNCAconc	182
PKNCAdata	185
PKNCAdose	188
PKNCAresults	190
print.PKNCAconc	191
print.PKNCAdata	191
print.provenance	192
print.summary_PKNCAresults	192
resolve_dose_tau	193
roundingSummarize	193
roundString	194
select_minimal_grouping_cols	195
set_intervals	195
setAttributeColumn	196
setDuration.PKNCAconc	197
setExcludeColumn	197
setRoute	198
signifString	199
sparse_auc_weight_linear	200

sparse_bootstrap 201

sparse_mean 202

sparse_pk_attribute 203

sparse_to_dense_pk 204

summary.PKNCAdata 204

summary.PKNCAresults 205

summary.PKNCAresults_sparse_bootstrap 207

superposition 207

time_calc 209

tss.monoexponential.generate.formula 210

update.PKNCAresults 211

var_sparse_auc 211

Index 213

add.interval.col	<i>Add columns for calculations within PKNCA intervals</i>
------------------	--

Description

Add columns for calculations within PKNCA intervals

Usage

```
add.interval.col(  
  name,  
  FUN,  
  values = c(FALSE, TRUE),  
  unit_type,  
  pretty_name,  
  depends = NULL,  
  desc = "",  
  sparse = FALSE,  
  formalsmap = list(),  
  FUN_sparse = NA_character_,  
  formalsmap_sparse = list(),  
  datatype = c("interval", "individual", "population"),  
  pptestcd_cdisc = NULL,  
  pptest_cdisc = NULL,  
  formula = NULL,  
  formula_note = NULL,  
  tier = "uncommon",  
  selection = NULL  
)
```

Arguments

name	The column name as a non-empty character string (length 1, may not be NA or ""). Names ending in <code>_ref</code> and the name <code>interval_id</code> are reserved for the reference-interval linkage columns of the interval specification and may not be used.
FUN	The function to run (as a character string) or NA if the parameter is automatically calculated when calculating another parameter.
values	Valid values for the column: either a function used to coerce/validate values (e.g. <code>as.numeric</code>) or a vector of allowed values (e.g. <code>c(FALSE, TRUE)</code>).
unit_type	The type of units to use for assigning and converting units. Must be one of the pre-defined unit types (see Details). This argument is required and has no default; omitting it raises an error.
pretty_name	The name of the parameter to use for printing in summary tables with units. (If an analysis does not include units, then the normal name is used.)
depends	Character vector of columns that must be run before this column.
desc	A human-readable description of the parameter. SDTM requires ≤ 40 characters; a longer description is accepted with a warning.
sparse	Retired. TRUE is an error: register a sparse-only parameter with <code>FUN = NA</code> and a <code>FUN_sparse</code> (plus <code>formalsmap_sparse</code>) instead, which is now what makes a parameter sparse-only. FALSE, the default, is accepted and does nothing.
formalsmap	A named list mapping parameter names in the function call to NCA parameter names. See the details for information on use of <code>formalsmap</code> .
FUN_sparse	The function to run (as a character string) when the data are sparse PK, or NA (the default) when the parameter has no sparse-specific estimator. With sparse data, a parameter that has one uses it; a parameter that does not falls back to <code>FUN</code> applied to the arithmetic-mean profile. See the details.
formalsmap_sparse	The <code>formalsmap</code> for <code>FUN_sparse</code> , which usually has a different calling convention than <code>FUN</code> (a sparse estimator needs <code>subject</code> , for example). May only be given when <code>FUN_sparse</code> is not NA.
datatype	The data type used for the calculation. The default is "interval", which is currently the only supported value. The "individual" and "population" data types are reserved for future use and will currently raise an error if selected.
pptestcd_cdisc	The CDISC PPTTESTCD code for this parameter. Can be a character string for simple mappings, a named list for route-dependent mappings with a route element whose value is itself a named list keyed by route (e.g. <code>list(route = list(extravascular = "CLF/F0", intravascular = "CLO"))</code>), or a named list with exactly the elements <code>dense</code> and <code>sparse</code> , for a parameter with a <code>FUN_sparse</code> whose sparse estimate has a code of its own (e.g. <code>list(dense = "AUCLST", sparse = "SPARSEAL")</code>). Defaults to <code>name</code> if not provided.
pptest_cdisc	The CDISC PPTTEST name for this parameter. Can be a character string or a named list (same structure as <code>pptestcd_cdisc</code>). Defaults to <code>desc</code> if not provided.

formula	Character value providing a LaTeX expression for how the parameter is calculated. Optional and used only for documentation.
formula_note	Character value providing additional context about the formula (e.g. assumptions or method details). Displayed alongside the formula in documentation tables.
tier	How commonly the parameter is reported: "common" for one that belongs in a default report for at least one context, or "uncommon" (the default) for one that is calculated only when asked for by name. See pknca_tiers() .
selection	A named list declaring what cannot be derived about where the parameter applies. Every element is optional, and the default of NULL derives everything: - concept The kind of quantity the parameter is (see pknca_concepts()). Needed only when the calculation function carries no pknca_concept() and the concept cannot be taken from depends. - route The routes of administration the parameter applies to (see pknca_routes()). Derived as intravenous for anything calculated from <code>c0</code> or needing a dose duration, and as any route otherwise. - dosing The dosing patterns the parameter applies to (see pknca_dosing()). Derived as single-dose for anything calculated from an extrapolation to infinity. A declared value propagates to every parameter calculated from this one. - secondary TRUE for a parameter that needs inputs from more than one profile, such as bioavailability, which compares two administrations, or renal clearance, which needs an amount excreted and a plasma AUC. One interval cannot supply those, so a secondary parameter is never chosen automatically; it stays available by name. A declared value propagates to every parameter calculated from this one.

Details

`FUN_sparse` gives a parameter a second calculation function for sparse PK. With sparse data, a parameter that has one is calculated with it – from the pooled individual samples, and with `formalsmap_sparse` in place of `formalsmap` – and its result is reported as a sparse result. A parameter with no `FUN_sparse` falls back to `FUN` applied to the arithmetic-mean profile, which is what sparse data have always done. A sparse estimator names its concentration inputs `conc/time` the way a dense one does; those draw from the pooled samples rather than the mean profile. The estimators PKNCA ships are linear-trapezoidal only, so `auc.method` does not apply to them.

The `formalsmap` argument enables mapping some alternate formal argument names to parameters. It is used to generalize functions that may use multiple similar arguments (such as the variants of mean residence time). The names of the list should correspond to function formal parameter names and the values should be one of the following:

- For the current interval:
 - character strings of NCA parameter name** The value of the parameter calculated for the current interval.
 - "conc"** Concentration measurements for the current interval.
 - "time"** Times associated with concentration measurements for the current interval (values start at 0 at the beginning of the current interval).

"volume" Volume associated with concentration measurements for the current interval (typically applies for excretion parameters like urine).

"duration.conc" Durations associated with concentration measurements for the current interval.

"dose" Dose amounts associated with the current interval.

"time.dose" Time of dose start associated with the current interval (values start at 0 at the beginning of the current interval).

"duration.dose" Duration of dose (typically infusion duration) for doses in the current interval.

"route" Route of dosing for the current interval.

"start" Time of interval start.

"end" Time of interval end.

"options" PKNCA.options governing calculations.

- For the current group:

"conc.group" Concentration measurements for the current group.

"time.group" Times associated with concentration measurements for the current group (values start at 0 at the beginning of the current interval).

"volume.group" Volume associated with concentration measurements for the current interval (typically applies for excretion parameters like urine).

"duration.conc.group" Durations associated with concentration measurements for the current group.

"dose.group" Dose amounts associated with the current group.

"time.dose.group" Time of dose start associated with the current group (values start at 0 at the beginning of the current interval).

"duration.dose.group" Duration of dose (typically infusion duration) for doses in the current group.

"route.group" Route of dosing for the current group.

- For sparse PK (NULL with dense PK, so a parameter naming one of these is only calculable with a sparse PKNCAconc):

"conc.sparse" The pooled individual concentration measurements for the current interval ("conc" is the arithmetic-mean profile built from them).

"time.sparse" Times associated with the pooled individual concentration measurements for the current interval (values start at 0 at the beginning of the current interval).

"conc.sparse.group" The pooled individual concentration measurements for the current group.

"time.sparse.group" Times associated with the pooled individual concentration measurements for the current group.

"subject" Subject identifiers for the pooled individual concentration measurements for the current interval.

- Constants:

a value wrapped in `base::I()` The value itself, passed to the function unchanged. Use this for an argument that selects a variant of a shared calculation function (for example, `auc.type = I("AUCall")`) rather than naming a data source or another parameter.

- For the reference interval:

a parameter name wrapped in `pknca_ref()` The value of that NCA parameter calculated in the reference interval, which is the interval named by the `<name>_ref` column of the interval specification. A parameter with any such argument is a secondary parameter; see the vignette "Secondary parameters".

Value

NULL (Calling this function has a side effect of changing the available intervals for calculations)

See Also

Other Interval specifications: `check.interval.specification()`, `choose.auc.intervals()`, `get.interval.cols()`, `get.parameter.deps()`, `interval_add_impute()`, `interval_add_param()`, `interval_add_secondary()`, `pknca_cdisc_codes()`, `pknca_check_parameter_classification()`, `pknca_concepts()`, `pknca_interval_table()`, `pknca_match_route()`, `pknca_parameter_table()`, `pknca_presets()`, `pknca_ref()`

Examples

```
## Not run:
add.interval.col("cmax",
  FUN="pk.calc.cmax",
  values=c(FALSE, TRUE),
  unit_type="conc",
  pretty_name="Cmax",
  desc="Maximum observed concentration")
add.interval.col("cmax.dn",
  FUN="pk.calc.dn",
  values=c(FALSE, TRUE),
  unit_type="conc_dosenorm",
  pretty_name="Cmax (dose-normalized)",
  desc="Maximum observed concentration, dose normalized",
  formalsmap=list(parameter="cmax"),
  depends="cmax")

## End(Not run)
```

addProvenance	<i>Add a hash and associated information to enable checking object provenance.</i>
---------------	--

Description

Add a hash and associated information to enable checking object provenance.

Usage

```
addProvenance(object, replace = FALSE)
```

Arguments

object	The object to add provenance
replace	Replace provenance if the object already has a provenance attribute. (If the object already has provenance and replace is FALSE, then an error will be raised.)

Value

The object with provenance as an added item

See Also

[checkProvenance\(\)](#)

adj.r.squared	<i>Calculate the adjusted r-squared value</i>
---------------	---

Description

Calculate the adjusted r-squared value

Usage

```
adj.r.squared(r.sq, n)
```

Arguments

r.sq	The r-squared value
n	The number of points

Value

The numeric adjusted r-squared value

See Also

Other Half-life and elimination: [pk.calc.aucpext\(\)](#), [pk.calc.half.life\(\)](#), [pk.calc.thalf.eff\(\)](#)

any_sparse_dense_in_interval

Determine if there are any sparse or dense calculations requested within an interval

Description

A calculation is sparse when only sparse data can produce it, which is a property of the registration rather than a flag; see [add.interval.col\(\)](#).

Usage

```
any_sparse_dense_in_interval(interval, sparse)
```

Arguments

interval	An interval specification
sparse	Are the concentration-time data sparse PK (commonly used in small nonclinical species or with terminal or difficult sampling) or dense PK (commonly used in clinical studies or larger nonclinical species)?

Value

A logical value indicating if the interval requests any sparse (if sparse=TRUE) or dense (if sparse=FALSE) calculations.

as.data.frame.PKNCResults

Extract the parameter results from a PKNCResults and return them as a data.frame.

Description

Extract the parameter results from a PKNCResults and return them as a data.frame.

Usage

```
## S3 method for class 'PKNCResults'
as.data.frame(
  x,
  ...,
  out_format = c("long", "wide", "cdisc"),
  filter_requested = FALSE,
  filter_excluded = FALSE,
  grpid_cols = NULL,
```

```

    grpid_numeric = NULL,
    out.format = deprecated()
)

```

Arguments

x The object to extract results from

... Ignored (for compatibility with generic `as.data.frame()`)

out_format Should the output be 'long' (default), 'wide', or 'cdisc'? When 'cdisc', the original PKNCA parameter name is kept in a new `pknca_parameter` column (lowercase so it cannot be mistaken for an SDTM PP variable; drop it before submission), the `PPTTESTCD` column is translated to CDISC standard codes, and a `PPTTEST` column with the CDISC test name is added. The translation is many-to-one – several PKNCA parameters can resolve to the same `PPTTESTCD` (every `AUCint` variant to "AUCINT", for example) – so `pknca_parameter` is the only column that still identifies which PKNCA calculation produced a row. Route-dependent parameters (e.g. `CL`, `VZ`, `MRT`) are resolved using the route information from the dose data.

Each row also gets its time point reference, as SDTMIG 3.4 defines it: `PPSTINT` and `PPENINT` give the interval start and end as ISO 8601 durations relative to the reference named in `PPTPTREF`, and `PPRFTDTC` gives that reference's date-time. The reference is the dose that starts the interval: the last included dose at or before the interval start for that subject and group (`PPTPTREF` "LAST DOSE PRIOR TO INTERVAL"), so a steady-state dosing interval reads "PT0H" to "PT24H". A subject (or group) without an included dose uses its first concentration instead (`PPTPTREF` "FIRST OBSERVATION"), as the date-time references of `PKNCAdata()` do. The durations are in the preferred time unit (`timeu_pref`) when it is given; `PPENINT` is NA for an interval to infinity, and `PPSTINT`, `PPENINT`, and `PPTPTREF` are NA when the subject has doses but none at or before the interval start. `PPRFTDTC` is only given when the concentration and dose times were date-times (see `PKNCAdata()`), and it can differ between the intervals of one subject.

`PPGRPID`, the record group identifier, is built from the grouping columns named in `grpid_cols` and the number of the row's interval, as "<prefix1><value1>.<prefix2><value2>.I<n>" (for example "A.P1.I01", "P2.I01", or just "I01" without `grpid_cols`). Within each combination of the subject, the analyte, and the `grpid_cols` columns, the intervals are numbered from 1 by start and then end; the rows of one interval (one per parameter) share the number. Other grouping columns (a treatment or a matrix, for example) do not enter `PPGRPID`: they stay as their own output columns, and intervals with the same window in different values of them share a number. The number is written with at least two digits, and with as many as the largest interval number needs, so that the text sorts in time order. The numbers, the width, and the checks of the values use every row of the results, so `filter_requested` and `filter_excluded` never renumber an interval or hide a bad value; an interval with no rows in the results has no number. The grouping columns also remain in the output.

filter_requested Only return rows with parameters that were specifically requested?

filter_excluded	Should excluded values be removed?
grp_id_cols	For out_format = "cdisc", the grouping columns that prefix the interval number in PPGRPID: a named character vector of grouping columns of the PKNCAconc() object, in prefix order, other than the subject and the analyte. Each name is a column, and each value is the text written before that column's value, such as c(Part = "", Period = "P"). The text before a value and the values may not contain ".", and a value may not be empty, and distinct values of a column may not give the same text. NULL (the default) uses the grp_id_cols of the PKNCAdata() object, and character() gives an interval-only identifier regardless of that default.
grp_id_numeric	For out_format = "cdisc", the names of columns in grp_id_cols whose values are whole numbers of at least 1 (such as the period), which are written as that number, so "01" becomes 1. A value that is not a finite whole number of at least 1 is an error. NULL (the default) uses the grp_id_numeric of the PKNCAdata() object for the columns of grp_id_cols that it names, and character() means no columns regardless of that default.
out.format	Deprecated in favor of out_format

Value

A data.frame (or usually a tibble) of results

as_PKNCAconc	<i>Convert an object into a PKNCAconc object</i>
--------------	--

Description

Convert an object into a PKNCAconc object

Usage

```
as_PKNCAconc(x, ...)  
  
as_PKNCAdose(x, ...)  
  
as_PKNCAdata(x, ...)  
  
as_PKNCAresults(x, ...)
```

Arguments

x	The object to convert
...	Passed to subsequent methods

Value

A converted object

Functions

- `as_PKNCAdose()`: Convert an object into a PKNCAdose object
- `as_PKNCAdata()`: Convert an object into a PKNCAdata object
- `as_PKNCAresults()`: Convert an object into a PKNCAresults object

<code>as_sparse_pk</code>	<i>Generate a sparse_pk object</i>
---------------------------	------------------------------------

Description

Generate a sparse_pk object

Usage

```
as_sparse_pk(conc, time, subject)
```

Arguments

<code>conc</code>	Measured concentrations
<code>time</code>	Time of the measurement of the concentrations
<code>subject</code>	Subject identifiers (may be any class; may not be null). A missing subject marks an imputed concentration (such as the zero that the <code>start_conc0</code> imputation adds at the start of an interval): a time where every subject is missing has a known concentration, which enters sparse estimates but not their variance.

Value

A `sparse_pk` object which is a list of lists. The inner lists have elements named: "time", The time of measurement; "conc", The concentration measured; "subject", The subject identifiers; "imputed", Whether the concentration at that time is imputed (known). The object will usually be modified by future functions to add more named elements to the inner list.

See Also

Other Sparse Methods: [pk.calc.aucinf.obs_sparse\(\)](#), [pk.calc.aucivlast_sparse\(\)](#), [pk.calc.auclast_sparse\(\)](#), [pk.calc.sparse_auc\(\)](#), [pk.calc.sparse_aumc\(\)](#), [sparse_auc_weight_linear\(\)](#), [sparse_bootstrap\(\)](#), [sparse_mean\(\)](#), [summary.PKNCAresults_sparse_bootstrap\(\)](#)

assert_aucmethod	<i>Assert that a value is a valid AUC method</i>
------------------	--

Description

Assert that a value is a valid AUC method

Usage

```
assert_aucmethod(method = c("lin up/log down", "linear", "lin-log"))
```

Arguments

method	The method for integration (one of 'lin up/log down', 'lin-log', or 'linear')
--------	---

Value

method or an informative error

assert_conc	<i>Verify that concentration measurements are valid</i>
-------------	---

Description

If the concentrations or times are invalid, will provide an error. Reasons for being invalid are

- time is not a number
- conc is not a number
- Any time value is NA
- time is not monotonically increasing
- conc and time are not the same length

Usage

```
assert_conc(conc, any_missing_conc = TRUE)
```

```
assert_time(time, sorted_time = TRUE)
```

```
assert_conc_time(conc, time, any_missing_conc = TRUE, sorted_time = TRUE)
```

Arguments

conc	Measured concentrations
any_missing_conc	Are any concentration values allowed to be NA?
time	Time of the measurement of the concentrations
sorted_time	Must the time be unique and monotonically increasing?

Details

Some cases may generate warnings but allow the data to proceed.

- A negative concentration is often but not always an error; it will generate a warning.

Value

conc or give an informative error

time or give an informative error

NULL, invisibly, when the data are valid; otherwise an informative error

Examples

```
assert_conc_time(conc = c(0, 2, 1), time = 0:2)
# Unsorted times are allowed when `sorted_time = FALSE`
assert_conc_time(conc = c(0, 2, 1), time = c(2, 0, 1), sorted_time = FALSE)
```

assert_dosetau	<i>Assert that a value is a dosing interval</i>
----------------	---

Description

Assert that a value is a dosing interval

Usage

```
assert_dosetau(tau)
```

Arguments

tau	The dosing interval
-----	---------------------

Value

tau or an informative error

assert_impute_method	<i>Check an imputation specification</i>
----------------------	--

Description

The specification is resolved the way [PKNCAdata\(\)](#) and [pk.nca\(\)](#) resolve it: the name of a column in intervals, NA to use the "impute" column of intervals when there is one, or otherwise a string of imputation methods. Every method named must exist.

Usage

```
assert_impute_method(impute, intervals = data.frame())
```

Arguments

impute	The imputation specification (a character scalar or NA)
intervals	The intervals data.frame (if the specification may name one of its columns)

Value

The resolved imputation strings (one per interval when read from a column), invisibly, or an error

See Also

[PKNCA_impute_fun_list\(\)](#), [PKNCA_impute_method](#)

Other Imputation: [PKNCA_impute_fun_list\(\)](#), [PKNCA_impute_method](#), [get_impute_column\(\)](#), [get_impute_method\(\)](#), [pknca_impute_methods\(\)](#)

Examples

```
assert_impute_method("start_predose,start_conc0")
assert_impute_method(
  "method",
  intervals = data.frame(start = 0, end = 24, method = "start_conc0")
)
try(assert_impute_method("start_misspelled"))
```

assert_intervals	<i>Assert Intervals</i>
------------------	-------------------------

Description

Verifies that an interval definition is valid for a PKNCAdata object. Valid means that intervals are a data.frame (or data.frame-like object), that the column names are either the groupings of the PKNCAconc part of the PKNCAdata object or that they are one of the NCA parameters allowed (i.e. names(get.interval.cols())). The column that the impute setting of the PKNCAdata object names (see [get_impute_method\(\)](#)) is also allowed, and it must be a character column, or a pknca_error_interval_impute_not_character error is raised. An interval_id identifies one interval: rows that share it may differ only in the parameters they request (and impute), not in start, end, or the groups, or a pknca_error_secondary_id_conflict error is raised. It will return the intervals argument unchanged, or it will raise an error.

Usage

```
assert_intervals(intervals, data)
```

Arguments

intervals	Proposed intervals
data	PKNCAdata object

Value

The intervals argument unchanged, or it will raise an error.

assert_intervaltime_single	<i>Assert that an interval is accurately defined as an interval, and return the interval</i>
----------------------------	--

Description

Assert that an interval is accurately defined as an interval, and return the interval

Usage

```
assert_intervaltime_single(interval = NULL, start = NULL, end = NULL)
```

Arguments

interval	Numeric vector of two numbers for the start and end time of integration
start	The start time of the interval
end	The end time of the interval

Value

interval (or c(start, end))

assert_lambdaz	<i>Assert that a lambda.z value is valid</i>
----------------	--

Description

Assert that a lambda.z value is valid

Usage

```
assert_lambdaz(
  lambda.z,
  any.missing = TRUE,
  .var.name = checkmate::vname(lambda.z)
)
```

Arguments

lambda.z	The elimination rate (in units of inverse time) for extrapolation
any.missing	[logical(1)] Are vectors with missing values allowed? Default is TRUE.
.var.name	[character(1)] Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname .

Value

lambda.z or an informative error

assert_number_between	<i>Confirm that a value is greater than another value</i>
-----------------------	---

Description

Confirm that a value is greater than another value

Usage

```
assert_number_between(
  x,
  ...,
  na.ok = FALSE,
  len = 1,
  .var.name = checkmate::vname(x)
)
```

Arguments

x	[any] Object to check.
...	Passed to assert_numeric_between()
na.ok	[logical(1)] Are missing values allowed? Default is FALSE.
len	Ignored (must be 1)
.var.name	[character(1)] Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname .

Value

x or an informative error

assert_numeric_between

Confirm that a value is greater than another value

Description

Confirm that a value is greater than another value

Usage

```
assert_numeric_between(
  x,
  any.missing = FALSE,
  null.ok = FALSE,
  lower_eq = -Inf,
  lower = -Inf,
  upper = Inf,
  upper_eq = Inf,
  ...,
  .var.name = checkmate::vname(x)
)
```

Arguments

x	[any] Object to check.
any.missing	[logical(1)] Are vectors with missing values allowed? Default is TRUE.
null.ok	[logical(1)] If set to TRUE, x may also be NULL. In this case only a type check of x is performed, all additional checks are disabled.

lower_eq, upper_eq	Values where equality is not allowed
lower	[numeric(1)] Lower value all elements of x must be greater than or equal to.
upper	[numeric(1)] Upper value all elements of x must be lower than or equal to.
...	Passed to <code>checkmate::assert_numeric()</code>
.var.name	[character(1)] Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname .

Value

x

assert_param_name	<i>Assert that a character vector only contains PKNCA parameter names</i>
-------------------	---

Description

Assert that a character vector only contains PKNCA parameter names

Usage

```
assert_param_name(param)
```

Arguments

param	A vector of parameter names to check
-------	--------------------------------------

Value

param or give an informative error

assert_PKNCAdata	<i>Assert that an object is a PKNCAdata object</i>
------------------	--

Description

Assert that an object is a PKNCAdata object

Usage

```
assert_PKNCAdata(object)
```

```
assert_PKNCAresults(object)
```

```
assert_PKNCAconc(object)
```

```
assert_PKNCAdose(object)
```

Arguments

object	The PKNCAdose object
--------	----------------------

Value

The object

Functions

- `assert_PKNCAresults()`: Assert that an object is a PKNCAresults object
- `assert_PKNCAconc()`: Assert that an object is a PKNCAconc object
- `assert_PKNCAdose()`: Assert that an object is a PKNCAdose object

assert_unit_col	<i>Assert that a value may either be a column name in the data (first) or a single unit value (second)</i>
-----------------	--

Description

Assert that a value may either be a column name in the data (first) or a single unit value (second)

Usage

```
assert_unit_col(unit, data)
```

```
assert_unit_value(unit)
```

```
assert_unit(unit, data)
```

Arguments

unit	The column name or unit value
data	The data.frame that contains a column named unit

Value

unit with an attribute of "unit_type" that is either "column" or "value", or NULL if is.null(unit)

Functions

- `assert_unit_col()`: Assert that a column name contains a character string (that could be a unit specification)
- `assert_unit_value()`: Assert that a value may be a single unit
The function does not verify that it is a real unit like "ng/mL" only that it is a single character string.

auc_integrate	<i>Support function for AUC integration</i>
---------------	---

Description

Support function for AUC integration

Usage

```
auc_integrate(
  conc,
  time,
  clast,
  tlast,
  lambda.z,
  interval_method,
  fun_linear,
  fun_log,
  fun_inf
)
```

Arguments

conc	Measured concentrations
time	Time of the measurement of the concentrations
clast	The last concentration above the limit of quantification
tlast	Time of last concentration above the limit of quantification (will be calculated, if not provided)
lambda.z	The elimination rate (in units of inverse time) for extrapolation

interval_method	The method for integrating each interval of conc
fun_linear	The function to use for integration of the linear part of the curve (not required for AUC or AUMC functions)
fun_log	The function to use for integration of the logarithmic part of the curve (if log integration is used; not required for AUC or AUMC functions)
fun_inf	The function to use for extrapolation from the final measurement to infinite time (not required for AUC or AUMC functions).

be_assess	<i>Assess bioequivalence against a regulatory framework</i>
-----------	---

Description

be_assess() performs a complete bioequivalence (BE) assessment of one or more noncompartmental endpoints against a regulatory framework, including reference scaling and the pass/fail decision. It accepts the results of `pk.nca()` directly (a PKNCAResults object) or a tidy long data.frame, so the same workflow that produces NCA parameters flows into the regulatory decision. It is a thin wrapper over `be_fit_models()` that adds the be_assess class and its print/summary methods.

Usage

```
be_assess(object, ...)

## Default S3 method:
be_assess(
  object,
  reference_col,
  reference_value,
  endpoints = c("cmax", "aucinf.obs", "aucinf.pred", "auclast"),
  regulator = "ABE",
  model_type = NULL,
  alpha = 0.1,
  subject = NULL,
  sequence = NULL,
  period = NULL,
  design = NULL,
  covariates = NULL,
  heteroscedastic = FALSE,
  ...
)

## S3 method for class 'PKNCAResults_sparse_bootstrap'
be_assess(
  object,
  reference_col,
```

```

reference_value,
endpoints = c("cmax", "aucinf.obs", "aucinf.pred", "auclast"),
regulator = "ABE",
model_type = NULL,
alpha = 0.1,
subject = NULL,
sequence = NULL,
period = NULL,
design = NULL,
covariates = NULL,
heteroscedastic = FALSE,
...
)

```

Arguments

object	A PKNCAResults object or a tidy long data.frame with a PPTTESTCD column of parameter names, a PPORRES/PPSTRES column of values, and subject/sequence/period/treatment columns. The results of a sparse bootstrap (from a PKNCAconc object made by sparse_bootstrap()) are compared with the percentile intervals of the bootstrap replicates (Shen and Machado 2017); see Details.
...	Arguments passed to the methods
reference_col	The column identifying the formulation/treatment.
reference_value	The value of reference_col that is the reference formulation.
endpoints	Character vector of NCA parameters (matched against PPTTESTCD) to assess.
regulator	The regulatory framework (see be_regulator()); one of "ABE", "EMA", "HC", "GCC", "FDA", "NTID", "HVNTID", or "descriptive" (no acceptance limits or pass/fail).
model_type	The model for the average-BE point estimate, one of "lmer" (mixed model, for crossover/replicate designs), "anova" (fixed-effects, for parallel designs), "isc" (intra-subject contrasts, the FDA reference-scaled path), "nlme" (mixed model with treatment-specific residual variances, for crossover/replicate designs), or "gls" (generalized least squares with treatment-specific residual variances, for parallel designs). When NULL (default) it is chosen from the design, the regulator, and heteroscedastic.
alpha	The significance level; the confidence interval has level 1 - alpha (default 0.10 gives the 90% interval).
subject, sequence, period	Column names for the subject, randomization sequence, and period. When NULL they are taken from the PKNCAResults object or detected from common column names. sequence may be absent.
design	An optional be_design() object; computed from the data when NULL.
covariates	An optional character vector of column names added to every model as additive fixed effects (numeric columns as linear terms, character or factor columns as factors). They must not be missing in any analyzed row. For a PKNCAResults

object they must be columns of `as.data.frame(object)`, which means grouping columns. The within-subject variances used for reference scaling and the intra-subject contrasts do not use covariates.

`heteroscedastic`

Logical. When TRUE, estimate a separate residual variance for each treatment (see Details).

Details

The model type is selected automatically from the design and the regulator. By design, a parallel study (one measurement per subject) uses a fixed-effects ANOVA (`"anova"`), while a crossover or replicate study (repeated measures per subject) uses the mixed model (`"lmer"`) for the geometric mean ratio and its confidence interval. The FDA reference-scaled frameworks (FDA RSABE, NTID, HVNTID) always use intra-subject contrasts (`"isc"`), as the guidances specify, regardless of design. Pass `model_type` to override (`"lmer"`, `"anova"`, `"isc"`, `"nlme"`, or `"gls"`). Within-subject variability uses the regulatory ANOVA estimator for the `"lmer"/"anova"/"isc"` paths and the treatment-specific mixed-model estimator for `"nlme"`.

With `heteroscedastic = TRUE` the residual variance is estimated separately for each treatment: a crossover or replicate study uses `nlme::lme()` with `varIdent(~ 1 | treatment)` (`"nlme"`), and a parallel study uses `nlme::gls()` with the same variance structure (`"gls"`). `lme4::lmer()` cannot estimate treatment-specific residual variances (its weights are fixed prior weights), so `heteroscedastic = TRUE` with `model_type = "lmer"` is an error. Without reference scaling, `"nlme"` accepts non-replicated designs and several test formulations; with reference scaling it needs a fully replicated design and one test formulation, because the per-formulation variances are then used as the within-subject variances.

`regulator = "descriptive"` estimates the geometric means, their ratio, and its confidence interval exactly as for `"ABE"` but applies no acceptance limits and reports no pass/fail, for comparisons such as food effect and drug-drug interaction studies.

For the results of a sparse bootstrap (Shen and Machado 2017; see [sparse_bootstrap\(\)](#)), there is one estimate per treatment, from the sparse data, rather than one per subject. The ratio of the test and reference estimates is the point estimate, and its confidence interval is the percentile interval of the ratios of the bootstrap replicates (`model_type = "bootstrap"`). The `gm_*` columns hold the estimates of each treatment with their percentile intervals. A parallel design resamples the treatments independently; for a crossover, give the treatment column as paired to [sparse_bootstrap\(\)](#). There is no within-subject variance, so only the `"ABE"` and `"descriptive"` frameworks apply, and the results may have no groups other than `reference_col` (filter them first).

Value

An object of class `be_assess` (a `data.frame`), ordered by endpoint (in the order requested) then by test formulation (reference-first), with one row per endpoint and test formulation and the columns `endpoint`, `test`, `n`, `design`, `units` (the endpoint's measurement units; the column is omitted with a warning when units are not provided), the reference and test geometric means with their 90% confidence intervals on the measurement scale (`gm_reference`, `gm_reference_lower`, `gm_reference_upper`, `gm_test`, `gm_test_lower`, `gm_test_upper`), `gmr_percent`, `ci_lower`, `ci_upper`, `cvwr_percent`, `cvwt_percent`, `swr`, `limit_lower`, `limit_upper`, `criterion`, `regulator`, `model_type`, and `pass`. `limit_*` are NA for the RSABE criterion and `criterion` is NA for the limit-based frameworks. For `regulator = "descriptive"` the `limit_lower`, `limit_upper`, `criterion`, and `pass` columns are

omitted. A caption attribute documents the model and the decision rule, or states that no regulatory decision was applied.

Methods (by class)

- `be_assess(PKNCAResults_sparse_bootstrap)`: Bioequivalence from the percentile intervals of a sparse bootstrap (Shen and Machado 2017)

References

Shen M, Machado SG. Bioequivalence evaluation of sparse sampling pharmacokinetics data using bootstrap resampling method. *Journal of Biopharmaceutical Statistics*. 2017;27(2):257-264. doi:10.1080/10543406.2016.1265543

See Also

`be_compare()` to assess one dataset under several frameworks, `be_within_var()`, and `be_regulator()`.

Other Bioequivalence: `be_compare()`, `be_dataset()`, `be_design()`, `be_expand_limits()`, `be_extract_param()`, `be_fit_model_single()`, `be_fit_models()`, `be_regulator()`, `be_table()`, `be_within_var()`

Examples

```
# A replicate-design crossover in long (PTESTCD/PPORRES) format
set.seed(1)
nsub <- 24
seqs <- rep(c("TRTR", "RTRT"), length.out = nsub)
b <- stats::rnorm(nsub, sd = 0.4)
d <- do.call(rbind, lapply(seq_len(nsub), function(i) {
  trt <- strsplit(seqs[i], "")[[1]]
  data.frame(
    subject = i, sequence = seqs[i], period = seq_along(trt), treatment = trt,
    PTESTCD = "auclast",
    PPORRES = exp(log(100) + ifelse(trt == "T", 0.04, 0) + b[i] +
      stats::rnorm(length(trt), sd = 0.45))
  )
}))
be_assess(d, reference_col = "treatment", reference_value = "R",
  endpoints = "auclast", regulator = "EMA")
```

be_compare

Compare a bioequivalence dataset across regulatory frameworks

Description

`be_compare()` runs `be_assess()` under several regulatory frameworks and stacks the results, so the same study can be judged side by side under the different reference-scaling rules. Frameworks that the design does not support (for example NTID on a partial replicate) are skipped with a warning. Rows for the "descriptive" framework have NA in the decision columns.

Usage

```
be_compare(
  object,
  reference_col,
  reference_value,
  endpoints = c("cmax", "aucinf.obs", "aucinf.pred", "auclast"),
  regulators = c("ABE", "EMA", "HC", "GCC", "FDA"),
  model_type = NULL,
  alpha = 0.1,
  subject = NULL,
  sequence = NULL,
  period = NULL,
  design = NULL,
  covariates = NULL,
  heteroscedastic = FALSE
)
```

Arguments

object	A PKNCAResults object or a tidy long data.frame with a PPTTESTCD column of parameter names, a PPORRES/PPSTRES column of values, and subject/sequence/period/treatment columns. The results of a sparse bootstrap (from a PKNCAResults object made by sparse_bootstrap()) are compared with the percentile intervals of the bootstrap replicates (Shen and Machado 2017); see Details.
reference_col	The column identifying the formulation/treatment.
reference_value	The value of reference_col that is the reference formulation.
endpoints	Character vector of NCA parameters (matched against PPTTESTCD) to assess.
regulators	A character vector of regulatory frameworks to compare (see be_regulator()).
model_type	The model for the average-BE point estimate, one of "lmer" (mixed model, for crossover/replicate designs), "anova" (fixed-effects, for parallel designs), "isc" (intra-subject contrasts, the FDA reference-scaled path), "nlme" (mixed model with treatment-specific residual variances, for crossover/replicate designs), or "gls" (generalized least squares with treatment-specific residual variances, for parallel designs). When NULL (default) it is chosen from the design, the regulator, and heteroscedastic.
alpha	The significance level; the confidence interval has level 1 - alpha (default 0.10 gives the 90% interval).
subject, sequence, period	Column names for the subject, randomization sequence, and period. When NULL they are taken from the PKNCAResults object or detected from common column names. sequence may be absent.
design	An optional be_design() object; computed from the data when NULL.
covariates	An optional character vector of column names added to every model as additive fixed effects (numeric columns as linear terms, character or factor columns as factors). They must not be missing in any analyzed row. For a PKNCAResults

object they must be columns of `as.data.frame(object)`, which means grouping columns. The within-subject variances used for reference scaling and the intra-subject contrasts do not use covariates.

heteroscedastic

Logical. When TRUE, estimate a separate residual variance for each treatment (see Details).

Value

An object of class `be_compare` (a `data.frame`) with the `be_assess()` columns for every endpoint, test formulation, and regulator.

See Also

`be_assess()`

Other Bioequivalence: `be_assess()`, `be_dataset()`, `be_design()`, `be_expand_limits()`, `be_extract_param()`, `be_fit_model_single()`, `be_fit_models()`, `be_regulator()`, `be_table()`, `be_within_var()`

Examples

```
set.seed(1)
nsub <- 24
seqs <- rep(c("TRTR", "RTRT"), length.out = nsub)
b <- stats::rnorm(nsub, sd = 0.4)
d <- do.call(rbind, lapply(seq_len(nsub), function(i) {
  trt <- strsplit(seqs[i], "")[[1]]
  data.frame(
    subject = i, sequence = seqs[i], period = seq_along(trt), treatment = trt,
    PPTTESTCD = "auclast",
    PPORRES = exp(log(100) + ifelse(trt == "T", 0.04, 0) + b[i] +
      stats::rnorm(length(trt), sd = 0.45))
  )
}))
be_compare(d, reference_col = "treatment", reference_value = "R",
  endpoints = "auclast", regulators = c("EMA", "HC", "GCC", "FDA"))
```

be_dataset

Build and validate a bioequivalence dataset

Description

`be_dataset()` prepares a noncompartmental result for bioequivalence calculation: it accepts a `PKNCAResults` object or a tidy long `data.frame`, resolves the value column, drops excluded/invalid rows, detects the subject/sequence/period columns, validates the reference, and sets the reference formulation as the first factor level. It standardizes the modeling columns (`.subject`, `.sequence`, `.period`, `.trt`, `.logval`) used by the downstream fitters.

Usage

```
be_dataset(object, ...)

## Default S3 method:
be_dataset(
  object,
  reference_col,
  reference_value,
  endpoints = c("cmax", "aucinf.obs", "aucinf.pred", "auclast"),
  subject = NULL,
  sequence = NULL,
  period = NULL,
  covariates = NULL,
  ...
)

## S3 method for class 'PKNCAResults_sparse_bootstrap'
be_dataset(
  object,
  reference_col,
  reference_value,
  endpoints = c("cmax", "aucinf.obs", "aucinf.pred", "auclast"),
  subject = NULL,
  sequence = NULL,
  period = NULL,
  covariates = NULL,
  ...
)
```

Arguments

object	A PKNCAResults object or a tidy long data.frame with a PPTTESTCD column of parameter names, a PPORRES/PPSTRES column of values, and subject/sequence/period/treatment columns. The results of a sparse bootstrap (from a PKNCAResults object made by sparse_bootstrap()) are compared with the percentile intervals of the bootstrap replicates (Shen and Machado 2017); see Details.
...	Arguments passed to the methods
reference_col	The column identifying the formulation/treatment.
reference_value	The value of reference_col that is the reference formulation.
endpoints	Character vector of NCA parameters (matched against PPTTESTCD) to assess.
subject, sequence, period	Column names for the subject, randomization sequence, and period. When NULL they are taken from the PKNCAResults object or detected from common column names. sequence may be absent.
covariates	An optional character vector of column names added to every model as additive fixed effects (numeric columns as linear terms, character or factor columns as

factors). They must not be missing in any analyzed row. For a `PKNCAResults` object they must be columns of `as.data.frame(object)`, which means grouping columns. The within-subject variances used for reference scaling and the intra-subject contrasts do not use covariates.

Details

The endpoint value is taken from the `PPSTRES` column when present, otherwise `PPORRES`. The measurement units are read from the matching units column – `PPSTRESU` for `PPSTRES`, or `PPORRESU` for `PPORRES` – which a `PKNCAResults` object provides automatically; a plain `data.frame` supplies units the same way by including the corresponding `PPSTRESU/PPORRESU` column. When no units column is present, units are unavailable and the `units` column is omitted from the assessment table.

Each covariate is copied to a standardized column named `.cov_<name>` (character columns become factors). A covariate may not be one of the subject, sequence, period, treatment, or value columns, and it may not be missing in any row that is analyzed.

Value

An object of class `be_dataset`: a list with `data` (the standardized long frame, including a `.units` column and one `.cov_<name>` column per covariate), `columns` (the resolved column names, including units and covariates), `reference_value`, `test_levels`, and `endpoints` (those present). For the results of a sparse bootstrap (see [sparse_bootstrap\(\)](#)), `data` has one row per bootstrap replicate, group, and endpoint instead of one per subject and period.

Methods (by class)

- `be_dataset(PKNCAResults_sparse_bootstrap)`: The replicates of a sparse bootstrap, one row per replicate, treatment, and endpoint

See Also

Other Bioequivalence: [be_assess\(\)](#), [be_compare\(\)](#), [be_design\(\)](#), [be_expand_limits\(\)](#), [be_extract_param\(\)](#), [be_fit_model_single\(\)](#), [be_fit_models\(\)](#), [be_regulator\(\)](#), [be_table\(\)](#), [be_within_var\(\)](#)

be_design	<i>Classify a bioequivalence crossover design</i>
-----------	---

Description

`be_design()` inspects the treatment-by-period pattern of a crossover study and reports the design type, the replication of the reference and test formulations, and which regulatory frameworks are feasible. Reference scaling (EMA/HC/GCC/FDA) requires the reference to be replicated; the narrow therapeutic index frameworks additionally require the test to be replicated.

Usage

`be_design(data, subject, sequence, period, treatment, reference_value)`

Arguments

data	A long data.frame with one row per concentration-derived observation (subject, period, treatment).
subject, sequence, period, treatment	Column names (length-1 character) identifying the subject, randomization sequence, period, and treatment. sequence may be NA to derive it from the treatment-by-period pattern.
reference_value	The value of treatment that is the reference formulation.

Value

An object of class `be_design`: a list with elements `design` (one of "parallel", "2x2x2", "full_replicate", "partial_replicate", "other"), `n_sequences`, `n_periods`, `n_treatments`, `n_subjects`, `sequences`, `treatments`, `reference`, `replicate_reference`, `replicate_test`, `reps_reference`, `reps_test`, `balanced`, and `feasible` (a named logical vector for `abe`, `abel`, `rsabe`, `ntid`, `hvtid`). `reps_reference` is the median, over subjects, of the number of times a subject received the reference. `reps_test` is the same median computed separately for each test formulation; with several test formulations it is the smallest of those medians, so `replicate_test` is TRUE only when every test formulation is replicated.

See Also

[be_assess\(\)](#)

Other Bioequivalence: [be_assess\(\)](#), [be_compare\(\)](#), [be_dataset\(\)](#), [be_expand_limits\(\)](#), [be_extract_param\(\)](#), [be_fit_model_single\(\)](#), [be_fit_models\(\)](#), [be_regulator\(\)](#), [be_table\(\)](#), [be_within_var\(\)](#)

Examples

```
d <- data.frame(
  subject = rep(1:4, each = 4),
  period = rep(1:4, times = 4),
  sequence = rep(c("TRTR", "RTRT"), each = 8),
  treatment = c("T", "R", "T", "R", "R", "T", "R", "T",
                "T", "R", "T", "R", "R", "T", "R", "T")
)
be_design(d, "subject", "sequence", "period", "treatment", reference_value = "R")
```

be_expand_limits

Reference-scaled acceptance limits for bioequivalence

Description

`be_expand_limits()` computes the (possibly widened) bioequivalence acceptance limits for a scaled-ABE framework given the within-reference standard deviation. For EMA and Health Canada the limits widen geometrically with `swR` and are capped; for the GCC they jump to a fixed wider band; below the switching variability they remain the conventional 80.00-125.00%.

Usage

```
be_expand_limits(swR, regulator)
```

Arguments

swR	The within-reference standard deviation of the log-transformed endpoint (for example from <code>be_within_var()</code>).
regulator	A regulator name (see <code>be_regulator()</code>) or a <code>be_regulator</code> object. Only the ABEL frameworks ("EMA", "HC", "GCC") widen; "ABE" always returns 80.00-125.00%, and "descriptive" (which has no limits) is an error.

Value

A named numeric vector `c(lower, upper)` of acceptance limits as percentages.

See Also

`be_regulator()`, `be_assess()`

Other Bioequivalence: `be_assess()`, `be_compare()`, `be_dataset()`, `be_design()`, `be_extract_param()`, `be_fit_model_single()`, `be_fit_models()`, `be_regulator()`, `be_table()`, `be_within_var()`

Examples

```
be_expand_limits(0.485, "EMA") # capped expansion
be_expand_limits(0.20, "EMA")  # below the switch: 80-125%
be_expand_limits(0.40, "GCC")  # fixed GCC band
```

be_extract_param	<i>Extract bioequivalence parameters from a fitted model</i>
------------------	--

Description

`be_extract_param()` turns a `be_fit_model_single()` fit into the parameters the regulatory decision needs. It always computes the intra-subject-contrast (ISC) point estimate, then dispatches on the model type to extract the model-based geometric means, the geometric mean ratio and its confidence interval, and the within-formulation standard deviations.

Usage

```
be_extract_param(fit, ds_ep, alpha = 0.1)
```

Arguments

fit	A <code>be_fit</code> object from <code>be_fit_model_single()</code> .
ds_ep	The standardized single-endpoint data.frame from <code>be_dataset()</code> .
alpha	The significance level; the confidence interval has level $1 - \alpha$.

Value

A data.frame with one row per test formulation, carrying the model and ISC point estimates/intervals, geometric means, and within-formulation variances.

See Also

Other Bioequivalence: [be_assess\(\)](#), [be_compare\(\)](#), [be_dataset\(\)](#), [be_design\(\)](#), [be_expand_limits\(\)](#), [be_fit_model_single\(\)](#), [be_fit_models\(\)](#), [be_regulator\(\)](#), [be_table\(\)](#), [be_within_var\(\)](#)

be_fit_model_single	<i>Fit the bioequivalence model(s) for one endpoint</i>
---------------------	---

Description

`be_fit_model_single()` fits the average-BE model for a single endpoint and dispatches on `model_type` to `be_fit_model_lmer()`, `be_fit_model_nlme()`, `be_fit_model_gls()`, or `be_fit_model_anova()`. This is the only place model fitting happens. For the "lmer", "anova", and "isc" types the within-formulation ANOVA variances are also fit here; for "nlme" they come from the single mixed model. Covariate columns carried by [be_dataset\(\)](#) (`.cov_<name>`) enter every model as additive fixed effects.

Usage

```
be_fit_model_single(
  ds_ep,
  model_type = c("lmer", "nlme", "anova", "isc", "glms"),
  scaling = TRUE
)
```

Arguments

<code>ds_ep</code>	The standardized single-endpoint data.frame from be_dataset() (<code>be_dataset(...)\$data</code> filtered to one endpoint).
<code>model_type</code>	One of "lmer", "nlme", "anova", "isc" (the intra-subject-contrast path, fit like "anova"), or "glms" (generalized least squares with treatment-specific residual variances, for parallel designs).
<code>scaling</code>	Logical; whether reference scaling is needed (controls whether the within-formulation variances are estimated).

Value

An object of class `be_fit`: a list with `model_type`, the fitted model, and (for lmer/anova/isc) `ref_var/test_var` within-formulation ANOVA variances. The endpoint's measurement units are attached as a `units` attribute (NULL when not provided).

See Also

Other Bioequivalence: [be_assess\(\)](#), [be_compare\(\)](#), [be_dataset\(\)](#), [be_design\(\)](#), [be_expand_limits\(\)](#), [be_extract_param\(\)](#), [be_fit_models\(\)](#), [be_regulator\(\)](#), [be_table\(\)](#), [be_within_var\(\)](#)

be_fit_models

Coordinate the bioequivalence calculation path

Description

`be_fit_models()` runs the single bioequivalence calculation path and returns the regulatory pass/fail table. It coordinates the stages in order: [be_dataset\(\)](#) to prepare the data, [be_design\(\)](#) to classify the design and choose the model, [be_fit_model_single\(\)](#) to fit, [be_extract_param\(\)](#) to extract the parameters, and [be_table\(\)](#) to apply the regulatory decision. [be_assess\(\)](#) and [be_compare\(\)](#) are thin verbs over it.

Usage

```
be_fit_models(
  object,
  reference_col,
  reference_value,
  endpoints = c("cmax", "aucinf.obs", "aucinf.pred", "auclast"),
  regulator = "ABE",
  model_type = NULL,
  alpha = 0.1,
  subject = NULL,
  sequence = NULL,
  period = NULL,
  design = NULL,
  covariates = NULL,
  heteroscedastic = FALSE
)
```

Arguments

object	A PKNCAresults object or a tidy long data.frame with a PPTTESTCD column of parameter names, a PPORRES/PPSTRES column of values, and subject/sequence/period/treatment columns. The results of a sparse bootstrap (from a PKNCAconc object made by sparse_bootstrap()) are compared with the percentile intervals of the bootstrap replicates (Shen and Machado 2017); see Details.
reference_col	The column identifying the formulation/treatment.
reference_value	The value of reference_col that is the reference formulation.
endpoints	Character vector of NCA parameters (matched against PPTTESTCD) to assess.

regulator	The regulatory framework (see be_regulator()); one of "ABE", "EMA", "HC", "GCC", "FDA", "NTID", "HVNTID", or "descriptive" (no acceptance limits or pass/fail).
model_type	The model for the average-BE point estimate, one of "lmer" (mixed model, for crossover/replicate designs), "anova" (fixed-effects, for parallel designs), "isc" (intra-subject contrasts, the FDA reference-scaled path), "nlme" (mixed model with treatment-specific residual variances, for crossover/replicate designs), or "gls" (generalized least squares with treatment-specific residual variances, for parallel designs). When NULL (default) it is chosen from the design, the regulator, and heteroscedastic.
alpha	The significance level; the confidence interval has level 1 - alpha (default 0.10 gives the 90% interval).
subject, sequence, period	Column names for the subject, randomization sequence, and period. When NULL they are taken from the PKNCAResults object or detected from common column names. sequence may be absent.
design	An optional be_design() object; computed from the data when NULL.
covariates	An optional character vector of column names added to every model as additive fixed effects (numeric columns as linear terms, character or factor columns as factors). They must not be missing in any analyzed row. For a PKNCAResults object they must be columns of <code>as.data.frame(object)</code> , which means grouping columns. The within-subject variances used for reference scaling and the intra-subject contrasts do not use covariates.
heteroscedastic	Logical. When TRUE, estimate a separate residual variance for each treatment (see Details).

Value

A data.frame with one row per endpoint and test formulation (the columns described in [be_assess\(\)](#)).

See Also

Other Bioequivalence: [be_assess\(\)](#), [be_compare\(\)](#), [be_dataset\(\)](#), [be_design\(\)](#), [be_expand_limits\(\)](#), [be_extract_param\(\)](#), [be_fit_model_single\(\)](#), [be_regulator\(\)](#), [be_table\(\)](#), [be_within_var\(\)](#)

be_regulator

Regulatory framework constants for bioequivalence assessment

Description

`be_regulator()` returns the scaling type, switching and capping rules, and regulatory constants that define a bioequivalence (BE) assessment framework. It is the single source of truth for these values: [be_assess\(\)](#) and the internal decision functions pull every constant from here rather than hard-coding it.

Usage

```
be_regulator(
  name = c("ABE", "EMA", "HC", "GCC", "FDA", "NTID", "HVNTID", "descriptive")
)
```

Arguments

name The regulatory framework, one of "ABE", "EMA", "HC", "GCC", "FDA", "NTID", "HVNTID", or "descriptive".

Details

The supported frameworks and their constants are:

- **ABE** – average bioequivalence; the universal unscaled criterion that the 90% confidence interval fall within 80.00-125.00%. No reference scaling.
- **EMA** – average bioequivalence with expanding limits (ABEL). Scaling begins when the within-reference coefficient of variation (CVwR) exceeds 30% ($\text{cvswitch} = 0.30$); the limits widen as $\exp(\pm k * \text{swR})$ with the regulatory constant $k = 0.76$ and the expansion is capped at $\text{CVwR} = 50\%$ (limits 69.84-143.19%). The constant 0.76 is $\log(1.25) / \sqrt{\log(1 + 0.30^2)}$ rounded to two decimals, as fixed by the EMA bioequivalence guideline.
- **HC** – Health Canada; ABEL with the same $k = 0.76$ but the expansion is capped at $\text{CVwR} = 57.382\%$ (upper limit 150%).
- **GCC** – Gulf Cooperation Council; fixed widened limits of 75.00-133.33% whenever $\text{CVwR} > 30\%$ (not CV-dependent).
- **FDA** – reference-scaled average bioequivalence (RSABE). The linearized (Howe/Hyslop) criterion uses $r_{\text{const}} = \log(1.25) / 0.25 = 0.8926$ ($\theta = r_{\text{const}}^2$) and switches from unscaled ABE when $\text{swR} \geq 0.294$ (CVwR of about 30%).
- **NTID** – FDA narrow therapeutic index drugs; uses $r_{\text{const}} = -\log(0.9) / 0.1 = 1.0536$ and always applies the scaled criterion (a fully replicated design is required).
- **HVNTID** – FDA highly variable narrow therapeutic index drugs; the RSABE-style scaled criterion with the NTID constant and the within-subject standard deviation ratio constraint.
- **descriptive** – no regulatory decision. The geometric means, their ratio, and its confidence interval are estimated exactly as for ABE, but no acceptance limits are applied and no pass/fail is reported. Use it for comparisons that are estimated rather than judged, such as food effect and drug-drug interaction studies.

All decision frameworks additionally impose the point-estimate constraint that the geometric mean ratio fall within 80.00-125.00%.

Value

An object of class `be_regulator`: a list with elements `name`, `scaling` (one of "none", "abel", "rsabe", "ntid", "hvntid"), `cvswitch`, `r_const`, `cvcap`, `switch_swr`, `pe_constr`, `est_method` ("anova" or "isc"), `decision` (FALSE only for "descriptive"), and `switch_basis`.

See Also

[be_expand_limits\(\)](#) for the ABEL acceptance limits and [be_assess\(\)](#) for the full assessment.
Other Bioequivalence: [be_assess\(\)](#), [be_compare\(\)](#), [be_dataset\(\)](#), [be_design\(\)](#), [be_expand_limits\(\)](#), [be_extract_param\(\)](#), [be_fit_model_single\(\)](#), [be_fit_models\(\)](#), [be_table\(\)](#), [be_within_var\(\)](#)

Examples

```
be_regulator("EMA")
be_regulator("FDA")
be_regulator("descriptive")
```

be_table	<i>Build the regulatory bioequivalence table</i>
----------	--

Description

`be_table()` takes the parameters from [be_extract_param\(\)](#) and a regulatory framework and produces the final per-endpoint pass/fail table. It selects the regulator-appropriate statistic (the model contrast for the expanding-limits frameworks, intra-subject contrasts for the FDA family) and calls the per-framework decision functions; it is the only place regulator decisions are made.

Usage

```
be_table(
  params,
  regulator,
  alpha = 0.1,
  design = NA_character_,
  model_type = NA_character_
)
```

Arguments

params	A data.frame of parameters from be_extract_param() (with an endpoint column added per endpoint).
regulator	A regulator name or a <code>be_regulator</code> object.
alpha	The significance level.
design	The design label (character) for the design column.
model_type	The model type label for the <code>model_type</code> column.

Value

A data.frame with one row per endpoint and test formulation and the pass/fail decision columns. For the "descriptive" framework the decision columns (`limit_lower`, `limit_upper`, `criterion`, and `pass`) are omitted.

See Also

Other Bioequivalence: [be_assess\(\)](#), [be_compare\(\)](#), [be_dataset\(\)](#), [be_design\(\)](#), [be_expand_limits\(\)](#), [be_extract_param\(\)](#), [be_fit_model_single\(\)](#), [be_fit_models\(\)](#), [be_regulator\(\)](#), [be_within_var\(\)](#)

be_within_var

Within-subject variability for reference-scaled bioequivalence

Description

`be_within_var()` estimates the within-subject standard deviations of the log-transformed endpoint separately for the reference (swR) and test (swT) formulations, the quantities that reference-scaling frameworks need. The default `model_type = "anova"` is an analysis of variance (ANOVA) of the replicate observations within each formulation; this is the estimator used by both Method A and Method B of the EMA replicateBE reference implementation and by the FDA moment-based approach, and it works for full and partial replicate designs. `model_type = "nlme"` instead fits a single mixed model with treatment-specific residual variances and requires a fully replicated design; it is provided as the alternative described in the FDA progesterone guidance and can differ slightly from "anova".

Usage

```
be_within_var(
  data,
  value,
  subject,
  period,
  treatment,
  reference_value,
  model_type = c("anova", "nlme"),
  alpha = 0.1
)
```

Arguments

<code>data</code>	A long data.frame with one row per observation for a <i>single</i> endpoint (subject, period, treatment, and the endpoint value).
<code>value</code>	The column name of the (untransformed) endpoint value; it is log-transformed internally.
<code>subject, period, treatment</code>	Column names identifying the subject, period, and treatment.
<code>reference_value</code>	The value of treatment that is the reference.
<code>model_type</code>	"anova" (the default and the regulatory standard) or "nlme" (mixed model with treatment-specific residual variances; full replicate only).
<code>alpha</code>	The significance level for the swT/swR ratio confidence bound (default 0.10 gives the 90% upper bound used for narrow therapeutic index drugs).

Value

An object of class `be_within_var`: a list with `swR`, `swT`, `s2wR`, `s2wT`, `cvwr_percent`, `cvwt_percent`, `df_wR`, `df_wT`, `sw_ratio` ($= swT/swR$), `sw_ratio_ci_upper` (the upper $1 - \alpha$ confidence bound of the ratio), `model_type`, and `alpha`. `swT` and the ratio are NA when the test formulation is not replicated (for example a partial replicate design).

See Also

[be_assess\(\)](#)

Other Bioequivalence: [be_assess\(\)](#), [be_compare\(\)](#), [be_dataset\(\)](#), [be_design\(\)](#), [be_expand_limits\(\)](#), [be_extract_param\(\)](#), [be_fit_model_single\(\)](#), [be_fit_models\(\)](#), [be_regulator\(\)](#), [be_table\(\)](#)

<code>business.mean</code>	<i>Generate functions to do the named function (e.g. mean) applying the business rules.</i>
----------------------------	---

Description

Generate functions to do the named function (e.g. mean) applying the business rules.

Usage

```
business.mean(x, ...)
```

```
business.sd(x, ...)
```

```
business.cv(x, ...)
```

```
business.geomean(x, ...)
```

```
business.geocv(x, ...)
```

```
business.min(x, ...)
```

```
business.max(x, ...)
```

```
business.median(x, ...)
```

```
business.range(x, ...)
```

Arguments

`x` vector to be passed to the various functions

`...` Additional arguments to be passed to the underlying function.

Value

The value of the various functions or NA if too many values are missing

Functions

- `business.sd()`: Compute the standard deviation with business rules.
- `business.cv()`: Compute the coefficient of variation with business rules.
- `business.geomean()`: Compute the geometric mean with business rules.
- `business.geocv()`: Compute the geometric coefficient of variation with business rules.
- `business.min()`: Compute the minimum with business rules.
- `business.max()`: Compute the maximum with business rules.
- `business.median()`: Compute the median with business rules.
- `business.range()`: Compute the range with business rules.

See Also

[pk.business\(\)](#)

check.conversion	<i>Check that the conversion to a data type does not change the number of NA values</i>
------------------	---

Description

Check that the conversion to a data type does not change the number of NA values

Usage

```
check.conversion(x, FUN, ...)
```

Arguments

x	the value to convert
FUN	the function to use for conversion
...	arguments passed to FUN

Value

`FUN(x, ...)` or an error if the set of NAs change.

`check.interval.specification`*Check the formatting of a calculation interval specification data frame.*

Description

Calculation interval specifications are data frames defining what calculations will be required and summarized from all time intervals. Note: parameters which are not requested may be calculated if it is required for (or computed at the same time as) a requested parameter.

Usage

```
check.interval.specification(x, impute = NA_character_)
```

Arguments

<code>x</code>	The data frame specifying what to calculate during each time interval
<code>impute</code>	The impute argument of PKNCAdata() , so that a column it names is treated as an imputation column (like <code>impute</code>) rather than as part of the definition of an interval.

Details

`start` and `end` time must always be given as columns, and the `start` must be before the `end`. Other columns define the parameters to be calculated and the groupings to apply the intervals to.

Data are selected for calculation within an interval by the time of the measurement or dose: a row is included when its time is at or after `start` and at or before `end` (a dose exactly at `end` is not included in the interval). For duration data (for example, urine collections or intravenous infusions), the time is the start of the collection or administration, and the duration is not considered during selection: a collection that starts within the interval and ends after `end` is included and contributes its full amount to the interval. For the simplest interpretation of results, align collection start and end times with interval boundaries.

Value

`x` The potentially updated data frame with the interval calculation specification.

See Also

The vignette "Selection of Calculation Intervals"

Other Interval specifications: [add.interval.col\(\)](#), [choose.auc.intervals\(\)](#), [get.interval.cols\(\)](#), [get.parameter.deps\(\)](#), [interval_add_impute\(\)](#), [interval_add_param\(\)](#), [interval_add_secondary\(\)](#), [pkncdisc_codes\(\)](#), [pkncdisc_check_parameter_classification\(\)](#), [pkncdisc_concepts\(\)](#), [pkncdisc_interval_table\(\)](#), [pkncdisc_match_route\(\)](#), [pkncdisc_parameter_table\(\)](#), [pkncdisc_presets\(\)](#), [pkncdisc_ref\(\)](#)

checkProvenance	<i>Check the hash of an object to confirm its provenance.</i>
-----------------	---

Description

Check the hash of an object to confirm its provenance.

Usage

```
checkProvenance(object)
```

Arguments

object	The object to check provenance for
--------	------------------------------------

Value

TRUE if the provenance is confirmed to be consistent, FALSE if the provenance is not consistent, or NA if provenance is not present. An object that was modified after it was created by a PKNCA function that marks provenance (for example `dplyr::filter()`, `dplyr::mutate()`, `exclude()`, or `normalize()` on a PKNCAresults object) has a hash such as "filtered from <hash>" instead of the original hash and always gives FALSE.

See Also

[addProvenance\(\)](#)

choose.auc.intervals	<i>Choose intervals to compute AUCs from time and dosing information</i>
----------------------	--

Description

Intervals are selected by the following metrics:

1. If there are no dose times, no intervals are generated and a "pknca_warning_no_dose_times_for_group" warning is given.
2. If only one dose is administered and any sample follows it, the interval runs from the dose to infinity with the parameters `pknca_interval_table()` gives for a single dose.
3. If more than one dose is administered, an interval is generated between any two consecutive doses that have samples at both dose times and at least one sample between them. It is a dosing interval when the samples run up to the next dose, and a single-dose profile bounded by the next dose when they stop partway and leave a washout.

4. For the final dose, the dosing interval (τ) is found with [find.tau\(\)](#). An interval one τ long is generated when a sample was taken at its end, with the parameters for a dose at steady state. It starts at the first dose of the last complete cycle, which is the last dose itself unless the regimen gives more than one dose per τ , so that the interval never contains a dose that was not recorded.
5. If samples continue beyond τ after the last dose, the half-life is calculated from the last dose onward. If the last dose has samples after it but gets neither of these, its profile is calculated to infinity as a single dose.

Usage

```
choose.auc.intervals(
  time.conc,
  time.dosing,
  options = list(),
  single.dose.aucs = NULL,
  route = "extravascular",
  sparse = FALSE,
  timeu = NULL,
  time.conc.nominal = NULL,
  time.dosing.nominal = NULL,
  dense.samples = 3,
  conc = NULL
)
```

Arguments

<code>time.conc</code>	Time of concentration measurement
<code>time.dosing</code>	Time of dosing
<code>options</code>	List of changes to the default PKNCA options (see <code>PKNCA.options()</code>)
<code>single.dose.aucs</code>	The AUC specification for single dosing.
<code>route</code>	How the drug was given, as one of pknca_routes() .
<code>sparse</code>	Is this a sparse sampling design? A sparse design imputes nothing; see pknca_interval_table() .
<code>timeu</code>	The time unit of <code>time.conc</code> and <code>time.dosing</code> ; NULL when it is not known, which is taken to be hours; or NA when it cannot be used (see find.dose.regimen()). PKNCAdata() gives the time unit of its concentration data.
<code>time.conc.nominal, time.dosing.nominal</code>	Nominal times of the samples and of the doses, on the same scale as <code>time.conc</code> and <code>time.dosing</code> and the same length, or NULL. They are used, and checked to be numeric, only to judge whether a dose is sampled densely after a change of regimen. PKNCAdata() gives the <code>time.nominal</code> columns of PKNCAconc() and PKNCAdose() .
<code>dense.samples</code>	The fewest samples within one dosing period after a dose that make it sampled densely, which gives it an interval after a change of regimen
<code>conc</code>	The concentrations at <code>time.conc</code> , or NULL. Used only to leave samples without a concentration out of the dense-sampling count.

Details

When `find.dose.regimen()` finds a change of regimen in the dose times (more than one segment) and the design is not sparse, the intervals follow the segments instead:

- One steady-state interval for each segment, when a sample ends it. An earlier segment's interval is its last complete cycle, from the cycle's first dose to the recorded dose one period later (the dose that starts the next segment, unless a dose was missed before it), so it never contains a later dose. The last segment's interval is its last cycle, one period at the segment's own period. Like the interval after the last dose, a steady-state interval needs only a sample at its end, so it may have as few as two samples.
- The interval after the first dose, as above.
- The interval after any other dose only when the dose is sampled densely: at least `dense.samples` samples (3 by default) strictly within one period of its segment after the dose. A sample counts by its nominal time when it has one (`time.conc.nominal`), from the dose's nominal time when that is given (`time.dosing.nominal`), so that a sample drawn a little late counts where it was scheduled; a sample without a nominal time counts by its actual time. Samples at the same time count once, and samples without a concentration (`conc` is NA) do not count; samples below the limit of quantification do. Nominal times must share the actual times' origin (time since the first dose, for example). When they restart instead, as times since the latest dose do, the actual times are used with a "pknca_warning_intervals_nominal_restart" warning.
- The last dose gets the intervals above at the period of the last segment, even when an earlier segment has more doses.

A "pknca_warning_intervals_by_segment" warning names the segments and the intervals chosen. It is also a "pknca_warning_tau_regimen_change" warning, which it replaces.

τ is matched to the nominal dosing intervals of `find.dose.regimen()` in the time unit `timeu`, taken to be hours when it is not given: dose times recorded a little early or late give the nominal interval, and an interval that matches none of them, such as dosing every hour, gives a "pknca_warning_tau_not_nominal" warning. With `timeu = NA` (a unit PKNCA cannot use), τ is found from the dose times alone.

Times are matched within a tolerance rather than exactly, so a sample drawn a little before its nominal time still bounds the interval it belongs to. The window is the `auto.interval.tolerance` option as a fraction of the interval's length, and it only reaches backward: a sample drawn after a boundary belongs to what follows that boundary, so a concentration drawn after a dose cannot stand in for the predose sample.

Setting the `auto.interval.method` option to "legacy" calculates the parameter lists PKNCA used before `pknca_interval_table()` was available: the `single.dose.aucs` option for single-dose data, and AUClast, Cmax, and Tmax for each interval of multiple-dose data. The intervals themselves are found the same way either way.

Value

A data frame with columns for `start`, `end`, and the parameters to calculate. See `check.interval.specification()` for column definitions. The data frame may have zero rows if no intervals could be found.

See Also

`pknca_interval_table()`, `pk.calc.auc()`, `pk.calc.half.life()`, `PKNCA.options()`

Other Interval specifications: `add.interval.col()`, `check.interval.specification()`, `get.interval.cols()`, `get.parameter.deps()`, `interval_add_impute()`, `interval_add_param()`, `interval_add_secondary()`, `pknca_cdisc_codes()`, `pknca_check_parameter_classification()`, `pknca_concepts()`, `pknca_interval_table()`, `pknca_match_route()`, `pknca_parameter_table()`, `pknca_presets()`, `pknca_ref()`

Other Interval determination: `find.dose.regimen()`, `find.tau()`, `resolve_dose_tau()`

Examples

```
# A single dose gives one profile to infinity
choose.auc.intervals(c(0, 1, 2, 4, 8, 24), 0)[, c("start", "end")]

# Daily dosing with a dense profile on the first and last day
choose.auc.intervals(
  c(0, 1, 2, 4, 8, 12, 24, 48, 72, 96, 120, 144, 145, 146, 148, 152, 156, 168, 192),
  seq(0, 144, by = 24)
)[, c("start", "end")]
```

`choose_interval_method`

Choose how to interpolate, extrapolate, or integrate data in each concentration interval

Description

Choose how to interpolate, extrapolate, or integrate data in each concentration interval

Usage

```
choose_interval_method(conc, time, tlast, method, auc.type, options)
```

Arguments

<code>conc</code>	Measured concentrations
<code>time</code>	Time of the measurement of the concentrations
<code>tlast</code>	Time of last concentration above the limit of quantification (will be calculated, if not provided)
<code>method</code>	The method for integration (one of 'lin up/log down', 'lin-log', or 'linear')
<code>auc.type</code>	The type of AUC to compute. Choices are 'AUCinf', 'AUClast', and 'AUCall'.
<code>options</code>	List of changes to the default PKNCA options (see <code>PKNCA.options()</code>)

Value

A character vector of methods for interpolation/extrapolation methods that is the same length as `conc` which indicates how to interpolate/integrate between each of the concentrations (all but the last value in the vector) and how to extrapolate after `tlast` (the last item in the vector). Possible values in the vector are: 'zero', 'linear', 'log', and 'extrap_log'

clean.conc.blq	<i>Handle BLQ values in the concentration measurements as requested by the user.</i>
----------------	--

Description

Handle BLQ values in the concentration measurements as requested by the user.

Usage

```
clean.conc.blq(
  conc,
  time,
  ...,
  options = list(),
  conc.blq = NULL,
  conc.na = NULL,
  check = TRUE
)
```

Arguments

conc	Measured concentrations
time	Time of the measurement of the concentrations
...	Additional arguments passed to clean.conc.na
options	List of changes to the default PKNCA options (see PKNCA.options())
conc.blq	How to handle a BLQ value that is between above LOQ values? See details for description.
conc.na	How to handle NA concentrations. (See clean.conc.na())
check	Run assert_conc_time() ?

Details

NA concentrations (and their associated times) will be handled as described in [clean.conc.na\(\)](#) before working with the BLQ values. The method for handling NA concentrations can affect the output of which points are considered BLQ and which are considered "middle". Values are considered BLQ if they are 0.

conc.blq can be set either a scalar indicating what should be done for all BLQ values or a list with elements either named "first", "middle" and "last" or "before.tmax" and "after.tmax" each set to a scalar.

The meaning of each of the list elements is:

first Values up to the first non-BLQ value. Note that if all values are BLQ, this includes all values.

middle Values that are BLQ between the first and last non-BLQ values.

last Values that are BLQ after the last non-BLQ value

before.tmax Values that are BLQ before the time at first maximum concentration

after.tmax Values that are BLQ after the time at first maximum concentration

The valid settings for each are:

"drop" Drop the BLQ values

"keep" Keep the BLQ values

a number Set the BLQ values to that number

Value

The concentration and time measurements (data frame) filtered and cleaned as requested relative to BLQ in the middle.

See Also

Other Data cleaners: [clean.conc.na\(\)](#)

clean.conc.na	<i>Handle NA values in the concentration measurements as requested by the user.</i>
---------------	---

Description

NA concentrations (and their associated times) will be removed then the BLQ values in the middle

Usage

```
clean.conc.na(conc, time, ..., options = list(), conc.na = NULL, check = TRUE)
```

Arguments

conc	Measured concentrations
time	Time of the measurement of the concentrations
...	Additional items to add to the data frame
options	List of changes to the default PKNCA options (see <code>PKNCA.options()</code>)
conc.na	How to handle NA concentrations? Either 'drop' or a number to impute.
check	Run assert_conc_time() ?

Value

The concentration and time measurements (data frame) filtered and cleaned as requested relative to NA in the concentration.

See Also

Other Data cleaners: [clean.conc.blq\(\)](#)

cov_holder

*Calculate the covariance for two time points with sparse sampling***Description**

The calculation follows equation A3 in Holder 2001 (see references below):

Usage

```
cov_holder(sparse_pk)
```

Arguments

sparse_pk A sparse_pk object from [as_sparse_pk\(\)](#)

Details

$$\hat{\sigma}_{ij} = \sum_{k=1}^{r_{ij}} \frac{(x_{ik} - \bar{x}_i)(x_{jk} - \bar{x}_j)}{(r_{ij} - 1) + \left(1 - \frac{r_{ij}}{r_i}\right) \left(1 - \frac{r_{ij}}{r_j}\right)}$$

If $r_{ij} = 0$, then $\hat{\sigma}_{ij}$ is defined as zero (rather than dividing by zero).

Where:

$\hat{\sigma}_{ij}$ The covariance of times i and j

r_i **and** r_j The number of subjects (usually animals) at times i and j, respectively

r_{ij} The number of subjects (usually animals) at both times i and j

x_{ik} **and** x_{jk} The concentration measured for animal k at times i and j, respectively

$\bar{x}_i$ **and** $\bar{x}_j$ The mean of the concentrations at times i and j, respectively

The Cauchy-Schwartz inequality is enforced for covariances to keep correlation coefficients between -1 and 1, inclusive, as described in equations 8 and 9 of Nedelman and Jia 1998.

Value

A matrix with one row and one column for each element of sparse_pk_attribute. The covariances are on the off diagonals, and for simplicity of use, it also calculates the variance on the diagonal elements.

References

Holder DJ. Comments on Nedelman and Jia's Extension of Satterthwaite's Approximation Applied to Pharmacokinetics. Journal of Biopharmaceutical Statistics. 2001;11(1-2):75-79. doi:10.1081/BIP-100104199

Nedelman JR, Jia X. An extension of Satterthwaite's approximation applied to pharmacokinetics. Journal of Biopharmaceutical Statistics. 1998;8(2):317-328. doi:10.1080/10543409808835241

defunct

*The following functions are defunct***Description**

The following functions are defunct

Usage

```
check.conc.time(...)
```

Arguments

... Ignored

Functions

- `check.conc.time()`: Defunct as of version 0.11

ensure_column_unit_exists

*Ensure Unit Columns Exist in PKNCA Object***Description**

Checks if specified unit columns exist in a PKNCA object (either PKNCAconc or PKNCAdose). If the columns do not exist, it creates them and assigns default values (NA or existing units).

Usage

```
ensure_column_unit_exists(pknca_obj, unit_name)
```

Arguments

`pknca_obj` A PKNCA object (either PKNCAconc or PKNCAdose).
`unit_name` A character vector of unit column names to ensure (concu, amountu, timeu...).

Details

The function performs the following steps:

1. Checks if the specified unit columns exist in the PKNCA object.
2. If a column does not exist, it creates the column and assigns default values.
3. If not default values are provided, it assigns NA to the new column.

Value

The updated PKNCA object with ensured unit columns.

exclude

Exclude data points or results from calculations or summarization.

Description

Exclude data points or results from calculations or summarization.

Usage

```
exclude(object, reason, mask, FUN)
```

```
## Default S3 method:
```

```
exclude(object, reason, mask, FUN)
```

Arguments

object	The object to exclude data from.
reason	The reason to add as a reason for exclusion.
mask	A logical vector or numeric index of values to exclude (see details).
FUN	A function to operate on the data (one group at a time) to select reasons for exclusions (see details).

Details

Only one of mask or FUN may be given. If FUN is given, it will be called with two arguments: a data.frame (or similar object) that consists of a single group of the data and the full object (e.g. the PKNCAconc object), FUN(current_group, object), and it must return a logical vector equivalent to mask or a character vector with the reason text given when data should be excluded or NA_character_ when the data should be included (for the current exclusion test).

Value

The object with updated information in the exclude column. The exclude column will contain the reason if mask or FUN indicate. If a previous reason for exclusion was given, then subsequent reasons for exclusion will be added to the first with a semicolon space ("; ") separator.

Methods (by class)

- exclude(default): The general case for data exclusion

See Also

Other Result exclusions: [exclude_nca](#), [pknca_exclude_rules\(\)](#), [pknca_missing_samples\(\)](#)

Examples

```
myconc <- PKNCAconc(data.frame(subject=1,
                                time=0:6,
                                conc=c(1, 2, 3, 2, 1, 0.5, 0.25)),
                    conc~time|subject)
exclude(myconc,
        reason="Carryover",
        mask=c(TRUE, rep(FALSE, 6)))
```

exclude_nca

*Exclude NCA parameters based on examining the parameter set.***Description**

Exclude NCA parameters based on examining the parameter set.

Usage

```
exclude_nca_span.ratio(min.span.ratio)

exclude_nca_max.aucinf.pext(max.aucinf.pext)

exclude_nca_count_conc_measured(
  min_count,
  exclude_param_pattern = c("^aucall", "^aucinf", "^aucint", "^auciv", "^auclast",
                             "^aumc", "^sparse_auc")
)

exclude_nca_min.hl.r.squared(min.hl.r.squared)

exclude_nca_min.hl.adj.r.squared(min.hl.adj.r.squared = 0.9)

exclude_nca_tmax_early(tmax_early = 0)

exclude_nca_tmax_0()

exclude_nca_tmax_coverage(min_subjects = 4, k_warn = 1.5, k_exclude = 3)
```

Arguments

min.span.ratio	The minimum acceptable span ratio (uses PKNCA.options("min.span.ratio") if not provided).
max.aucinf.pext	The maximum acceptable percent AUC extrapolation (uses PKNCA.options("max.aucinf.pext") if not provided).
min_count	Minimum number of measured concentrations

exclude_param_pattern	Character vector of regular expression patterns to exclude
min.hl.r.squared	The minimum acceptable r-squared value for half-life (uses <code>PKNCA.options("min.hl.r.squared")</code> if not provided).
min.hl.adj.r.squared	The minimum acceptable adjusted r-squared for half-life (uses 0.9 if not provided).
tmax_early	The time for Tmax which is considered too early to be a valid NCA result
min_subjects	The fewest subjects with a Tmax in a group for the group to be checked (smaller groups are not checked, with a message)
k_warn	The multiple of the interquartile range that sets the inner fences; a subject with no sample within them gets a warning
k_exclude	The multiple of the interquartile range that sets the outer fences (at least k_warn); a subject with no sample within them is excluded

Value

A function to give to `exclude()` as FUN. Its `pknca_affected_parameters` attribute lists the parameters it can exclude, and its `pknca_options` attribute lists the `PKNCA.options()` entries its thresholds came from (see `pknca_exclude_rules()`).

Functions

- `exclude_nca_span_ratio()`: Exclude based on the half-life span ratio
- `exclude_nca_max_aucinf_pext()`: Exclude based on the percent of AUC extrapolated to infinity (both observed and predicted)
- `exclude_nca_count_conc_measured()`: Exclude based on the count of concentrations measured and not below the lower limit of quantification (affects AUC and AUMC parameters)
- `exclude_nca_min.hl.r.squared()`: Exclude based on half-life r-squared
- `exclude_nca_min.hl.adj.r.squared()`: Exclude based on half-life adjusted r-squared
- `exclude_nca_tmax_early()`: Exclude based on implausibly early Tmax (often used for extravascular dosing with a Tmax value of 0)
- `exclude_nca_tmax_0()`: Exclude based on implausibly early Tmax (special case for `tmax_early = 0`)
- `exclude_nca_tmax_coverage()`: Exclude based on whether a subject has a sample within Tukey's fences around the Tmax values of its group; the rule flags a subject with no sample within the inner fences, and excludes only a subject with no sample within the outer fences

Tmax coverage

`exclude_nca_tmax_coverage()` compares each subject's samples with the Tmax values of its group; it flags a subject whose samples miss the usual Tmax times, and excludes only beyond the outer fences. The group is the summary group of `summary.PKNCAresults()`: every grouping column except the subject, with the interval start and end (so each part, treatment, analyte, and

interval is its own group). From the first and third quartiles of the Tmax values of every subject in the group (the subject being judged included), Tukey's inner fences are the quartiles extended by `k_warn` times the interquartile range, and the outer fences by `k_exclude` times it. The quartiles are those of `stats::quantile(type = 7)`, the default type; with few subjects, the types differ (Tmax values of 1, 1, 2, 2, 1, and 8 give a third quartile of 2 with type 7, 3.5 with type 6, and 2.5 with type 8). Tmax values that are missing or already excluded are not used. The ranges never start before the interval start, where a reported range starts when its lower fence is below it. A sample at the interval start counts only when the subject's dose at or before the interval start is an intravascular bolus (route intravascular and no duration or a duration of zero in `PKNCAdose()`), so a predose sample does not count after an extravascular dose or when the results have no dose data.

The samples are the concentration rows with an actual time within the interval (the rows that `pk.nca()` used) that have a concentration and are not excluded. When the concentration data have a nominal time (`time.nominal` in `PKNCaconc()`), the samples are placed by their nominal time minus the interval start, so the nominal times must share the origin of the interval times (for example, the nominal time since the first dose); otherwise, they are placed by their actual time. The Tmax values are placed the same way: with nominal times, a subject's Tmax is the nominal time of the sample it was found at, so an offset between the actual and nominal times moves the fences and the samples together. A subject whose Tmax sample has no nominal time does not enter the fences, and a subject whose samples in the interval have no nominal time at all is not judged, with a `pknca_message_tmax_coverage_no_nominal_subject` message. Times are never converted between units, and the time unit is only used in the text.

- A subject with no sample within the outer fences is excluded: every parameter of that interval is excluded with a reason that gives the outer range and the subject's sample nearest to it.
- A subject with a sample within the outer fences but none within the inner fences is flagged as a possible data issue and not excluded: a `pknca_warning_tmax_coverage_outlier` warning has the fields `group` (a one-row data.frame with the group and subject), `tmax_range` (the inner range, relative to the interval start), and `nearest_sample` (the subject's sample nearest to it, relative to the interval start).
- With nominal times, a subject with a sample within the inner fences that is missing some of the group's nominal times within them (see `pknca_missing_samples()`) gets a `pknca_message_tmax_coverage_partial` message that Cmax and Tmax may be unreliable. Its fields are `group`, `tmax_range`, `time_nominal_missing`, and `reason` (as in `pknca_missing_samples()`). Without nominal times, there is no schedule to compare with, and no message is given.
- The warnings and messages can be caught with `withCallingHandlers()`.
- A group with fewer than `min_subjects` subjects with a Tmax on the basis of the fences (one subject, for example) is not checked, and a `pknca_message_tmax_coverage_few_subjects` message says so once per group. Quartiles of fewer than four values describe the spread of the group poorly.
- When the nominal times disagree with the interval, the group is not checked, and a `pknca_warning_tmax_coverage_no` warning says so once per group. They disagree when no nominal time of the group is after the interval start, when most samples of the group with an actual time in the interval have a nominal time outside it, or when a subject's nominal times go back to the start of its schedule while its actual times increase within the interval, as when the nominal times restart at each dose.
- Sparse data have one Tmax per group, from the mean profile, so they are not checked, and a `pknca_message_tmax_coverage_sparse` message says so once per call to `exclude()`.

See Also

Other Result exclusions: [exclude\(\)](#), [pknca_exclude_rules\(\)](#), [pknca_missing_samples\(\)](#)

Examples

```
my_conc <- PKNCAconc(data.frame(conc=1.1^(3:0),
                                time=0:3,
                                subject=1),
                    conc~time|subject)
my_data <- PKNCAdata(my_conc,
                    intervals=data.frame(start=0, end=Inf,
                                          aucinf.obs=TRUE,
                                          aucpext.obs=TRUE))

my_result <- pk.nca(my_data)
my_result_excluded <- exclude(my_result,
                              FUN=exclude_nca_max_aucinf_pext())
as.data.frame(my_result_excluded)
```

exclude_nca_by_param *Exclude based on NCA parameter thresholds*

Description

Exclude rows from NCA results based on specified thresholds for a given parameter. This function allows users to define minimum and/or maximum acceptable values for a parameter and excludes rows that fall outside these thresholds.

Usage

```
exclude_nca_by_param(
  parameter,
  min_thr = NULL,
  max_thr = NULL,
  affected_parameters = parameter
)
```

Arguments

parameter	The name of the PKNCA parameter to evaluate (e.g., "span.ratio").
min_thr	The minimum acceptable value for the parameter. If not provided, is not applied.
max_thr	The maximum acceptable value for the parameter. If not provided, is not applied.
affected_parameters	Character vector of PKNCA parameters that will be marked as excluded. By default is the defined parameter.

Value

A function that can be used with `PKNCA::exclude` to mark through the 'exclude' column the rows in the PKNCA results based on the specified thresholds for a parameter. Its `pknc_a_affected_parameters` attribute lists the parameters it can mark (see [pknc_a_exclude_rules\(\)](#)).

Examples

```
# Example dataset
my_data <- PKNCA::PKNCAdata(
  PKNCA::PKNCAconc(data.frame(conc = 5:1,
                                time = 0:4,
                                subject = 1),
                    conc ~ time | subject),
  PKNCA::PKNCAdose(data.frame(subject = 1, dose = 100, time = 0),
                    dose ~ time | subject)
)
my_result <- PKNCA::pk.nca(my_data)

# Exclude rows where span.ratio is less than 2
excluded_result <- PKNCA::exclude(
  my_result,
  FUN = exclude_nca_by_param("span.ratio", min_thr = 2)
)
as.data.frame(excluded_result)
```

filter.PKNCAresults *dplyr filtering for PKNCA*

Description

Filtering a `PKNCAconc` or `PKNCAdose` object filters its data. Filtering a `PKNCAresults` object filters the result table and, when the filter only uses group columns (the columns after the `|` in the `PKNCAconc` formula), the concentration data, the dose data, and the intervals of `x$data` as well, so that the filtered results are consistent with their data. Only those three are filtered; `group_ref` and units of the `PKNCAdata` are not.

Usage

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

## S3 method for class 'PKNCAconc'
filter(.data, ..., .by = NULL, .preserve = FALSE)

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

Arguments

<code>.data</code>	A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from <code>dbplyr</code> or <code>dtplyr</code>). See <i>Methods</i> , below, for more details.
<code>...</code>	<data-masking> Expressions that return a logical vector, defined in terms of the variables in <code>.data</code> . If multiple expressions are included, they are combined with the <code>&</code> operator. To combine expressions using <code> </code> instead, wrap them in when_any() . Only rows for which all expressions evaluate to TRUE are kept (for <code>filter()</code>) or dropped (for <code>filter_out()</code>).
<code>.by</code>	<tidy-select> Optionally, a selection of columns to group by for just this operation, functioning as an alternative to group_by() . For details and examples, see ?dplyr_by .
<code>.preserve</code>	Relevant when the <code>.data</code> input is grouped. If <code>.preserve = FALSE</code> (the default), the grouping structure is recalculated based on the resulting data, otherwise the grouping is kept as is.

Details

The filter is evaluated once, on the result table. The groups that remain there are the groups kept: the rows of the concentration data, dose data, and intervals are kept when their values in the group columns that they share with the result table match a remaining group. A table that has none of those group columns (for example, intervals with no group columns) is left whole, and groups that have no rows in the result table are dropped from the data. Because the filter is evaluated only on the results, any filter that aggregates over rows (such as `ID == max(ID)`) keeps the same groups in every table.

A filter that uses any other column of the result table (such as `PPTSTCD`, `PPORRES`, or `exclude`), or that uses no column of the result table, filters the result table only. Columns are recognized when the expressions name them (`part == "SAD"`, `.data$part == "SAD"`, and the columns of `.by`). Variables of the calling environment are not treated as columns. Filters that choose columns without naming them (`.data[[var]]`, `across()`, `if_any()`, `if_all()`, `pick()`, and tidyselect helpers such as `all_of()` or `starts_with()`) filter the result table only, with a message that the data were not filtered. The columns `start` and `end` belong to the result table, so a filter on them leaves the data unchanged.

`mutate()` and the joins change only the result table; assigning into the object directly is the caller's responsibility and is not tracked.

A `PKNCAresults` object that a `PKNCA` dplyr verb changes (`filter()`, `mutate()`, `group_by()`, `ungroup()`, and the joins) no longer matches the hash that [checkProvenance\(\)](#) verifies: the hash is replaced by a marker such as "filtered from <hash>" and [checkProvenance\(\)](#) returns FALSE. A call that leaves the object identical keeps its provenance.

See Also

Other dplyr verbs: [group_by.PKNCAresults\(\)](#), [inner_join.PKNCAresults\(\)](#), [mutate.PKNCAresults\(\)](#)

filter_interval	<i>Subset data to the rows used for calculations within an interval</i>
-----------------	---

Description

Rows are selected by their time falling within the interval: $\text{start} \leq \text{time}$ and $\text{time} \leq \text{end}$ (or $\text{time} < \text{end}$ when `include_end=FALSE`). For duration data (for example, urine collections or intravenous infusions), time is the start of the collection or administration, and the duration column is not consulted during selection: a record whose duration starts within the interval and ends after the interval end is selected and contributes its full amount to calculations within the interval. For the simplest interpretation of results, align collection start and end times with interval boundaries.

Usage

```
filter_interval(data, start, end, include_na = FALSE, include_end = TRUE)
```

Arguments

data	A data.frame with a column named time (and, for duration data, a column named duration, which is ignored during selection)
start, end	The beginning and end times of the interval
include_na	Should rows with an NA time be kept?
include_end	Should a row with $\text{time} == \text{end}$ be kept?

Value

The rows of data selected for the interval

find.dose.regimen	<i>Find the dosing regimen from the dose times of one subject</i>
-------------------	---

Description

The dose times are sorted and times that repeat are dropped. The regimen is then described as one or more segments, each a stretch of dosing that repeats over a period, found in this order:

1. Equally spaced doses repeat over their spacing.
2. A cycle of 2 to 4 doses per period that every dose follows is looked for next. The period is the median span of that many spacings, refined by least squares, and each dose's time within the period falls into one of the positions of the cycle. The spacings between the positions must differ from one another by more than `tol`, every dose must be within `snap.tol` of the period of its position, and every position must hold at least two doses (two complete periods). The positions must also explain the scatter of the doses: the doses sit at most half as far from their positions as the spacings sit from their median, and the positions fit the doses better

than evenly spaced positions by more than chance (an F test at the 0.999 level). Daily doses recorded a few hours early or late are therefore daily dosing, not a cycle. Twice-daily doses at 08:00 and 16:00 repeat every 24 hours with offsets 0 and 8. When the period is not one of the given `tau.choices` and a whole number of periods is, that multiple is used if the doses hold two complete repeats of it (so a 48 hour choice selects two days of twice-daily dosing only when there are at least nine doses); the built-in nominal set never selects a longer period.

3. Doses whose spacings are all within `tol` of their median are evenly spaced with scatter, and repeat over the median spacing.
4. Otherwise, each spacing is labeled with the candidate it is within `tol` of (on the log scale), and the rest are clustered on the log scale. Doses squeezed into one interval between two runs of the regimen are set aside as extra doses. Runs of at least `min.run` spacings with one label anchor segments, and the runs must hold most of the spacings. A spacing outside the runs is a missed dose when it is a whole number (up to `max.missed`) of the segment's interval and is irregular when it is not; a dose given early or late is the dose between two irregular spacings. Neighboring anchors with the same label are one segment, so a change of regimen gives several segments.
5. When that reading has missed or irregular doses or more than one segment, a cycle of 2 to 4 doses that at least 70% of the doses follow is used instead if it explains more spacings, or as many with no more missed, off-schedule, or extra doses and changes of regimen. This keeps three-times-daily dosing with a dose not given as three times daily, and daily dosing after twice-daily dosing as a change of regimen.
6. If none of that fits, there is no repeating interval.

Usage

```
find.dose.regimen(
  x,
  tau.choices = NULL,
  timeu = NULL,
  options = list(),
  tol = 0.2,
  snap.tol = 0.1,
  min.run = 2,
  max.missed = 4
)
```

Arguments

<code>x</code>	Dose times for one subject, as numbers
<code>tau.choices</code>	The nominal intervals to match, in the unit of <code>x</code> , or NA to use the built-in set. Names are used as labels. NULL takes the <code>tau.choices</code> option.
<code>timeu</code>	The time unit of <code>x</code> (such as "hr" or "day"); NULL (or "") when it is not known, which is taken to be hours; or NA when it is known not to be a unit PKNCA can use, which leaves the intervals to the data. Any other unit that cannot be converted to hours is an error, except that without the units package a unit other than "hr" leaves the intervals to the data with a message, so an unrecognized unit such as "zzz" is an error only when the units package is installed.

options	List of changes to the default PKNCA options (see <code>PKNCA.options()</code>)
tol	Relative tolerance for grouping spacings with one another and with a candidate
snap.tol	Relative tolerance for reporting an interval as a candidate and for a dose to be on a cycle's pattern
min.run	The fewest consecutive spacings with one label that form a segment
max.missed	The largest multiple of the interval that is read as missed doses

Details

The interval of a segment is snapped to the nearest candidate within `snap.tol` (source "nominal"), and is otherwise the value found in the data (source "auto"). The value found in the data is the median of the spacings (or spans) that set it, so scatter in the recorded times does not move it. The candidates are `tau.choices` when it is given. Otherwise they are the built-in nominal intervals, converted from hours to `timeu`: Q4H (4 hours), QID (6), TID (8), BID (12), QD (24), QOD (48), Q72H (72), QW (168), Q2W (336), Q3W (504), Q4W (672), Q6W (1008), Q8W (1344), and Q12W (2016).

Without a time unit, the times are taken to be hours, as PKNCA takes date-times to be, so dose times 24 and 26 hours apart are once-daily dosing. Times recorded in another unit without saying so are then compared with intervals in hours, and an interval that matches none of them gives a "pknca_warning_tau_not_nominal" warning; give `timeu` to avoid that. `timeu = NA` says that the unit is not one PKNCA can use, and the intervals then come from the data alone (every source is "auto"). Converting a unit other than "hr" needs the `units` package; without it, such a unit gives a "pknca_message_tau_units_fallback" message and the intervals come from the data alone, so results for data in other units can differ between installations with and without that package.

Value

A data frame with one row per segment and the columns:

segment	The segment number
start, end	The first and last dose time of the segment; a dose where the regimen changes ends one segment and starts the next
n_doses	The number of doses in the segment
interval	The period that the dose pattern repeats over, NA for a single dose or when nothing repeats
label	The name of the candidate for an evenly spaced regimen with the segment's number of doses per period (twice-daily doses at 0 and 10 hours are "BID"), or NA
source	"nominal" when interval is a candidate and "auto" when it was found in the data
n_intervals_used	The spacings that set the interval
n_missed	Doses read as missed
n_irregular	Doses off schedule or extra
note	A description of anything unusual
offsets	A list column of the dose times within one period, relative to the first dose of the period

With no dose times, the data frame has no rows.

Conditions

`pknca_warning_tau_irregular_dosing` A segment has missed doses (the message names the dose times before them) or doses off schedule (the message names those doses).

`pknca_warning_tau_not_nominal` There were candidates to match (any timeu other than NA) and a segment matched none of them. The message includes the regimen table.

`pknca_warning_tau_regimen_change` More than one segment was found. The message includes the regimen table.

`pknca_warning_dose_regimen` The parent class of the three warnings above, to handle them together.

`pknca_message_tau_units_fallback` timeu is not "hr" and the units package is not installed to convert it, so the intervals come from the data alone.

`pknca_error_regimen_time_unit` timeu cannot be converted to hours.

`pknca_error_regimen_time_class` x is a date-time or difftime rather than a number.

See Also

[find.tau\(\)](#), which gives the interval of the segment with the most doses

Other Interval determination: [choose.auc.intervals\(\)](#), [find.tau\(\)](#), [resolve_dose_tau\(\)](#)

Examples

```
# Daily dosing with the times as recorded
find.dose.regimen(c(0, 23.6, 48, 72.4, 96), timeu = "hr")
# Twice daily at breakfast and dinner repeats daily
find.dose.regimen(c(0, 10, 24, 34, 48, 58), timeu = "hr")
# Daily, then twice daily
suppressWarnings(
  find.dose.regimen(c(0, 24, 48, 72, 84, 96, 108, 120), timeu = "hr")
)
```

find.tau

Find the repeating interval within a vector of doses

Description

The regimen is found with [find.dose.regimen\(\)](#), and the interval is the period of its segment with the most doses (the latest of those when segments tie). In brief, the dose times are sorted and times that repeat are dropped; equally spaced doses give their spacing; a pattern of more than one dose per interval that repeats over at least two complete intervals gives the interval it repeats over; doses scattered within a tolerance of one spacing give their median spacing; and otherwise runs of the same spacing form the regimen, with the gaps between them read as missed doses (a whole number of intervals) or doses off schedule. Times are compared on the log scale within `tol` and `snap.tol` of [find.dose.regimen\(\)](#), so a dose recorded at 23.6 hours is a daily dose.

Usage

```
find.tau(
  x,
  na.action = stats::na.omit,
  options = list(),
  tau.choices = NULL,
  timeu = NULL
)
```

Arguments

<code>x</code>	the vector to find the interval within
<code>na.action</code>	What to do with NAs in <code>x</code>
<code>options</code>	List of changes to the default PKNCA options (see <code>PKNCA.options()</code>)
<code>tau.choices</code>	The nominal intervals to match, in the unit of <code>x</code> , or NA to use the built-in set. Names are used as labels. NULL takes the <code>tau.choices</code> option.
<code>timeu</code>	The time unit of <code>x</code> (such as "hr" or "day"); NULL (or "") when it is not known, which is taken to be hours; or NA when it is known not to be a unit PKNCA can use, which leaves the intervals to the data. Any other unit that cannot be converted to hours is an error, except that without the units package a unit other than "hr" leaves the intervals to the data with a message, so an unrecognized unit such as "zzz" is an error only when the units package is installed.

Details

Looking for a repeating pattern before reading anything as a missed dose is what keeps a regimen with a regular gap in it, such as dosing three times a day at 0, 6, and 12 hours, from being reported as a 6 hour interval with a dose missing overnight.

The intervals are matched to `tau.choices` when it is given. When it is NA (the default), they are matched to the built-in nominal intervals listed in [find.dose.regimen\(\)](#), in the unit `timeu`. **Without a time unit, the times are taken to be hours**, so `find.tau(c(0, 24, 50))` is 24 (once daily) rather than the median spacing of 25; times in another unit should give `timeu`, or they are compared with intervals in hours and an interval that matches none of them gives a "pknca_warning_tau_not_nominal" warning. The warnings of [find.dose.regimen\(\)](#) are given here as well, notably "pknca_warning_tau_irregular_dosing" for missed doses or doses off schedule.

Value

NA when there are no dose times or no interval repeats, 0 for a single dose time, and otherwise the repeating interval.

See Also

Other Interval determination: [choose.auc.intervals\(\)](#), [find.dose.regimen\(\)](#), [resolve_dose_tau\(\)](#)

Examples

```
# Equally spaced doses give their spacing
find.tau(c(0, 24, 48, 72))
# Twice-daily dosing repeats daily although no two doses are a day apart
find.tau(c(0, 10, 24, 34, 48, 58), tau.choices = c(12, 24))
# Dose times as recorded, with the unit known
find.tau(c(0, 23.6, 48, 72.4, 96), timeu = "hr")
```

findOperator	<i>Find the first occurrence of an operator in a formula and return the left, right, or both sides of the operator.</i>
--------------	---

Description

Find the first occurrence of an operator in a formula and return the left, right, or both sides of the operator.

Usage

```
findOperator(x, op, side)
```

Arguments

x	The formula to parse
op	The operator to search for (e.g. +, -, *, /, ...)
side	Which side of the operator would you like to see: 'left', 'right', or 'both'.

Value

The side of the operator requested, NA if requesting the left side of a unary operator, and NULL if the operator is not found.

See Also

Other Formula parsing: [parse_formula_to_cols\(\)](#)

fit_half_life	<i>Perform the half-life fit given the data. The function simply fits the data without any validation. No selection of points or any other components are done.</i>
---------------	---

Description

Perform the half-life fit given the data. The function simply fits the data without any validation. No selection of points or any other components are done.

Usage

```
fit_half_life(data, tlast)
```

Arguments

data	The data to fit. Must have two columns named "log_conc" and "time"
tlast	The time of last observed concentration above the limit of quantification.

Value

A named list with one value each for "r.squared", "adj.r.squared", "lambda.z.corrxy", "lambda.z", "clast.pred", "lambda.z.time.first", "lambda.z.time.last", "lambda.z.n.points", "half.life", and "span.ratio". [pk.calc.half.life\(\)](#) fits one candidate per span of terminal points and builds a data.frame from the candidate it selects.

See Also

[pk.calc.half.life\(\)](#)

fit_half_life_tobit	<i>Perform a Tobit half-life fit given the data. The function fits the data using maximum likelihood without any point selection or validation.</i>
---------------------	---

Description

Perform a Tobit half-life fit given the data. The function fits the data using maximum likelihood without any point selection or validation.

Usage

```
fit_half_life_tobit(data, tlast, optim_control = list())
```

Arguments

data	The data to fit. Must have columns named "log_conc", "time", "log_lloq", and "mask_blq". log_conc for BLQ observations is not used (the likelihood uses log_lloq instead).
tlast	The time of last observed concentration above the lower limit of quantification.
optim_control	A list of control parameters passed to <code>stats::optim()</code> .

Value

A data.frame with one row and columns named "lambda.z", "clast.pred", "lambda.z.time.first", "lambda.z.time.last", "lambda.z.n.points", "lambda.z.n.points_blq", "half.life", "span.ratio", "tobit_residual", and "adj_tobit_residual". Returns NA for all columns if `stats::optim()` does not converge, and emits a warning.

See Also

`pk.calc.half.life()`, `fit_half_life_tobit_LL()`

fit_half_life_tobit_LL

Negative log-likelihood for Tobit half-life regression

Description

Helper function used by `fit_half_life_tobit()` via `stats::optim()`. For observations above the LLOQ, the normal density contributes to the likelihood. For censored (BLQ) observations, the normal CDF up to the LLOQ contributes.

Usage

```
fit_half_life_tobit_LL(par, log_conc, time, mask_blq, log_lloq)
```

Arguments

par	A 3-element numeric vector: <code>c(log_c0, lambda_z, log_resid_error)</code>
log_conc	Natural log of observed concentration (may be <code>-Inf</code> for BLQ; those values are not used when <code>mask_blq</code> is TRUE)
time	Numeric time vector
mask_blq	Logical vector; TRUE where the observation is below the LLOQ
log_lloq	Natural log of the lower limit of quantification

Value

The negative sum of the log-likelihood (a scalar)

See Also

fit_half_life_tobit()

formula.PKNCAconc	<i>Extract the formula from a PKNCAconc object.</i>
-------------------	---

Description

Extract the formula from a PKNCAconc object.

Usage

```
## S3 method for class 'PKNCAconc'
formula(x, ...)

## S3 method for class 'PKNCAdose'
formula(x, ...)
```

Arguments

x	The object to extract the formula from.
...	Unused

Value

A formula object

geomean	<i>Compute the geometric mean, sd, and CV</i>
---------	---

Description

Compute the geometric mean, sd, and CV

Usage

```
geomean(x, na.rm = FALSE)

geosd(x, na.rm = FALSE)

geocv(x, na.rm = FALSE)
```

Arguments

x	A vector to compute the geometric mean of
na.rm	Should missing values be removed?

Value

The scalar value of the geometric mean, geometric standard deviation, or geometric coefficient of variation.

Functions

- `geosd()`: Compute the geometric standard deviation, $\exp(\text{sd}(\log(x)))$.
- `geocv()`: Compute the geometric coefficient of variation, $\sqrt{\exp(\text{sd}(\log(x))^2) - 1} * 100$.

References

Kirkwood T. B.L. Geometric means and measures of dispersion. *Biometrics* 1979; 35: 908-909

Examples

```
geomean(1:3)
geosd(1:3)
geocv(1:3)
```

```
get.best.model
```

Extract the best model from a list of models using the AIC.

Description

Extract the best model from a list of models using the AIC.

Usage

```
get.best.model(object, ...)
```

Arguments

<code>object</code>	the list of models
<code>...</code>	Passed to <code>AIC()</code>

Value

The model which is assessed as best. If more than one are equal, the first is chosen.

get.interval.cols	<i>Get the columns that can be used in an interval specification</i>
-------------------	--

Description

Get the columns that can be used in an interval specification

Usage

```
get.interval.cols()
```

Value

A list with named elements for each parameter. Each list element contains the parameter definition.

See Also

[check.interval.specification\(\)](#) and the vignette "Selection of Calculation Intervals"

Other Interval specifications: [add.interval.col\(\)](#), [check.interval.specification\(\)](#), [choose.auc.intervals\(\)](#), [get.parameter.deps\(\)](#), [interval_add_impute\(\)](#), [interval_add_param\(\)](#), [interval_add_secondary\(\)](#), [pkncdisc_codes\(\)](#), [pkncdisc_check_parameter_classification\(\)](#), [pkncdisc_concepts\(\)](#), [pkncdisc_interval_table\(\)](#), [pkncdisc_match_route\(\)](#), [pkncdisc_parameter_table\(\)](#), [pkncdisc_presets\(\)](#), [pkncdisc_ref\(\)](#)

Examples

```
get.interval.cols()
```

get.parameter.deps	<i>Get all columns that depend on a parameter</i>
--------------------	---

Description

Get all columns that depend on a parameter

Usage

```
get.parameter.deps(x, recursive = FALSE)
```

Arguments

x	The parameter name (as a character string)
recursive	Search backward to the inputs x is calculated from, rather than forward to the parameters calculated from x. See the details.

Details

The two directions answer different questions. The default answers "what becomes invalid if x changes?". `recursive = TRUE` answers "what does x need?", and its result mixes parameter names with the raw inputs the calculation ends at, such as "conc", "time", "dose", and "time.dose".

Value

With `recursive = FALSE` (default), a character vector of parameter names that depend on the parameter x ; empty if none do. With `recursive = TRUE`, the unique set of everything x is calculated from, following each dependency to the end.

See Also

Other Interval specifications: [add.interval.col\(\)](#), [check.interval.specification\(\)](#), [choose.auc.intervals\(\)](#), [get.interval.cols\(\)](#), [interval.add.impute\(\)](#), [interval.add.param\(\)](#), [interval.add.secondary\(\)](#), [pknca.cdisc.codes\(\)](#), [pknca.check.parameter.classification\(\)](#), [pknca.concepts\(\)](#), [pknca.interval.table\(\)](#), [pknca.match.route\(\)](#), [pknca.parameter.table\(\)](#), [pknca.presets\(\)](#), [pknca.ref\(\)](#)

get_halflife_curve	<i>Interpolate and extrapolate concentrations along the half-life fit</i>
--------------------	---

Description

Concentrations are given as concentrations, not log-concentrations. Take the natural log of conc if the log-concentration is wanted.

Usage

```
get_halflife_curve(
  object,
  tout = NULL,
  n = 50,
  extrapolate_earlier = FALSE,
  extrapolate_later = TRUE
)
```

Arguments

object	A PKNCAResults or PKNCAdata object
tout	Times for output. The same times are used for every group and interval. If NULL (the default), n equally-spaced times spanning the concentrations used for the fit are used instead.
n	The number of equally-spaced times to generate when tout is NULL. It is ignored when tout is given.
extrapolate_earlier, extrapolate_later	Should concentrations be extrapolated before the first (extrapolate_earlier) or after the last (extrapolate_later) concentration used for the fit? Times outside the fit that are not extrapolated give an NA concentration.

Details

Like `stats::approx()`, give either `tout` for specific times or `n` for equally-spaced times. With `n`, the times span the concentrations used for the fit (`time_first` to `time_last` from `get_half-life_fit()`), so `tout` is required to extrapolate.

Value

A data.frame with the grouping columns, the interval start and end times, and the columns:

- `time`: the time of the concentration, on the same scale as the times in the concentration data
- `conc`: the concentration on the half-life fit at that time

`conc` is NA where the half-life could not be calculated or was excluded, and where extrapolation was requested but not allowed. Groups without a fit give a single row with an NA time when `tout` is NULL, since no times can be generated for them.

See Also

`get_half-life_fit()` for the slope and intercept of the fit, and `get_half-life_points()` for the concentrations used for it

Examples

```
o_conc <- PKNCAconc(Theoph, conc~Time|Subject)
o_data <- PKNCAdata(o_conc, intervals = data.frame(start = 0, end = Inf, half.life = TRUE))
o_nca <- pk.nca(o_data)
# Equally-spaced times across the fit
head(get_half-life_curve(o_nca))
# Specific times, extrapolating past the last concentration used
get_half-life_curve(o_nca, tout = c(12, 24, 36))
```

`get_half-life_fit`

Get the half-life fit line for each interval

Description

The half-life fit is the log-linear regression of concentration on time, $\log(\text{conc}) = \text{intercept} + \text{slope} \times \text{time}$. Concentrations along the line are $\exp(\text{intercept} + \text{slope} \times \text{time})$.

Usage

```
get_half-life_fit(object)
```

Arguments

`object` A PKNCAresults or PKNCAdata object

Details

Times in a `PKNCAResults` object are relative to the start of the interval, but `time_first`, `time_last`, and the time scale of `intercept` are on the same scale as the times in the concentration data so that the line can be drawn with the observed concentrations.

Value

A `data.frame` with one row for each group and interval where half-life was calculated. Along with the grouping columns and the interval start and end times, it has the columns:

- `intercept`: the natural log of the concentration where the line crosses time 0
- `slope`: the slope of the line, $-\lambda \cdot z$
- `time_first`, `time_last`: the first and last times of the concentrations used for the fit

`intercept` and `slope` are NA when the half-life could not be calculated or was excluded.

See Also

`get_half-life_points()` to see which concentrations were used for the fit

Examples

```
o_conc <- PKNCAconc(Theoph, conc~Time|Subject)
o_data <- PKNCAdata(o_conc, intervals = data.frame(start = 0, end = Inf, half.life = TRUE))
o_nca <- pk.nca(o_data)
get_half-life_fit(o_nca)
```

<code>get_half-life_points</code>	<i>Determine which concentrations were used for half-life calculation</i>
-----------------------------------	---

Description

Determine which concentrations were used for half-life calculation

Usage

```
get_half-life_points(object)
```

Arguments

`object` A `PKNCAResults` or `PKNCAdata` object

Value

A logical vector with `TRUE` if the point was used for half-life (including concentrations below the limit of quantification within the range of times for calculation), `FALSE` if it was not used for half-life but the half-life was calculated for the interval, and `NA` if half-life was not calculated for the interval. If a row is excluded from all calculations, it is set to `NA` as well.

Examples

```
o_conc <- PKNCAconc(Theoph, conc~Time|Subject)
o_data <- PKNCAdata(o_conc, intervals = data.frame(start = 0, end = Inf, half.life = TRUE))
o_nca <- pk.nca(o_data)
get_half-life_points(o_nca)
```

get_impute_column	<i>Get the name of the intervals column that the imputation methods come from</i>
-------------------	---

Description

get_impute_column() applies the first two of the three readings of impute described in [get_impute_method\(\)](#) and says which column [get_impute_method\(\)](#) reads the methods from. It does not read the column, so it does not check that the column is character.

Usage

```
get_impute_column(intervals, impute)
```

Arguments

intervals	the data.frame of intervals
impute	the imputation definition – either the name of a column in intervals (character scalar), NA to look for a generic "impute" column, or the imputation method string. Must be an atomic scalar; a list (even of length 1) is rejected.

Value

The name of the column of intervals that the methods come from (character scalar): impute when it names a column of intervals, or "impute" when impute is NA and intervals has a column named "impute". NULL when impute is itself the method string, or is NA and there is no "impute" column, so there is no column.

See Also

Other Imputation: [PKNCA_impute_fun_list\(\)](#), [PKNCA_impute_method](#), [assert_impute_method\(\)](#), [get_impute_method\(\)](#), [pknca_impute_methods\(\)](#)

Examples

```
intervals <- data.frame(start = 0, end = 24, impute = "start_conc0", my_impute = "start_cmin")
get_impute_column(intervals, impute = "my_impute")
get_impute_column(intervals, impute = NA)
# A method string is not a column
get_impute_column(intervals, impute = "start_predose")
# NA without an "impute" column is not a column either
get_impute_column(intervals[, c("start", "end")], impute = NA)
```

get_impute_method	<i>Get the imputation methods for the intervals</i>
-------------------	---

Description

get_impute_method() is the rule `pk.nca()` uses to find the imputation methods for each interval, from the `impute` argument of `PKNCAdata()` and the intervals. It reads `impute` one of three ways, in this order:

Usage

```
get_impute_method(intervals, impute)
```

Arguments

<code>intervals</code>	the data.frame of intervals
<code>impute</code>	the imputation definition – either the name of a column in <code>intervals</code> (character scalar), NA to look for a generic "impute" column, or the imputation method string. Must be an atomic scalar; a list (even of length 1) is rejected.

Details

1. If `impute` is the name of a column in `intervals`, the methods are that column's values, one per interval. The name may be any column name, not only "impute".
2. Otherwise, if `impute` is NA and `intervals` has a column named "impute", the methods are that column's values, one per interval.
3. Otherwise `impute` is itself the method (or comma- or space-separated methods) for every interval, and the result is `impute` as a single value. An NA that finds no "impute" column therefore gives a single `NA_character_`, meaning no imputation.

get_impute_method() only reads the specification. It does not check that the methods exist; `assert_impute_method()` does. `get_impute_column()` gives the name of the column the methods are read from, which tells a column from a method string even for an intervals table with one row.

Value

A character vector of imputation specifications: one per row of intervals when read from a column, and of length one otherwise (the method string, or `NA_character_` for no imputation). A column that is not character is an error, except that a column of only NA (which is logical when made with NA) is read as no imputation.

See Also

Other Imputation: `PKNCA_impute_fun_list()`, `PKNCA_impute_method`, `assert_impute_method()`, `get_impute_column()`, `pknca_impute_methods()`

Examples

```

intervals <- data.frame(start = 0, end = 24, impute = "start_conc0")
# The column named by `impute`
get_impute_method(intervals, impute = "impute")
# NA finds the column named "impute"
get_impute_method(intervals, impute = NA)
# NA without an "impute" column means no imputation
get_impute_method(intervals[, c("start", "end")], impute = NA)
# Anything else is the method for every interval
get_impute_method(intervals, impute = "start_predose")

```

getAttributeColumn	<i>Retrieve the value of an attribute column.</i>
--------------------	---

Description

Retrieve the value of an attribute column.

Usage

```
getAttributeColumn(object, attr_name, warn_missing = c("attr", "column"))
```

Arguments

object	The object to extract the attribute value from.
attr_name	The name of the attribute to extract
warn_missing	Give a warning if the "attr"ibute or "column" is missing. Character vector with zero, one, or both of "attr" and "column".

Value

The value of the attribute (or NULL if the attribute is not set or the column does not exist)

getColumnValueOrNot	<i>Get the value from a column in a data frame if the value is a column there, otherwise, the value should be a scalar or the length of the data.</i>
---------------------	---

Description

Get the value from a column in a data frame if the value is a column there, otherwise, the value should be a scalar or the length of the data.

Usage

```
getColumnValueOrNot(data, value, prefix = "X")
```

Arguments

data	A data.frame or similar object
value	A character string giving the name of a column in the data, a scalar, or a vector the same length as the data
prefix	The prefix to use if a column must be added (it will be used as the full column name if it is not already in the dataset or it will be prepended to the maximum column name if not.)

Value

A list with elements named "data", "name" giving the data with a column named "name" with the value in that column.

getDataName.PKNCAconc *Get the name of the element containing the data for the current object.*

Description

Get the name of the element containing the data for the current object.

Usage

```
## S3 method for class 'PKNCAconc'
getDataName(object)

## S3 method for class 'PKNCAdose'
getDataName(object)

## S3 method for class 'PKNCAresults'
getDataName(object)

getDataName(object)

## Default S3 method:
getDataName(object)
```

Arguments

object	The object to get the data name from.
--------	---------------------------------------

Value

A character scalar with the name of the data object (or NULL if the method does not apply).

Methods (by class)

- `getDataName(default)`: If no data name exists, returns NULL.

See Also

Other PKNCA object extractors: [getDepVar\(\)](#), [getIndepVar\(\)](#)

getDepVar	<i>Get the dependent variable (left hand side of the formula) from a PKNCA object.</i>
-----------	--

Description

Get the dependent variable (left hand side of the formula) from a PKNCA object.

Usage

```
getDepVar(x, ...)
```

Arguments

x	The object to extract the formula from
...	Unused

Value

The vector of the dependent variable from the object.

See Also

Other PKNCA object extractors: [getDataName.PKNCAconc\(\)](#), [getIndepVar\(\)](#)

getGroups.PKNCAconc	<i>Get the groups (right hand side after the from a PKNCA object).</i>
---------------------	--

Description

Get the groups (right hand side after the | from a PKNCA object).
Get the groups (right hand side after the | from a PKNCA object).

Usage

```
## S3 method for class 'PKNCAconc'
getGroups(
  object,
  form = stats::formula(object),
  level,
  data = as.data.frame(object),
  sep
)

## S3 method for class 'PKNCAdata'
getGroups(object, ...)

## S3 method for class 'PKNCAdose'
getGroups(...)

## S3 method for class 'PKNCAresults'
getGroups(
  object,
  form = formula(as_PKNCAconc(object)),
  level,
  data = as.data.frame(object),
  sep
)
```

Arguments

<code>object</code>	The object to extract the data from
<code>form</code>	The formula to extract the data from (defaults to the formula from <code>object</code>)
<code>level</code>	optional. If included, this specifies the level(s) of the groups to include. If a numeric scalar, include the first <code>level</code> number of groups. If a numeric vector, include each of the groups specified by the number. If a character vector, include the named group levels.
<code>data</code>	The data to extract the groups from (defaults to the data from <code>object</code>)
<code>sep</code>	Unused (kept for compatibility with the <code>nlme</code> package)
<code>...</code>	Arguments passed to other <code>getGroups</code> functions

Value

A data frame with the (selected) group columns.

A data frame with the (selected) group columns.

getIndepVar	<i>Get the independent variable (right hand side of the formula) from a PKNCA object.</i>
-------------	---

Description

Get the independent variable (right hand side of the formula) from a PKNCA object.

Usage

```
getIndepVar(x, ...)
```

Arguments

x	The object to extract the formula from
...	Unused

Value

The vector of the independent variable from the object.

See Also

Other PKNCA object extractors: [getDataName.PKNCAconc\(\)](#), [getDepVar\(\)](#)

group_by.PKNCAresults	<i>dplyr grouping for PKNCA</i>
-----------------------	---------------------------------

Description

dplyr grouping for PKNCA

Usage

```
## S3 method for class 'PKNCAresults'
group_by(.data, ..., .add = FALSE, .drop = dplyr::group_by_drop_default(.data))

## S3 method for class 'PKNCAconc'
group_by(.data, ..., .add = FALSE, .drop = dplyr::group_by_drop_default(.data))

## S3 method for class 'PKNCAdose'
group_by(.data, ..., .add = FALSE, .drop = dplyr::group_by_drop_default(.data))

## S3 method for class 'PKNCAresults'
ungroup(x, ...)
```

```
## S3 method for class 'PKNCAconc'
ungroup(x, ...)
```

```
## S3 method for class 'PKNCAdose'
ungroup(x, ...)
```

Arguments

.data	A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from dbplyr or dtplyr). See <i>Methods</i> , below, for more details.
...	<data-masking> In group_by(), variables or computations to group by. Computations are always done on the ungrouped data frame. To perform computations on the grouped data, you need to use a separate mutate() step before the group_by(). Computations are not allowed in nest_by(). In ungroup(), variables to remove from the grouping.
.add	When FALSE, the default, group_by() will override existing groups. To add to the existing groups, use .add = TRUE.
.drop	Drop groups formed by factor levels that don't appear in the data? The default is TRUE except when .data has been previously grouped with .drop = FALSE. See group_by_drop_default() for details.
x	A tbl()

See Also

Other dplyr verbs: [filter.PKNCAresults\(\)](#), [inner_join.PKNCAresults\(\)](#), [mutate.PKNCAresults\(\)](#)

group_vars.PKNCAconc *Get grouping variables for a PKNCA object*

Description

Get grouping variables for a PKNCA object

Usage

```
## S3 method for class 'PKNCAconc'
group_vars(x)
```

```
## S3 method for class 'PKNCAdata'
group_vars(x)
```

```
## S3 method for class 'PKNCAdose'
group_vars(x)
```

```
## S3 method for class 'PKNCAresults'
group_vars(x)
```

Arguments

x The PKNCA object

Value

A character vector (possibly empty) of the grouping variables

Functions

- group_vars(PKNCAdata): Get group_vars for a PKNCAdata object from the PKNCAconc object within
- group_vars(PKNCAdose): Get group_vars for a PKNCAdose object
- group_vars(PKNCAresults): Get group_vars for a PKNCAresults object from the PKNCAconc object within

inner_join.PKNCAresults

dplyr joins for PKNCA

Description

dplyr joins for PKNCA

Usage

```
## S3 method for class 'PKNCAresults'
inner_join(
  x,
  y,
  by = NULL,
  copy = FALSE,
  suffix = c(".x", ".y"),
  ...,
  keep = FALSE
)

## S3 method for class 'PKNCAresults'
left_join(
  x,
  y,
  by = NULL,
  copy = FALSE,
  suffix = c(".x", ".y"),
  ...,
  keep = FALSE
)
```

```
## S3 method for class 'PKNCAresults'
right_join(
  x,
  y,
  by = NULL,
  copy = FALSE,
  suffix = c(".x", ".y"),
  ...,
  keep = FALSE
)
```

```
## S3 method for class 'PKNCAresults'
full_join(
  x,
  y,
  by = NULL,
  copy = FALSE,
  suffix = c(".x", ".y"),
  ...,
  keep = FALSE
)
```

```
## S3 method for class 'PKNCAconc'
inner_join(
  x,
  y,
  by = NULL,
  copy = FALSE,
  suffix = c(".x", ".y"),
  ...,
  keep = FALSE
)
```

```
## S3 method for class 'PKNCAconc'
left_join(
  x,
  y,
  by = NULL,
  copy = FALSE,
  suffix = c(".x", ".y"),
  ...,
  keep = FALSE
)
```

```
## S3 method for class 'PKNCAconc'
right_join(
  x,
```

```
y,
by = NULL,
copy = FALSE,
suffix = c(".x", ".y"),
...,
keep = FALSE
)

## S3 method for class 'PKNCAconc'
full_join(
  x,
  y,
  by = NULL,
  copy = FALSE,
  suffix = c(".x", ".y"),
  ...,
  keep = FALSE
)

## S3 method for class 'PKNCAdose'
inner_join(
  x,
  y,
  by = NULL,
  copy = FALSE,
  suffix = c(".x", ".y"),
  ...,
  keep = FALSE
)

## S3 method for class 'PKNCAdose'
left_join(
  x,
  y,
  by = NULL,
  copy = FALSE,
  suffix = c(".x", ".y"),
  ...,
  keep = FALSE
)

## S3 method for class 'PKNCAdose'
right_join(
  x,
  y,
  by = NULL,
  copy = FALSE,
  suffix = c(".x", ".y"),
```

```

    ...,
    keep = FALSE
  )

## S3 method for class 'PKNCAdose'
full_join(
  x,
  y,
  by = NULL,
  copy = FALSE,
  suffix = c(".x", ".y"),
  ...,
  keep = FALSE
)

```

Arguments

x, y	A pair of data frames, data frame extensions (e.g. a tibble), or lazy data frames (e.g. from dbplyr or dtplyr). See <i>Methods</i> , below, for more details.
by	A join specification created with join_by(), or a character vector of variables to join by. If NULL, the default, <code>*_join()</code> will perform a natural join, using all variables in common across x and y. A message lists the variables so that you can check they're correct; suppress the message by supplying <code>by</code> explicitly. To join on different variables between x and y, use a join_by() specification. For example, <code>join_by(a == b)</code> will match <code>x\$a</code> to <code>y\$b</code>. To join by multiple variables, use a join_by() specification with multiple expressions. For example, <code>join_by(a == b, c == d)</code> will match <code>x\$a</code> to <code>y\$b</code> and <code>x\$c</code> to <code>y\$d</code>. If the column names are the same between x and y, you can shorten this by listing only the variable names, like <code>join_by(a, c)</code>. join_by() can also be used to perform inequality, rolling, and overlap joins. See the documentation at ?join_by for details on these types of joins. For simple equality joins, you can alternatively specify a character vector of variable names to join by. For example, <code>by = c("a", "b")</code> joins <code>x\$a</code> to <code>y\$a</code> and <code>x\$b</code> to <code>y\$b</code>. If variable names differ between x and y, use a named character vector like <code>by = c("x_a" = "y_a", "x_b" = "y_b")</code>. To perform a cross-join, generating all combinations of x and y, see cross_join().
copy	If x and y are not from the same data source, and <code>copy</code> is TRUE, then y will be copied into the same src as x. This allows you to join tables across srcs, but it is a potentially expensive operation so you must opt into it.
suffix	If there are non-joined duplicate variables in x and y, these suffixes will be added to the output to disambiguate them. Should be a character vector of length 2.
...	Other parameters passed onto methods.
keep	Should the join keys from both x and y be preserved in the output? If NULL, the default, joins on equality retain only the keys from x, while joins on inequality retain the keys from both inputs.

- If TRUE, all keys from both inputs are retained.
- If FALSE, only keys from x are retained. For right and full joins, the data in key columns corresponding to rows that only exist in y are merged into the key columns from x. Can't be used when joining on inequality conditions.

See Also

Other dplyr verbs: [filter.PKNCResults\(\)](#), [group_by.PKNCResults\(\)](#), [mutate.PKNCResults\(\)](#)

interp.extrap.conc	<i>Interpolate concentrations between measurements or extrapolate concentrations after the last measurement.</i>
--------------------	--

Description

interpolate.conc() and extrapolate.conc() returns an interpolated (or extrapolated) concentration. interp.extrap.conc() will choose whether interpolation or extrapolation is required and will also operate on many concentrations. These will typically be used to estimate the concentration between two measured concentrations or after the last measured concentration. Of note, these functions will not extrapolate prior to the first point.

Usage

```
interp.extrap.conc(  
  conc,  
  time,  
  time.out,  
  lambda.z = NA,  
  clast = pk.calc.clast.obs(conc, time),  
  options = list(),  
  method = NULL,  
  auc.type = "AUCinf",  
  interp.method,  
  extrap.method,  
  ...,  
  conc.blq = NULL,  
  conc.na = NULL,  
  check = TRUE  
)
```

```
interpolate.conc(  
  conc,  
  time,  
  time.out,  
  options = list(),  
  method = NULL,  
  interp.method,
```

```

    conc.blq = NULL,
    conc.na = NULL,
    conc.origin = 0,
    ...,
    check = TRUE
)

extrapolate.conc(
  conc,
  time,
  time.out,
  lambda.z = NA,
  clast = pk.calc.clast.obs(conc, time),
  auc.type = "AUCinf",
  extrap.method,
  options = list(),
  conc.na = NULL,
  conc.blq = NULL,
  ...,
  check = TRUE
)

interp.extrap.conc.dose(
  conc,
  time,
  time.dose,
  route.dose = "extravascular",
  duration.dose = NA,
  time.out,
  out.after = FALSE,
  options = list(),
  conc.blq = NULL,
  conc.na = NULL,
  ...,
  check = TRUE
)

```

Arguments

<code>conc</code>	Measured concentrations
<code>time</code>	Time of the measurement of the concentrations
<code>time.out</code>	Time when interpolation is requested (vector for <code>interp.extrap.conc()</code> , scalar otherwise)
<code>lambda.z</code>	The elimination rate (in units of inverse time) for extrapolation
<code>clast</code>	The last observed concentration above the limit of quantification. If not given, <code>clast</code> is calculated from pk.calc.clast.obs()
<code>options</code>	List of changes to the default PKNCA options (see <code>PKNCA.options()</code>)

method	The method for integration (one of 'lin up/log down', 'lin-log', or 'linear')
auc.type	The type of AUC to compute. Choices are 'AUCinf', 'AUClast', and 'AUCall'.
interp.method, extrapol.method	deprecated in favor of method and auc.type
...	Additional arguments passed to interpolate.conc() or extrapolate.conc().
conc.blq	How to handle a BLQ value that is between above LOQ values? See details for description.
conc.na	How to handle NA concentrations. (See clean.conc.na())
check	Run assert_conc_time() ?
conc.origin	The concentration before the first measurement. conc.origin is typically used to set predose values to zero (default), set a predose concentration for endogenous compounds, or set predose concentrations to NA if otherwise unknown.
time.dose	Time of the dose
route.dose	What is the route of administration ("intravascular" or "extravascular"). See the details for how this parameter is used.
duration.dose	What is the duration of administration? See the details for how this parameter is used.
out.after	Should interpolation occur from the data before (FALSE) or after (TRUE) the interpolated point? See the details for how this parameter is used. It only has a meaningful effect at the instant of an IV bolus dose.

Details

An NA value for the `lambda.z` parameter will prevent extrapolation.

extrap.method 'AUCinf' Use `lambda.z` to extrapolate beyond the last point with the half-life.

'AUCall' If the last point is above the limit of quantification or missing, this is identical to 'AUCinf'. If the last point is below the limit of quantification, then linear interpolation between the Clast and the next BLQ is used for that interval and all additional points are extrapolated as 0.

'AUClast' Extrapolates all points after the last above the limit of quantification as 0.

`duration.dose` and `direction.out` are ignored if `route.dose == "extravascular"`. `direction.out` is ignored if `duration.dose > 0`.

`route.dose` and `duration.dose` affect how interpolation/extrapolation of the concentration occurs at the time of dosing. If `route.dose == "intravascular"` and `duration.dose == 0` then extrapolation occurs for an IV bolus using [pk.calc.c0\(\)](#) with the data after dosing. Otherwise (either `route.dose == "extravascular"` or `duration.dose > 0`), extrapolation occurs using the concentrations before dosing and estimating the half-life (or more precisely, estimating `lambda.z`). Finally, `direction.out` can change the direction of interpolation in cases with `route.dose == "intravascular"` and `duration.dose == 0`. When `direction.out == "before"` interpolation occurs only with data before the dose (as is the case for `route.dose == "extravascular"`), but if `direction.out == "after"` interpolation occurs from the data after dosing.

Value

The interpolated or extrapolated concentration value as a scalar double (or vector for `interp.extrap.conc()`).

Functions

- `interpolate.conc()`: Interpolate concentrations through Tlast (inclusive)
- `extrapolate.conc()`: Extrapolate concentrations after Tlast
- `interp.extrap.conc.dose()`: Interpolate and extrapolate concentrations without interpolating or extrapolating beyond doses.

See Also

`pk.calc.clast.obs()`, `pk.calc.half.life()`, `pk.calc.c0()`

`interp_extrap_conc_method`

Interpolate or extrapolate concentrations using the provided method

Description

Interpolate or extrapolate concentrations using the provided method

Usage

`interpolate_conc_linear(conc_1, conc_2, time_1, time_2, time_out)`

`interpolate_conc_log(conc_1, conc_2, time_1, time_2, time_out)`

`extrapolate_conc_lambdaz(clast, lambda.z, tlast, time_out)`

Arguments

<code>conc_1, conc_2</code>	The concentration at time1 and time2
<code>time_1, time_2</code>	The time value associated with conc1 and conc2
<code>time_out</code>	Time when interpolation is requested
<code>clast</code>	The concentration at the last time above the lower LOQ
<code>lambda.z</code>	The elimination rate (in units of inverse time) for extrapolation
<code>tlast</code>	The time of the last concentration above the lower limit of quantification (LOQ)

Value

The interpolated or extrapolated value using the correct method

interval_add_impute	<i>Add or remove an imputation method in the intervals of a PKNCAdata object</i>
---------------------	--

Description

Add or remove an imputation method in the intervals of a PKNCAdata object

Usage

```
interval_add_impute(
  data,
  target_impute,
  after = Inf,
  target_params = NULL,
  target_groups = NULL,
  ...
)

interval_remove_impute(
  data,
  target_impute,
  target_params = NULL,
  target_groups = NULL,
  ...
)
```

Arguments

data	A PKNCAdata object or a data.frame of intervals.
target_impute	The imputation method to add or remove, as a character string (see PKNCA_impute_method).
after	Where to insert the method within any imputation already present, following base::append() : 0 makes it first and Inf (the default) makes it last. A method that is already present is moved to the requested position rather than duplicated.
target_params	Restrict the change to these NCA parameters. NULL (the default) applies it to every parameter being calculated.
target_groups	A data.frame of group values restricting the change to matching intervals. Every column must match (and) for at least one row of target_groups (or). NULL (the default) applies the change to all intervals.
...	Ignored.

Value

The input with the imputation updated. An interval is split into more than one row when its parameters no longer share the same imputation, and rows that come to share every value are merged.

See Also

[interval_add_param\(\)](#), [PKNCAdata\(\)](#), and the vignette "Data Imputation"

Other Interval specifications: [add.interval.col\(\)](#), [check.interval.specification\(\)](#), [choose.auc.intervals\(\)](#), [get.interval.cols\(\)](#), [get.parameter.deps\(\)](#), [interval_add_param\(\)](#), [interval_add_secondary\(\)](#), [pknca_cdisc_codes\(\)](#), [pknca_check_parameter_classification\(\)](#), [pknca_concepts\(\)](#), [pknca_interval_table\(\)](#), [pknca_match_route\(\)](#), [pknca_parameter_table\(\)](#), [pknca_presets\(\)](#), [pknca_ref\(\)](#)

Examples

```
intervals <- data.frame(start = 0, end = 24, cmax = TRUE, half.life = TRUE)
# Impute a starting concentration for everything in the interval
interval_add_impute(intervals, target_impute = "start_conc0")
# ... but not for half-life, which splits the interval into two rows
interval_add_impute(intervals, target_impute = "start_conc0", target_params = "cmax")
```

interval_add_param	<i>Add or remove parameters in the intervals of a PKNCAdata object</i>
--------------------	--

Description

Add or remove parameters in the intervals of a PKNCAdata object

Usage

```
interval_add_param(
  data,
  param = NULL,
  param_pattern = NULL,
  target_groups = NULL,
  ...
)

interval_remove_param(
  data,
  param = NULL,
  param_pattern = NULL,
  target_groups = NULL,
  ...
)
```

Arguments

data	A PKNCAdata object or a data.frame of intervals.
param	A character vector of NCA parameter names.
param_pattern	A character vector of regular expressions matching NCA parameter names. One or both of param and param_pattern must be given.

`target_groups` A data.frame of group values restricting the change to matching intervals. Every column must match (and) for at least one row of `target_groups` (or). NULL (the default) applies the change to all intervals.

... Ignored.

Value

The input with the parameters added or removed.

See Also

[interval_add_impute\(\)](#), [get.interval.cols\(\)](#)

Other Interval specifications: [add.interval.col\(\)](#), [check.interval.specification\(\)](#), [choose.auc.intervals\(\)](#), [get.interval.cols\(\)](#), [get.parameter.deps\(\)](#), [interval_add_impute\(\)](#), [interval_add_secondary\(\)](#), [pkncdisc_codes\(\)](#), [pkncdisc_check_parameter_classification\(\)](#), [pkncdisc_concepts\(\)](#), [pkncdisc_interval_table\(\)](#), [pkncdisc_match_route\(\)](#), [pkncdisc_parameter_table\(\)](#), [pkncdisc_presets\(\)](#), [pkncdisc_ref\(\)](#)

Examples

```
intervals <- data.frame(start = 0, end = 24, cmax = TRUE, tmax = TRUE)
interval_add_param(intervals, param = c("auclast", "half.life"))
# At least one parameter must remain, so removing every one is an error
interval_remove_param(intervals, param_pattern = "^tmax$")
```

interval_add_secondary

Link a secondary parameter to the interval it is calculated against

Description

A secondary parameter needs a result from a second profile: renal clearance divides an amount excreted in urine by a plasma AUC, an accumulation ratio compares one dosing interval with another, and a metabolite ratio compares two analytes. This adds the request and the linkage columns (`interval_id` on the reference interval and `<param>_ref` on the intervals calculating the parameter) that [pk.nca\(\)](#) reads.

Usage

```
interval_add_secondary(
  data,
  param,
  reference = NULL,
  target_groups = NULL,
  ref_id = NULL,
  ...
)
```

Arguments

data	A PKNCAdata object or a data.frame of intervals.
param	The name of one secondary NCA parameter (see pknca_parameter_table() for which parameters are secondary).
reference	A data.frame describing the reference interval: its columns are start, end, and/or group columns of the intervals, every column must match (and) for at least one of its rows (or). A named list is accepted and coerced. When no interval matches, one is created and the creation is reported with a <code>pknca_message_secondary_created_interval</code> message. NULL (the default) takes the <code>group_ref</code> given to PKNCAdata() (resolved for <code>param</code> when it is parameter-specific), or derives the reference from the data (see Details).
target_groups	A data.frame of group values restricting the parameter request to matching intervals, with the same matching rules as <code>reference</code> . NULL (the default) requests it on every interval that is not a reference interval.
ref_id	The <code>interval_id</code> to give the reference interval. The default of NULL keeps an identifier the reference rows already have and otherwise generates one matching the class of the <code>interval_id</code> column. A <code>ref_id</code> that differs from the identifier the reference rows already have renames that interval, which is an error when another interval points at the old identifier or already has <code>ref_id</code> . Reference rows that already carry more than one <code>interval_id</code> are an error (<code>pknca_error_secondary_ref_ambiguous_spec</code>) when <code>ref_id</code> is given, as it cannot name more than one interval.
...	Ignored.

Details

The reference interval gains whatever the linked calculation reads from it (the plasma AUC for a renal clearance, for example) without being announced, as calculating any dependency is.

An interval that already names a different reference for `param` is left alone with a `pknca_warning_secondary_ref_exists` warning, so that the helper never silently re-points an analysis.

With `reference = NULL` the reference is derived from the data, which needs a PKNCAdata object (a bare intervals data.frame carries no concentrations to derive it from). The `group_ref` given to [PKNCAdata\(\)](#) is used when it is set. Otherwise the derivation applies where the parameter is measured on an interval collection while everything it references is a spot sample (renal clearance) and the concentration data declare a collection volume: each interval takes the profile with no collection volume that differs from its own in the fewest group columns. This is the same derivation [pk.nca\(\)](#) makes on its own, written out into the returned intervals instead of being ephemeral. Anything the derivation cannot resolve is an error here (`pknca_error_secondary_needs_ref`) rather than the NA result the automatic path gives, so narrow the request with `target_groups` when some intervals are not meant to calculate `param`.

When the parameter pairs an interval collection with spot-sample references (renal clearance: an amount excreted against a plasma AUC) and the reference interval is created by the PKNCAdata method, the created interval spans the collections whole: a collection that begins inside the interval contributes its full amount (see [pk.nca\(\)](#)), so a collection running past the interval's end extends the created reference's end to time + duration of the latest collection. An explicit end in reference

overrides the extension, and the data.frame method (which has no concentration data) copies the test interval's times unchanged.

Value

The input with the parameter requested and the linkage columns set, after `check.interval.specification()`.

See Also

`interval_add_param()`, `pknca_ref()`, `pk.nca()`

Other Interval specifications: `add.interval.col()`, `check.interval.specification()`, `choose.auc.intervals()`, `get.interval.cols()`, `get.parameter.deps()`, `interval_add_impute()`, `interval_add_param()`, `pknca_cdisc_codes()`, `pknca_check_parameter_classification()`, `pknca_concepts()`, `pknca_interval_table()`, `pknca_match_route()`, `pknca_parameter_table()`, `pknca_presets()`, `pknca_ref()`

Examples

```
intervals <-
  data.frame(
    PCSPEC = c("plasma", "urine"),
    start = 0, end = 24,
    auclast = c(TRUE, FALSE),
    ae = c(FALSE, TRUE)
  )
interval_add_secondary(
  intervals,
  param = "clr.last",
  reference = data.frame(PCSPEC = "plasma")
)
```

```
is_sparse_pk.PKNCAconc
```

Is a PKNCA object used for sparse PK?

Description

Is a PKNCA object used for sparse PK?

Usage

```
## S3 method for class 'PKNCAconc'
is_sparse_pk(object)

## S3 method for class 'PKNCAdata'
is_sparse_pk(object)

## S3 method for class 'PKNCAresults'
is_sparse_pk(object)

is_sparse_pk(object)
```

Arguments

object The object to see if it includes sparse PK

Value

TRUE if sparse and FALSE if dense (not sparse)

<code>model.frame.PKNCAconc</code>	<i>Extract the columns used in the formula (in order) from a PKNCAconc or PKNCAdose object.</i>
------------------------------------	---

Description

Extract the columns used in the formula (in order) from a PKNCAconc or PKNCAdose object.

Usage

```
## S3 method for class 'PKNCAconc'
model.frame(formula, ...)

## S3 method for class 'PKNCAdose'
model.frame(formula, ...)
```

Arguments

formula The object to use (parameter name is formula to use the generic function)
... Unused

Value

A data frame with the columns from the object in formula order.

<code>mutate.PKNCAresults</code>	<i>dplyr mutate-based modification for PKNCA</i>
----------------------------------	--

Description

dplyr mutate-based modification for PKNCA

Usage

```
## S3 method for class 'PKNCResults'  
mutate(.data, ...)  
  
## S3 method for class 'PKNCaconc'  
mutate(.data, ...)  
  
## S3 method for class 'PKNCAdose'  
mutate(.data, ...)
```

Arguments

- .data A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from dbplyr or dtplyr). See *Methods*, below, for more details.
- ... [<data-masking>](#) Name-value pairs. The name gives the name of the column in the output.
The value can be:
- A vector of length 1, which will be recycled to the correct length.
 - A vector the same length as the current group (or the whole data frame if ungrouped).
 - NULL, to remove the column.
 - A data frame or tibble, to create multiple columns in the output.

See Also

Other dplyr verbs: [filter.PKNCResults\(\)](#), [group_by.PKNCResults\(\)](#), [inner_join.PKNCResults\(\)](#)

normalize	<i>Normalize parameters in a PKNCResults object or data.frame</i>
-----------	---

Description

Normalize parameters in a PKNCResults object or data.frame

Usage

```
normalize(object, norm_table, parameters, suffix)
```

Arguments

- object A PKNCResults object or result data.frame
- norm_table data.frame with group columns, normalization values (normalization), and units (unit)
- parameters character vector of parameter names to normalize
- suffix character value to add for the normalized parameter names

Value

A data.frame with normalized parameters

normalize_by_col	<i>Normalize PKNCA results by a column of the concentration data</i>
------------------	--

Description

Normalize PKNCA results by a column of the concentration data

Usage

```
normalize_by_col(object, col, unit, parameters, suffix)
```

Arguments

object	A PKNCAresults object
col	The column name from PKNCAconc to use for the normalization groups
unit	The unit of the previous column for normalization. Can be a column name in PKNCAconc or a single value.
parameters	Character vector of parameter names to normalize
suffix	Suffix to add to the normalized parameter code names

Value

A data.frame with normalized parameters

normalize_exclude	<i>Normalize the exclude column by setting blanks to NA</i>
-------------------	---

Description

Normalize the exclude column by setting blanks to NA

Usage

```
normalize_exclude(object)
```

Arguments

object	The object to extract the exclude column from
--------	---

Value

The exclude vector where NA indicates not to exclude and anything else indicates to exclude.

parse_formula_to_cols *Convert a formula representation to the columns for input data*

Description

Convert a formula representation to the columns for input data

Usage

```
parse_formula_to_cols(form)
```

Arguments

form the formula (or something coercible into a formula) to extract into its parts

Value

A list of column names for various formula parts

See Also

Other Formula parsing: [findOperator\(\)](#)

pk.business	<i>Run any function with a maximum missing fraction of X and 0s possibly counting as missing. The maximum fraction missing comes from PKNCA.options("max.missing").</i>
-------------	---

Description

Note that all missing values are removed prior to calling the function.

Usage

```
pk.business(FUN, zero.missing = FALSE, max.missing)
```

Arguments

FUN	function to run. The function is called as FUN(x, ...) with missing values removed.
zero.missing	Are zeros counted as missing? If TRUE then include them in the missing count.
max.missing	The maximum fraction of the data allowed to be missing (a number between 0 and 1, inclusive).

Value

A version of FUN that can be called with parameters that are checked for missingness (and zeros) with missing (and zeros) removed before the call. If max.missing is exceeded, then NA is returned.

Examples

```
my_mean <- pk.business(FUN=mean)
mean(c(1:3, NA))
# Less than half missing results in the summary statistic of the available
# values.
my_mean(c(1:3, NA))
# More than half missing results in a missing value
my_mean(c(1:3, rep(NA, 4)))
```

pk.calc.ae

Calculate amount excreted (typically in urine or feces)

Description

Calculate amount excreted (typically in urine or feces)

Usage

```
pk.calc.ae(conc, volume, check = TRUE)
```

Arguments

conc	Measured concentrations
volume	The volume (or mass) of the sample
check	Should the concentration and volume data be checked?

Details

ae is `sum(conc*volume)`.

The units for the concentration and volume should match such that `sum(conc*volume)` has units of mass or moles.

Value

The amount excreted during the interval

See Also

[pk.calc.clr\(\)](#), [pk.calc.fe\(\)](#)

Other Urine/Excretion parameters: [pk.calc.clr\(\)](#), [pk.calc.erint\(\)](#), [pk.calc.erlst\(\)](#), [pk.calc.ermx\(\)](#), [pk.calc.ertlst\(\)](#), [pk.calc.ertmx\(\)](#), [pk.calc.fe\(\)](#), [pk.calc.volpk\(\)](#)

pk.calc.aucabove	<i>Calculate the AUC above a given concentration</i>
------------------	--

Description

Concentrations below the given concentration (conc_above) will be set to zero.

Usage

```
pk.calc.aucabove(conc, time, conc_above = NA_real_, ..., options = list())
```

Arguments

conc	Measured concentrations
time	Time of the measurement of the concentrations
conc_above	The concentration threshold to calculate AUC above. Must be finite (Inf/-Inf are not allowed); if NA, no AUC is calculated.
...	Extra arguments. Currently, the only extra argument that is used is method as described in the details section.
options	List of changes to the default PKNCA options (see PKNCA.options())

Value

The AUC of the concentration above the limit

pk.calc.aucinf.obs_sparse	<i>Sparse estimators for the AUC and AUMC to infinity</i>
---------------------------	---

Description

These are the FUN_sparse of aucinf.obs and aumcinf.obs: with sparse PK, [pk.nca\(\)](#) estimates them as the sparse AUClast or AUMClast (see [pk.calc.auclast_sparse\(\)](#)) plus the extrapolation from the mean concentration at tlast with the lambda.z of the mean profile, with the standard error of Yuan (1993) and its extension to the AUMC. The sparse_lambda_z_se option of [PKNCA.options\(\)](#) chooses whether the standard error adds the uncertainty of lambda.z with the delta method ("delta", the default) or treats it as known ("none", Yuan's method); see vignette("v24-sparse-auc-to-infinity").

Usage

```
pk.calc.aucinf.obs_sparse(
  conc,
  time,
  subject,
  lambda.z,
  lambda.z.time.first,
  lambda.z.time.last,
  lambda.z.n.points,
  ...,
  options = list()
)

pk.calc.aumcinf.obs_sparse(
  conc,
  time,
  subject,
  lambda.z,
  lambda.z.time.first,
  lambda.z.time.last,
  lambda.z.n.points,
  ...,
  options = list()
)
```

Arguments

conc	Measured concentrations
time	Time of the measurement of the concentrations
subject	Subject identifiers (may be any class; may not be null). A missing subject marks an imputed concentration (such as the zero that the <code>start_conc0</code> imputation adds at the start of an interval): a time where every subject is missing has a known concentration, which enters sparse estimates but not their variance.
lambda.z	The elimination rate of the mean profile
lambda.z.time.first, lambda.z.time.last, lambda.z.n.points	The first and last time and the number of points of the half-life fit to the mean profile
...	For functions other than <code>pk.calc.auxc</code> , these values are passed to <code>pk.calc.auxc</code>
options	List of changes to the default PKNCA options (see <code>PKNCA.options()</code>)

Value

A data.frame with the point estimate, its standard error, and the degrees of freedom, named for the parameter (`aucinf.obs`, `aucinf.obs_se`, and `aucinf.obs_df`, or the `aumcinf.obs` equivalents)

Functions

- `pk.calc.aumcinf.obs_sparse()`: Sparse AUMCinf,obs

References

Yuan J. Estimation of variance for AUC in animal studies. *Journal of Pharmaceutical Sciences*. 1993;82(7):761-763. doi:10.1002/jps.2600820718

See Also

Other Sparse Methods: [as_sparse_pk\(\)](#), [pk.calc.aucivlast_sparse\(\)](#), [pk.calc.auclast_sparse\(\)](#), [pk.calc.sparse_auc\(\)](#), [pk.calc.sparse_aumc\(\)](#), [sparse_auc_weight_linear\(\)](#), [sparse_bootstrap\(\)](#), [sparse_mean\(\)](#), [summary.PKNCResults_sparse_bootstrap\(\)](#)

pk.calc.aucivlast_sparse

Sparse AUC and AUMC for IV bolus dosing with C₀ back-extrapolation

Description

These are the FUN_sparse of aucivlast, aucivall, aucivinf.obs, aumcivlast, aumcivall, and aumcivinf.obs: with sparse PK and no concentration at time 0, C_0 is back-extrapolated from the mean profile with the methods of [pk.calc.c0\(\)](#) (log-linear from the first two means when the profile declines), and the area from time 0 to the first sample is added to the sparse AUC (see [pk.calc.auclast_sparse\(\)](#), [pk.calc.sparse_auc\(\)](#) for AUCall, and [pk.calc.aucivinf.obs_sparse\(\)](#)). C_0 is a function of the means, so its uncertainty is added to the standard error with the delta method. With the linear trapezoidal rule, the AUMC from time 0 to the first sample does not depend on C_0 . A nonzero concentration measured at time 0 is used as C_0 ; a zero there, measured or imputed (as by the start_conc0 imputation), is not, as for [pk.calc.c0\(\)](#).

Usage

```
pk.calc.aucivlast_sparse(conc, time, subject, ..., options = list())
```

```
pk.calc.aucivall_sparse(conc, time, subject, ..., options = list())
```

```
pk.calc.aucivinf.obs_sparse(
  conc,
  time,
  subject,
  lambda.z,
  lambda.z.time.first,
  lambda.z.time.last,
  lambda.z.n.points,
  ...,
  options = list()
)
```

```
pk.calc.aumcivlast_sparse(conc, time, subject, ..., options = list())
```

```
pk.calc.aumcivall_sparse(conc, time, subject, ..., options = list())

pk.calc.aumcivinf.obs_sparse(
  conc,
  time,
  subject,
  lambda.z,
  lambda.z.time.first,
  lambda.z.time.last,
  lambda.z.n.points,
  ...,
  options = list()
)
```

Arguments

conc	Measured concentrations
time	Time of the measurement of the concentrations
subject	Subject identifiers (may be any class; may not be null). A missing subject marks an imputed concentration (such as the zero that the <code>start_conc0</code> imputation adds at the start of an interval): a time where every subject is missing has a known concentration, which enters sparse estimates but not their variance.
...	For functions other than <code>pk.calc.auxc</code> , these values are passed to <code>pk.calc.auxc</code>
options	List of changes to the default PKNCA options (see <code>PKNCA.options()</code>)
lambda.z	The elimination rate of the mean profile
lambda.z.time.first, lambda.z.time.last, lambda.z.n.points	The first and last time and the number of points of the half-life fit to the mean profile

Value

A data.frame with the point estimate, its standard error, and the degrees of freedom, named for the parameter (for example, `aucivlast`, `aucivlast_se`, and `aucivlast_df`)

Functions

- `pk.calc.aucivall_sparse()`: Sparse AUCall for IV bolus dosing
- `pk.calc.aucivinf.obs_sparse()`: Sparse AUCinf,obs for IV bolus dosing
- `pk.calc.aumcivlast_sparse()`: Sparse AUMClast for IV bolus dosing
- `pk.calc.aumcivall_sparse()`: Sparse AUMCall for IV bolus dosing
- `pk.calc.aumcivinf.obs_sparse()`: Sparse AUMCinf,obs for IV bolus dosing

See Also

Other Sparse Methods: [as_sparse_pk\(\)](#), [pk.calc.aucinf.obs_sparse\(\)](#), [pk.calc.auclast_sparse\(\)](#), [pk.calc.sparse_auc\(\)](#), [pk.calc.sparse_aumc\(\)](#), [sparse_auc_weight_linear\(\)](#), [sparse_bootstrap\(\)](#), [sparse_mean\(\)](#), [summary.PKNCAresults_sparse_bootstrap\(\)](#)

pk.calc.auclast_sparse

Sparse estimators for the AUC and AUMC to the last measured concentration

Description

These are the FUN_sparse of auclast, aumclast, aucall, and aumcall: with sparse PK, [pk.nca\(\)](#) estimates those parameters from the pooled individual samples with the Bailer point estimate and the Nedelman-Jia/Holder standard error rather than integrating the arithmetic-mean profile. They wrap [pk.calc.sparse_auc\(\)](#) and [pk.calc.sparse_aumc\(\)](#) (with `auc.type = "AUClast" or "AUCall"`), reporting the results under the unified parameter names. AUCall adds the triangle from tlast to the next time, whose mean is zero.

Usage

```
pk.calc.auclast_sparse(conc, time, subject, ..., options = list())
```

```
pk.calc.aumclast_sparse(conc, time, subject, ..., options = list())
```

```
pk.calc.aucall_sparse(conc, time, subject, ..., options = list())
```

```
pk.calc.aumcall_sparse(conc, time, subject, ..., options = list())
```

Arguments

conc	Measured concentrations
time	Time of the measurement of the concentrations
subject	Subject identifiers (may be any class; may not be null). A missing subject marks an imputed concentration (such as the zero that the <code>start_conc0</code> imputation adds at the start of an interval): a time where every subject is missing has a known concentration, which enters sparse estimates but not their variance.
...	For functions other than <code>pk.calc.auxc</code> , these values are passed to <code>pk.calc.auxc</code>
options	List of changes to the default PKNCA options (see <code>PKNCA.options()</code>)

Details

The sparse variance theory is defined for the linear trapezoidal rule only, so these ignore the `auc.method` option; [pk.nca\(\)](#) says so when the option is set to anything else.

Value

A data.frame with the point estimate, its standard error, and the degrees of freedom, named for the parameter (for example, `auclast`, `auclast_se`, and `auclast_df`)

Functions

- pk.calc.aumclast_sparse(): Sparse AUMClast
- pk.calc.aucall_sparse(): Sparse AUCall
- pk.calc.aumcall_sparse(): Sparse AUMCall

See Also

Other Sparse Methods: [as_sparse_pk\(\)](#), [pk.calc.aucinf.obs_sparse\(\)](#), [pk.calc.aucivlast_sparse\(\)](#), [pk.calc.sparse_auc\(\)](#), [pk.calc.sparse_aumc\(\)](#), [sparse_auc_weight_linear\(\)](#), [sparse_bootstrap\(\)](#), [sparse_mean\(\)](#), [summary.PKNCAresults_sparse_bootstrap\(\)](#)

pk.calc.aucpext	<i>Calculate the AUC percent extrapolated</i>
-----------------	---

Description

Calculate the AUC percent extrapolated

Usage

```
pk.calc.aucpext(auclast, aucinf)
```

Arguments

auclast	the area under the curve from time 0 to the last measurement above the limit of quantification
aucinf	the area under the curve from time 0 to infinity

Details

aucpext is $100 \times (1 - \text{auclast} / \text{aucinf})$.

Value

The numeric value of the AUC percent extrapolated or NA_real_ if any of the following are true is.na(aucinf), is.na(auclast), aucinf <= 0, or auclast <= 0.

See Also

Other Half-life and elimination: [adj.r.squared\(\)](#), [pk.calc.half.life\(\)](#), [pk.calc.thalf.eff\(\)](#)

pk.calc.auxc*Compute the Area Under the (Moment) Curve*

Description

Compute the area under the curve (AUC) and the area under the moment curve (AUMC) for pharmacokinetic (PK) data. AUC and AUMC are used for many purposes when analyzing PK in drug development.

Usage

```
pk.calc.auxc(  
  conc,  
  time,  
  interval = c(0, Inf),  
  clast = pk.calc.clast.obs(conc, time, check = FALSE),  
  lambda.z = NA,  
  auc.type = c("AUClast", "AUCinf", "AUCall"),  
  options = list(),  
  method = NULL,  
  conc.blq = NULL,  
  conc.na = NULL,  
  check = TRUE,  
  fun_linear,  
  fun_log,  
  fun_inf  
)  
  
pk.calc.auc(conc, time, ..., options = list())  
  
pk.calc.auc.last(conc, time, ..., options = list())  
  
pk.calc.auc.inf(conc, time, ..., options = list(), lambda.z)  
  
pk.calc.auc.inf.obs(conc, time, clast.obs, ..., options = list(), lambda.z)  
  
pk.calc.auc.inf.pred(conc, time, clast.pred, ..., options = list(), lambda.z)  
  
pk.calc.auc.all(conc, time, ..., options = list())  
  
pk.calc.aumc(conc, time, ..., options = list())  
  
pk.calc.aumc.last(conc, time, ..., options = list())  
  
pk.calc.aumc.inf(conc, time, ..., options = list(), lambda.z)  
  
pk.calc.aumc.inf.obs(conc, time, clast.obs, ..., options = list(), lambda.z)
```

```
pk.calc.aumc.inf.pred(conc, time, clast.pred, ..., options = list(), lambda.z)
```

```
pk.calc.aumc.all(conc, time, ..., options = list())
```

Arguments

conc	Measured concentrations
time	Time of the measurement of the concentrations
interval	Numeric vector of two numbers for the start and end time of integration
clast, clast.obs, clast.pred	The last concentration above the limit of quantification; this is used for AUCinf calculations. If provided as clast.obs (observed clast value, default), AUCinf is AUCinf.obs. If provided as clast.pred, AUCinf is AUCinf.pred.
lambda.z	The elimination rate (in units of inverse time) for extrapolation
auc.type	The type of AUC to compute. Choices are 'AUCinf', 'AUClast', and 'AUCall'.
options	List of changes to the default PKNCA options (see PKNCA.options())
method	The method for integration (one of 'lin up/log down', 'lin-log', or 'linear')
conc.blq	How to handle a BLQ value that is between above LOQ values? See details for description.
conc.na	How to handle NA concentrations. (See clean.conc.na())
check	Run assert_conc_time() ?
fun_linear	The function to use for integration of the linear part of the curve (not required for AUC or AUMC functions)
fun_log	The function to use for integration of the logarithmic part of the curve (if log integration is used; not required for AUC or AUMC functions)
fun_inf	The function to use for extrapolation from the final measurement to infinite time (not required for AUC or AUMC functions).
...	For functions other than pk.calc.auxc, these values are passed to pk.calc.auxc

Details

pk.calc.auc.last is simply a shortcut setting the interval parameter to c(0, "last").

Extrapolation beyond Clast occurs using the half-life and Clast,obs; Clast,pred is not yet supported.

If all conc input are zero, then the AU(M)C is zero.

You probably do not want to call pk.calc.auxc(). Usually, you will call one of the other functions for calculating AUC like pk.calc.auc.last(), pk.calc.auc.inf.obs(), etc.

Value

A numeric value for the AU(M)C.

Functions

- `pk.calc.auc()`: Compute the area under the curve
- `pk.calc.auc.last()`: Compute the AUClast.
- `pk.calc.auc.inf()`: Compute the AUCinf
- `pk.calc.auc.inf.obs()`: Compute the AUCinf with the observed Clast.
- `pk.calc.auc.inf.pred()`: Compute the AUCinf with the predicted Clast.
- `pk.calc.auc.all()`: Compute the AUCall.
- `pk.calc.aumc()`: Compute the area under the moment curve
- `pk.calc.aumc.last()`: Compute the AUMClast.
- `pk.calc.aumc.inf()`: Compute the AUMCinf
- `pk.calc.aumc.inf.obs()`: Compute the AUMCinf with the observed Clast.
- `pk.calc.aumc.inf.pred()`: Compute the AUMCinf with the predicted Clast.
- `pk.calc.aumc.all()`: Compute the AUMCall.

References

Gabrielsson J, Weiner D. "Section 2.8.1 Computation methods - Linear trapezoidal rule." Pharmacokinetic & Pharmacodynamic Data Analysis: Concepts and Applications, 4th Edition. Stockholm, Sweden: Swedish Pharmaceutical Press, 2000. 162-4.

Gabrielsson J, Weiner D. "Section 2.8.3 Computation methods - Log-linear trapezoidal rule." Pharmacokinetic & Pharmacodynamic Data Analysis: Concepts and Applications, 4th Edition. Stockholm, Sweden: Swedish Pharmaceutical Press, 2000. 164-7.

See Also

[clean.conc.blq\(\)](#)

Other AUC calculations: [pk.calc.auxcint\(\)](#), [pk.calc.auxciv\(\)](#)

Examples

```
myconc <- c(0, 1, 2, 1, 0.5, 0.25, 0)
mytime <- c(0, 1, 2, 3, 4, 5, 6)
pk.calc.auc(myconc, mytime, interval=c(0, 6))
pk.calc.auc(myconc, mytime, interval=c(0, Inf))
```

pk.calc.auxcint	<i>Calculate AUXC (AUC or AUMC) over an interval with interpolation/extrapolation</i>
-----------------	---

Description

Calculates AUC or AUMC over a given interval, optionally interpolating or extrapolating concentrations.

Usage

```
pk.calc.auxcint(
  conc,
  time,
  interval = NULL,
  start = NULL,
  end = NULL,
  clast = pk.calc.clast.obs(conc, time),
  lambda.z = NA,
  time.dose = NULL,
  route = "extravascular",
  duration.dose = 0,
  auc.type = c("AUClast", "AUCinf", "AUCall"),
  options = list(),
  method = NULL,
  conc.blq = NULL,
  conc.na = NULL,
  check = TRUE,
  fun_linear,
  fun_log,
  fun_inf,
  ...
)

pk.calc.aucint(conc, time, ..., options = list())

pk.calc.aucint.last(
  conc,
  time,
  start = NULL,
  end = NULL,
  time.dose,
  route = "extravascular",
  duration.dose = 0,
  ...,
  options = list()
)
```

```
pk.calc.aucint.all(  
  conc,  
  time,  
  start = NULL,  
  end = NULL,  
  time.dose,  
  route = "extravascular",  
  duration.dose = 0,  
  ...,  
  options = list()  
)  
  
pk.calc.aucint.inf.obs(  
  conc,  
  time,  
  start = NULL,  
  end = NULL,  
  time.dose,  
  lambda.z,  
  clast.obs,  
  route = "extravascular",  
  duration.dose = 0,  
  ...,  
  options = list()  
)  
  
pk.calc.aucint.inf.pred(  
  conc,  
  time,  
  start = NULL,  
  end = NULL,  
  time.dose,  
  lambda.z,  
  clast.pred,  
  route = "extravascular",  
  duration.dose = 0,  
  ...,  
  options = list()  
)  
  
pk.calc.aumcint(conc, time, ..., options = list())  
  
pk.calc.aumcint.last(  
  conc,  
  time,  
  start = NULL,  
  end = NULL,
```

```
    time.dose,  
    route = "extravascular",  
    duration.dose = 0,  
    ...,  
    options = list()  
)  
  
pk.calc.aumcint.all(  
    conc,  
    time,  
    start = NULL,  
    end = NULL,  
    time.dose,  
    route = "extravascular",  
    duration.dose = 0,  
    ...,  
    options = list()  
)  
  
pk.calc.aumcint.inf.obs(  
    conc,  
    time,  
    start = NULL,  
    end = NULL,  
    time.dose,  
    lambda.z,  
    clast.obs,  
    route = "extravascular",  
    duration.dose = 0,  
    ...,  
    options = list()  
)  
  
pk.calc.aumcint.inf.pred(  
    conc,  
    time,  
    start = NULL,  
    end = NULL,  
    time.dose,  
    lambda.z,  
    clast.pred,  
    route = "extravascular",  
    duration.dose = 0,  
    ...,  
    options = list()  
)
```

Arguments

<code>conc</code>	Measured concentrations
<code>time</code>	Time of the measurement of the concentrations
<code>interval</code>	Numeric vector of two numbers for the start and end time of integration
<code>start</code>	The start time of the interval
<code>end</code>	The end time of the interval
<code>clast, clast.obs, clast.pred</code>	The last concentration above the limit of quantification; this is used for AUCinf calculations. If provided as <code>clast.obs</code> (observed clast value, default), AUCinf is AUCinf,obs. If provided as <code>clast.pred</code> , AUCinf is AUCinf,pred.
<code>lambda.z</code>	The elimination rate (in units of inverse time) for extrapolation
<code>time.dose, route, duration.dose</code>	The time of doses, route of administration, and duration of dose used with interpolation and extrapolation of concentration data (see interp.extrap.conc.dose()). If NULL or if every <code>time.dose</code> is NA (an analysis with no dosing data), interp.extrap.conc() is used instead and the calculation is not dose-aware. If only some <code>time.dose</code> values are NA, the result is NA because that dose cannot be placed on the timeline.
<code>auc.type</code>	The type of AUC to compute. Choices are 'AUCinf', 'AUClast', and 'AUCall'.
<code>options</code>	List of changes to the default PKNCA options (see <code>PKNCA.options()</code>)
<code>method</code>	The method for integration (one of 'lin up/log down', 'lin-log', or 'linear')
<code>conc.blq</code>	How to handle a BLQ value that is between above LOQ values? See details for description.
<code>conc.na</code>	How to handle NA concentrations. (See clean.conc.na())
<code>check</code>	Run assert_conc_time() ?
<code>fun_linear, fun_log, fun_inf</code>	Integration functions for linear, logarithmic, and infinite extrapolation methods.
<code>...</code>	Additional arguments passed to <code>pk.calc.auxc</code> and <code>interp.extrap.conc</code>

Value

The AUXC for an interval of time as a number

Functions

- `pk.calc.aucint()`: Calculate AUC over an interval
- `pk.calc.aucint.last()`: Interpolate or extrapolate concentrations for AUClast
- `pk.calc.aucint.all()`: Interpolate or extrapolate concentrations for AUCall
- `pk.calc.aucint.inf.obs()`: Interpolate or extrapolate concentrations for AUCinf.obs
- `pk.calc.aucint.inf.pred()`: Interpolate or extrapolate concentrations for AUCinf.pred
- `pk.calc.aumcint()`: Calculate AUMC over an interval
- `pk.calc.aumcint.last()`: Interpolate or extrapolate concentrations for AUMClast
- `pk.calc.aumcint.all()`: Interpolate or extrapolate concentrations for AUMCall
- `pk.calc.aumcint.inf.obs()`: Interpolate or extrapolate concentrations for AUMCinf.obs
- `pk.calc.aumcint.inf.pred()`: Interpolate or extrapolate concentrations for AUMCinf.pred

Doses bound the profile

When dose times are given, the concentrations used to extrapolate the profile being integrated, if extrapolation is required, end at the first dose at or after end: a concentration measured after that dose is affected by the dose and is neither integrated nor used for interpolation/extrapolation into this interval. Concentrations from before the interval are used to estimate the concentration at start, and that estimate does not interpolate across a dose either (see [interp.extrap.conc.dose\(\)](#)).

Only start and end are estimated. A dose within the interval is integrated across using the concentrations measured on either side of it, because the interval asks for the profiles on both sides of that dose to be integrated together.

The region after Tlast

The region of the interval after the last measurable concentration is handled the same way as the matching AUC parameter:

AUClast contributes zero.

AUCall contributes the triangle from clast to the first below-the-limit-of-quantification measurement and zero after that.

AUCinf is extrapolated with `lambda.z` (in other words, with the half-life), always using the logarithmic trapezoidal rule to align with the exponential decay that the half-life describes. When `lambda.z` is not estimable and the interval is finite, AUCall is used instead.

When the interval ends at or before Tlast, no extrapolation happens and `lambda.z` is not used at all. The extrapolation that was used is reported in the method (PPANMETH) column.

See Also

[PKNCA.options\(\)](#), [interp.extrap.conc.dose\(\)](#)

Other AUC calculations: [pk.calc.auxc\(\)](#), [pk.calc.auxciv\(\)](#)

Other AUMC calculations: [pk.calc.auxciv\(\)](#)

pk.calc.auxciv

Calculate AUXC (AUC or AUMC) for IV dosing with C0 back-extrapolation

Description

Calculates AUC or AUMC for intravenous dosing, with optional back-extrapolation to C0.

Usage

```
pk.calc.auxciv(  
  conc,  
  time,  
  c0,  
  auxc,  
  fun_auxc_last,  
  fun_auxc,  
  auc.type = NULL,  
  lambda.z = NA,  
  clast = NA,  
  ...,  
  options = list(),  
  check = TRUE  
)
```

```
pk.calc.auciv(  
  conc,  
  time,  
  c0,  
  auc,  
  auc.type = NULL,  
  lambda.z = NA,  
  clast = NA,  
  ...,  
  options = list(),  
  check = TRUE  
)
```

```
pk.calc.auciv_pbext(  
  conc,  
  time,  
  auc,  
  auciv,  
  ...,  
  options = list(),  
  check = TRUE  
)
```

```
pk.calc.aumciv(  
  conc,  
  time,  
  c0,  
  aumc,  
  auc.type = NULL,  
  lambda.z = NA,  
  clast = NA,  
  ...,
```

```

    options = list(),
    check = TRUE
  )

```

Arguments

conc	Measured concentrations
time	Time of the measurement of the concentrations
c0	The concentration at time 0, typically calculated using pk.calc.c0()
auxc	The AUXC calculated using conc and time without c0 (it may be calculated using any method)
fun_auxc_last	Function to calculate the AUXC for the last interval (e.g., pk.calc.auc.last or pk.calc.aumc.last)
fun_auxc	Function to calculate the AUXC over the full interval when auxc could not be calculated (e.g., pk.calc.auc or pk.calc.aumc)
auc.type	The type of AUXC for fun_auxc to calculate. NULL (the default) means that the AUXC will not be calculated from c0 and the measured data, so NA is returned when auxc is NA.
lambda.z	The elimination rate (in units of inverse time) for extrapolation
clast	The last concentration above the limit of quantification, used by fun_auxc when auc.type is "AUCinf" (clast.obs gives AUCinf,obs and clast.pred gives AUCinf,pred)
...	For functions other than pk.calc.auxc , these values are passed to pk.calc.auxc
options	List of changes to the default PKNCA options (see PKNCA.options())
check	Run assert_conc_time() ?
auc	The AUC calculated without C0 back-extrapolation
auciv	The AUC calculated using c0
aumc	The AUMC calculated using conc and time without c0

Details

The AUXC for intravenous (IV) dosing extrapolates the AUXC back from the first measurement to time 0 using c0 and the AUXC calculated by another method (e.g., [auclast](#) or [aumclast](#)).

How the calculation proceeds depends on what is measured at time = 0:

A concentration is measured at time = 0 The AUXC between time = 0 and the next time point is calculated with the measured concentration ([auxc_first](#)) and with c0 ([auxc_second](#)). The final AUXC is $\text{auxc} + \text{auxc_second} - \text{auxc_first}$.

No concentration at time = 0 and auxc was calculated [auxc](#) comes from a method that extrapolates back to time = 0 with [conc.origin](#) (zero; the [aucint](#) family), so that segment is replaced by the one using c0 in the same way.

No concentration at time = 0 and auxc is NA [auxc](#) is NA because an AUXC may not start before the first measurement. c0 supplies that measurement, so the AUXC is calculated here from c0 and the measured data using [fun_auxc](#) and [auc.type](#).

The calculation for back-extrapolation is $100 \times (1 - \text{auc}/\text{auciv})$. It requires a measured concentration at time 0; without one, auc does not describe the observed part of auciv (it either cannot be calculated or, for the aucint family, already extrapolates back to time 0 with conc.origin), so NA is returned.

Value

The AUXC calculated using c0

pk.calc.auciv_pctbackextrap: The AUC percent back-extrapolated

Functions

- pk.calc.auciv(): Calculate AUC for intravenous dosing with C0 back-extrapolation
- pk.calc.auciv_pbext(): Calculate the percent back-extrapolated AUC for IV administration
- pk.calc.aumciv(): Calculate AUMC for intravenous dosing with C0 back-extrapolation

See Also

Other AUC calculations: [pk.calc.auxc\(\)](#), [pk.calc.auxcint\(\)](#)

Other AUMC calculations: [pk.calc.aumcint\(\)](#)

pk.calc.c0

Estimate the concentration at dosing time for an IV bolus dose.

Description

Estimate the concentration at dosing time for an IV bolus dose.

Usage

```
pk.calc.c0(
  conc,
  time,
  time.dose = 0,
  method = c("c0", "logslope", "c1", "cmin", "set0"),
  check = TRUE
)

pk.calc.c0.method.logslope(conc, time, time.dose = 0, check = TRUE)

pk.calc.c0.method.c0(conc, time, time.dose = 0, check = TRUE)

pk.calc.c0.method.c1(conc, time, time.dose = 0, check = TRUE)

pk.calc.c0.method.set0(conc, time, time.dose = 0, check = TRUE)

pk.calc.c0.method.cmin(conc, time, time.dose = 0, check = TRUE)
```

Arguments

conc	Measured concentrations
time	Time of the measurement of the concentrations
time.dose	The time when dosing occurred
method	The order of methods to test (see details)
check	Run <code>assert_conc_time()</code> ?

Details

Methods available for interpolation are below, and each has its own specific function.

`c0` If the observed conc at `time.dose` is nonzero, return that. This method should usually be used first for single-dose IV bolus data in case nominal time zero is measured.

`logslope` Compute the semilog line between the first two measured times, and use that line to extrapolate backward to `time.dose`

`c1` Use the first point after `time.dose`

`cmin` Set `c0` to `cmin` during the interval. This method should usually be used for multiple-dose oral data and IV infusion data.

`set0` Set `c0` to zero (regardless of any other data). This method should usually be used first for single-dose oral data.

Value

The estimated concentration at time 0.

Functions

- `pk.calc.c0.method.logslope()`: Semilog regress the first and second points after `time.dose`. This method will return NA if the second conc after `time.dose` is 0 or greater than the first.
- `pk.calc.c0.method.c0()`: Use $C_0 = \text{conc}[\text{time \%in\% time.dose}]$ if it is nonzero.
- `pk.calc.c0.method.c1()`: Use $C_0 = C_1$.
- `pk.calc.c0.method.set0()`: Use $C_0 = 0$ (typically used for single dose oral and IV infusion)
- `pk.calc.c0.method.cmin()`: Use $C_0 = C_{\text{min}}$ (typically used for multiple dose oral and IV infusion but not IV bolus)

See Also

Other NCA parameters for concentrations during the intervals: `pk.calc.cav()`, `pk.calc.ceoi()`, `pk.calc.clast.obs()`, `pk.calc.cmax()`, `pk.calc.count_conc()`, `pk.calc.ctrough()`

pk.calc.cav	<i>Calculate the average concentration during an interval.</i>
-------------	--

Description

Calculate the average concentration during an interval.

Usage

```
pk.calc.cav(auc, start, end)
```

Arguments

auc	The area under the curve during the interval
start	The start time of the interval
end	The end time of the interval

Details

cav is $\text{auc}/(\text{end}-\text{start})$.

Value

The Cav (average concentration during the interval)

See Also

Other NCA parameters for concentrations during the intervals: [pk.calc.c0\(\)](#), [pk.calc.ceoi\(\)](#), [pk.calc.clast.obs\(\)](#), [pk.calc.cmax\(\)](#), [pk.calc.count_conc\(\)](#), [pk.calc.ctrough\(\)](#)

pk.calc.ceoi	<i>Determine the concentration at the end of infusion</i>
--------------	---

Description

Determine the concentration at the end of infusion

Usage

```
pk.calc.ceoi(conc, time, duration.dose = NA, check = TRUE)
```

Arguments

conc	Measured concentrations
time	Time of the measurement of the concentrations
duration.dose	The duration for the dosing administration (typically from IV infusion)
check	Run assert_conc_time() ?

Value

The concentration at the end of the infusion, NA if duration.dose is NA, or NA if all time != duration.dose

See Also

Other NCA parameters for concentrations during the intervals: [pk.calc.c0\(\)](#), [pk.calc.cav\(\)](#), [pk.calc.clast.obs\(\)](#), [pk.calc.cmax\(\)](#), [pk.calc.count_conc\(\)](#), [pk.calc.ctrough\(\)](#)

pk.calc.cl

Calculate the (observed oral) clearance

Description

Calculate the (observed oral) clearance

Usage

```
pk.calc.cl(dose, auc)
```

Arguments

dose	the dose administered
auc	The area under the concentration-time curve.

Details

cl is dose/auc.

If dose is the same length as the other inputs, then the output will be the same length as all of the inputs; the function assumes that you are calculating for multiple intervals simultaneously. If the inputs other than dose are scalars and dose is a vector, then the function assumes multiple doses were given in a single interval, and the sum of the doses will be used for the calculation.

Value

the numeric value of the total (CL) or observed oral clearance (CL/F)

References

Gabrielsson J, Weiner D. "Section 2.5.1 Derivation of clearance." Pharmacokinetic & Pharmacodynamic Data Analysis: Concepts and Applications, 4th Edition. Stockholm, Sweden: Swedish Pharmaceutical Press, 2000. 86-7.

See Also

Other Clearance and volume parameters: [pk.calc.kel\(\)](#), [pk.calc.vz\(\)](#)

pk.calc.clast.obs	<i>Determine the last observed concentration above the limit of quantification (LOQ).</i>
-------------------	---

Description

If all concentrations are missing, NA_real_ is returned. If all concentrations are zero (below the limit of quantification) or missing, zero is returned. If Tlast is NA (due to no non-missing above LOQ measurements), this will return NA_real_.

Usage

```
pk.calc.clast.obs(conc, time, check = TRUE)
```

Arguments

conc	Measured concentrations
time	Time of the measurement of the concentrations
check	Run <code>assert_conc_time()</code> ?

Value

The last observed concentration above the LOQ

See Also

Other NCA parameters for concentrations during the intervals: `pk.calc.c0()`, `pk.calc.cav()`, `pk.calc.ceoi()`, `pk.calc.cmax()`, `pk.calc.count_conc()`, `pk.calc.ctrough()`

pk.calc.clr	<i>Calculate renal clearance</i>
-------------	----------------------------------

Description

Calculate renal clearance

Usage

```
pk.calc.clr(ae, auc)
```

Arguments

ae	The amount excreted in urine (as a numeric scalar or vector)
auc	The area under the curve (as a numeric scalar or vector)

Details

clr is $\text{sum}(\text{ae})/\text{auc}$.

The units for the ae and auc should match such that ae/auc has units of volume/time.

Value

The renal clearance as a number

See Also

[pk.calc.ae\(\)](#), [pk.calc.fe\(\)](#)

Other Urine/Excretion parameters: [pk.calc.ae\(\)](#), [pk.calc.erint\(\)](#), [pk.calc.erlst\(\)](#), [pk.calc.emax\(\)](#), [pk.calc.ertlst\(\)](#), [pk.calc.ertmax\(\)](#), [pk.calc.fe\(\)](#), [pk.calc.volpk\(\)](#)

pk.calc.cmax

Determine maximum observed PK concentration

Description

Determine maximum observed PK concentration

Usage

```
pk.calc.cmax(conc, check = TRUE)
```

```
pk.calc.cmin(conc, check = TRUE)
```

Arguments

conc	Measured concentrations
check	Run assert_conc() ?

Value

a number for the maximum concentration or NA if all concentrations are missing

Functions

- [pk.calc.cmin\(\)](#): Determine the minimum observed PK concentration

See Also

Other NCA parameters for concentrations during the intervals: [pk.calc.c0\(\)](#), [pk.calc.cav\(\)](#), [pk.calc.ceoi\(\)](#), [pk.calc.clast.obs\(\)](#), [pk.calc.count_conc\(\)](#), [pk.calc.ctrough\(\)](#)

Examples

```
conc_data <- Theoph[Theoph$Subject == 1,]
pk.calc.cmin(conc_data$conc)
```

pk.calc.count_conc	Count the number of concentration measurements in an interval
--------------------	---

Description

count_conc and count_conc_measured are typically used for quality control on the data to ensure that there are a sufficient number of non-missing samples for a calculation and to ensure that data are consistent between individuals.

Usage

```
pk.calc.count_conc(conc, check = TRUE)
```

```
pk.calc.count_conc_measured(conc, check = TRUE)
```

Arguments

conc	Measured concentrations
check	Run assert_conc() ?

Value

a count of the non-missing concentrations (0 if all concentrations are missing)

a count of the non-missing, measured (not below or above the limit of quantification) concentrations (0 if all concentrations are missing). "Measured" here means above the limit of quantification; an imputed concentration above it is counted (see the "Imputed concentrations" section).

Functions

- `pk.calc.count_conc_measured()`: Count the number of concentration measurements that are not missing, above, or below the limit of quantification in an interval

Imputed concentrations

Both counts are taken after imputation, and neither distinguishes an imputed concentration from a measured one. `count_conc` counts every non-missing concentration, so any imputation that adds a point increases it. `count_conc_measured` counts concentrations above the limit of quantification, so whether an imputed point is counted depends on its value rather than on its being imputed: the zero added by [PKNCA_impute_method_start_conc0\(\)](#) is not counted, while the concentration carried to the start time by [PKNCA_impute_method_start_predose\(\)](#) and the minimum added by [PKNCA_impute_method_start_cmin\(\)](#) are.

To count only measured samples, calculate the counts in an interval with no imputation.

See Also

Other NCA parameters for concentrations during the intervals: [pk.calc.c0\(\)](#), [pk.calc.cav\(\)](#), [pk.calc.ceoi\(\)](#), [pk.calc.clast.obs\(\)](#), [pk.calc.cmax\(\)](#), [pk.calc.ctrough\(\)](#)

pk.calc.ctrough	<i>Determine the trough (end of interval) concentration</i>
-----------------	---

Description

Determine the trough (end of interval) concentration

Usage

```
pk.calc.ctrough(conc, time, end)
```

```
pk.calc.cstart(conc, time, start)
```

Arguments

conc	Measured concentrations
time	Time of the measurement of the concentrations
end	The end time of the interval
start	The start time of the interval

Value

The concentration when time == end. If none match, then NA

Functions

- `pk.calc.cstart()`: Concentration at the beginning of the interval

See Also

Other NCA parameters for concentrations during the intervals: [pk.calc.c0\(\)](#), [pk.calc.cav\(\)](#), [pk.calc.ceoi\(\)](#), [pk.calc.clast.obs\(\)](#), [pk.calc.cmax\(\)](#), [pk.calc.count_conc\(\)](#)

pk.calc.deg.fluc	<i>Determine the degree of fluctuation</i>
------------------	--

Description

Determine the degree of fluctuation

Usage

```
pk.calc.deg.fluc(cmax, cmin, cav)
```

```
pk.calc.swing(cmax, cmin)
```

Arguments

cmax	The maximum observed concentration
cmin	The minimum observed concentration
cav	The average concentration in the interval

Details

deg.fluc is $100 * (cmax - cmin) / cav$.

swing is $100 * (cmax - cmin) / cmin$.

Value

The degree of fluctuation around the average concentration.

The swing above the minimum concentration. If cmin is zero, then the result is infinity.

Functions

- `pk.calc.swing()`: PK swing

See Also

Other Multiple-dose PK parameters: [pk.calc.ptr\(\)](#)

`pk.calc.dn`*Determine dose normalized NCA parameter*

Description

Determine dose normalized NCA parameter

Usage

```
pk.calc.dn(parameter, dose)
```

Arguments

parameter	Parameter to dose normalize
dose	Dose in units compatible with the area under the curve

Value

a number for dose normalized AUC

Examples

```
pk.calc.dn(90, 10)
```

pk.calc.erint	<i>Calculate the excretion rate over the interval</i>
---------------	---

Description

Calculate the excretion rate over the interval

Usage

```
pk.calc.erint(ae, start, end)
```

Arguments

ae	The amount excreted during the interval (see pk.calc.ae())
start	The start time of the interval
end	The end time of the interval

Details

The excretion rate is the amount recovered during the interval divided by the duration of the interval, $ae/(end - start)$. It uses the interval rather than the sum of the collection durations, so a gap between collections lowers the rate. An interval ending at infinity has no duration to divide by and gives NA.

Value

The excretion rate during the interval

See Also

[pk.calc.ae\(\)](#), [pk.calc.ermx\(\)](#)

Other Urine/Excretion parameters: [pk.calc.ae\(\)](#), [pk.calc.clr\(\)](#), [pk.calc.erlst\(\)](#), [pk.calc.ermx\(\)](#), [pk.calc.ertlst\(\)](#), [pk.calc.ertmx\(\)](#), [pk.calc.fe\(\)](#), [pk.calc.volpk\(\)](#)

pk.calc.erlst	<i>Calculate the last measurable excretion rate</i>
---------------	---

Description

Calculate the last measurable excretion rate

Usage

```
pk.calc.erlst(conc, volume, time, duration.conc, check = TRUE)
```

Arguments

conc	The concentration in the excreta (e.g., urine or feces)
volume	The volume (or mass) of the sample
time	The starting time of the collection interval
duration.conc	The duration of the collection interval
check	Should the concentration data be checked?

Details

Collections are ordered by their midpoint time, matching `pk.calc.ertlst()`, which gives the midpoint time of this same collection. When no collection has a positive excretion rate, the result is 0.

Value

The last measurable (positive) excretion rate, or NA if not available

See Also

`pk.calc.ertlst()`, `pk.calc.ymax()`

Other Urine/Excretion parameters: `pk.calc.ae()`, `pk.calc.clr()`, `pk.calc.erint()`, `pk.calc.ymax()`, `pk.calc.ertlst()`, `pk.calc.ertmax()`, `pk.calc.fe()`, `pk.calc.volpk()`

pk.calc.ymax	<i>Calculate the maximum excretion rate</i>
--------------	---

Description

Calculate the maximum excretion rate

Usage

```
pk.calc.ymax(conc, volume, time, duration.conc, check = TRUE)
```

Arguments

conc	The concentration in the excreta (e.g., urine or feces)
volume	The volume (or mass) of the sample
time	The starting time of the collection interval
duration.conc	The duration of the collection interval
check	Should the concentration data be checked?

Value

The maximum excretion rate, or NA if not available

See Also

Other Urine/Excretion parameters: [pk.calc.ae\(\)](#), [pk.calc.clr\(\)](#), [pk.calc.erint\(\)](#), [pk.calc.erlst\(\)](#), [pk.calc.ertlst\(\)](#), [pk.calc.ertmax\(\)](#), [pk.calc.fe\(\)](#), [pk.calc.volpk\(\)](#)

<code>pk.calc.ertlst</code>	<i>Calculate the midpoint collection time of the last measurable excretion rate</i>
-----------------------------	---

Description

Calculate the midpoint collection time of the last measurable excretion rate

Usage

```
pk.calc.ertlst(conc, volume, time, duration.conc, check = TRUE)
```

Arguments

<code>conc</code>	The concentration in the excreta (e.g., urine or feces)
<code>volume</code>	The volume (or mass) of the sample
<code>time</code>	The starting time of the collection interval
<code>duration.conc</code>	The duration of the collection interval
<code>check</code>	Should the concentration and time data be checked?

Value

The midpoint collection time of the last measurable excretion rate, or NA/0 if not available

See Also

Other Urine/Excretion parameters: [pk.calc.ae\(\)](#), [pk.calc.clr\(\)](#), [pk.calc.erint\(\)](#), [pk.calc.erlst\(\)](#), [pk.calc.ermx\(\)](#), [pk.calc.ertmax\(\)](#), [pk.calc.fe\(\)](#), [pk.calc.volpk\(\)](#)

pk.calc.ertmax	<i>Calculate the midpoint collection time of the maximum excretion rate</i>
----------------	---

Description

Calculate the midpoint collection time of the maximum excretion rate

Usage

```
pk.calc.ertmax(  
  conc,  
  volume,  
  time,  
  duration.conc,  
  options = list(),  
  check = TRUE,  
  first.tmax = NULL  
)
```

Arguments

conc	The concentration in the excreta (e.g., urine or feces)
volume	The volume (or mass) of the sample
time	The starting time of the collection interval
duration.conc	The duration of the collection interval
options	List of changes to the default PKNCA options (see <code>PKNCA.options()</code>)
check	Should the concentration and time data be checked?
first.tmax	If TRUE, return the first time of maximum excretion rate; otherwise, return the last

Value

The midpoint collection time of the maximum excretion rate, or NA if not available

See Also

Other Urine/Excretion parameters: [pk.calc.ae\(\)](#), [pk.calc.clr\(\)](#), [pk.calc.erint\(\)](#), [pk.calc.erlst\(\)](#), [pk.calc.ermx\(\)](#), [pk.calc.ertlst\(\)](#), [pk.calc.fe\(\)](#), [pk.calc.volpk\(\)](#)

`pk.calc.f`*Calculate the absolute (or relative) bioavailability*

Description

Calculate the absolute (or relative) bioavailability

Usage

```
pk.calc.f(dose1, auc1, dose2, auc2)
```

Arguments

dose1	The dose administered in route or method 1
auc1	The AUC from 0 to infinity or 0 to tau administered in route or method 1
dose2	The dose administered in route or method 2
auc2	The AUC from 0 to infinity or 0 to tau administered in route or method 2

Details

f is $(auc2/dose2)/(auc1/dose1)$.

`pk.calc.fe`*Calculate fraction excreted (typically in urine or feces)*

Description

Calculate fraction excreted (typically in urine or feces)

Usage

```
pk.calc.fe(ae, dose)
```

Arguments

ae	The amount excreted (as a numeric scalar or vector)
dose	The dose (as a numeric scalar or vector)

Details

fe is $\text{sum}(ae)/\text{dose}$

The units for ae and dose should be the same so that ae/dose is a unitless fraction.

Value

The fraction of dose excreted

See Also

[pk.calc.ae\(\)](#), [pk.calc.clr\(\)](#)

Other Urine/Excretion parameters: [pk.calc.ae\(\)](#), [pk.calc.clr\(\)](#), [pk.calc.erint\(\)](#), [pk.calc.erlst\(\)](#), [pk.calc.ermx\(\)](#), [pk.calc.ertlst\(\)](#), [pk.calc.ertmx\(\)](#), [pk.calc.volpk\(\)](#)

pk.calc.half.life	<i>Compute the half-life and associated parameters</i>
-------------------	--

Description

The terminal elimination half-life is estimated from the final points in the concentration-time curve using semi-log regression ($\log(\text{conc}) \sim \text{time}$, the "log-linear" method) or Tobit regression ("tobit" method) with automated selection of the points for calculation (unless manually.selected.points is TRUE).

Usage

```
pk.calc.half.life(  
  conc,  
  time,  
  tmax,  
  tlast,  
  time.dose = NULL,  
  duration.dose = 0,  
  lloq = NULL,  
  hl_method = c("log-linear", "tobit"),  
  manually.selected.points = FALSE,  
  options = list(),  
  min.hl.points = NULL,  
  max.hl.points = NULL,  
  min.hl.start.time = NULL,  
  adj.r.squared.factor = NULL,  
  r.squared.factor = NULL,  
  tobit_n_points_penalty = NULL,  
  tobit_optim_control = NULL,  
  conc.blq = NULL,  
  conc.na = NULL,  
  first.tmax = NULL,  
  allow.tmax.in.half.life = NULL,  
  check = TRUE  
)
```

Arguments

<code>conc</code>	Measured concentrations
<code>time</code>	Time of the measurement of the concentrations
<code>tmax</code>	Time of maximum concentration (will be calculated and included in the return data frame if not given)
<code>tlast</code>	Time of last concentration above the limit of quantification (will be calculated and included in the return data frame if not given)
<code>time.dose</code>	Time of the dose for the current interval (must be the same length as dose)
<code>duration.dose</code>	The duration of the dose administration for the current interval (typically zero for extravascular and intravascular bolus and nonzero for intravascular infusion)
<code>lloq</code>	Lower limit of quantification. A scalar or a vector the same length as <code>conc</code> . Required when <code>hl_method = "tobit"</code> .
<code>hl_method</code>	The method used to estimate the half-life. "log-linear" (default) uses ordinary least-squares regression on log-transformed concentrations. "tobit" uses maximum-likelihood Tobit regression that properly accounts for BLQ observations. The global default can be changed via <code>PKNCA.options(hl_method = "tobit")</code> .
<code>manually.selected.points</code>	Have the input points (<code>conc</code> and <code>time</code>) been manually selected? The impact of setting this to <code>TRUE</code> is that no selection for the best points will be done. When <code>TRUE</code> , this option causes the options of <code>adj.r.squared.factor</code> , <code>r.squared.factor</code> , <code>min.hl.points</code> , <code>max.hl.points</code> , <code>min.hl.start.time</code> , and <code>allow.tmax.in.half.life</code> to be ignored.
<code>options</code>	List of changes to the default PKNCA options (see <code>PKNCA.options()</code>)
<code>min.hl.points</code>	The minimum number of points that must be included to calculate the half-life. For <code>hl_method = "tobit"</code> this counts only above-LLOQ points.
<code>max.hl.points</code>	The maximum number of points that automatic point selection may use (Inf, the default, for no maximum). It must be more than <code>min.hl.points</code> . For <code>hl_method = "tobit"</code> this counts only above-LLOQ points.
<code>min.hl.start.time</code>	The earliest time at which automatic point selection may start the half-life: points before it are not used. It is on the scale of time, which within <code>pk.nca()</code> is the time since the start of the interval. The default, 0, does not restrict the points.
<code>adj.r.squared.factor</code>	The allowance in adjusted r-squared for adding another point (log-linear method only). Giving it takes <code>r.squared.factor</code> out of use, and setting it to NA selects points with <code>r.squared.factor</code> instead.
<code>r.squared.factor</code>	The allowance in r-squared for adding another point (log-linear method only). Giving it takes <code>adj.r.squared.factor</code> out of use, so exactly one of the two is ever in use; NA (the default) selects points with <code>adj.r.squared.factor</code> . Unlike the adjusted r-squared, the r-squared does not reward more points, so it generally selects fewer of them; with <code>min.hl.points = 2</code> the two-point fit has an r-squared of 1 and is therefore always selected.

tobit_n_points_penalty	The penalty exponent on the number of points for Tobit window selection. See PKNCA.options() .
tobit_optim_control	A list of control parameters passed to stats::optim() for the Tobit fit. See PKNCA.options() .
conc.blq	How to handle a BLQ value that is between above LOQ values? See details for description.
conc.na	How to handle NA concentrations. (See clean.conc.na())
first.tmax	If there is more than one time point with the maximum value (Cmax or ERmax), which time should be selected for Tmax/ERTmax? If 'TRUE', the first will be selected. If not, then the last is considered Tmax/ERTmax.
allow.tmax.in.half.life	Allow the concentration point for tmax to be included in the half-life slope calculation.
check	Run assert_conc_time() ?

Details

See the "Half-Life Calculation" and "Half-Life Calculation with Tobit Regression" vignettes for more details on the calculation methods.

If `manually.selected.points` is FALSE (default), the half-life is calculated by computing the best fit line for all points at or after `tmax` (based on the value of `allow.tmax.in.half.life`).

For `hl_method = "log-linear"`, the best half-life is chosen by the following rules in order:

- At least `min.hl.points` and at most `max.hl.points` points included, the first of them no earlier than `min.hl.start.time`
- A `lambda.z > 0` and at the same time the best adjusted r-squared (within `adj.r.squared.factor`) or, when `adj.r.squared.factor` is NA, the best r-squared (within `r.squared.factor`)
- The one with the most points included

For `hl_method = "tobit"`, BLQ observations are retained and treated as left-censored. The best window is the one minimizing `tobit_residual * n ^ tobit_n_points_penalty` (default: `raw_tobit_residual`) among windows with `lambda.z > 0`, at least `min.hl.points` and at most `max.hl.points` above-LLOQ points, and a first point no earlier than `min.hl.start.time`. On ties the largest window (most total points) is preferred.

If `manually.selected.points` is TRUE, the `conc` and `time` data are used as-is without any form of point selection. When TRUE, `adj.r.squared.factor`, `r.squared.factor`, `min.hl.points`, `max.hl.points`, `min.hl.start.time`, and `allow.tmax.in.half.life` are ignored.

Value

A data frame with one row. Columns depend on `hl_method`:

Columns returned by both methods:

tmax Time of maximum observed concentration (only included if not given as an input)

tlast Time of last observed concentration above the LOQ (only included if not given as an input)

lambda.z elimination rate

lambda.z.time.first first time for half-life calculation

lambda.z.time.last last time for half-life calculation

lambda.z.n.points number of points in half-life calculation (all points for Tobit, including BLQ)

clast.pred Concentration at tlast as predicted by the half-life line

half.life half-life

span.ratio ratio of the above-LLOQ time span to the half-life

Additional columns for `hl_method = "log-linear"`:

r.squared coefficient of determination

adj.r.squared adjusted coefficient of determination

lambda.z.corrxy correlation between time and log-conc for the half-life points

Additional columns for `hl_method = "tobit"`:

lambda.z.n.points_biq number of BLQ points included in the fit

tobit_residual estimated residual standard deviation from the Tobit fit (on the log-concentration scale)

adj_tobit_residual adjusted Tobit residual (analogous to adjusted r-squared; penalizes smaller windows)

References

Gabrielsson J, Weiner D. "Section 2.8.4 Strategies for estimation of lambda-z." Pharmacokinetic & Pharmacodynamic Data Analysis: Concepts and Applications, 4th Edition. Stockholm, Sweden: Swedish Pharmaceutical Press, 2000. 167-9.

See Also

Other Half-life and elimination: [adj.r.squared\(\)](#), [pk.calc.aucpext\(\)](#), [pk.calc.thalf.eff\(\)](#)

pk.calc.kel

Calculate the elimination rate (Kel)

Description

Calculate the elimination rate (Kel)

Usage

`pk.calc.kel(mrt)`

Arguments

mrt	the mean residence time
kel	is 1/mrt, not to be confused with lambda.z.

Value

the numeric value of the elimination rate

See Also

Other Clearance and volume parameters: [pk.calc.cl\(\)](#), [pk.calc.vz\(\)](#)

pk.calc.mrt	<i>Calculate the mean residence time (MRT) for single-dose data or linear multiple-dose data.</i>
-------------	---

Description

Calculate the mean residence time (MRT) for single-dose data or linear multiple-dose data.

Usage

```
pk.calc.mrt(auc, aumc)

pk.calc.mrt.iv(auc, aumc, duration.dose)

pk.calc.mrt.md(auctau, aumctau, aucinf, tau)

pk.calc.mrt.md.iv(auctau, aumctau, aucinf, tau, duration.dose)
```

Arguments

auc	the AUC from 0 to infinity or 0 to tau
aumc	the AUMC from 0 to infinity or 0 to tau
duration.dose	The duration of the dose (usually an infusion duration for an IV infusion)
auctau	the AUC from time 0 to the end of the dosing interval (tau).
aumctau	the AUMC from time 0 to the end of the dosing interval (tau).
aucinf	the AUC from time 0 to infinity (typically using single-dose data)
tau	The dosing interval

Details

mrt is $\text{aumc}/\text{auc} - \text{duration.dose}/2$ where $\text{duration.dose} = 0$ for oral administration.

mrt.md is $\text{aumctau}/\text{auctau} + \text{tau} * (\text{aucinf} - \text{auctau})/\text{auctau}$ and should only be used for multiple dosing with equal intervals between doses. Note that if $\text{aucinf} == \text{auctau}$ (as would be the assumption with linear kinetics), the equation becomes the same as the single-dose MRT.

These are the parameters to use when PK are nonlinear. With linear PK, MRT and Vss can be measured from a single dose and the single-dose parameters (mrt.obs, vss.obs, and similar) describe steady state as well. When PK are nonlinear they do not, so MRT and Vss have to be measured over a steady-state dosing interval, which is what these parameters do. The ratio $\text{aumctau}/\text{auctau}$ on its own (mrt.last over a dosing interval) is not MRT at steady state and underestimates it substantially; the $\text{tau} * (\text{aucinf} - \text{auctau})/\text{auctau}$ term accounts for the drug still in the body at the end of the interval.

Within `pk.nca()`, tau is detected from the dose times of the group with `find.tau()`. When the dosing data hold a single dose (a common steady-state design where only the profiled dose is recorded), nothing repeats and nothing can be detected, so give tau as a column of the interval specification instead; a tau column always takes precedence over detection. If tau can be neither given nor detected, the parameter is NA with a warning.

mrt.ivmd is mrt.md less half of the infusion duration, the same correction that mrt.iv applies to the single-dose MRT. Without it, MRT and everything derived from it are high by $\text{duration.dose}/2$.

Value

the numeric value of the mean residence time

Functions

- `pk.calc.mrt.iv()`: MRT for an IV infusion
- `pk.calc.mrt.md()`: MRT for multiple-dose data with nonlinear kinetics
- `pk.calc.mrt.md.iv()`: MRT for multiple-dose data with nonlinear kinetics given as an IV infusion

pk.calc.ptr	<i>Determine the peak-to-trough ratio</i>
-------------	---

Description

Determine the peak-to-trough ratio

Usage

`pk.calc.ptr(cmax, ctrough)`

Arguments

cmax	The maximum observed concentration
ctrough	The last concentration in an interval

Details

ptr is cmax/ctrough.

Value

The ratio of cmax to ctrough (if ctrough == 0, NA)

See Also

Other Multiple-dose PK parameters: [pk.calc.deg.fluc\(\)](#)

pk.calc.ratio	<i>Calculate the ratio of a parameter between two intervals</i>
---------------	---

Description

Calculate the ratio of a parameter between two intervals

Usage

```
pk.calc.ratio(test, reference)
```

Arguments

test	The parameter value in the current (test) interval
reference	The parameter value in the reference interval

Value

test/reference, or NA where the reference is missing or is not above zero (a ratio to a zero or negative reference is not interpretable).

See Also

[interval_add_secondary\(\)](#)

Examples

```
pk.calc.ratio(test = 10, reference = 20)
```

pk.calc.sparse_auc *Calculate AUC and related parameters using sparse NCA methods*

Description

The AUC is calculated as:

Usage

```
pk.calc.sparse_auc(
    conc,
    time,
    subject,
    method = "linear",
    auc.type = "AUClast",
    ...,
    options = list()
)

pk.calc.sparse_auclast(conc, time, subject, ..., options = list())
```

Arguments

conc	Measured concentrations
time	Time of the measurement of the concentrations
subject	Subject identifiers (may be any class; may not be null). A missing subject marks an imputed concentration (such as the zero that the start_conc0 imputation adds at the start of an interval): a time where every subject is missing has a known concentration, which enters sparse estimates but not their variance.
method	The method for integration (one of 'lin up/log down', 'lin-log', or 'linear')
auc.type	The type of AUC to compute. Choices are 'AUCinf', 'AUClast', and 'AUCall'.
...	For functions other than pk.calc.auxc, these values are passed to pk.calc.auxc
options	List of changes to the default PKNCA options (see PKNCA.options())

Details

$$AUC = \sum_i w_i \bar{C}_i$$

Where:

AUC is the estimated area under the concentration-time curve

w_i is the weight applied to the concentration at time i (related to the time which it affects, see [sparse_auc_weight_linear\(\)](#))

$\bar{C}_i$ is the average concentration at time i

Functions

- `pk.calc.sparse_auclast()`: Compute the AUClast for sparse PK

See Also

Other Sparse Methods: `as_sparse_pk()`, `pk.calc.aucinf.obs_sparse()`, `pk.calc.aucivlast_sparse()`, `pk.calc.auclast_sparse()`, `pk.calc.sparse_aumc()`, `sparse_auc_weight_linear()`, `sparse_bootstrap()`, `sparse_mean()`, `summary.PKNCAresults_sparse_bootstrap()`

<code>pk.calc.sparse_aumc</code>	<i>Calculate AUMC and related parameters using sparse NCA methods</i>
----------------------------------	---

Description

The AUMC is calculated as:

Usage

```
pk.calc.sparse_aumc(
  conc,
  time,
  subject,
  method = "linear",
  auc.type = "AUClast",
  ...,
  options = list()
)

pk.calc.sparse_aumclast(conc, time, subject, ..., options = list())
```

Arguments

<code>conc</code>	Measured concentrations
<code>time</code>	Time of the measurement of the concentrations
<code>subject</code>	Subject identifiers (may be any class; may not be null). A missing subject marks an imputed concentration (such as the zero that the <code>start_conc0</code> imputation adds at the start of an interval): a time where every subject is missing has a known concentration, which enters sparse estimates but not their variance.
<code>method</code>	The method for integration (one of 'lin up/log down', 'lin-log', or 'linear')
<code>auc.type</code>	The type of AUC to compute. Choices are 'AUCinf', 'AUClast', and 'AUCall'.
<code>...</code>	For functions other than <code>pk.calc.auxc</code> , these values are passed to <code>pk.calc.auxc</code>
<code>options</code>	List of changes to the default PKNCA options (see <code>PKNCA.options()</code>)

Details

$$AUMC = \sum_i w_i \overline{t_i C_i}$$

Where:

$AUMC$ is the estimated area under the first moment curve

w_i is the weight applied to time i (same as for AUC, see [sparse_auc_weight_linear\(\)](#))

$\overline{t_i C_i}$ is the average of the moment (time $\times$ concentration) at time i

Value

A data.frame with columns:

sparse_aumc	The estimated AUMC
sparse_aumc_se	Standard error of the AUMC estimate
sparse_aumc_df	Degrees of freedom for the variance estimate

Functions

- `pk.calc.sparse_aumclast()`: Compute the AUMClast for sparse PK

See Also

Other Sparse Methods: [as_sparse_pk\(\)](#), [pk.calc.aucinf.obs_sparse\(\)](#), [pk.calc.aucivlast_sparse\(\)](#), [pk.calc.auclast_sparse\(\)](#), [pk.calc.sparse_auc\(\)](#), [sparse_auc_weight_linear\(\)](#), [sparse_bootstrap\(\)](#), [sparse_mean\(\)](#), [summary.PKNCResults_sparse_bootstrap\(\)](#)

pk.calc.thalf.eff	<i>Calculate the effective half-life</i>
-------------------	--

Description

Calculate the effective half-life

Usage

```
pk.calc.thalf.eff(mrt)
```

Arguments

mrt	the mean residence time to infinity
-----	-------------------------------------

Details

thalf.eff is $\log(2) \times \text{mrt}$.

Value

the numeric value of the effective half-life

See Also

Other Half-life and elimination: [adj.r.squared\(\)](#), [pk.calc.aucpext\(\)](#), [pk.calc.half.life\(\)](#)

pk.calc.time_above	<i>Determine time at or above a set value</i>
--------------------	---

Description

Interpolation is performed aligning with `PKNCA.options("auc.method")`. Extrapolation outside of the measured times is not yet implemented. The method may be changed by giving a named method argument, as well.

Usage

```
pk.calc.time_above(conc, time, conc_above, ..., options = list(), check = TRUE)
```

Arguments

conc	Measured concentrations
time	Time of the measurement of the concentrations
conc_above	The concentration to be above
...	Extra arguments. Currently, the only extra argument that is used is method as described in the details section.
options	List of changes to the default PKNCA options (see <code>PKNCA.options()</code>)
check	Run assert_conc_time() ?

Details

For 'lin up/log down', if clast is above conc_above and there are concentrations BLQ after that, linear down is used to extrapolate to the BLQ concentration (equivalent to AUCall).

Value

the time above the given concentration

pk.calc.tlag	<i>Determine the observed lag time (time before the first concentration above the limit of quantification or above the first concentration in the interval)</i>
--------------	---

Description

Determine the observed lag time (time before the first concentration above the limit of quantification or above the first concentration in the interval)

Usage

```
pk.calc.tlag(conc, time)
```

Arguments

conc	Measured concentrations
time	Time of the measurement of the concentrations

Value

The time associated with the first increasing concentration

See Also

Other NCA time parameters: [pk.calc.tlast\(\)](#), [pk.calc.tmax\(\)](#), [pk.calc.tmin\(\)](#)

pk.calc.tlast	<i>Determine time of last observed concentration above the limit of quantification.</i>
---------------	---

Description

NA will be returned if all conc are NA or 0.

Usage

```
pk.calc.tlast(conc, time, check = TRUE)
```

```
pk.calc.tfirst(conc, time, check = TRUE)
```

Arguments

conc	Measured concentrations
time	Time of the measurement of the concentrations
check	Run assert_conc_time() ?

Value

The time of the last observed concentration measurement

Functions

- `pk.calc.tfirst()`: Determine the first concentration above the limit of quantification.

See Also

Other NCA time parameters: `pk.calc.tlag()`, `pk.calc.tmax()`, `pk.calc.tmin()`

`pk.calc.tmax`

Determine time of maximum observed PK concentration

Description

Input restrictions are:

1. the conc and time must be the same length,
2. the time may have no NAs,

NA will be returned if:

1. the length of conc and time is 0
2. all conc is 0 or NA

Usage

```
pk.calc.tmax(conc, time, options = list(), first.tmax = NULL, check = TRUE)
```

Arguments

<code>conc</code>	Measured concentrations
<code>time</code>	Time of the measurement of the concentrations
<code>options</code>	List of changes to the default PKNCA options (see <code>PKNCA.options()</code>)
<code>first.tmax</code>	If there is more than one time point with the maximum value (Cmax or ERmax), which time should be selected for Tmax/ERTmax? If 'TRUE', the first will be selected. If not, then the last is considered Tmax/ERTmax.
<code>check</code>	Run <code>assert_conc_time()</code> ?

Value

The time of the maximum concentration

See Also

Other NCA time parameters: `pk.calc.tlag()`, `pk.calc.tlast()`, `pk.calc.tmin()`

Examples

```
conc_data <- Theoph[Theoph$Subject == 1,]
pk.calc.tmax(conc = conc_data$conc, time = conc_data$Time)
```

pk.calc.tmin

*Determine time of minimum observed PK concentration***Description**

Input restrictions are:

1. the conc and time must be the same length,
2. the time may have no NAs,

NA will be returned if:

1. the length of conc and time is 0
2. all conc is NA

Usage

```
pk.calc.tmin(conc, time, options = list(), first.tmin = NULL, check = TRUE)
```

Arguments

conc	Measured concentrations
time	Time of the measurement of the concentrations
options	List of changes to the default PKNCA options (see PKNCA.options())
first.tmin	If there is more than one time point with the minimum value (Cmin), which time should be selected for Tmin? If 'TRUE', the first will be selected. If not, then the last is considered Tmin.
check	Run assert_conc_time() ?

Value

The time of the minimum concentration

See Also

Other NCA time parameters: [pk.calc.tlag\(\)](#), [pk.calc.tlast\(\)](#), [pk.calc.tmax\(\)](#)

Examples

```
conc_data <- Theoph[Theoph$Subject == 1,]
pk.calc.tmin(conc = conc_data$conc, time = conc_data$Time)
```

pk.calc.totdose	<i>Extract the dose used for calculations</i>
-----------------	---

Description

Extract the dose used for calculations

Usage

```
pk.calc.totdose(dose)
```

Arguments

dose	the dose administered
------	-----------------------

Value

The total dose for an interval

pk.calc.volpk	<i>Calculate the total urine volume</i>
---------------	---

Description

Calculate the total urine volume

Usage

```
pk.calc.volpk(volume)
```

Arguments

volume	The volume (or mass) of the sample
--------	------------------------------------

Value

The sum of urine volumes for the interval

See Also

Other Urine/Excretion parameters: [pk.calc.ae\(\)](#), [pk.calc.clr\(\)](#), [pk.calc.erint\(\)](#), [pk.calc.erlst\(\)](#), [pk.calc.ermx\(\)](#), [pk.calc.ertlst\(\)](#), [pk.calc.ertmx\(\)](#), [pk.calc.fe\(\)](#)

`pk.calc.vz`*Calculate the terminal volume of distribution (Vz)*

Description

Calculate the terminal volume of distribution (Vz)

Usage

```
pk.calc.vz(cl, lambda.z)
```

```
pk.calc.vss(cl, mrt)
```

Arguments

<code>cl</code>	the clearance (or apparent observed clearance)
<code>lambda.z</code>	The elimination rate (in units of inverse time) for extrapolation
<code>mrt</code>	the mean residence time

Details

`vz` is `cl/lambda.z`.

`vss` is `cl*mrt`.

Value

the volume of distribution at steady-state

Functions

- `pk.calc.vss()`: Steady-state volume of distribution (Vss)

See Also

Other Clearance and volume parameters: [pk.calc.cl\(\)](#), [pk.calc.kel\(\)](#)

pk.nca	<i>Compute NCA parameters for each interval for each subject.</i>
--------	---

Description

The `pk.nca` function computes the NCA parameters from a `PKNCAdata` object. All options for the calculation and input data are set in prior functions (`PKNCAconc`, `PKNCAdose`, and `PKNCAdata`). Options for calculations are set either in `PKNCAdata` or with the current default options in `PKNCA.options`.

Usage

```
pk.nca(data, verbose = FALSE)
```

Arguments

<code>data</code>	A <code>PKNCAdata</code> object
<code>verbose</code>	Indicate, by <code>message()</code> , the current state of calculation.

Details

When performing calculations, all time results are relative to the start of the interval. For example, if an interval starts at 168 hours, ends at 192 hours, and the maximum concentration is at 169 hours, `tmax=169-168=1`.

Data are selected for each interval by the measurement or dose time: rows with a time at or after the interval start and at or before the interval end are included (a dose exactly at the interval end is not included in the interval). For duration data (for example, urine collections or intravenous infusions), the time is the start of the collection or administration, and the duration is not considered during selection: a collection that starts within the interval and ends after the interval end contributes its full amount to the interval. For the simplest interpretation of results, align collection start and end times with interval boundaries.

Value

A `PKNCAResults` object.

See Also

[PKNCAdata\(\)](#), [PKNCA.options\(\)](#), [summary.PKNCAResults\(\)](#), [as.data.frame.PKNCAResults\(\)](#), [exclude\(\)](#)

pk.nca.interval	Compute all PK parameters for a single concentration-time data set
-----------------	--

Description

For one subject/time range, compute all available PK parameters. All the internal options should be set by `PKNCA.options()` prior to running. The only part that changes with a call to this function is the concentration and time.

Usage

```
pk.nca.interval(  
  conc,  
  time,  
  volume,  
  duration.conc,  
  dose,  
  time.dose,  
  duration.dose,  
  route,  
  conc.group = NULL,  
  time.group = NULL,  
  volume.group = NULL,  
  duration.conc.group = NULL,  
  dose.group = NULL,  
  time.dose.group = NULL,  
  duration.dose.group = NULL,  
  route.group = NULL,  
  conc.sparse = NULL,  
  time.sparse = NULL,  
  conc.sparse.group = NULL,  
  time.sparse.group = NULL,  
  impute_method = NA_character_,  
  include_half.life = NULL,  
  exclude_half.life = NULL,  
  lloq = NULL,  
  subject = NULL,  
  interval,  
  options = list(),  
  timeu = NULL  
)
```

Arguments

conc	Measured concentrations
time	Time of the measurement of the concentrations

volume, volume.group	The volume (or mass) of the concentration measurement for the current interval or all data for the group (typically for urine and fecal measurements)
duration.conc, duration.conc.group	The duration of the concentration measurement for the current interval or all data for the group (typically for urine and fecal measurements)
dose, dose.group	Dose amount (may be a scalar or vector) for the current interval or all data for the group
time.dose	Time of the dose for the current interval (must be the same length as dose)
duration.dose	The duration of the dose administration for the current interval (typically zero for extravascular and intravascular bolus and nonzero for intravascular infusion)
route, route.group	The route of dosing for the current interval or all data for the group
conc.group	All concentrations measured for the group
time.group	Time of all concentrations measured for the group
time.dose.group	Time of the dose for all data for the group (must be the same length as dose.group)
duration.dose.group	The duration of the dose administration for all data for the group (typically zero for extravascular and intravascular bolus and nonzero for intravascular infusion)
conc.sparse, time.sparse	The pooled individual concentrations and their times for the current interval with sparse PK (conc and time are the arithmetic-mean profile built from them). NULL for dense PK.
conc.sparse.group, time.sparse.group	The pooled individual concentrations and their times for all data for the group with sparse PK. NULL for dense PK.
impute_method	The method to use for imputation as a character string
exclude_half.life, include_half.life	Manual half-life point selection, given as a logical value per concentration measurement (or, in PKNCAconc() , the name of such a column in the data). <code>exclude_half.life</code> drops the flagged points; automatic curve-stripping point selection is still performed on the remaining (non-excluded) points and is not bypassed. <code>include_half.life</code> names the exact points to use, bypassing automatic curve-stripping point selection. Each value is TRUE, FALSE, or NA (undefined); the column/vector is treated as "in use" for an interval unless it is entirely NA (so an all-FALSE column still counts as in use), so leave it NA (rather than FALSE) where the mechanism should not apply. The column must be logical and must exist in the data; anything else is an error. Only one of <code>exclude_half.life</code> and <code>include_half.life</code> may be in use for a given interval. See the "Half-Life Calculation" vignette for more details on the use of these arguments.
lloq	An optional scalar or vector (the same length as conc) with the lower limit of quantification passed to pk.calc.half.life() for the Tobit half-life method.
subject	Subject identifiers for the pooled sparse samples

interval	One row of an interval definition (see check.interval.specification() for how to define the interval.
options	List of changes to the default PKNCA options (see <code>PKNCA.options()</code>)
timeu	The time unit of the group's times: NULL when it is not known, which is taken to be hours, or NA when it cannot be used. A τ detected from the dose times is matched to the nominal dosing intervals for that unit (see find.dose.regimen()).

Value

A data frame with one row per result, with columns PptestCD, PPORRES, PPANMETH, and exclude. Its "sparse" attribute is a logical vector saying, for each row, whether the parameter that produced it is registered as a sparse PK parameter.

See Also

[check.interval.specification\(\)](#)

pk.nca.intervals	<i>Compute NCA for multiple intervals</i>
------------------	---

Description

Compute NCA for multiple intervals

Usage

```
pk.nca.intervals(
  data_conc,
  data_dose,
  data_intervals,
  options,
  impute,
  data_sparse_conc = NULL,
  verbose = FALSE,
  timeu = NULL,
  warning_prefix = ""
)
```

Arguments

data_conc	A data.frame or tibble with standardized column names as output from <code>prepare_PKNCAconc()</code> . With sparse PK this is the arithmetic-mean profile.
data_dose	A data.frame or tibble with standardized column names as output from <code>prepare_PKNCAdose()</code>
data_intervals	A data.frame or tibble with standardized column names as output from <code>prepare_PKNCAintervals()</code>
options	List of changes to the default PKNCA options (see <code>PKNCA.options()</code>)
impute	The column name in <code>data_intervals</code> to use for imputation

data_sparse_conc	For sparse PK, a data.frame or tibble of the pooled individual samples (including a subject column) as output from <code>prepare_PKNCAconc()</code> ; NULL for dense PK
verbose	Indicate, by <code>message()</code> , the current state of calculation.
timeu	The time unit of the group's times: NULL when it is not known, which is taken to be hours, or NA when it cannot be used. A τ detected from the dose times is matched to the nominal dosing intervals for that unit (see find.dose.regimen()).
warning_prefix	The text naming the group, put before each dose regimen warning

Value

A list with elements "dense" and "sparse", each a data.frame of the NCA results calculated from that concentration representation (or, when no calculation was possible at all, the warning condition saying why)

pk.tss	<i>Compute the time to steady-state (tss)</i>
--------	---

Description

Compute the time to steady-state (tss)

Usage

```
pk.tss(..., type = c("monoexponential", "stepwise.linear"), check = TRUE)
```

Arguments

...	Passed to pk.tss.monoexponential() or pk.tss.stepwise.linear() .
type	The type of Tss to calculate, either <code>stepwise.linear</code> or <code>monoexponential</code>
check	See pk.tss.data.prep()

Value

A data frame with columns as defined from `pk.tss.monoexponential` and/or `pk.tss.stepwise.linear`.

See Also

Other Time to steady-state calculations: [pk.tss.monoexponential\(\)](#), [pk.tss.stepwise.linear\(\)](#)

pk.tss.data.prep	<i>Clean up the time to steady-state parameters and return a data frame for use by the tss calculators.</i>
------------------	---

Description

Clean up the time to steady-state parameters and return a data frame for use by the tss calculators.

Usage

```
pk.tss.data.prep(
  conc,
  time,
  subject,
  treatment,
  subject.dosing,
  time.dosing,
  options = list(),
  conc.blq = NULL,
  conc.na = NULL,
  check = TRUE,
  ...
)
```

Arguments

conc	Measured concentrations
time	Time of the measurement of the concentrations
subject	Subject identifiers (used as a random effect in the model)
treatment	Treatment description (if missing, all subjects are assumed to be on the same treatment)
subject.dosing	Subject number for dosing
time.dosing	Time of dosing
options	List of changes to the default PKNCA options (see <code>PKNCA.options()</code>)
conc.blq	How to handle a BLQ value that is between above LOQ values? See details for description.
conc.na	How to handle NA concentrations. (See clean.conc.na())
check	Run assert_conc_time() ?
...	Discarded inputs to allow generic calls between tss methods.

Value

a data frame with columns for concentration, time, subject, and treatment.

`pk.tss.monoexponential`*Compute the time to steady state using nonlinear, mixed-effects modeling of trough concentrations.*

Description

Trough concentrations are selected as concentrations at the time of dosing. An exponential curve is then fit through the data with a different magnitude by treatment (as a factor) and a random steady-state concentration and time to steady-state by subject (see `random.effects` argument).

Usage

```
pk.tss.monoexponential(  
  ...,  
  tss.fraction = 0.9,  
  output = c("population", "popind", "individual", "single"),  
  check = TRUE,  
  verbose = FALSE  
)
```

Arguments

<code>...</code>	See pk.tss.data.prep()
<code>tss.fraction</code>	The fraction of steady-state required for calling steady-state
<code>output</code>	Which types of outputs should be produced? <code>population</code> is the population estimate for time to steady-state (from an nlme model), <code>popind</code> is the individual estimate (from an nlme model), <code>individual</code> fits each individual separately with a gnls model (requires more than one individual; use <code>single</code> for one individual), and <code>single</code> fits all the data to a single gnls model.
<code>check</code>	See pk.tss.data.prep() .
<code>verbose</code>	Describe models as they are run, show convergence of the model (passed to the nlme function), and additional details while running.

Value

A scalar float for the first time when steady-state is achieved or NA if it is not observed.

References

Maganti, L., Panebianco, D.L. & Maes, A.L. Evaluation of Methods for Estimating Time to Steady State with Examples from Phase 1 Studies. AAPS J 10, 141–147 (2008). <https://doi.org/10.1208/s12248-008-9014-y>

See Also

Other Time to steady-state calculations: [pk.tss\(\)](#), [pk.tss.stepwise.linear\(\)](#)

```
pk.tss.monoexponential.individual
```

A helper function to estimate individual and single outputs for mono-exponential time to steady-state.

Description

This function is not intended to be called directly. Please use `pk.tss.monoexponential`.

Usage

```
pk.tss.monoexponential.individual(
  data,
  output = c("individual", "single"),
  verbose = FALSE
)
```

Arguments

<code>data</code>	a data frame as prepared by <code>pk.tss.data.prep()</code> . It must contain at least columns for subject, time, conc, and <code>tss.constant</code> .
<code>output</code>	a character vector requesting the output types.
<code>verbose</code>	Show verbose output.

Details

If no model converges, then the `tss.monoexponential.single` and/or `tss.monoexponential.individual` column will be set to NA.

Value

A data frame with either one row (if population output is provided) or one row per subject (if `popind` is provided). The columns will be named `tss.monoexponential.population` and/or `tss.monoexponential.popind`.

```
pk.tss.monoexponential.population
```

A helper function to estimate population and popind outputs for mono-exponential time to steady-state.

Description

This function is not intended to be called directly. Please use `pk.tss.monoexponential`.

Usage

```
pk.tss.monoexponential.population(
  data,
  output = c("population", "popind"),
  verbose = FALSE
)
```

Arguments

data	a data frame as prepared by <code>pk.tss.data.prep()</code> . It must contain at least columns for subject, time, conc, and tss.constant.
output	a character vector requesting the output types.
verbose	Show verbose output.

Details

If no model converges, then the `tss.monoexponential.population` column will be set to NA. If the best model does not include a random effect for subject on Tss then the `tss.monoexponential.popind` column of the output will be set to NA.

Value

A data frame with either one row (if population output is provided) or one row per subject (if popind is provided). The columns will be named `tss.monoexponential.population` and/or `tss.monoexponential.popind`.

`pk.tss.stepwise.linear`

Compute the time to steady state using stepwise test of linear trend

Description

A linear slope is fit through the data to find when it becomes non-significant. Note that this is less preferred than the `pk.tss.monoexponential` due to the fact that with more time or more subjects the performance of the test changes (see reference).

Usage

```
pk.tss.stepwise.linear(
  ...,
  min.points = 3,
  level = 0.95,
  verbose = FALSE,
  check = TRUE
)
```

Arguments

...	See pk.tss.data.prep()
min.points	The minimum number of points required for the fit
level	The confidence level required for assessment of steady-state
verbose	Describe models as they are run, show convergence of the model (passed to the nlme function), and additional details while running.
check	See pk.tss.data.prep()

Details

The model is fit with a different magnitude by treatment (as a factor, if given) and a random slope by subject (if given). A minimum of min.points is required to fit the model.

Value

A scalar float for the first time when steady-state is achieved or NA if it is not observed.

References

Maganti L, Panebianco DL, Maes AL. Evaluation of Methods for Estimating Time to Steady State with Examples from Phase 1 Studies. AAPS Journal 10(1):141-7. doi:10.1208/s12248-008-9014-y

See Also

Other Time to steady-state calculations: [pk.tss\(\)](#), [pk.tss.monoexponential\(\)](#)

pk_nca_result_to_df	<i>Convert the grouping info and list of results for each group into a results data.frame</i>
---------------------	---

Description

Convert the grouping info and list of results for each group into a results data.frame

Usage

```
pk_nca_result_to_df(group_info, result)
```

Arguments

group_info	A data.frame of grouping columns
result	A list of data.frames with the results from NCA parameter calculations

Value

A data.frame with group_info and result combined, warnings filtered out, and results unnested.

Description

Compute pharmacokinetic (PK) noncompartmental analysis (NCA) parameters.

Details

PKNCA has been cross-validated with both Phoenix WinNonlin(R) and Pumas (click here for the [cross-validation article](#))

A common workflow would load data from a file or database into a data.frame then run the following code.

Author(s)

Maintainer: Bill Denney <wdenney@humanpredictions.com> ([ORCID](#))

Authors:

- Bill Denney <wdenney@humanpredictions.com> ([ORCID](#))
- Clare Buckeridge <clare.buckeridge@pfizer.com>
- Gerardo Jose Rodriguez <gerardo.jrac@gmail.com> ([ORCID](#))

Other contributors:

- Sridhar Duvvuri [contributor]
- Natalie Sang <nataliesang143@gmail.com> [contributor]

See Also

Useful links:

- <https://pknca.humanpredictions.com/>
- <https://github.com/humanpred/pknca>
- Report bugs at <https://github.com/humanpred/pknca/issues>

Examples

```
## Not run:
# Load concentration-time data into a data.frame called d.conc
# with columns named "conc", "time", and "subject".
my.conc <- PKNCAconc(d.conc, conc~time|subject)
# Load dose-time data into a data.frame called d.dose
# with columns named "dose", "time", and "subject".
my.dose <- PKNCAdose(d.dose, dose~time|subject)
# Combine the concentration-time and dose-time data into an object
# ready for calculations.
```

```
my.data <- PKNCAdata(my.conc, my.dose)
# Perform the calculations
my.results <- pk.nca(my.data)
# Look at summary results
summary(my.results)
# Look at a listing of results
as.data.frame(my.results)

## End(Not run)
```

PKNCA.choose.option	<i>Choose either the value from an option list or the current set value for an option.</i>
---------------------	--

Description

Choose either the value from an option list or the current set value for an option.

Usage

```
PKNCA.choose.option(name, value = NULL, options = list())
```

Arguments

name	The option name requested.
value	A value to check for the option (NULL to choose not to check the value).
options	List of changes to the default PKNCA options (see <code>PKNCA.options()</code>)

Value

The value of the option first from the `options` list and if it is not there then from the current settings.

See Also

Other PKNCA calculation and summary settings: [PKNCA.options\(\)](#), [PKNCA.set.summary\(\)](#), [PKNCA_options_defaults\(\)](#)

PKNCA.options	<i>Set default options for PKNCA functions</i>
---------------	--

Description

This function will set the default PKNCA options. If given no inputs, it will provide the current option set. If given name/value pairs, it will set the option (as in the [options\(\)](#) function). If given a name, it will return the value for the parameter. If given the default option as true, it will provide the default options.

Usage

```
PKNCA.options(..., default = FALSE, check = FALSE, name, value)
```

Arguments

...	options to set or get the value for
default	(re)sets all default options
check	check a single option given, but do not set it (for validation of the values when used in another function)
name	An option name to use with the value.
value	An option value (paired with the name) to set or check (if NULL, the current value of the option is returned).

Details

Options are either for calculation or summary functions. Calculation options are required for a calculation function to report a result (otherwise the reported value will be NA). Summary options are used during summarization and are used for assessing what values are included in the summary.

See the vignette 'Options for Controlling PKNCA' for a current list of options (`vignette("Options-for-Controlling-PKN", package="PKNCA")`).

Value

If...

no arguments are given returns the current options.

a value is set (including the defaults) returns NULL

a single value is requested the current value of that option is returned as a scalar

multiple values are requested the current values of those options are returned as a list

See Also

[PKNCA.options.describe\(\)](#)

Other PKNCA calculation and summary settings: [PKNCA.choose.option\(\)](#), [PKNCA.set.summary\(\)](#), [PKNCA_options_defaults\(\)](#)

Examples

```
PKNCA.options()
PKNCA.options(default=TRUE)
PKNCA.options("auc.method")
PKNCA.options(name="auc.method")
PKNCA.options(auc.method="lin up/log down", min.hl.points=3)
```

```
PKNCA.options.describe
```

Describe a PKNCA.options option by name.

Description

Describe a PKNCA.options option by name.

Usage

```
PKNCA.options.describe(name)
```

Arguments

name	The option name requested.
------	----------------------------

Value

A character string of the description.

See Also

[PKNCA.options\(\)](#)

```
PKNCA.set.summary
```

Define how NCA parameters are summarized.

Description

Define how NCA parameters are summarized.

Usage

```
PKNCA.set.summary(
  name,
  description,
  point,
  spread,
  rounding = list(signif = 3),
  reset = FALSE,
  spread_for = NULL
)
```

Arguments

name	The parameter name or a vector of parameter names. It must have already been defined (see add.interval.col()).
description	A single-line description of the summary
point	The function to calculate the point estimate for the summary. The function will be called as <code>point(x)</code> and must return a scalar value (typically a number, NA, or a string).
spread	Optional. The function to calculate the spread (or variability). The function will be called as <code>spread(x)</code> and must return a scalar or two-long vector (typically a number, NA, or a string).
rounding	Instructions for how to round the value of point and spread. It may either be a list or a function. If it is a list, then it must have a single entry with a name of either "signif" or "round" and a value of the digits to round. If a function, it is expected to return a scalar number or character string with the correct results for an input of either a scalar or a two-long vector.
reset	Reset all the summary instructions to no instruction (this is not intended for general use)
spread_for	Optional. The name of another parameter that name gives the spread of, such as "auclast" for "auclast_se". Where results for name are present, summary.PKNCAresults() summarizes spread_for with these instructions: point is applied to the values of spread_for, spread to the values of name, and description describes the summary. name then has no summary column of its own. spread must be given.

Value

All current summary settings (invisibly)

See Also

[summary.PKNCAresults\(\)](#)

Other PKNCA calculation and summary settings: [PKNCA.choose.option\(\)](#), [PKNCA.options\(\)](#), [PKNCA_options_defaults\(\)](#)

Examples

```
## Not run:
PKNCA.set.summary(
  name="half.life",
  description="arithmetic mean and standard deviation",
  point=business.mean,
  spread=business.sd,
  rounding=list(signif=3)
)

## End(Not run)
```

pknca_cdisc_codes	<i>Every registered CDISC PPTESTCD/PPTEST code, with CT membership</i>
-------------------	--

Description

Enumerates the pptestcd_cdisc/pptest_cdisc values registered on every NCA parameter (see [add.interval.col\(\)](#) and [get.interval.cols\(\)](#)), expanding a route- or sparse-keyed mapping into one row per variant, and flags whether each code is a real CDISC PKPARMCD term. Generated from the registry at call time; never hand-written, so it always reflects the parameters actually registered (including by another package).

Usage

```
pknca_cdisc_codes()
```

Details

A parameter with no CDISC PKPARMCD equivalent (for example a sample count or a standard error, neither of which CDISC assigns its own PP parameter code) keeps a sponsor-defined code with `in_ct` FALSE; that is expected for some "uncommon" tier parameters and, for a few documented exceptions, for "common" tier ones as well (see the comments at their `add.interval.col()` registrations).

Value

A `data.frame` with one row per parameter/variant and columns `parameter`, `tier`, `variant` ("single", a route, or "dense"/"sparse"), `pptestcd_cdisc`, `pptest_cdisc`, and `in_ct` (whether `pptestcd_cdisc` is a code in the CDISC PKPARMCD codelist, checked against the installed `cdiscdata` package; NA for every row, with a message, if `cdiscdata` is not installed).

See Also

[pknca_parameter_table\(\)](#), [add.interval.col\(\)](#)

Other Interval specifications: [add.interval.col\(\)](#), [check.interval.specification\(\)](#), [choose.auc.intervals\(\)](#), [get.interval.cols\(\)](#), [get.parameter.deps\(\)](#), [interval_add_impute\(\)](#), [interval_add_param\(\)](#), [interval_add_secondary\(\)](#), [pknca_check_parameter_classification\(\)](#), [pknca_concepts\(\)](#), [pknca_interval_table\(\)](#), [pknca_match_route\(\)](#), [pknca_parameter_table\(\)](#), [pknca_presets\(\)](#), [pknca_ref\(\)](#)

Examples

```
head(pknca_cdisc_codes())
```

pknca_check_parameter_classification

Report parameters that PKNCA cannot classify for automatic selection

Description

Intended for packages that register their own NCA parameters: call it in your tests to find parameters that will never be selected automatically because they carry no concept.

Usage

```
pknca_check_parameter_classification(param = NULL)
```

Arguments

param Parameter names to describe. The default is every registered parameter.

Value

A data.frame of the unclassifiable parameters, with the same columns as [pknca_parameter_table\(\)](#). Zero rows means everything is classified.

See Also

[pknca_concept\(\)](#), [pknca_parameter_table\(\)](#)

Other Interval specifications: [add.interval.col\(\)](#), [check.interval.specification\(\)](#), [choose.auc.intervals\(\)](#), [get.interval.cols\(\)](#), [get.parameter.deps\(\)](#), [interval_add_impute\(\)](#), [interval_add_param\(\)](#), [interval_add_secondary\(\)](#), [pknca_cdisc_codes\(\)](#), [pknca_concepts\(\)](#), [pknca_interval_table\(\)](#), [pknca_match_route\(\)](#), [pknca_parameter_table\(\)](#), [pknca_presets\(\)](#), [pknca_ref\(\)](#)

Examples

```
pknca_check_parameter_classification()
```

pknca_concept

The concept a parameter calculation function computes

Description

The concept is the kind of quantity a user asks for – "clearance" rather than the fifteen registered clearance parameters. It is stored as an attribute on the calculation function so that it sits with the code that computes it, and so that a parameter added by another package can carry one too.

Usage

```
pknca_concept(x)

pknca_concept(x) <- value
```

Arguments

x A parameter calculation function (for example `pk.calc.cmax`).

value A concept name; see [pknca_concepts\(\)](#) for the ones PKNCA uses.

Details

A function that computes different concepts depending on which parameter it is registered for – `pk.calc.dn()` is the example within PKNCA – should have no concept. Those parameters take the concept of the parameter they depend on. When neither applies, set the concept for the individual parameter with the selection argument of [add.interval.col\(\)](#).

Value

`pknca_concept()` gives the concept, or NULL when none is set. The replacement form gives the function with the attribute set.

See Also

[pknca_concepts\(\)](#), [add.interval.col\(\)](#), and the vignette "Writing Parameter Functions"

Examples

```
pknca_concept(pk.calc.cmax)
```

pknca_concepts	<i>Concepts, tiers, and contexts used to classify NCA parameters</i>
----------------	--

Description

Concepts, tiers, and contexts used to classify NCA parameters

Usage

```
pknca_concepts()

pknca_tiers()

pknca_routes(synonyms = FALSE)

pknca_dosing()

pknca_sample_types()
```

Arguments

synonyms If TRUE, `pknca_routes()` returns the table of spellings that data may use for a route instead of the route names (see [Route synonyms](#)).

Details

A tier of "common" marks a parameter that belongs in a default report for at least one context; "uncommon" marks one that is calculated only when asked for by name. "uncommon" is the default, so a parameter registered without a tier is never selected automatically.

Value

`pknca_concepts()` gives the concept names PKNCA uses, `pknca_tiers()` the reporting tiers, `pknca_routes()` the routes of administration, `pknca_dosing()` the dosing patterns, and `pknca_sample_types()` the sample collection types.

Route synonyms

`pknca_routes(synonyms = TRUE)` returns a data.frame with one row per spelling and the columns

- **synonym:** The spelling, in lower case with single spaces, as it may appear in data (for example "po", "intravenous bolus"). It includes the common abbreviations and the CDISC SDTM ROUTE controlled terminology terms that are extravascular or intravascular routes.
- **route:** One of `pknca_routes()`, or "iv" when the spelling is intravascular but does not say which intravascular route it is (for example "intravenous" does not say bolus or infusion). "iv" is not a value of `pknca_routes()` without synonyms, because PKNCA cannot calculate with a route that does not say how the drug entered.
- **dose_route:** "extravascular" or "intravascular", the values that `PKNCAdose()` accepts for route.

`pknca_match_route()` looks spellings up in this table.

See Also

[pknca_concept\(\)](#), [add.interval.col\(\)](#)

Other Interval specifications: [add.interval.col\(\)](#), [check.interval.specification\(\)](#), [choose.auc.intervals\(\)](#), [get.interval.cols\(\)](#), [get.parameter.deps\(\)](#), [interval_add_impute\(\)](#), [interval_add_param\(\)](#), [interval_add_secondary\(\)](#), [pknca_cdisc_codes\(\)](#), [pknca_check_parameter_classification\(\)](#), [pknca_interval_table\(\)](#), [pknca_match_route\(\)](#), [pknca_parameter_table\(\)](#), [pknca_presets\(\)](#), [pknca_ref\(\)](#)

Examples

```
pknca_concepts()
```

pknca_difftime_to_unit

Convert a difftime to a number in a time unit

Description

pknca_difftime_to_unit() expresses a duration in the time unit of an analysis, such as a collection duration that is the difference of two date-times, so that it can be used with PKNCAconc() and PKNCAdose() times and durations given in that unit. The same conversion turns date-times and difftime durations into numbers in [pk.nca\(\)](#).

Usage

```
pknca_difftime_to_unit(x, unit)
```

Arguments

x	A difftime vector (NA values stay NA, and a zero-length vector gives a zero-length result)
unit	The time unit to express x in, a single string such as "hr" or "day"

Details

A difftime of any of its units (seconds, minutes, hours, days, or weeks) is converted exactly, through seconds. unit may be any time unit that the units package knows, such as "hr", "min", "day", or "week". Hours ("hr") need no conversion, so they work without the units package; every other unit needs the units package installed, which stops with an error from rlang if it is missing.

To convert a number that is not a difftime, say its unit first with [base::as.difftime\(\)](#).

Value

A numeric vector the length of x: the duration in unit, with no unit or other attributes

Errors

An x that is not a difftime stops with the class pknca_error_difftime_not_difftime, because a plain number has no unit to convert from. A unit that is not a single string, or that is not a time unit the units package can convert hours to, stops with the class pknca_error_difftime_unit.

Examples

```
# Hours need no conversion beyond the difftime's own unit
pknca_difftime_to_unit(as.difftime(90, units = "mins"), unit = "hr")

# Any other unit needs the units package
pknca_difftime_to_unit(as.difftime(36, units = "hours"), unit = "day")
# The difference of two date-times
start <- as.POSIXct("2026-10-05 08:00:00", tz = "UTC")
```

```
end <- as.POSIXct("2026-10-05 20:30:00", tz = "UTC")
pknca_difftime_to_unit(difftime(end, start), unit = "min")
```

pknca_exclude_rules *List the automatic NCA result exclusion rules*

Description

Every `exclude_nca_*`() rule is registered with its description and the descriptions of its arguments. Its arguments and their defaults come from the factory function, and the parameters it can exclude and the `PKNCA.options()` entries it uses come from the exclusion function that the factory returns when called with its defaults (an argument without a default falls back to its option, as it does in use).

Usage

```
pknca_exclude_rules()
```

Value

A tibble with one row per rule and the columns:

rule The function name (for example, "exclude_nca_span_ratio")

description The one-line description

arguments A list column of data.frames with one row per argument and the columns argument, default (the deparsed default, or NA when there is none), and description

callable_with_defaults Can the rule be created without giving any arguments?

options A list column of the `PKNCA.options()` names that the rule uses when created with its defaults, or NULL when it cannot be

affected_parameters A list column of the NCA parameters that the rule, created with its defaults, can exclude, or NULL when it cannot be created without arguments

Rules

Rule	Description
<code>exclude_nca_by_param()</code>	Exclude based on NCA parameter thresholds
<code>exclude_nca_count_conc_measured()</code>	Exclude based on the count of concentrations measured and not below the lower
<code>exclude_nca_max_aucinf_pext()</code>	Exclude based on the percent of AUC extrapolated to infinity (both observed and
<code>exclude_nca_min_hl_adj_r_squared()</code>	Exclude based on half-life adjusted r-squared
<code>exclude_nca_min_hl_r_squared()</code>	Exclude based on half-life r-squared
<code>exclude_nca_span_ratio()</code>	Exclude based on the half-life span ratio
<code>exclude_nca_tmax_0()</code>	Exclude based on implausibly early Tmax (special case for <code>tmax_early = 0</code>)
<code>exclude_nca_tmax_coverage()</code>	Exclude based on whether a subject has a sample within Tukey's fences around t
<code>exclude_nca_tmax_early()</code>	Exclude based on implausibly early Tmax (often used for extravascular dosing w

See Also

[exclude_nca](#), [exclude\(\)](#)

Other Result exclusions: [exclude\(\)](#), [exclude_nca](#), [pknca_missing_samples\(\)](#)

Examples

```
rules <- pknca_exclude_rules()
rules[, c("rule", "description")]
```

pknca_find_units_param

Find NCA parameters with a given unit type

Description

Find NCA parameters with a given unit type

Usage

```
pknca_find_units_param(unit_type)
```

Arguments

unit_type The type of unit as assigned with `add.interval.col`

Value

A character vector of parameters with a given unit type

PKNCA_impute_fun_list *Separate out a vector of PKNCA imputation methods into a list of functions*

Description

Each imputation string is split at commas and spaces, and each method name is expanded to its function name by adding PKNCA_impute_method_ to the beginning. An error will be raised if the functions are not found.

Usage

```
PKNCA_impute_fun_list(x)
```

Arguments

x The character vector of PKNCA imputation method strings (without the PKNCA_impute_method_ part, like "start_predose,start_conc0")

Value

A list with one element per element of x, each a character vector of function names to run in order (or NA_character_ for no imputation).

See Also

[assert_impute_method\(\)](#), [PKNCA_impute_method](#)

Other Imputation: [PKNCA_impute_method](#), [assert_impute_method\(\)](#), [get_impute_column\(\)](#), [get_impute_method\(\)](#), [pknca_impute_methods\(\)](#)

Examples

```
PKNCA_impute_fun_list(c("start_predose,start_conc0", NA))
```

PKNCA_impute_method *Methods for imputation of data with PKNCA*

Description

Methods for imputation of data with PKNCA

Usage

```
PKNCA_impute_method_start_conc0(conc, time, start = 0, ..., options = list())
```

```
PKNCA_impute_method_start_cmin(conc, time, start, end, ..., options = list())
```

```
PKNCA_impute_method_start_predose(
```

```
  conc,
  time,
  start,
  end,
  conc.group,
  time.group,
  ...,
  max_shift = NA_real_,
  options = list()
)
```

```
PKNCA_impute_method_start_predose_conc0(
```

```
  conc,
  time,
  start,
  end,
  conc.group,
  time.group,
  ...,
  max_shift = NA_real_,
  options = list()
)
```

```
PKNCA_impute_method_end_conc_drop(conc, time, end, ..., options = list())
```

Arguments

<code>conc</code>	Measured concentrations
<code>time</code>	Time of the measurement of the concentrations
<code>start</code>	The start time of the interval
<code>...</code>	ignored
<code>options</code>	List of changes to the default PKNCA options (see <code>PKNCA.options()</code>)
<code>end</code>	The end time of the interval
<code>conc.group</code>	All concentrations measured for the group
<code>time.group</code>	Time of all concentrations measured for the group
<code>max_shift</code>	The maximum amount of time to shift a concentration forward (defaults to 5% of the interval duration, i.e. $0.05 \times (\text{end} - \text{start})$, if <code>is.finite(end)</code> , and when <code>is.infinite(end)</code> , defaults to 5% of the time from start to <code>max(time)</code>)

Value

A data.frame with one column named conc with imputed concentrations and one column named time with the times.

Functions

- `PKNCA_impute_method_start_conc0()`: Set the concentration at the start time to 0, even if a nonzero concentration exists at that time (usually used with single-dose data). Forcing the start concentration to zero is intentional: an existing start-time value is replaced with 0, including a nonzero predose measurement shifted to the start time by `start_predose`, so the imputation chain "`start_predose, start_conc0`" gives the same result as "`start_conc0`" alone. To carry a predose measurement to the start time, use `start_predose` without `start_conc0`. When no observation exists at the start time, a new row with a concentration of 0 is added.
- `PKNCA_impute_method_start_cmin()`: Add a new concentration of the minimum during the interval at the start time (usually used with multiple-dose data)
- `PKNCA_impute_method_start_predose()`: Shift a predose concentration to become the time zero concentration (only if a time zero concentration does not exist). The most recent predose sample with a measured concentration is shifted; samples with a missing concentration are skipped.
- `PKNCA_impute_method_start_predose_conc0()`: Use a predose concentration as the start concentration when one is available and 0 when it is not. A concentration measured at the start time is kept as-is, which is what distinguishes this from `start_conc0`: a measured concentration at the time of an intravenous bolus dose is the C0 for that dose, and `start_conc0` would replace it with 0. The chain "`start_predose, start_conc0`" cannot express this, because `start_conc0` overwrites whatever `start_predose` shifted.
- `PKNCA_impute_method_end_conc_drop()`: Drop a concentration measured exactly at the end of the interval, if one is present (usually used with multiple-dose data when a point at the interval boundary belongs to the next dose, e.g. an imputed C0)

See Also

`pknca_impute_methods()` lists the methods.

Other Imputation: `PKNCA_impute_fun_list()`, `assert_impute_method()`, `get_impute_column()`, `get_impute_method()`, `pknca_impute_methods()`

`pknca_impute_methods` *List the imputation methods*

Description

Every `PKNCA_impute_method_*`() function is registered with its description; its arguments and their defaults come from the function itself. The method names are what imputation strings use (see `PKNCA_impute_fun_list()` and the `impute` argument of `PKNCAdata()`).

Usage

```
pknca_impute_methods()
```

Details

Registration only describes PKNCA's own methods. A method is still found by its function name: a user-defined `PKNCA_impute_method_<name>()` function that PKNCA can see (for example, in the global environment) is used by the imputation string "`<name>`" without being registered, and it is not listed here.

Value

A tibble with one row per method, sorted by method, and the columns:

- method** The name used in imputation strings (for example, "`start_conc0`")
- fun** The function name (for example, "`PKNCA_impute_method_start_conc0`")
- description** The one-line description
- arguments** A list column of data.frames with one row per argument and the columns `argument` and `default` (the deparsed default, or NA when there is none)

Methods

Method	Description
"end_conc_drop"	Drop a concentration measured exactly at the end of the interval, if one is present (usually used with <code>start_conc0</code>)
"start_cmin"	Add a new concentration of the minimum during the interval at the start time (usually used with <code>start_conc0</code>)
"start_conc0"	Set the concentration at the start time to 0, even if a nonzero concentration exists at that time (usually used with <code>start_predose</code>)
"start_predose"	Shift a predose concentration to become the time zero concentration (only if a time zero concentration exists)
"start_predose_conc0"	Use a predose concentration as the start concentration when one is available and 0 when it is not.

See Also

Other Imputation: [PKNCA_impute_fun_list\(\)](#), [PKNCA_impute_method](#), [assert_impute_method\(\)](#), [get_impute_column\(\)](#), [get_impute_method\(\)](#)

Examples

```
pknca_impute_methods()[, c("method", "description")]
```

pknca_interval_table *Build an interval specification from a description of the analysis*

Description

Chooses the NCA parameters to calculate for an interval, and the imputation to calculate them from, given when the interval runs and how the drug was given. The result is a data.frame suitable for the intervals argument of [PKNCAdata\(\)](#).

Usage

```
pknca_interval_table(
  start,
  end,
  dosing = "single",
  route = "extravascular",
  sample_type = "spot",
  sparse = FALSE,
  tier = "common",
  include = NULL,
  exclude = NULL,
  impute = NULL,
  preset = NULL,
  clast_type = "obs",
  ...
)
```

Arguments

start, end	The start and end time of the interval. Both may be vectors, giving one set of rows per interval.
dosing	Was the drug given once ("single"), repeatedly without assuming steady state ("multiple"), or repeatedly at steady state ("steady_state")? Asking for "steady_state" also gives the parameters that apply to any repeated dose. See pknca_dosing() .
route	How the drug was given; see pknca_routes() .
sample_type	Were concentrations measured in samples taken at a point in time ("spot", the usual case for blood, plasma, and serum) or in a collection over an interval ("interval", the usual case for urine and feces)? See pknca_sample_types() .
sparse	Is this a sparse sampling design? The parameters chosen are the same either way – the AUCs and AUMCs with a sparse estimator are estimated with the sparse methods when the data are sparse, and the rest are calculated from the arithmetic-mean profile – but the imputation differs: the sparse estimators accept only an imputed zero, so a sparse single dose imputes zero at the start ("start_conc0"), and other sparse designs impute nothing.

tier	"common" (the default) gives the parameters usually reported for the context; "all" gives every parameter it can calculate.
include, exclude	NCA parameters or concepts (see pknca_concepts()) to add to or remove from what the context gives. A parameter named in both is an error.
impute	The imputation to use, as for PKNCAdata() . The default of NULL chooses one from the context; NA uses none.
preset	A named set of arguments to start from; see pknca_presets() . Arguments given explicitly override the preset.
clast_type	Should the extrapolation to infinity use the observed ("obs") or predicted ("pred") last concentration?
...	Columns to add to every row, typically the groups the interval applies to.

Details

The interval is assumed to start at a dose, which is what makes an imputation at the start meaningful. Pass `impute = NA` for an interval that starts partway through a profile.

Which AUC the interval is built on follows from dosing: a single dose uses `AUClast` and the extrapolation to infinity, and a repeated dose uses the `AUCint` family, which interpolates at both interval boundaries. The clearance, volume, and mean residence time that follow from that AUC are chosen to match.

Value

A data.frame with one or more rows per interval: start, end, a logical column per parameter, an impute column when any imputation applies, and any columns given in ... An interval is split into more than one row when some of its parameters must not be calculated from the imputed data.

See Also

[pknca_parameter_table\(\)](#) for how each parameter is classified, [PKNCAdata\(\)](#), and the vignette "Selection of Calculation Intervals"

Other Interval specifications: [add.interval.col\(\)](#), [check.interval.specification\(\)](#), [choose.auc.intervals\(\)](#), [get.interval.cols\(\)](#), [get.parameter.deps\(\)](#), [interval_add_impute\(\)](#), [interval_add_param\(\)](#), [interval_add_secondary\(\)](#), [pknca_cdisc_codes\(\)](#), [pknca_check_parameter_classification\(\)](#), [pknca_concepts\(\)](#), [pknca_match_route\(\)](#), [pknca_parameter_table\(\)](#), [pknca_presets\(\)](#), [pknca_ref\(\)](#)

Examples

```
# A single oral dose
pknca_interval_table(0, 24, dosing = "single", route = "extravascular")

# At steady state, with the fluctuation parameters added
pknca_interval_table(
  144, 168,
  dosing = "steady_state", route = "extravascular",
  include = "fluctuation"
)
```

```
# A urine collection
pknca_interval_table(0, 24, dosing = "single", route = "extravascular",
                     sample_type = "interval")
```

pknca_match_route	<i>Match spellings of a route of administration to the route PKNCA uses</i>
-------------------	---

Description

Match spellings of a route of administration to the route PKNCA uses

Usage

```
pknca_match_route(x)
```

Arguments

x	A character vector (or factor) of route spellings, such as "PO" or "INTRAVENOUS BOLUS". Matching ignores case and leading, trailing, and repeated white space.
---	--

Value

A data.frame with one row for each element of x and the columns route (one of [pknca_routes\(\)](#), or "iv" when the spelling is intravascular but does not say which intravascular route it is) and dose_route ("extravascular" or "intravascular", the values [PKNCAdose\(\)](#) accepts for route). Both are NA when the spelling is not known or x is NA.

See Also

The Route synonyms section of [pknca_routes\(\)](#) for the spellings that are known.

Other Interval specifications: [add.interval.col\(\)](#), [check.interval.specification\(\)](#), [choose.auc.intervals\(\)](#), [get.interval.cols\(\)](#), [get.parameter.deps\(\)](#), [interval_add_impute\(\)](#), [interval_add_param\(\)](#), [interval_add_secondary\(\)](#), [pknca_cdisc_codes\(\)](#), [pknca_check_parameter_classification\(\)](#), [pknca_concepts\(\)](#), [pknca_interval_table\(\)](#), [pknca_parameter_table\(\)](#), [pknca_presets\(\)](#), [pknca_ref\(\)](#)

Examples

```
pknca_match_route(c("PO", "Intravenous", "IV BOLUS", "unknown"))
```

pknca_missing_samples *Find the samples missing from each subject's nominal sampling schedule*

Description

The schedule of a group is every nominal time that has a row in the concentration data of that group, where the group is every grouping variable of the concentration data except the subject (for example, the study, treatment, and analyte). Each subject of the group is expected to have a usable concentration at every time of its group's schedule, and a time without one is reported. Missing samples are therefore found only when some subject of the group has a row at that nominal time; a time that no subject has a row for is not part of the schedule. Recording a missed sample as a row with an NA concentration keeps the time in the schedule even when every subject missed it.

Usage

```
pknca_missing_samples(object)
```

Arguments

object	A PKNCAconc, PKNCAdata, or PKNCAResults object (the concentration data of a PKNCAdata or PKNCAResults object are checked). Its concentration data must have a nominal time (the <code>time.nominal</code> argument of PKNCAconc()).
--------	--

Details

A nominal time of a subject's schedule is reported with one of these reasons:

- "no row": the subject has no row at that nominal time.
- "NA concentration": every row of the subject at that nominal time has an NA concentration.
- "excluded": the subject has a concentration at that nominal time, but every row with one is excluded (with the `exclude` column of [PKNCAconc\(\)](#) or with [exclude\(\)](#)), so the calculations do not use it.

A concentration below the limit of quantification (zero) is a usable sample. Rows with an NA nominal time (unscheduled samples) are not part of any schedule, and they neither add a time to it nor fill one. Nominal times are compared as they are, so they may be in any unit.

The nominal times must share one origin for each subject (for example, the time since the first dose). Nominal times that restart at each dose repeat for each dose, so one dose's samples would fill another dose's missing times. They are found when a subject's nominal times go back to the start of its schedule (its first nominal time after zero, or earlier) while its actual times increase; two samples drawn out of order are not a restart. The groups with such a subject are not reported, and a `pknca_warning_missing_samples_nominal_restart` warning names them.

Sparse data (see the `sparse` argument of [PKNCAconc\(\)](#)) have no schedule that each subject follows: each subject gives a few of the group's samples by design. For sparse data, only the rows that exist are checked (reasons "NA concentration" and "excluded"), and a message says that absent rows are not reported.

Value

A data.frame with one row per missing sample and the columns: the grouping columns of the concentration data other than the subject, the subject column, the nominal time column (with the names they have in the concentration data), and reason. It is sorted by those columns and has no rows when no sample is missing.

See Also

[exclude_nca_tmax_coverage\(\)](#), which excludes the results of a subject whose samples miss the Tmax range of its group

Other Result exclusions: [exclude\(\)](#), [exclude_nca](#), [pknca_exclude_rules\(\)](#)

Examples

```
d_conc <-
  data.frame(
    subject = rep(1:3, each = 4),
    time_nominal = rep(c(0, 1, 2, 4), 3),
    time = rep(c(0, 1, 2, 4), 3),
    conc = c(0, 5, 3, 1, 0, 4, NA, 1, 0, 6, 2, 1)
  )
# Subject 3 has no sample at 4 hours
d_conc <- d_conc[-12, ]
o_conc <- PKNCAconc(d_conc, conc ~ time | subject, time.nominal = "time_nominal")
pknca_missing_samples(o_conc)
```

PKNCA_options_defaults

Get the default values of PKNCA options without changing them

Description

Unlike `PKNCA.options(default = TRUE)`, which resets the current options to their defaults, this only reads the default values.

Usage

```
PKNCA_options_defaults(name = NULL)
```

Arguments

name The option name(s) requested, or NULL for all options.

Value

For one name, the default value of that option; otherwise, a named list of default values (all options when name is NULL).

See Also

[PKNCA.options\(\)](#), [PKNCA.options.describe\(\)](#)

Other PKNCA calculation and summary settings: [PKNCA.choose.option\(\)](#), [PKNCA.options\(\)](#), [PKNCA.set.summary\(\)](#)

Examples

```
PKNCA_options_defaults("min.span.ratio")
# The current options are not changed
PKNCA.options(min.span.ratio = 3)
PKNCA_options_defaults("min.span.ratio")
PKNCA.options("min.span.ratio")
PKNCA.options(default = TRUE)
```

pknca_parameter_table *How each NCA parameter is classified for interval selection*

Description

How each NCA parameter is classified for interval selection

Usage

```
pknca_parameter_table(param = NULL)
```

Arguments

param Parameter names to describe. The default is every registered parameter.

Details

A parameter whose concept could not be resolved has NA for concept. That is not an error: it is calculated normally when asked for by name, but it is never selected automatically.

Value

A data.frame with one row per parameter and columns for the concept, tier, sample_type, whether it is sparse, dose_normalized, or secondary (needing inputs from more than one profile), and the route and dosing contexts it applies to (comma-separated).

See Also

[pknca_concept\(\)](#), [get.interval.cols\(\)](#)

Other Interval specifications: [add.interval.col\(\)](#), [check.interval.specification\(\)](#), [choose.auc.intervals\(\)](#), [get.interval.cols\(\)](#), [get.parameter.deps\(\)](#), [interval_add_impute\(\)](#), [interval_add_param\(\)](#), [interval_add_secondary\(\)](#), [pknca_cdisc_codes\(\)](#), [pknca_check_parameter_classification\(\)](#), [pknca_concepts\(\)](#), [pknca_interval_table\(\)](#), [pknca_match_route\(\)](#), [pknca_presets\(\)](#), [pknca_ref\(\)](#)

Examples

```
head(pknca_parameter_table())
```

pknca_presets

Named argument sets for building an interval specification

Description

Named argument sets for building an interval specification

Usage

```
pknca_presets()
```

Value

A named list of the arguments each preset gives to [pknca_interval_table\(\)](#). Arguments given to that function directly override the preset.

See Also

[pknca_interval_table\(\)](#)

Other Interval specifications: [add.interval.col\(\)](#), [check.interval.specification\(\)](#), [choose.auc.intervals\(\)](#), [get.interval.cols\(\)](#), [get.parameter.deps\(\)](#), [interval_add_impute\(\)](#), [interval_add_param\(\)](#), [interval_add_secondary\(\)](#), [pknca_cdisc_codes\(\)](#), [pknca_check_parameter_classification\(\)](#), [pknca_concepts\(\)](#), [pknca_interval_table\(\)](#), [pknca_match_route\(\)](#), [pknca_parameter_table\(\)](#), [pknca_ref\(\)](#)

Examples

```
names(pknca_presets())
pknca_presets()$bioequivalence
```

pknca_ref

Mark a formalsmap entry as coming from the reference interval

Description

Used in the formalsmap argument of [add.interval.col\(\)](#) to declare that an argument takes the value of param calculated in the *reference* interval (the interval named by the <parameter>_ref column of the interval specification) rather than in the current interval.

Usage

```
pknca_ref(param)
```

```
is_pknca_ref(x)
```

Arguments

param	The name of the NCA parameter to take from the reference interval (a single non-missing character string). It does not need to be registered yet when <code>pknca_ref()</code> is called; it is validated when the parameter is calculated.
x	An object to test.

Value

An object of class `pknca_ref`.

See Also

`add.interval.col()`, the vignette "Secondary parameters"

Other Interval specifications: `add.interval.col()`, `check.interval.specification()`, `choose.auc.intervals()`, `get.interval.cols()`, `get.parameter.deps()`, `interval_add_impute()`, `interval_add_param()`, `interval_add_secondary()`, `pknca_cdisc_codes()`, `pknca_check_parameter_classification()`, `pknca_concepts()`, `pknca_interval_table()`, `pknca_match_route()`, `pknca_parameter_table()`, `pknca_presets()`

Examples

```
pknca_ref("aucinf.obs")
```

`pknca_unit_conversion` *Perform unit conversion (if possible) on PKNCA results*

Description

Perform unit conversion (if possible) on PKNCA results

Usage

```
pknca_unit_conversion(result, units, allow_partial_missing_units = FALSE)
```

Arguments

result	The results data.frame
units	The unit conversion table
allow_partial_missing_units	Should missing units be allowed for some but not all parameters?

Value

The result table with units converted

`pknca_unit_reconcile_factor`*Find the conversion factor between two units, when there is one*

Description

The conversions of `pknca_units_table()` are requested by the user and so fail loudly when they cannot be made. This answers the different question of whether two units PKNCA derived itself can be reconciled – units that are unrelated (a concentration and an amount) or outside udunits ("IU/mL") are an expected answer of "no" rather than a mistake.

Usage

```
pknca_unit_reconcile_factor(from, to)
```

Arguments

from, to	Single unit strings, as they appear in the "PPORRESU" and "PPSTRESU" columns of a unit conversion table
----------	---

Value

The number that a value in from units is multiplied by to express it in to units, or NA_real_ when the two are not convertible or the units package is not installed.

See Also

[pknca_units_table\(\)](#)

`pknca_units_add_paren` *Add parentheses to a unit value, if needed*

Description

Add parentheses to a unit value, if needed

Usage

```
pknca_units_add_paren(unit)
```

Arguments

unit	The text of the unit
------	----------------------

Value

The unit with parentheses around it, if needed

pknca_units_conversion_factor

Find the factor converting a value from one unit to another

Description

Find the factor converting a value from one unit to another

Usage

```
pknca_units_conversion_factor(from, to)
```

Arguments

from, to	Single unit strings, as they appear in the "PPORRESU" and "PPSTRESU" columns of a unit conversion table
----------	---

Value

The number that a value in from units is multiplied by to express it in to units. Units that the units package cannot convert between are an error.

pknca_units_table

Create a unit assignment and conversion table

Description

This data.frame is typically used for the units argument for [PKNCAdata\(\)](#). If a unit is not given, then all of the units derived from that unit will be NA.

Usage

```
pknca_units_table(concu, ...)

## Default S3 method:
pknca_units_table(
  concu,
  doseu,
  amountu,
  timeu,
  concu_pref = NULL,
  doseu_pref = NULL,
  amountu_pref = NULL,
  timeu_pref = NULL,
  conversions = data.frame(),
```

```

    ...
)

## S3 method for class 'PKNCAdata'
pknca_units_table(concu, ..., conversions = data.frame())

```

Arguments

concu, doseu, amountu, timeu	Units for concentration, dose, amount, and time in the source data
...	Additional arguments (not used)
concu_pref, doseu_pref, amountu_pref, timeu_pref	Preferred units for reporting; conversions will be automatically.
conversions	An optional data.frame with columns of c("PPORRESU", "PPSTRESU", "conversion_factor") for the original calculation units, the standardized units, and a conversion factor to multiply the initial value by to get a standardized value. This argument overrides any preferred unit conversions from concu_pref, doseu_pref, amountu_pref, or timeu_pref.

Value

A unit conversion table with columns for "PPTTESTCD" and "PPORRESU" if conversions is not given, and adding "PPSTRESU" and "conversion_factor" if conversions is given.

Secondary parameters and reference groups

A secondary parameter (renal clearance, bioavailability, a ratio; see `vignette("v09-secondary-parameters")`) takes one value from the interval requesting it and another from a reference interval, so its units are a quotient of the two groups' units. When the PKNCAdata method finds more than one set of units among the groups, the table therefore gains a `<group column>_ref` column for each of its group columns and, for each secondary parameter, one row per pair of (own group, reference group) with "PPORRESU" composed from the correct side of each. Rows describing a result that names no reference – every primary parameter, and a secondary parameter calculated without a reference – hold NA in the `_ref` columns, and results carrying no reference group join to them. A quotient whose two sides are convertible into one another is dimensionless, and such a row also gets "PPSTRESU" of "fraction" with the "conversion_factor" that turns the raw quotient into that number.

With one set of units for every group there is no reference side to name and no `_ref` column is added.

See Also

The `units` argument for [PKNCAdata\(\)](#)

Examples

```

pknca_units_table() # only parameters that are unitless
pknca_units_table(
  concu="ng/mL", doseu="mg/kg", amountu="mg", timeu="hr"
)

```

```

# Automatic unit conversion needs the units package
pknca_units_table(
  concu="ng/mL", doseu="mg/kg", amountu="mg", timeu="hr",
  # Convert clearance and volume units to more understandable units with
  # automatic unit conversion
  conversions=data.frame(
    PPORRESU=c("(mg/kg)/(hr*ng/mL)", "(mg/kg)/(ng/mL)"),
    PPSTRESU=c("mL/hr/kg", "mL/kg")
  )
)
pknca_units_table(
  concu="mg/L", doseu="mg/kg", amountu="mg", timeu="hr",
  # Convert clearance and volume units to molar units (assuming
  conversions=data.frame(
    PPORRESU=c("mg/L", "(mg/kg)/(hr*ng/mL)", "(mg/kg)/(ng/mL)"),
    PPSTRESU=c("mmol/L", "mL/hr/kg", "mL/kg"),
    # Manual conversion of concentration units from ng/mL to mmol/L (assuming
    # a molecular weight of 138.121 g/mol)
    conversion_factor=c(1/138.121, NA, NA)
  )
)

# This will make all time-related parameters use "day" even though the
# original units are "hr"
pknca_units_table(
  concu = "ng/mL", doseu = "mg/kg", timeu = "hr", amountu = "mg",
  timeu_pref = "day"
)

```

PKNCAconc

Create a PKNCAconc object

Description

Create a PKNCAconc object

Usage

```
PKNCAconc(data, ...)
```

```
## Default S3 method:
PKNCAconc(data, ...)
```

```
## S3 method for class 'tbl_df'
PKNCAconc(data, ...)
```

```
## S3 method for class 'data.frame'
```

```

PKNCAconc(
  data,
  formula,
  subject,
  time.nominal,
  exclude = NULL,
  duration,
  volume,
  exclude_half.life,
  include_half.life,
  lloq,
  sparse = FALSE,
  ...,
  concu = NULL,
  amountu = NULL,
  timeu = NULL,
  concu_pref = NULL,
  amountu_pref = NULL,
  timeu_pref = NULL
)

```

Arguments

data	A data frame with concentration (or amount for urine/feces), time, and the groups defined in formula.
...	Ignored.
formula	The formula defining the concentration~time groups or amount~time groups for urine/feces (In the remainder of the documentation, "concentration" will be used to describe concentration or amount.) One special aspect of the groups part of the formula is that the last group to the left of any / is assumed to be the subject unless the subject argument is given. The time may be numeric, or it may be a date-time (POSIXct) or a date (Date); see the "Date-time input" section.
subject	The column indicating the subject number. If not provided, this defaults to the last grouping variable to the left of a / (for example, Subject with concentration~time Study+Subject) or the last grouping variable when there is no / (for example, Subject with concentration~time Study+Subject). When there are no grouping variables (single-subject data), no subject column is set.
time.nominal	(optional) The name of the nominal time column (if the main time variable is actual time. The time.nominal is not used during calculations; it is available to assist with data summary and checking.
exclude	(optional) The name of a column with concentrations to exclude from calculations and summarization. If given, the column should have values of NA or "" for concentrations to include and non-empty text for concentrations to exclude.
duration	(optional) The duration of collection as is typically used for concentration measurements in urine or feces. The time of a measurement is the start of the

collection, and only the time is used when selecting data for a calculation interval; the duration is not considered. A collection starting within an interval and ending after the interval end contributes its full amount to that interval, so for the simplest interpretation of results, align collection start and end times with interval boundaries. A duration column is added to the data only when this is given; requesting an excretion rate parameter (ermax, ertmax, ertlst) without it is an error. A numeric duration is in the time unit of the analysis; a difftime duration is converted to that unit in `PKNCAdata()`.

volume	(optional) The volume (or mass) of collection as is typically used for urine or feces measurements. A volume column is added to the data only when this is given; requesting a parameter that needs it (ae, fe, volpk, and similar) without it is an error.
exclude_half.life, include_half.life	Manual half-life point selection, given as a logical value per concentration measurement (or, in <code>PKNCAconc()</code> , the name of such a column in the data). <code>exclude_half.life</code> drops the flagged points; automatic curve-stripping point selection is still performed on the remaining (non-excluded) points and is not bypassed. <code>include_half.life</code> names the exact points to use, bypassing automatic curve-stripping point selection. Each value is TRUE, FALSE, or NA (undefined); the column/vector is treated as "in use" for an interval unless it is entirely NA (so an all-FALSE column still counts as in use), so leave it NA (rather than FALSE) where the mechanism should not apply. The column must be logical and must exist in the data; anything else is an error. Only one of <code>exclude_half.life</code> and <code>include_half.life</code> may be in use for a given interval. See the "Half-Life Calculation" vignette for more details on the use of these arguments.
lloq	(optional) The lower limit of quantification used by the Tobit half-life method (<code>hl_method = "tobit"</code>). Either the name of a column in data giving the per-observation LLOQ or a numeric scalar applied to all observations. Values must be numeric, finite, and greater than zero; missing values (NA) are allowed, and NaN is not. A column of numbers is stored as given (an integer column stays integer and a units column stays a units column). A column or scalar that is entirely NA of any class (such as the logical NA of a column read from a file with no LLOQ, or an all-NA integer column) is accepted and stored as double NA. Anything else is the error <code>pknca_error_conc_invalid_lloq</code> . When provided, it is passed through to <code>pk.calc.half.life()</code> . With sparse PK, the half-life is calculated from the mean profile, and the LLOQ of each time is the median of the LLOQs of its samples (missing LLOQs are ignored). See the "Half-Life Calculation with Tobit Regression" vignette for more details.
sparse	Are the concentration-time data sparse PK (commonly used in small nonclinical species or with terminal or difficult sampling) or dense PK (commonly used in clinical studies or larger nonclinical species)?
concu, amountu, timeu	Either unit values (e.g. "ng/mL") or column names within the data where units are provided. For a date-time (POSIXct or Date) time column, <code>timeu</code> must be a unit value, and <code>timeu_pref</code> takes precedence over it (see the "Date-time input" section).
concu_pref, amountu_pref, timeu_pref	Preferred units for reporting (not column names). For a date-time time column,

the times are converted directly to `timeu_pref`, which then is also `timeu`.

Value

A `PKNCAconc` object that can be used for automated NCA.

Date-time input

The concentration time (and the dose time in `PKNCAdose()`) may be a date-time (`POSIXct`) or a date (`Date`; a date is taken as 08:00 on that date, a typical time of a first PK sample, with a warning). Date-times have no numeric unit, so they are converted directly to the time unit used for calculations and reports: `timeu_pref` when given (it takes precedence over `timeu`, and `timeu` is set to it), otherwise `timeu`, otherwise hours (without units). The unit must be a single time unit value (like "hr" or "day"), not a column name. A numeric duration (here or in `PKNCAdose()`) is in that unit, and a `difftime` duration is converted to it exactly.

The times remain date-times in the `PKNCAconc`, `PKNCAdose`, and `PKNCAdata` objects. `pk.nca()` converts them to numeric time relative to the first dose (or first concentration) in each group; see the "Date-time input" section of `PKNCAdata()`. The nominal time (`time.nominal`) is not converted and usually stays numeric.

See Also

Other PKNCA objects: `PKNCAdata()`, `PKNCAdose()`, `PKNCAresults()`

PKNCAdata

Create a PKNCAdata object.

Description

`PKNCAdata()` combines `PKNCAconc` and `PKNCAdose` objects and adds in the intervals for PK calculations.

Usage

```
PKNCAdata(data.conc, data.dose, ...)

## S3 method for class 'PKNCAconc'
PKNCAdata(data.conc, data.dose, ...)

## S3 method for class 'PKNCAdose'
PKNCAdata(data.conc, data.dose, ...)

## Default S3 method:
PKNCAdata(
  data.conc,
  data.dose,
  ...,

```

```

    formula.conc,
    formula.dose,
    impute = NA_character_,
    intervals,
    units,
    options = list(),
    group_ref = NULL,
    grpid_cols = NULL,
    grpid_numeric = NULL
  )

```

Arguments

<code>data.conc</code>	Concentration data as a <code>PKNCAconc</code> object or a data frame
<code>data.dose</code>	Dosing data as a <code>PKNCAdose</code> object (see details)
<code>...</code>	arguments passed to <code>PKNCAdata.default</code>
<code>formula.conc</code>	Formula for making a <code>PKNCAconc</code> object with <code>data.conc</code> . This must be given if <code>data.conc</code> is a <code>data.frame</code> , and it must not be given if <code>data.conc</code> is a <code>PKNCAconc</code> object.
<code>formula.dose</code>	Formula for making a <code>PKNCAdose</code> object with <code>data.dose</code> . This must be given if <code>data.dose</code> is a <code>data.frame</code> , and it must not be given if <code>data.dose</code> is a <code>PKNCAdose</code> object.
<code>impute</code>	Methods for imputation. <code>NA</code> for to search for the column named "impute" in the intervals or no imputation if that column does not exist, a comma-or space-separated list of names, or the name of a column in the intervals <code>data.frame</code> (any column name works, not only "impute", and the column must be character). <code>get_impute_method()</code> gives the exact rule. See <code>vignette("v08-data-imputation", package="PKNCA")</code> for more details.
<code>intervals</code>	A data frame with the AUC interval specifications as defined in <code>check.interval.specification()</code> . If missing, this will be automatically chosen by <code>choose.auc.intervals()</code> . (see details) With date-time data, <code>start</code> and <code>end</code> may be date-times (see the "Date-time input" section).
<code>units</code>	A <code>data.frame</code> of unit assignments and conversions as created by <code>pknca_units_table()</code>
<code>options</code>	List of changes to the default PKNCA options (see <code>PKNCA.options()</code>)
<code>group_ref</code>	The reference profiles for automatically-linked secondary parameters, as a <code>data.frame</code> of group values, optionally parameter-specific (see Details). <code>NULL</code> (the default) derives the reference from the data.
<code>grpid_cols</code>	The grouping columns that prefix the interval number in the <code>PPGRPID</code> column of <code>as.data.frame(results, out_format = "cdisc")</code> , as a named character vector of <code>PKNCAconc()</code> grouping columns (other than the subject and analyte), each with the text written before its value (see <code>as.data.frame.PKNCAresults()</code>). It is the default for the <code>grpid_cols</code> argument there, so every output of one analysis agrees. <code>NULL</code> (the default) gives an interval-only identifier.
<code>grpid_numeric</code>	The names of the columns in <code>grpid_cols</code> whose values are whole numbers of at least 1, such as the period, written as that number in <code>PPGRPID</code> . It is the default for the <code>grpid_numeric</code> argument of <code>as.data.frame.PKNCAresults()</code> . <code>NULL</code> (the default) names no columns.

Details

If `data.dose` is not given or is NA, then the intervals must be given. At least one of `data.dose` and intervals must be given.

A secondary parameter is calculated from a result in another interval, and the interval specification links the two with an `interval_id` column and a `<parameter>_ref` pointer (see [interval_add_secondary\(\)](#)). An `interval_id` identifies one interval: rows that share it (an interval split by imputation, for example) may differ only in the parameters they request, never in start, end, or the groups. Where a request has no pointer and could not otherwise be calculated, `pk.nca()` derives the reference profile from the data: a parameter measured on an interval collection whose inputs are spot samples (renal clearance) takes the nearest profile with no collection volume, and `group_ref` restricts – or, for anything else, supplies – the profiles that may be used. The derived reference interval is created for the calculation only and is not added to the intervals in the result. When more than one profile is equally close, the affected results are NA with the reason in the `exclude` column and a `pknca_warning_secondary_auto_reference` warning.

`group_ref` takes three forms. A `data.frame` of group values applies to every secondary parameter: its columns must be group columns of the concentration data, every column must match (and) for at least one of its rows (or), and every value must appear in the data – for example, with groups crossing TRTP, PCTEST, and PCSPEC, `group_ref = data.frame(PCSPEC = "PLASMA")` directs renal-clearance references to the plasma profiles and `group_ref = data.frame(PCTEST = "midazolam")` directs metabolite ratios to the parent analyte. The same `data.frame` with a parameter column applies each row only to the secondary parameter it names, and the columns a parameter's rows leave NA do not apply to it, so one table can steer renal clearance by PCSPEC and a metabolite ratio by PCTEST: `group_ref = data.frame(parameter = c("clr.obs", "ratio.aucinf.obs"), PCSPEC = c("PLASMA", NA), PCTEST = c(NA, "midazolam"))`. A named list of `data.frames`, one per parameter, says the same thing: `group_ref = list(clr.obs = data.frame(PCSPEC = "PLASMA"), ratio.aucinf.obs = data.frame(PCTEST = "midazolam"))`.

Value

A PKNCAdata object with concentration, dose, interval, and calculation options stored (note that PKNCAdata objects can also have results after a NCA calculations are done to the data).

Date-time input

The concentration and dose times may be date-times (POSIXct) or dates (Date). `PKNCAdata()` checks them and keeps them as they are, and `pk.nca()` converts them to numeric time before it calculates, so the intervals can still be changed after `PKNCAdata()`:

- The time reference is the first dose (ignoring excluded doses) within each combination of the grouping variables (and the subject) shared by the concentration and dose formulas. With `dose~time|Part+Subject`, each subject's first dose in each study part (or period, for a crossover) is time 0; with `dose~time|Subject`, each subject's first dose of the study is time 0. The dose formula must include the subject, so that one reference is never shared by several subjects.
- A subject (group) without an included dose time uses its first concentration (the first one not excluded) as the reference, with a warning. Without dosing data, every reference is the first concentration, within each combination of the concentration grouping variables to the left of any / (so analytes share their subject's reference).

- Sparse data use one reference per group rather than per subject, because every subject in a sparse group shares the group's dosing.
- Numeric time is in the time unit of the `PKNCAconc()` object: `timeu_pref` when given, otherwise `timeu`, otherwise hours (without units). Numeric concentration collection and dosing durations are in that unit, and `difftime` durations are converted to it.
- Manually specified intervals may be numeric times relative to the time reference, in that unit, or date-times. Date-time `start` and `end` (`POSIXct`, or `Date` for 08:00, with a warning) are converted relative to the reference of the group each row applies to. A row that does not name every reference group (for example, a row without `Subject`) applies to every matching group and becomes one row per group, because one absolute window is a different relative window for each subject. A date-time `start` may pair with `end = Inf` (or a `POSIXct Inf`), which stays infinite; the `start` must be finite, both bounds must otherwise be date-times, and the time zone must match the data. `PKNCAdata()` and `set_intervals()` check date-time intervals, and `pk.nca()` converts them; converted intervals have an `interval_time_kind` column ("`datetime`", or "`relative`" for numeric rows added later). The conversion gives the window only: a window starting before a subject's first measurement still needs an imputation rule (`impute`) for a concentration at its start.
- The results of `pk.nca()` keep the converted data that the calculation used (`results$data`), with the time reference of each group in its `time_reference` element and the `time_reference_type` column saying whether it is the "`first_dose`" or the "`first_conc`"; `as.data.frame(results, out_format = "cdisc")` gives, in the `PPRFTDTC` column, the date-time of the reference of each row: the dose that starts its interval, or the first concentration for a "`first_conc`" group (see `as.data.frame.PKNCResults()`).

Both times must be date-times (or dates), not one numeric and one date-time; date-times must have the same time zone; and the dose formula must include the subject of dense data. Otherwise, it is an error. Differences are elapsed time, so a change to or from daylight saving time is handled correctly when the time zone is a named zone (like "`America/New_York`").

See Also

`choose.auc.intervals()`, `pk.nca()`, `pknc_units_table()`

Other PKNCA objects: `PKNCAconc()`, `PKNCAdose()`, `PKNCResults()`

PKNCAdose

Create a PKNCAdose object

Description

Create a PKNCAdose object

Usage

```
PKNCAdose(data, ...)
```

```
## Default S3 method:
```

```

PKNCAdose(data, ...)

## S3 method for class 'tbl_df'
PKNCAdose(data, ...)

## S3 method for class 'data.frame'
PKNCAdose(
  data,
  formula,
  route,
  rate,
  duration,
  time.nominal,
  exclude = NULL,
  ...,
  doseu = NULL,
  doseu_pref = NULL
)

```

Arguments

<code>data</code>	A data frame with time and the groups defined in formula.
<code>...</code>	Ignored.
<code>formula</code>	The formula defining the <code>dose.amount~time groups</code> where time is the time of the dosing and <code>dose.amount</code> is the amount administered at that time (see Details). The time may be numeric, or it may be a date-time (POSIXct) or a date (Date) when the concentration time is too (see the "Date-time input" section of PKNCAconc()).
<code>route</code>	Define the route of administration. The value may be either a column name from the data (checked first) or a character string of either "extravascular" or "intravascular" (checked second). If given as a column name, then every value of the column must be either "extravascular" or "intravascular".
<code>rate, duration</code>	(optional) for "intravascular" dosing, the rate or duration of dosing. If given as a character string, it is the name of a column from the data, and if given as a number, it is the value for all doses. Only one may be given, and if neither is given, then the dose is assumed to be a bolus (<code>duration=0</code>). If rate is given, then the dose amount must be given (the left hand side of the formula). A numeric duration is in the time unit of the analysis; a <code>difftime</code> duration is converted to that unit in PKNCAdata() .
<code>time.nominal</code>	(optional) The name of the nominal time column (if the main time variable is actual time. The <code>time.nominal</code> is not used during calculations; it is available to assist with data summary and checking.
<code>exclude</code>	(optional) The name of a column with doses to exclude from calculations and summarization. If given, the column should have values of NA or "" for doses to include and non-empty text for doses to exclude.
<code>doseu</code>	Either unit values (e.g. "mg") or column names within the data where units are provided.

doseu_pref Preferred units for reporting (not column names)

Details

The formula for a PKNCAdose object can be given three ways: one-sided (missing left side), one-sided (missing right side), or two-sided. Each of the three ways can be given with or without groups. When given one-sided missing the left side, the left side can either be omitted or can be given as a period (.): ~time|treatment+subject and .~time|treatment+subject are identical, and dose-related NCA parameters will all be reported as not calculable (for example, clearance). When given one-sided missing the right side, the right side must be specified as a period (.): dose~.|treatment+subject, and only a single row may be given per group. When the right side is missing, PKNCA assumes that the same dose is given in every interval. When given as a two-sided formula, both the dose amount and time are used directly from the data.

Value

A PKNCAdose object that can be used for automated NCA.

See Also

Other PKNCA objects: [PKNCAconc\(\)](#), [PKNCAdata\(\)](#), [PKNCAResults\(\)](#)

PKNCAResults	<i>Generate a PKNCAResults object</i>
--------------	---------------------------------------

Description

This function should not be run directly. The object is created for summarization.

Usage

```
PKNCAResults(result, data, exclude = NULL)
```

Arguments

result	a data frame with NCA calculation results and groups. Each row is one interval and each column is a group name or the name of an NCA parameter.
data	The PKNCAdata used to generate the result
exclude	(optional) The name of a column with concentrations to exclude from calculations and summarization. If given, the column should have values of NA or "" for concentrations to include and non-empty text for concentrations to exclude.

Value

A PKNCAResults object with each of the above within.

See Also

Other PKNCA objects: [PKNCAconc\(\)](#), [PKNCAdata\(\)](#), [PKNCAdose\(\)](#)

print.PKNCAconc	<i>Print and/or summarize a PKNCAconc or PKNCAdose object.</i>
-----------------	--

Description

Print and/or summarize a PKNCAconc or PKNCAdose object.

Usage

```
## S3 method for class 'PKNCAconc'
print(x, n = 6, summarize = FALSE, ...)

## S3 method for class 'PKNCAconc'
summary(object, n = 0, summarize = TRUE, ...)

## S3 method for class 'PKNCAdose'
print(x, n = 6, summarize = FALSE, ...)

## S3 method for class 'PKNCAdose'
summary(object, n = 0, summarize = TRUE, ...)
```

Arguments

x	The object to print
n	The number of rows of data to show (see head())
summarize	Summarize the nested number of groups
...	Arguments passed to print.formula and print.data.frame
object	The object to summarize

print.PKNCAdata	<i>Print a PKNCAdata object</i>
-----------------	---------------------------------

Description

Print a PKNCAdata object

Usage

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

Arguments

x	The object to print
...	Arguments passed on to print.PKNCAconc() and print.PKNCAdose()

<code>print.provenance</code>	<i>Print the summary of a provenance object</i>
-------------------------------	---

Description

Print the summary of a provenance object

Usage

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

Arguments

<code>x</code>	The object to be printed
<code>...</code>	Ignored

Value

invisible text of the printed information

<code>print.summary_PKNCResults</code>	<i>Print the results summary</i>
--	----------------------------------

Description

Print the results summary

Usage

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

Arguments

<code>x</code>	A <code>summary_PKNCResults</code> object
<code>...</code>	passed to <code>print.data.frame</code> (<code>row.names</code> is always set to <code>FALSE</code>)

Value

`x` invisibly

See Also

```
summary\_PKNCResults\(\)
```

resolve_dose_tau	<i>Determine the dosing interval (tau) to use for a calculation interval</i>
------------------	--

Description

A tau column in the interval specification takes precedence over detection so that designs where only the steady-state dose is present in the dosing data (nothing repeats, so nothing can be detected) can still be calculated. Otherwise tau is detected from the group's dose times with [find.tau\(\)](#), which gives a "pknca_warning_tau_irregular_dosing" warning when the doses are spaced as though one was missed.

Usage

```
resolve_dose_tau(interval, time.dose, options = list(), timeu = NULL)
```

Arguments

interval	One row of an interval definition (see check.interval.specification())
time.dose	The dose times for the whole group (not just the interval; an interval one tau long contains a single dose, so nothing repeats within it)
options	List of changes to the default PKNCA options (see PKNCA.options())
timeu	The time unit of time.dose (see choose.auc.intervals())

Value

The dosing interval, or NA_real_ when it cannot be determined

See Also

Other Interval determination: [choose.auc.intervals\(\)](#), [find.dose.regimen\(\)](#), [find.tau\(\)](#)

roundingSummarize	<i>During the summarization of PKNCAresults, do the rounding of values based on the instructions given.</i>
-------------------	---

Description

During the summarization of PKNCAresults, do the rounding of values based on the instructions given.

Usage

```
roundingSummarize(x, name)
```

Arguments

x	The values to summarize
name	The NCA parameter name (matching a parameter name in <code>PKNCA.set.summary()</code>)

Value

A string of the rounded value

roundString	<i>Round a value to a defined number of digits printing out trailing zeros, if applicable.</i>
-------------	--

Description

Round a value to a defined number of digits printing out trailing zeros, if applicable.

Usage

```
roundString(x, digits = 0, sci_range = Inf, sci_sep = "e", si_range)
```

Arguments

x	The number to round
digits	integer indicating the number of decimal places
sci_range	See help for <code>signifString()</code> (and you likely want to round with <code>signifString</code> if you want to use this argument)
sci_sep	The separator to use for scientific notation strings (typically this will be either "e" or "x10^" for computer- or human-readable output).
si_range	Deprecated, please use <code>sci_range</code>

Details

Values that are not standard numbers like Inf, NA, and NaN are returned as "Inf", "NA", and NaN.

Value

A string with the value

See Also

`round()`, `signifString()`

select_minimal_grouping_cols

Find Minimal Grouping Columns for Strata Reconstruction

Description

This function identifies the smallest set of columns in a data frame whose unique combinations can reconstruct the grouping structure defined by the specified strata columns. It removes duplicate, constant, and redundant columns, then searches for the minimal combination that uniquely identifies each stratum.

Usage

```
select_minimal_grouping_cols(df, strata_cols)
```

Arguments

df	A data frame.
strata_cols	Column names in df whose unique combination defines the strata.

Value

A data frame containing the strata columns and their minimal set of grouping columns.

set_intervals

Set Intervals

Description

Takes in two objects, the PKNCAdata object and the proposed intervals. It will then check that the intervals are valid, given the data object. If the intervals are valid, it will set them in the object. It will return the data object with the intervals set.

Usage

```
set_intervals(data, intervals)
```

Arguments

data	PKNCAdata object
intervals	Proposed intervals

Details

With date-time data, start and end may be date-times; they are kept as given and converted to numeric time relative to each group's time reference by [pk.nca\(\)](#) (see the "Date-time input" section of [PKNCAdata\(\)](#)).

Value

The data object with the intervals set.

setAttributeColumn	<i>Add an attribute to an object where the attribute is added as a name to the names of the object.</i>
--------------------	---

Description

Add an attribute to an object where the attribute is added as a name to the names of the object.

Usage

```
setAttributeColumn(
  object,
  attr_name,
  col_or_value,
  col_name,
  default_value,
  stop_if_default,
  warn_if_default,
  message_if_default
)
```

Arguments

object	The object to set the attribute column on.
attr_name	The attribute name to set
col_or_value	If this exists as a column in the data, it is used as the col_name. If not, this becomes the default_value.
col_name	The name of the column within the dataset to use (if missing, uses attr_name)
default_value	The value to fill in the column if the column does not exist (the column is filled with NA if it does not exist and no value is provided).
stop_if_default, warn_if_default, message_if_default	A character string to provide as an error, a warning, or a message to the user if the default_value is used. They are tested in order (if stop, the code stops; if warning, the message is ignored; and message last).

Value

The object with the attribute column added to the data.

See Also

[getAttributeColumn\(\)](#)

setDuration.PKNCAconc *Set the duration of dosing or measurement*

Description

Set the duration of dosing or measurement

Usage

```
## S3 method for class 'PKNCAconc'
setDuration(object, duration, ...)

setDuration(object, ...)

## S3 method for class 'PKNCAdose'
setDuration(object, duration, rate, dose, ...)
```

Arguments

object	An object to set a duration on
duration	The value to set for the duration or the name of the column in the data to use for the duration.
...	Arguments passed to another setDuration function
rate	(for PKNCAdose objects only) The rate of infusion
dose	(for PKNCAdose objects only) The dose amount

Value

The object with duration set

setExcludeColumn *Set the exclude parameter on an object*

Description

This function adds the exclude column to an object. To change the exclude value, use the [exclude\(\)](#) function.

Usage

```
setExcludeColumn(object, exclude = NULL, dataname = "data")
```

Arguments

object	The object to set the exclude column on.
exclude	The column name to set as the exclude value.
dataname	The name of the data.frame within the object to add the exclude column to.

Value

The object with an exclude column and attribute

setRoute	<i>Set the dosing route</i>
----------	-----------------------------

Description

Set the dosing route

Usage

```
setRoute(object, ...)  
  
## S3 method for class 'PKNCAdose'  
setRoute(object, route, ...)
```

Arguments

object	A PKNCAdose object
...	Arguments passed to another setRoute function
route	A character string indicating one of the following: the column from the data which indicates the route of administration, a scalar indicating the route of administration for all subjects, or a vector indicating the route of administration for each dose in the dataset.

Value

The object with an updated route

signifString	<i>Round a value to a defined number of significant digits printing out trailing zeros, if applicable.</i>
--------------	--

Description

Round a value to a defined number of significant digits printing out trailing zeros, if applicable.

Usage

```
signifString(x, ...)

## S3 method for class 'data.frame'
signifString(x, ...)

## Default S3 method:
signifString(x, digits = 6, sci_range = 6, sci_sep = "e", si_range, ...)
```

Arguments

x	The number to round
...	Arguments passed to methods.
digits	integer indicating the number of significant digits
sci_range	integer (or Inf) indicating when to switch to scientific notation instead of floating point. Zero indicates always use scientific; Inf indicates to never use scientific notation; otherwise, scientific notation is used when $\text{abs}(\log_{10}(x)) > \text{si_range}$.
sci_sep	The separator to use for scientific notation strings (typically this will be either "e" or "x10^" for computer- or human-readable output).
si_range	Deprecated, please use sci_range

Details

Values that are not standard numbers like Inf, NA, and NaN are returned as "Inf", "NA", and NaN.

Value

A string with the value

See Also

[signif\(\)](#), [roundString\(\)](#)

sparse_auc_weight_linear

Calculate the weight for sparse AUC calculation with the linear-trapezoidal rule

Description

The weight is used as the w_i parameter in `pk.calc.sparse_auc()`

Usage

```
sparse_auc_weight_linear(sparse_pk)
```

Arguments

sparse_pk A sparse_pk object from `as_sparse_pk()`

Details

$$w_i = \frac{\delta_{time,i-1,i} + \delta_{time,i,i+1}}{2}$$

$$\delta_{time,i,i+1} = t_{i+1} - t_i$$

Where:

w_i is the weight at time i

$\delta_{time,i-1,i}$ **and** $\delta_{time,i,i+1}$ are the changes between time i-1 and i or i and i+1 (zero outside of the time range)

t_i is the time at time i

Value

A numeric vector of weights for sparse AUC calculations the same length as sparse_pk

See Also

Other Sparse Methods: `as_sparse_pk()`, `pk.calc.aucinf.obs_sparse()`, `pk.calc.aucivlast_sparse()`, `pk.calc.auclast_sparse()`, `pk.calc.sparse_auc()`, `pk.calc.sparse_aumc()`, `sparse_bootstrap()`, `sparse_mean()`, `summary.PKNCAresults_sparse_bootstrap()`

sparse_bootstrap	<i>Bootstrap resampling of sparse PK data</i>
------------------	---

Description

sparse_bootstrap() makes a new sparse `PKNCAconc()` object holding the original data and `n_boot` bootstrap replicates of it, following the stratified nonparametric bootstrap of Shen and Machado (2017). Each replicate resamples the animals with replacement within each stratum, where the stratum is the set of times an animal was sampled: each sampling time with serial sacrifice, or each batch in a batch design. An animal's samples stay together, so the correlation between samples from the same animal is kept. `pk.nca()` then calculates every parameter for every replicate the same way as for the original data (from the arithmetic-mean profile, or with the sparse estimators). Its results summarize the replicates with `summary()`, and `be_assess()` compares groups with the percentile intervals of the replicates.

Usage

```
sparse_bootstrap(
  object,
  n_boot = 200,
  seed = NULL,
  paired = NULL,
  replicate_col = "bootstrap"
)
```

Arguments

object	A sparse <code>PKNCAconc</code> object
n_boot	The number of bootstrap replicates. The default of 200 is enough for the mean and standard deviation of the parameters (Takemoto et al. 2006); percentile confidence intervals need more replicates to be stable.
seed	The random seed for the resampling. When <code>NULL</code> , one is drawn (from the current random number stream) and stored in the result, so the replicates can always be reproduced. The random number state of the session is restored afterward.
paired	Group columns whose levels each animal has, so they are resampled together (for example, the treatment of a crossover design)
replicate_col	The name of the replicate group column to add

Details

The replicates are a new group (`replicate_col`) with the values "original" (the data as given) and "bootstrap1" through "bootstrap<n_boot>". A resampled animal gets a new subject identifier (`<replicate_col>_subject`, the original subject with the draw number) so that an animal drawn twice counts as two animals.

Groups named in `paired`, and the analytes (the groups after / in the formula), are resampled together: an animal drawn for a replicate brings its samples from every level of those groups. Use

paired for a crossover, where each animal (or eye, as in Shen and Machado 2017) receives each treatment. Other groups (such as the treatment of a parallel design) are resampled independently.

The "original" replicate holds the data as given, so the same `pk.nca()` run also gives the estimates.

Value

A sparse PKNCAconc object with the original data and the replicates. Its bootstrap element records `n_boot`, `seed`, `paired`, `replicate_col`, and the original subject column.

References

Shen M, Machado SG. Bioequivalence evaluation of sparse sampling pharmacokinetics data using bootstrap resampling method. *Journal of Biopharmaceutical Statistics*. 2017;27(2):257-264. doi:10.1080/10543406.2016.1265543

Takemoto S, Yamaoka K, Nishikawa M, Takakura Y. Histogram analysis of pharmacokinetic parameters by bootstrap resampling from one-point sampling data in animal experiments. *Drug Metabolism and Pharmacokinetics*. 2006;21(6):458-464. doi:10.2133/dmpk.21.458

See Also

Other Sparse Methods: `as_sparse_pk()`, `pk.calc.aucinf.obs_sparse()`, `pk.calc.aucivlast_sparse()`, `pk.calc.auclast_sparse()`, `pk.calc.sparse_auc()`, `pk.calc.sparse_aumc()`, `sparse_auc_weight_linear()`, `sparse_mean()`, `summary.PKNCAresults_sparse_bootstrap()`

Examples

```
d_sparse <-
  data.frame(
    time = rep(c(0, 1, 2, 4, 8, 24), each = 3),
    conc = c(0, 0, 0, 5, 6, 4, 8, 7, 9, 6, 5, 7, 3, 2.5, 3.5, 0.6, 0.4, 0.5)
  )
d_sparse$animal <- seq_len(nrow(d_sparse))
o_conc <- PKNCAconc(d_sparse, conc ~ time | animal, sparse = TRUE)
o_conc_boot <- sparse_bootstrap(o_conc, n_boot = 20, seed = 1)
o_data_boot <-
  PKNCAdata(o_conc_boot, intervals = data.frame(start = 0, end = 24, auclast = TRUE, cmax = TRUE))
summary(pk.nca(o_data_boot))
```

sparse_mean

Calculate the mean concentration at all time points for use in sparse NCA calculations

Description

Choices for the method of calculation (the argument `sparse_mean_method`) are:

Usage

```
sparse_mean(
  sparse_pk,
  sparse_mean_method = c("arithmetic mean, <=50% BLQ", "arithmetic mean")
)
```

Arguments

`sparse_pk` A `sparse_pk` object from [as_sparse_pk\(\)](#)
`sparse_mean_method` The method used to calculate the sparse mean (see details)

Details

"arithmetic mean" Arithmetic mean (ignoring number of BLQ samples)

"arithmetic mean, <=50% BLQ" If >50% of the measurements are BLQ, zero. Otherwise, the arithmetic mean of all samples (including the BLQ as zero).

Value

A vector the same length as `sparse_pk` with the mean concentration at each of those times.

See Also

Other Sparse Methods: [as_sparse_pk\(\)](#), [pk.calc.aucinf.obs_sparse\(\)](#), [pk.calc.aucivlast_sparse\(\)](#), [pk.calc.auclast_sparse\(\)](#), [pk.calc.sparse_auc\(\)](#), [pk.calc.sparse_aumc\(\)](#), [sparse_auc_weight_linear\(\)](#), [sparse_bootstrap\(\)](#), [summary.PKNCAresults_sparse_bootstrap\(\)](#)

`sparse_pk_attribute` *Set or get a sparse_pk object attribute*

Description

Set or get a `sparse_pk` object attribute

Usage

```
sparse_pk_attribute(sparse_pk, ...)
```

Arguments

`sparse_pk` A `sparse_pk` object from [as_sparse_pk\(\)](#)
`...` Either a character string (to get that value) or a named vector the same length as `sparse_pk` to set the value.

Value

Either the attribute value or an updated `sparse_pk` object

sparse_to_dense_pk	<i>Extract the mean concentration-time profile as a data.frame</i>
--------------------	--

Description

Extract the mean concentration-time profile as a data.frame

Usage

```
sparse_to_dense_pk(sparse_pk)
```

Arguments

sparse_pk A sparse_pk object from [as_sparse_pk\(\)](#)

Value

A data.frame with names of "conc" and "time"

summary.PKNCAdata	<i>Summarize a PKNCAdata object showing important details about the concentration, dosing, and interval information.</i>
-------------------	--

Description

Summarize a PKNCAdata object showing important details about the concentration, dosing, and interval information.

Usage

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

Arguments

object The PKNCAdata object to summarize.
 ... arguments passed on to [print.PKNCAdata\(\)](#)

summary.PKNCAresults	<i>Summarize PKNCA results</i>
----------------------	--------------------------------

Description

Summarize PKNCA results

Usage

```
## S3 method for class 'PKNCAresults'
summary(
  object,
  ...,
  drop_group = as_PKNCAconc(object)$columns$subject,
  drop_param = character(),
  summarize_n = NA,
  not_requested = ".",
  not_calculated = "NC",
  drop.group = deprecated(),
  summarize.n.per.group = deprecated(),
  not.requested.string = deprecated(),
  not.calculated.string = deprecated(),
  pretty_names = NULL,
  caption_prefix = NULL
)
```

Arguments

object	The results to summarize
...	Ignored.
drop_group	Which group(s) should be dropped from the formula?
drop_param	Which parameters should be excluded from the summary?
summarize_n	Should a column for N be added (TRUE or FALSE)? NA means to automatically detect adding N if the data has a subject column indicated. Note that N is maximum number of parameter results for any parameter; if no parameters are requested for a group, then N will be NA.
not_requested	A character string to use when a parameter summary was not requested for a parameter within an interval.
not_calculated	A character string to use when a parameter summary was requested, but the point estimate AND spread calculations (if applicable) returned NA. It is described in the caption when it is used.
drop.group, summarize.n.per.group, not.requested.string, not.calculated.string	Deprecated use drop_group, not_requested, not_calculated, or summarize_n, instead

`pretty_names` Should pretty names (easier to understand in a report) be used? TRUE is yes, FALSE is no, and NULL is yes if units are used and no if units are not used.

`caption_prefix` Prefix to prepend to the generated table caption.

Details

Excluded results will not be included in the summary.

A parameter whose summary instructions name it as the spread of another parameter (`spread_for` in `PKNCA.set.summary()`, such as the standard error of a sparse AUClast) has no column of its own. Where its results are present, the other parameter is summarized with its instructions instead: for a sparse AUClast, the estimate with its standard error. The caption gives the summary used. A summary row with more than one sparse estimate is an error, since each standard error describes one estimate.

Value

A data frame of NCA parameter results summarized according to the summarization settings.

See Also

`PKNCA.set.summary()`, `print.summary_PKNCAresults()`

Examples

```
conc_obj <- PKNCAconc(as.data.frame(datasets::Theoph), conc ~ Time | Subject)
d_dose <-
  unique(datasets::Theoph[
    datasets::Theoph$Time == 0,
    c("Dose", "Time", "Subject")
  ])
dose_obj <- PKNCAdose(d_dose, Dose ~ Time | Subject)
data_obj_automatic <- PKNCAdata(conc_obj, dose_obj)
results_obj_automatic <- pk.nca(data_obj_automatic)
# To get standard results run summary
summary(results_obj_automatic)
# To enable numeric conversion and extraction, do not give a spread function
# and subsequently run as.numeric on the result columns.
PKNCA.set.summary(
  name = c("auclast", "cmax", "half.life", "aucinf.obs"),
  point = business.geomean,
  description = "geometric mean"
)
PKNCA.set.summary(
  name = c("tmax"),
  point = business.median,
  description = "median"
)
summary(results_obj_automatic, not_requested = "NA")
```

summary.PKNCAresults_sparse_bootstrap	<i>Summarize the results of a sparse bootstrap</i>
---------------------------------------	--

Description

The bootstrap replicates (see [sparse_bootstrap\(\)](#)) are summarized the way subjects are for dense data (see [summary.PKNCAresults\(\)](#)), with each parameter’s summary statistics from [PKNCA.set.summary\(\)](#): for example, the geometric mean and geometric coefficient of variation of the replicates’ AUClast. N is the number of animals in the study, and the caption gives the number of bootstrap replicates and the random seed. The original data and the sparse standard errors of single replicates are not part of the summary. For confidence intervals and comparisons between groups, see [be_assess\(\)](#).

Usage

```
## S3 method for class 'PKNCAresults_sparse_bootstrap'
summary(object, ..., drop_group = character(), summarize_n = NA)
```

Arguments

object	The results to summarize
...	Ignored.
drop_group	Groups to drop from the summary, in addition to the bootstrap replicate
summarize_n	Should a column for N, the number of animals in the study, be added? NA (the default) and TRUE add it.

Value

A data frame of the summarized bootstrap results (see [summary.PKNCAresults\(\)](#))

See Also

Other Sparse Methods: [as_sparse_pk\(\)](#), [pk.calc.aucinf.obs_sparse\(\)](#), [pk.calc.aucivlast_sparse\(\)](#), [pk.calc.auclast_sparse\(\)](#), [pk.calc.sparse_auc\(\)](#), [pk.calc.sparse_aumc\(\)](#), [sparse_auc_weight_linear\(\)](#), [sparse_bootstrap\(\)](#), [sparse_mean\(\)](#)

superposition	<i>Compute noncompartmental superposition for repeated dosing</i>
---------------	---

Description

Compute noncompartmental superposition for repeated dosing

Usage

```

superposition(conc, ...)

## S3 method for class 'PKNCAconc'
superposition(conc, ...)

## S3 method for class 'numeric'
superposition(
  conc,
  time,
  dose.input = NULL,
  tau,
  dose.times = 0,
  dose.amount,
  n.tau = Inf,
  options = list(),
  lambda.z,
  clast.pred = FALSE,
  tlast,
  additional.times = numeric(),
  check.blq = TRUE,
  method = NULL,
  auc.type = "AUCinf",
  steady.state.tol = 0.001,
  ...
)

```

Arguments

<code>conc</code>	Measured concentrations
<code>...</code>	Additional arguments passed to the <code>half.life</code> function if required to compute <code>lambda.z</code> .
<code>time</code>	Time of the measurement of the concentrations
<code>dose.input</code>	The dose given to generate the <code>conc</code> and <code>time</code> inputs. If missing, output doses will be assumed to be equal to the input dose.
<code>tau</code>	The dosing interval
<code>dose.times</code>	The time of dosing within the dosing interval. The <code>min(dose.times)</code> must be ≥ 0 , and the <code>max(dose.times)</code> must be $< \tau$. There may be more than one dose times given as a vector.
<code>dose.amount</code>	The doses given for the output. Linear proportionality will be used from the input to output if they are not equal. The length of <code>dose.amount</code> must be either 1 or matching the length of <code>dose.times</code> .
<code>n.tau</code>	The number of τ dosing intervals to simulate or <code>Inf</code> for steady-state.
<code>options</code>	List of changes to the default PKNCA options (see <code>PKNCA.options()</code>)
<code>lambda.z</code>	The elimination rate (in units of inverse time) for extrapolation

<code>clast.pred</code>	To use predicted as opposed to observed Clast, either give the value for <code>clast.pred</code> here or set it to <code>true</code> (for automatic calculation from the half-life).
<code>tlast</code>	The time of last observed concentration above the limit of quantification. This is calculated if not provided.
<code>additional.times</code>	Times to include in the final outputs in addition to the standard times (see details). All <code>min(additional.times)</code> must be ≥ 0 , and the <code>max(additional.times)</code> must be $\leq \tau$.
<code>check.blq</code>	Must the first concentration measurement be below the limit of quantification?
<code>method</code>	The method for integration (one of 'lin up/log down', 'lin-log', or 'linear')
<code>auc.type</code>	The type of AUC to compute. Choices are 'AUCinf', 'AUClast', and 'AUCall'.
<code>steady.state.tol</code>	The tolerance for assessing if steady-state has been achieved (between 0 and 1, exclusive).

Details

The returned superposition times will include all of the following times: 0 (zero), `dose.times`, `time modulo tau` (shifting time for each dose time as well), `additional.times`, and `tau`.

When concentrations are extrapolated as zero after `tlast` (for example, `auc.type = "AUClast"` when `tlast < tau`), some concentrations in the profile can never become nonzero. Steady-state (`n.tau = Inf`) is then assessed on the nonzero concentrations, and a warning is given that zero concentrations remain in the steady-state profile.

Value

A data frame with columns named "conc" and "time".

See Also

[interp.extrap.conc\(\)](#)

<code>time_calc</code>	<i>Times relative to an event (typically dosing)</i>
------------------------	--

Description

Times relative to an event (typically dosing)

Usage

```
time_calc(time_event, time_obs, units = NULL)
```

Arguments

time_event	A vector of times for events
time_obs	A vector of times for observations
units	Passed to base::as.numeric.difftime()

Value

A data.frame with columns for:

event_number_before The index of time_event that is the last one before time_obs or NA if none are before.

event_number_after The index of time_event that is the first one after time_obs or NA if none are after.

time_before The minimum time that the current time_obs is before a time_event, 0 if at least one time_obs == time_event.

time_after The minimum time that the current time_obs is after a time_event, 0 if at least one time_obs == time_event.

time_after_first The time after the first event (may be negative or positive).

time_after and time_before are calculated if they are at the same time as a dose, they equal zero, and otherwise, they are calculated relative to the dose number in the event_number_* columns.

tss.monoexponential.generate.formula

A helper function to generate the formula and starting values for the parameters in monoexponential models.

Description

A helper function to generate the formula and starting values for the parameters in monoexponential models.

Usage

```
tss.monoexponential.generate.formula(data)
```

Arguments

data	The data used for the model
------	-----------------------------

Value

a list with elements for each of the variables

update.PKNCResults	<i>Update existing PKNCResults with new data</i>
--------------------	--

Description

The only thing that can change in the update is the concentration or dose data. All other items (column definitions, etc) must remain the same. If options are not set in data, then the default PKNC options will be considered identical.

Usage

```
## S3 method for class 'PKNCResults'
update(object, data, ...)
```

Arguments

object	The PKNCResults data
data	The new PKNCdata object
...	Ignored

Details

If more than the allowed settings change, then a full recalculation will occur.

This function is typically used with Shiny apps which may repeat analyses with small changes (e.g. point exclusion).

Value

The PKNCResults with updated new data

var_sparse_auc	<i>Calculate the variance for the AUC of sparsely sampled PK</i>
----------------	--

Description

Equation 7.vii in Nedelman and Jia, 1998 is used for this calculation:

Usage

```
var_sparse_auc(sparse_pk)
```

Arguments

sparse_pk	A sparse_pk object from as_sparse_pk()
-----------	--

Details

$$\text{var}(\hat{AUC}) = \sum_{i=0}^m \left(\frac{w_i^2 s_i^2}{r_i} \right) + 2 \sum_{i < j} \left(\frac{w_i w_j r_{ij} s_{ij}}{r_i r_j} \right)$$

The degrees of freedom are the Satterthwaite approximation of equation 6 of the same paper for any sampling design, including subjects with more than one sample (see the "df" attribute of the return value and Details of `cov_holder()` for the covariance).

When the animals form groups sampled at exactly the same times, with no time sampled in more than one group (serial sacrifice, batch, and complete designs), the same variance and degrees of freedom are calculated from the weighted sum of each animal's samples, which is much faster for many animals. For serial sacrifice, the degrees of freedom are then equation 6a of Nedelman, Gibiansky, and Lau (1995).

Value

The variance of the AUC estimate with a "df" attribute containing its degrees of freedom.

References

- Nedelman JR, Jia X. An extension of Satterthwaite's approximation applied to pharmacokinetics. *Journal of Biopharmaceutical Statistics*. 1998;8(2):317-328. doi:10.1080/10543409808835241
- Holder DJ. Comments on Nedelman and Jia's Extension of Satterthwaite's Approximation Applied to Pharmacokinetics. *Journal of Biopharmaceutical Statistics*. 2001;11(1-2):75-79. doi:10.1081/BIP-100104199
- Nedelman JR, Gibiansky E, Lau DTW. Applying Bailer's method for AUC confidence intervals to sparse sampling. *Pharmaceutical Research*. 1995;12(1):124-128.

Index

- * **AUC calculations**
 - pk.calc.auxc, [105](#)
 - pk.calc.auxcint, [108](#)
 - pk.calc.auxciv, [112](#)
- * **AUMC calculations**
 - pk.calc.auxcint, [108](#)
 - pk.calc.auxciv, [112](#)
- * **Bioequivalence**
 - be_assess, [25](#)
 - be_compare, [28](#)
 - be_dataset, [30](#)
 - be_design, [32](#)
 - be_expand_limits, [33](#)
 - be_extract_param, [34](#)
 - be_fit_model_single, [35](#)
 - be_fit_models, [36](#)
 - be_regulator, [37](#)
 - be_table, [39](#)
 - be_within_var, [40](#)
- * **Clearance and volume parameters**
 - pk.calc.cl, [118](#)
 - pk.calc.kel, [132](#)
 - pk.calc.vz, [144](#)
- * **Concentration interpolation and extrapolation**
 - interp.extrap.conc, [85](#)
- * **Data cleaners**
 - clean.conc.blq, [48](#)
 - clean.conc.na, [49](#)
- * **Formula parsing**
 - findOperator, [64](#)
 - parse_formula_to_cols, [97](#)
- * **Half-life and elimination**
 - adj.r.squared, [11](#)
 - pk.calc.aucpext, [104](#)
 - pk.calc.half.life, [129](#)
 - pk.calc.thalf.eff, [138](#)
- * **Imputation**
 - assert_impute_method, [18](#)
 - get_impute_column, [73](#)
 - get_impute_method, [74](#)
 - PKNCA_impute_fun_list, [167](#)
 - PKNCA_impute_method, [167](#)
 - pknca_impute_methods, [169](#)
- * **Internal**
 - any_sparse_dense_in_interval, [12](#)
 - assert_intervaltime_single, [19](#)
 - assert_param_name, [22](#)
 - auc_integrate, [24](#)
 - choose_interval_method, [47](#)
 - cov_holder, [50](#)
 - ensure_column_unit_exists, [51](#)
 - filter_interval, [59](#)
 - getDataName.PKNCAconc, [76](#)
 - interp_extrap_conc_method, [88](#)
 - parse_formula_to_cols, [97](#)
 - pk_nca_result_to_df, [154](#)
 - pknca_find_units_param, [166](#)
 - pknca_unit_conversion, [178](#)
 - pknca_unit_reconcile_factor, [179](#)
 - pknca_units_add_paren, [179](#)
 - pknca_units_conversion_factor, [180](#)
 - resolve_dose_tau, [193](#)
 - select_minimal_grouping_cols, [195](#)
 - sparse_pk_attribute, [203](#)
 - sparse_to_dense_pk, [204](#)
- * **Interval determination**
 - choose.auc.intervals, [44](#)
 - find.dose.regimen, [59](#)
 - find.tau, [62](#)
 - resolve_dose_tau, [193](#)
- * **Interval specifications**
 - add.interval.col, [6](#)
 - check.interval.specification, [43](#)
 - choose.auc.intervals, [44](#)
 - get.interval.cols, [69](#)
 - get.parameter.deps, [69](#)
 - interval_add_impute, [89](#)

- interval_add_param, 90
- interval_add_secondary, 91
- pknca_cdisc_codes, 160
- pknca_check_parameter_classification, 161
- pknca_concepts, 162
- pknca_interval_table, 171
- pknca_match_route, 173
- pknca_parameter_table, 176
- pknca_presets, 177
- pknca_ref, 177
- * **Mean residence time**
 - pk.calc.mrt, 133
- * **Multiple-dose PK parameters**
 - pk.calc.deg.fluc, 122
 - pk.calc.ptr, 134
- * **NCA parameter calculations**
 - pk.calc.half.life, 129
- * **NCA parameters for concentrations during the intervals**
 - pk.calc.c0, 115
 - pk.calc.cav, 117
 - pk.calc.ceoi, 117
 - pk.calc.clast.obs, 119
 - pk.calc.cmax, 120
 - pk.calc.count_conc, 121
 - pk.calc.ctrough, 122
- * **NCA time parameters**
 - pk.calc.tlag, 140
 - pk.calc.tlast, 140
 - pk.calc.tmax, 141
 - pk.calc.tmin, 142
- * **PKNCA calculation and summary settings**
 - PKNCA.choose.option, 156
 - PKNCA.options, 157
 - PKNCA.set.summary, 158
 - PKNCA_options_defaults, 175
- * **PKNCA object extractors**
 - getDataName.PKNCAconc, 76
 - getDepVar, 77
 - getIndepVar, 79
- * **PKNCA objects**
 - PKNCAconc, 182
 - PKNCAdata, 185
 - PKNCAdose, 188
 - PKNCAresults, 190
- * **Result exclusions**
 - exclude, 52
 - exclude_nca, 53
 - pknca_exclude_rules, 165
 - pknca_missing_samples, 174
- * **Sparse Methods**
 - as_sparse_pk, 15
 - pk.calc.aucinf.obs_sparse, 99
 - pk.calc.aucivlast_sparse, 101
 - pk.calc.auclast_sparse, 103
 - pk.calc.sparse_auc, 136
 - pk.calc.sparse_aumc, 137
 - sparse_auc_weight_linear, 200
 - sparse_bootstrap, 201
 - sparse_mean, 202
 - summary.PKNCAresults_sparse_bootstrap, 207
- * **Superposition**
 - superposition, 207
- * **Time to steady-state calculations**
 - pk.tss, 149
 - pk.tss.monoexponential, 151
 - pk.tss.stepwise.linear, 153
- * **Urine/Excretion parameters**
 - pk.calc.ae, 98
 - pk.calc.clr, 119
 - pk.calc.erint, 124
 - pk.calc.erlst, 124
 - pk.calc.ermx, 125
 - pk.calc.ertlst, 126
 - pk.calc.ertmax, 127
 - pk.calc.fe, 128
 - pk.calc.volpk, 143
- * **dplyr verbs**
 - filter.PKNCAresults, 57
 - group_by.PKNCAresults, 79
 - inner_join.PKNCAresults, 81
 - mutate.PKNCAresults, 94
- ?dplyr_by, 58
- ?join_by, 84
- add.interval.col, 6
- add.interval.col(), 12, 43, 47, 69, 70, 90, 91, 93, 159–163, 172, 173, 176–178
- addProvenance, 10
- addProvenance(), 44
- adj.r.squared, 11
- adj.r.squared(), 104, 132, 139
- any_sparse_dense_in_interval, 12
- as.data.frame(), 13
- as.data.frame.PKNCAresults, 12

- `as.data.frame.PKNCAresults()`, 145, 186, 188
- `as_PKNCAconc`, 14
- `as_PKNCAdata` (`as_PKNCAconc`), 14
- `as_PKNCAdose` (`as_PKNCAconc`), 14
- `as_PKNCAresults` (`as_PKNCAconc`), 14
- `as_sparse_pk`, 15
- `as_sparse_pk()`, 50, 101, 102, 104, 137, 138, 200, 202–204, 207, 211
- `assert_aucmethod`, 16
- `assert_conc`, 16
- `assert_conc()`, 120, 121
- `assert_conc_time` (`assert_conc`), 16
- `assert_conc_time()`, 48, 49, 87, 106, 111, 114, 116, 117, 119, 131, 139–142, 150
- `assert_dosetau`, 17
- `assert_impute_method`, 18
- `assert_impute_method()`, 73, 74, 167, 169, 170
- `assert_intervals`, 19
- `assert_intervaltime_single`, 19
- `assert_lambdaz`, 20
- `assert_number_between`, 20
- `assert_numeric_between`, 21
- `assert_param_name`, 22
- `assert_PKNCAconc` (`assert_PKNCAdata`), 23
- `assert_PKNCAdata`, 23
- `assert_PKNCAdose` (`assert_PKNCAdata`), 23
- `assert_PKNCAresults` (`assert_PKNCAdata`), 23
- `assert_time` (`assert_conc`), 16
- `assert_unit` (`assert_unit_col`), 23
- `assert_unit_col`, 23
- `assert_unit_value` (`assert_unit_col`), 23
- `auc_integrate`, 24
- `base::append()`, 89
- `base::as.difftime()`, 164
- `base::I()`, 9
- `be_assess`, 25
- `be_assess()`, 28, 30, 32–37, 39–41, 201, 207
- `be_compare`, 28
- `be_compare()`, 28, 32–37, 39–41
- `be_dataset`, 30
- `be_dataset()`, 28, 30, 33–37, 39–41
- `be_design`, 32
- `be_design()`, 26, 28–30, 32, 34–37, 39–41
- `be_expand_limits`, 33
- `be_expand_limits()`, 28, 30, 32, 33, 35–37, 39–41
- `be_extract_param`, 34
- `be_extract_param()`, 28, 30, 32–34, 36, 37, 39–41
- `be_fit_model_single`, 35
- `be_fit_model_single()`, 28, 30, 32–37, 39–41
- `be_fit_models`, 36
- `be_fit_models()`, 25, 28, 30, 32–36, 39–41
- `be_regulator`, 37
- `be_regulator()`, 26, 28–30, 32–37, 40, 41
- `be_table`, 39
- `be_table()`, 28, 30, 32–37, 39, 41
- `be_within_var`, 40
- `be_within_var()`, 28, 30, 32–37, 39, 40
- `business.cv` (`business.mean`), 41
- `business.geocv` (`business.mean`), 41
- `business.geomean` (`business.mean`), 41
- `business.max` (`business.mean`), 41
- `business.mean`, 41
- `business.median` (`business.mean`), 41
- `business.min` (`business.mean`), 41
- `business.range` (`business.mean`), 41
- `business.sd` (`business.mean`), 41
- `check.conc.time` (`defunct`), 51
- `check.conversion`, 42
- `check.interval.specification`, 43
- `check.interval.specification()`, 10, 46, 47, 69, 70, 90, 91, 93, 148, 160, 161, 163, 172, 173, 176–178, 186, 193
- `checkProvenance`, 44
- `checkProvenance()`, 11, 58
- `choose.auc.intervals`, 44
- `choose.auc.intervals()`, 10, 43, 62, 63, 69, 70, 90, 91, 93, 160, 161, 163, 172, 173, 176–178, 186, 188, 193
- `choose_interval_method`, 47
- `clean.conc.blq`, 48
- `clean.conc.blq()`, 49, 107
- `clean.conc.na`, 49
- `clean.conc.na()`, 48, 49, 87, 106, 111, 131, 150
- `cov_holder`, 50
- `cov_holder()`, 212
- `cross_join`, 84
- `defunct`, 51

- dplyr::filter(), 44
- dplyr::mutate(), 44
- ensure_column_unit_exists, 51
- exclude, 52
- exclude(), 44, 54–56, 145, 166, 174, 175, 197
- exclude_nca, 52, 53, 166, 175
- exclude_nca_by_param, 56
- exclude_nca_by_param(), 165
- exclude_nca_count_conc_measured (exclude_nca), 53
- exclude_nca_count_conc_measured(), 165
- exclude_nca_max.aucinf.pext (exclude_nca), 53
- exclude_nca_max.aucinf.pext(), 165
- exclude_nca_min.hl.adj.r.squared (exclude_nca), 53
- exclude_nca_min.hl.adj.r.squared(), 165
- exclude_nca_min.hl.r.squared (exclude_nca), 53
- exclude_nca_min.hl.r.squared(), 165
- exclude_nca_span.ratio(exclude_nca), 53
- exclude_nca_span.ratio(), 165
- exclude_nca_tmax_0(exclude_nca), 53
- exclude_nca_tmax_0(), 165
- exclude_nca_tmax_coverage (exclude_nca), 53
- exclude_nca_tmax_coverage(), 165, 175
- exclude_nca_tmax_early(exclude_nca), 53
- exclude_nca_tmax_early(), 165
- extrapolate.conc(interp.extrap.conc), 85
- extrapolate_conc_lambdaz (interp_extrap_conc_method), 88
- filter.PKNCAconc (filter.PKNCAresults), 57
- filter.PKNCAdose (filter.PKNCAresults), 57
- filter.PKNCAresults, 57
- filter.PKNCAresults(), 80, 85, 95
- filter_interval, 59
- find.dose.regimen, 59
- find.dose.regimen(), 45–47, 62, 63, 148, 149, 193
- find.tau, 62
- find.tau(), 45, 47, 62, 134, 193
- findOperator, 64
- findOperator(), 97
- fit_half_life, 65
- fit_half_life_tobit, 65
- fit_half_life_tobit_LL, 66
- formula.PKNCAconc, 67
- formula.PKNCAdose (formula.PKNCAconc), 67
- full_join.PKNCAconc (inner_join.PKNCAresults), 81
- full_join.PKNCAdose (inner_join.PKNCAresults), 81
- full_join.PKNCAresults (inner_join.PKNCAresults), 81
- geocv (geomean), 67
- geomean, 67
- geosd (geomean), 67
- get.best.model, 68
- get.interval.cols, 69
- get.interval.cols(), 10, 43, 47, 70, 90, 91, 93, 160, 161, 163, 172, 173, 176–178
- get.parameter.deps, 69
- get.parameter.deps(), 10, 43, 47, 69, 90, 91, 93, 160, 161, 163, 172, 173, 176–178
- get_halflife_curve, 70
- get_halflife_fit, 71
- get_halflife_fit(), 71
- get_halflife_points, 72
- get_halflife_points(), 71, 72
- get_impute_column, 73
- get_impute_column(), 18, 74, 167, 169, 170
- get_impute_method, 74
- get_impute_method(), 18, 19, 73, 167, 169, 170, 186
- getAttributeColumn, 75
- getAttributeColumn(), 196
- getColumnValueOrNot, 75
- getDataName (getDataName.PKNCAconc), 76
- getDataName.PKNCAconc, 76
- getDataName.PKNCAconc(), 77, 79
- getDepVar, 77
- getDepVar(), 77, 79
- getGroups.PKNCAconc, 77
- getGroups.PKNCAdata (getGroups.PKNCAconc), 77
- getGroups.PKNCAdose (getGroups.PKNCAconc), 77

- getGroups.PKNCAResults
 - (getGroups.PKNCAResults), 77
- getIndepVar, 79
- getIndepVar(), 77
- group_by(), 58
- group_by.PKNCAResults
 - (group_by.PKNCAResults), 79
- group_by.PKNCADose
 - (group_by.PKNCAResults), 79
- group_by.PKNCAResults, 79
- group_by.PKNCAResults(), 58, 85, 95
- group_by_drop_default(), 80
- group_vars.PKNCAResults, 80
- group_vars.PKNCAData
 - (group_vars.PKNCAResults), 80
- group_vars.PKNCADose
 - (group_vars.PKNCAResults), 80
- group_vars.PKNCAResults
 - (group_vars.PKNCAResults), 80
- head(), 191
- inner_join.PKNCAResults
 - (inner_join.PKNCAResults), 81
- inner_join.PKNCADose
 - (inner_join.PKNCAResults), 81
- inner_join.PKNCAResults, 81
- inner_join.PKNCAResults(), 58, 80, 95
- interp.extrap.conc, 85
- interp.extrap.conc(), 111, 209
- interp.extrap.conc.dose(), 111, 112
- interp.extrap.conc.method, 88
- interpolate.conc (interp.extrap.conc), 85
- interpolate_conc_linear
 - (interp.extrap.conc.method), 88
- interpolate_conc_log
 - (interp.extrap.conc.method), 88
- interval_add_impute, 89
- interval_add_impute(), 10, 43, 47, 69, 70, 91, 93, 160, 161, 163, 172, 173, 176–178
- interval_add_param, 90
- interval_add_param(), 10, 43, 47, 69, 70, 90, 93, 160, 161, 163, 172, 173, 176–178
- interval_add_secondary, 91
- interval_add_secondary(), 10, 43, 47, 69, 70, 90, 91, 135, 160, 161, 163, 172, 173, 176–178, 187
- interval_remove_impute
 - (interval_add_impute), 89
- interval_remove_param
 - (interval_add_param), 90
- is_pknca_ref (pknca_ref), 177
- is_sparse_pk (is_sparse_pk.PKNCAResults), 93
- is_sparse_pk.PKNCAResults, 93
- join_by(), 84
- left_join.PKNCAResults
 - (inner_join.PKNCAResults), 81
- left_join.PKNCADose
 - (inner_join.PKNCAResults), 81
- left_join.PKNCAResults
 - (inner_join.PKNCAResults), 81
- model.frame.PKNCAResults, 94
- model.frame.PKNCADose
 - (model.frame.PKNCAResults), 94
- mutate.PKNCAResults (mutate.PKNCAResults), 94
- mutate.PKNCADose (mutate.PKNCAResults), 94
- mutate.PKNCAResults, 94
- mutate.PKNCAResults(), 58, 80, 85
- normalize, 95
- normalize(), 44
- normalize_by_col, 96
- normalize_exclude, 96
- options(), 157
- parse_formula_to_cols, 97
- parse_formula_to_cols(), 64
- pk.business, 97
- pk.business(), 42
- pk.calc.ae, 98
- pk.calc.ae(), 120, 124–127, 129, 143
- pk.calc.auc (pk.calc.auxc), 105
- pk.calc.auc(), 46
- pk.calc.aucabove, 99
- pk.calc.aucall_sparse
 - (pk.calc.aucall_sparse), 103
- pk.calc.aucinf.obs_sparse, 99

- pk.calc.aucinf.obs_sparse(), [15](#), [101](#),
[102](#), [104](#), [137](#), [138](#), [200](#), [202](#), [203](#),
[207](#)
- pk.calc.aucint(pk.calc.auxcint), [108](#)
- pk.calc.auciv(pk.calc.auxciv), [112](#)
- pk.calc.auciv_pbext(pk.calc.auxciv),
[112](#)
- pk.calc.aucivall_sparse
(pk.calc.aucivlast_sparse), [101](#)
- pk.calc.aucivinf.obs_sparse
(pk.calc.aucivlast_sparse), [101](#)
- pk.calc.aucivlast_sparse, [101](#)
- pk.calc.aucivlast_sparse(), [15](#), [101](#), [104](#),
[137](#), [138](#), [200](#), [202](#), [203](#), [207](#)
- pk.calc.auclast_sparse, [103](#)
- pk.calc.auclast_sparse(), [15](#), [99](#), [101](#),
[102](#), [137](#), [138](#), [200](#), [202](#), [203](#), [207](#)
- pk.calc.aucpext, [104](#)
- pk.calc.aucpext(), [11](#), [132](#), [139](#)
- pk.calc.aumc(pk.calc.auxc), [105](#)
- pk.calc.aumcall_sparse
(pk.calc.auclast_sparse), [103](#)
- pk.calc.aumcinf.obs_sparse
(pk.calc.aucinf.obs_sparse), [99](#)
- pk.calc.aumcint(pk.calc.auxcint), [108](#)
- pk.calc.aumciv(pk.calc.auxciv), [112](#)
- pk.calc.aumcivall_sparse
(pk.calc.aucivlast_sparse), [101](#)
- pk.calc.aumcivinf.obs_sparse
(pk.calc.aucivlast_sparse), [101](#)
- pk.calc.aumcivlast_sparse
(pk.calc.aucivlast_sparse), [101](#)
- pk.calc.aumclast_sparse
(pk.calc.auclast_sparse), [103](#)
- pk.calc.auxc, [105](#)
- pk.calc.auxc(), [112](#), [115](#)
- pk.calc.auxcint, [108](#)
- pk.calc.auxcint(), [107](#), [115](#)
- pk.calc.auxciv, [112](#)
- pk.calc.auxciv(), [107](#), [112](#)
- pk.calc.c0, [115](#)
- pk.calc.c0(), [87](#), [88](#), [101](#), [114](#), [117–122](#)
- pk.calc.cav, [117](#)
- pk.calc.cav(), [116](#), [118–122](#)
- pk.calc.ceoi, [117](#)
- pk.calc.ceoi(), [116](#), [117](#), [119–122](#)
- pk.calc.cl, [118](#)
- pk.calc.cl(), [133](#), [144](#)
- pk.calc.clast.obs, [119](#)
- pk.calc.clast.obs(), [86](#), [88](#), [116–118](#),
[120–122](#)
- pk.calc.clr, [119](#)
- pk.calc.clr(), [98](#), [124–127](#), [129](#), [143](#)
- pk.calc.cmax, [120](#)
- pk.calc.cmax(), [116–119](#), [121](#), [122](#)
- pk.calc.cmin(pk.calc.cmax), [120](#)
- pk.calc.count_conc, [121](#)
- pk.calc.count_conc(), [116–120](#), [122](#)
- pk.calc.count_conc_measured
(pk.calc.count_conc), [121](#)
- pk.calc.cstart(pk.calc.ctrough), [122](#)
- pk.calc.ctrough, [122](#)
- pk.calc.ctrough(), [116–121](#)
- pk.calc.deg.fluc, [122](#)
- pk.calc.deg.fluc(), [135](#)
- pk.calc.dn, [123](#)
- pk.calc.erint, [124](#)
- pk.calc.erint(), [98](#), [120](#), [125–127](#), [129](#),
[143](#)
- pk.calc.erlst, [124](#)
- pk.calc.erlst(), [98](#), [120](#), [124](#), [126](#), [127](#),
[129](#), [143](#)
- pk.calc.ermx, [125](#)
- pk.calc.ermx(), [98](#), [120](#), [124–127](#), [129](#),
[143](#)
- pk.calc.ertlst, [126](#)
- pk.calc.ertlst(), [98](#), [120](#), [124–127](#), [129](#),
[143](#)
- pk.calc.ertmx, [127](#)
- pk.calc.ertmx(), [98](#), [120](#), [124–126](#), [129](#),
[143](#)
- pk.calc.f, [128](#)
- pk.calc.fe, [128](#)
- pk.calc.fe(), [98](#), [120](#), [124–127](#), [143](#)
- pk.calc.half.life, [129](#)
- pk.calc.half.life(), [11](#), [46](#), [65](#), [66](#), [88](#),
[104](#), [139](#), [147](#), [184](#)
- pk.calc.kel, [132](#)
- pk.calc.kel(), [118](#), [144](#)
- pk.calc.mrt, [133](#)
- pk.calc.ptr, [134](#)
- pk.calc.ptr(), [123](#)
- pk.calc.ratio, [135](#)
- pk.calc.sparse_auc, [136](#)
- pk.calc.sparse_auc(), [15](#), [101–104](#), [138](#),
[200](#), [202](#), [203](#), [207](#)

- pk.calc.sparse_auclast
(pk.calc.sparse_auc), 136
- pk.calc.sparse_aumc, 137
- pk.calc.sparse_aumc(), 15, 101–104, 137,
200, 202, 203, 207
- pk.calc.sparse_aumclast
(pk.calc.sparse_aumc), 137
- pk.calc.swing(pk.calc.deg.fluc), 122
- pk.calc.tfirfirst(pk.calc.tlast), 140
- pk.calc.thalf.eff, 138
- pk.calc.thalf.eff(), 11, 104, 132
- pk.calc.time_above, 139
- pk.calc.tlag, 140
- pk.calc.tlag(), 141, 142
- pk.calc.tlast, 140
- pk.calc.tlast(), 140–142
- pk.calc.tmax, 141
- pk.calc.tmax(), 140–142
- pk.calc.tmin, 142
- pk.calc.tmin(), 140, 141
- pk.calc.totdose, 143
- pk.calc.volpk, 143
- pk.calc.volpk(), 98, 120, 124–127, 129
- pk.calc.vss(pk.calc.vz), 144
- pk.calc.vz, 144
- pk.calc.vz(), 118, 133
- pk.nca, 145
- pk.nca(), 18, 25, 55, 74, 91–93, 99, 103, 130,
134, 164, 185, 187, 188, 195, 201,
202
- pk.nca.interval, 146
- pk.nca.intervals, 148
- pk.tss, 149
- pk.tss(), 151, 154
- pk.tss.data.prep, 150
- pk.tss.data.prep(), 149, 151–154
- pk.tss.monoexponential, 151
- pk.tss.monoexponential(), 149, 154
- pk.tss.monoexponential.individual, 152
- pk.tss.monoexponential.population, 152
- pk.tss.stepwise.linear, 153
- pk.tss.stepwise.linear(), 149, 151
- pk_nca_result_to_df, 154
- PKNCA, 155
- PKNCA-package (PKNCA), 155
- PKNCA.choose.option, 156
- PKNCA.choose.option(), 157, 159, 176
- PKNCA.options, 157
- PKNCA.options(), 46, 54, 99, 112, 131, 145,
146, 156, 158, 159, 165, 176
- PKNCA.options.describe, 158
- PKNCA.options.describe(), 157, 176
- PKNCA.set.summary, 158
- PKNCA.set.summary(), 156, 157, 176, 194,
206, 207
- pknca_cdisc_codes, 160
- pknca_cdisc_codes(), 10, 43, 47, 69, 70, 90,
91, 93, 161, 163, 172, 173, 176–178
- pknca_check_parameter_classification,
161
- pknca_check_parameter_classification(),
10, 43, 47, 69, 70, 90, 91, 93, 160,
163, 172, 173, 176–178
- pknca_concept, 161
- pknca_concept(), 8, 161, 163, 176
- pknca_concept<- (pknca_concept), 161
- pknca_concepts, 162
- pknca_concepts(), 8, 10, 43, 47, 69, 70, 90,
91, 93, 160–162, 172, 173, 176–178
- pknca_difftime_to_unit, 164
- pknca_dosing(pknca_concepts), 162
- pknca_dosing(), 8, 171
- pknca_exclude_rules, 165
- pknca_exclude_rules(), 52, 54, 56, 57, 175
- pknca_find_units_param, 166
- PKNCA_impute_fun_list, 167
- PKNCA_impute_fun_list(), 18, 73, 74, 169,
170
- PKNCA_impute_method, 18, 73, 74, 89, 167,
167, 170
- PKNCA_impute_method_end_conc_drop
(PKNCA_impute_method), 167
- PKNCA_impute_method_start_cmin
(PKNCA_impute_method), 167
- PKNCA_impute_method_start_cmin(), 121
- PKNCA_impute_method_start_conc0
(PKNCA_impute_method), 167
- PKNCA_impute_method_start_conc0(), 121
- PKNCA_impute_method_start_predose
(PKNCA_impute_method), 167
- PKNCA_impute_method_start_predose(),
121
- PKNCA_impute_method_start_predose_conc0
(PKNCA_impute_method), 167
- pknca_impute_methods, 169
- pknca_impute_methods(), 18, 73, 74, 167,

- [169](#)
- `pknca_interval_table`, [171](#)
- `pknca_interval_table()`, [10](#), [43–47](#), [69](#), [70](#), [90](#), [91](#), [93](#), [160](#), [161](#), [163](#), [173](#), [176–178](#)
- `pknca_match_route`, [173](#)
- `pknca_match_route()`, [10](#), [43](#), [47](#), [69](#), [70](#), [90](#), [91](#), [93](#), [160](#), [161](#), [163](#), [172](#), [176–178](#)
- `pknca_missing_samples`, [174](#)
- `pknca_missing_samples()`, [52](#), [55](#), [56](#), [166](#)
- `PKNCA_options_defaults`, [175](#)
- `PKNCA_options_defaults()`, [156](#), [157](#), [159](#)
- `pknca_parameter_table`, [176](#)
- `pknca_parameter_table()`, [10](#), [43](#), [47](#), [69](#), [70](#), [90–93](#), [160](#), [161](#), [163](#), [172](#), [173](#), [177](#), [178](#)
- `pknca_presets`, [177](#)
- `pknca_presets()`, [10](#), [43](#), [47](#), [69](#), [70](#), [90](#), [91](#), [93](#), [160](#), [161](#), [163](#), [172](#), [173](#), [176](#), [178](#)
- `pknca_ref`, [177](#)
- `pknca_ref()`, [10](#), [43](#), [47](#), [69](#), [70](#), [90](#), [91](#), [93](#), [160](#), [161](#), [163](#), [172](#), [173](#), [176](#), [177](#)
- `pknca_routes` (`pknca_concepts`), [162](#)
- `pknca_routes()`, [8](#), [45](#), [171](#), [173](#)
- `pknca_sample_types` (`pknca_concepts`), [162](#)
- `pknca_sample_types()`, [171](#)
- `pknca_tiers` (`pknca_concepts`), [162](#)
- `pknca_tiers()`, [8](#)
- `pknca_unit_conversion`, [178](#)
- `pknca_unit_reconcile_factor`, [179](#)
- `pknca_units_add_paren`, [179](#)
- `pknca_units_conversion_factor`, [180](#)
- `pknca_units_table`, [180](#)
- `pknca_units_table()`, [179](#), [186](#), [188](#)
- `PKNCAconc`, [182](#)
- `PKNCAconc()`, [14](#), [45](#), [55](#), [147](#), [174](#), [184](#), [186](#), [188–190](#), [201](#)
- `PKNCAdata`, [185](#)
- `PKNCAdata()`, [13](#), [14](#), [18](#), [43](#), [45](#), [74](#), [90](#), [92](#), [145](#), [169](#), [171](#), [172](#), [180](#), [181](#), [184](#), [185](#), [189](#), [190](#), [195](#)
- `PKNCAdose`, [188](#)
- `PKNCAdose()`, [45](#), [55](#), [163](#), [173](#), [185](#), [188](#), [190](#)
- `PKNCAresults`, [190](#)
- `PKNCAresults()`, [185](#), [188](#), [190](#)
- `print.PKNCAconc`, [191](#)
- `print.PKNCAconc()`, [191](#)
- `print.PKNCAdata`, [191](#)
- `print.PKNCAdata()`, [204](#)
- `print.PKNCAdose` (`print.PKNCAconc`), [191](#)
- `print.PKNCAdose()`, [191](#)
- `print.provenance`, [192](#)
- `print.summary_PKNCAresults`, [192](#)
- `print.summary_PKNCAresults()`, [206](#)
- `resolve_dose_tau`, [193](#)
- `resolve_dose_tau()`, [47](#), [62](#), [63](#)
- `right_join.PKNCAconc` (`inner_join.PKNCAresults`), [81](#)
- `right_join.PKNCAdose` (`inner_join.PKNCAresults`), [81](#)
- `right_join.PKNCAresults` (`inner_join.PKNCAresults`), [81](#)
- `round()`, [194](#)
- `roundingSummarize`, [193](#)
- `roundString`, [194](#)
- `roundString()`, [199](#)
- `select_minimal_grouping_cols`, [195](#)
- `set_intervals`, [195](#)
- `set_intervals()`, [188](#)
- `setAttributeColumn`, [196](#)
- `setDuration` (`setDuration.PKNCAconc`), [197](#)
- `setDuration.PKNCAconc`, [197](#)
- `setExcludeColumn`, [197](#)
- `setRoute`, [198](#)
- `signif()`, [199](#)
- `signifString`, [199](#)
- `signifString()`, [194](#)
- `sparse_auc_weight_linear`, [200](#)
- `sparse_auc_weight_linear()`, [15](#), [101](#), [102](#), [104](#), [136–138](#), [202](#), [203](#), [207](#)
- `sparse_bootstrap`, [201](#)
- `sparse_bootstrap()`, [15](#), [26](#), [27](#), [29](#), [31](#), [32](#), [36](#), [101](#), [102](#), [104](#), [137](#), [138](#), [200](#), [203](#), [207](#)
- `sparse_mean`, [202](#)
- `sparse_mean()`, [15](#), [101](#), [102](#), [104](#), [137](#), [138](#), [200](#), [202](#), [207](#)
- `sparse_pk_attribute`, [203](#)
- `sparse_to_dense_pk`, [204](#)
- `stats::approx()`, [71](#)
- `stats::optim()`, [66](#), [131](#)
- `summary()`, [201](#)
- `summary.PKNCAconc` (`print.PKNCAconc`), [191](#)
- `summary.PKNCAdata`, [204](#)
- `summary.PKNCAdose` (`print.PKNCAconc`), [191](#)

`summary.PKNCAresults`, 205
`summary.PKNCAresults()`, 54, 145, 159, 192, 207
`summary.PKNCAresults_sparse_bootstrap`, 207
`summary.PKNCAresults_sparse_bootstrap()`, 15, 101, 102, 104, 137, 138, 200, 202, 203
`superposition`, 207

`tbl()`, 80
`time_calc`, 209
`tss.monoexponential.generate.formula`, 210

`ungroup.PKNCAconc`
 (`group_by.PKNCAresults`), 79
`ungroup.PKNCAdose`
 (`group_by.PKNCAresults`), 79
`ungroup.PKNCAresults`
 (`group_by.PKNCAresults`), 79
`update.PKNCAresults`, 211

`var_sparse_auc`, 211
`vname`, 20–22

`when_any()`, 58
`withCallingHandlers()`, 55