

# Package ‘PKNCA’

May 7, 2026

**Type** Package

**Title** Perform Pharmacokinetic Non-Compartmental Analysis

**Version** 0.12.1

**Imports** checkmate, dplyr ( $\geq 0.5.0$ ), digest, nlme, purrr, rlang,  
stats, tidyr, tibble, utils, lifecycle

**Suggests** covr, cowplot, ggplot2, knitr, labeling, pander, pmxTools,  
rmarkdown, spelling, testthat ( $\geq 3.0.0$ ), units, withr

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

**License** AGPL-3

**URL** <https://humanpred.github.io/pknca/>,  
<https://github.com/humanpred/pknca>,  
<http://humanpred.github.io/pknca/>

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

**NeedsCompilation** no

**VignetteBuilder** knitr

**RoxygenNote** 7.3.2

**Config/testthat/edition** 3

**Encoding** UTF-8

**Language** en-US

**Author** Bill Denney [aut, cre] (ORCID: <https://orcid.org/0000-0002-5759-428X>),  
Clare Buckeridge [aut],  
Gerardo Jose Rodriguez [aut] (ORCID:  
<https://orcid.org/0000-0003-1413-0060>),  
Sridhar Duvvuri [ctb]

**Maintainer** Bill Denney [wdenney@humanpredictions.com](mailto:wdenney@humanpredictions.com)

**Repository** CRAN

**Date/Publication** 2025-08-19 04:40:02 UTC

## Contents

add.interval.col . . . . .	5
addProvenance . . . . .	7
adj.r.squared . . . . .	8
any_sparse_dense_in_interval . . . . .	8
as.data.frame.PKNCAresults . . . . .	9
assert_aucmethod . . . . .	9
assert_conc . . . . .	10
assert_dosetau . . . . .	11
assert_intervals . . . . .	11
assert_intervaltime_single . . . . .	12
assert_lambda . . . . .	12
assert_number_between . . . . .	13
assert_numeric_between . . . . .	14
assert_PKNCAdata . . . . .	15
assert_unit_col . . . . .	15
as_PKNCAconc . . . . .	16
as_sparse_pk . . . . .	17
auc_integrate . . . . .	17
business.mean . . . . .	18
check.conversion . . . . .	19
check.interval.deps . . . . .	20
check.interval.specification . . . . .	20
checkProvenance . . . . .	21
choose.auc.intervals . . . . .	22
choose_interval_method . . . . .	23
clean.conc.blq . . . . .	23
clean.conc.na . . . . .	25
cov_holder . . . . .	26
defunct . . . . .	27
exclude . . . . .	27
exclude_nca . . . . .	28
exclude_nca_by_param . . . . .	30
filter.PKNCAresults . . . . .	31
find.tau . . . . .	32
findOperator . . . . .	33
fit_half_life . . . . .	33
formula.PKNCAconc . . . . .	34
geomean . . . . .	35
get.best.model . . . . .	36
get.first.model . . . . .	36
get.interval.cols . . . . .	37
get.parameter.deps . . . . .	37
getAttributeColumn . . . . .	38
getColumnValueOrNot . . . . .	38
getDataName.PKNCAconc . . . . .	39
getDepVar . . . . .	40

getGroups.PKNCAconc . . . . .	40
getIndepVar . . . . .	41
get_halflife_points . . . . .	42
get_impute_method . . . . .	43
group_by.PKNCAresults . . . . .	43
group_vars.PKNCAconc . . . . .	44
inner_join.PKNCAresults . . . . .	45
interp.extrap.conc . . . . .	49
interp_extrap_conc_method . . . . .	52
is_sparse_pk.PKNCAconc . . . . .	53
model.frame.PKNCAconc . . . . .	53
mutate.PKNCAresults . . . . .	54
normalize_exclude . . . . .	55
parse_formula_to_cols . . . . .	55
pk.business . . . . .	56
pk.calc.ae . . . . .	56
pk.calc.aucabove . . . . .	57
pk.calc.aucint . . . . .	58
pk.calc.auciv . . . . .	60
pk.calc.aucpext . . . . .	62
pk.calc.auxc . . . . .	62
pk.calc.c0 . . . . .	65
pk.calc.cav . . . . .	67
pk.calc.ceoi . . . . .	67
pk.calc.cl . . . . .	68
pk.calc.clast.obs . . . . .	68
pk.calc.clr . . . . .	69
pk.calc.cmax . . . . .	70
pk.calc.count_conc . . . . .	71
pk.calc.cstart . . . . .	71
pk.calc.ctrough . . . . .	72
pk.calc.deg.fluc . . . . .	73
pk.calc.dn . . . . .	73
pk.calc.f . . . . .	74
pk.calc.fe . . . . .	74
pk.calc.half.life . . . . .	75
pk.calc.kel . . . . .	77
pk.calc.mrt . . . . .	77
pk.calc.mrt.md . . . . .	78
pk.calc.ptr . . . . .	79
pk.calc.sparse_auc . . . . .	79
pk.calc.swing . . . . .	80
pk.calc.thalf.eff . . . . .	81
pk.calc.time_above . . . . .	81
pk.calc.tlag . . . . .	82
pk.calc.tlast . . . . .	83
pk.calc.tmax . . . . .	83
pk.calc.totdose . . . . .	84

pk.calc.vss . . . . .	85
pk.calc.vz . . . . .	85
pk.nca . . . . .	86
pk.nca.interval . . . . .	86
pk.nca.intervals . . . . .	88
pk.tss . . . . .	89
pk.tss.data.prep . . . . .	90
pk.tss.monoexponential . . . . .	91
pk.tss.monoexponential.individual . . . . .	92
pk.tss.monoexponential.population . . . . .	92
pk.tss.stepwise.linear . . . . .	93
PKNCA . . . . .	94
PKNCA.choose.option . . . . .	96
PKNCA.options . . . . .	96
PKNCA.options.describe . . . . .	98
PKNCA.set.summary . . . . .	98
PKNCAconc . . . . .	99
PKNCAdata . . . . .	101
PKNCAdose . . . . .	103
PKNCAresults . . . . .	104
pknca_find_units_param . . . . .	105
PKNCA_impute_fun_list . . . . .	106
PKNCA_impute_method . . . . .	106
pknca_units_add_paren . . . . .	107
pknca_units_table . . . . .	108
pknca_unit_conversion . . . . .	109
pk_nca_result_to_df . . . . .	110
print.PKNCAconc . . . . .	110
print.PKNCAdata . . . . .	111
print.provenance . . . . .	112
print.summary_PKNCAresults . . . . .	112
roundingSummarize . . . . .	113
roundString . . . . .	113
setAttributeColumn . . . . .	114
setDuration.PKNCAconc . . . . .	115
setExcludeColumn . . . . .	116
setRoute . . . . .	116
set_intervals . . . . .	117
signifString . . . . .	117
sort.interval.cols . . . . .	118
sparse_auc_weight_linear . . . . .	119
sparse_mean . . . . .	120
sparse_pk_attribute . . . . .	120
sparse_to_dense_pk . . . . .	121
summary.PKNCAdata . . . . .	121
summary.PKNCAresults . . . . .	122
superposition . . . . .	123
time_calc . . . . .	125

tss.monoexponential.generate.formula . . . . .	126
var_sparse_auc . . . . .	127

<b>Index</b>	<b>128</b>
--------------	------------

---

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(),
  datatype = c("interval", "individual", "population")
)
```

## Arguments

name	The column name as a character string
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
unit_type	The type of units to use for assigning and converting units.
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 (<=40 characters to comply with SDTM)
sparse	Is the calculation for sparse PK?
formalsmap	A named list mapping parameter names in the function call to NCA parameter names. See the details for information on use of formalsmap.
datatype	The type of data used for the calculation

## Details

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.

## Value

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

**See Also**

Other Interval specifications: [check.interval.deps\(\)](#), [check.interval.specification\(\)](#), [choose.auc.intervals\(\)](#), [get.interval.cols\(\)](#), [get.parameter.deps\(\)](#)

**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

---

any_sparse_dense_in_interval	<i>Determine if there are any sparse or dense calculations requested within an interval</i>
------------------------------	---

---

**Description**

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

**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.PKNCAresults`

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

---

### Description

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

### Usage

```
## S3 method for class 'PKNCAresults'
as.data.frame(
  x,
  ...,
  out_format = c("long", "wide"),
  filter_requested = FALSE,
  filter_excluded = FALSE,
  out.format = deprecated()
)
```

### Arguments

<code>x</code>	The object to extract results from
<code>...</code>	Ignored (for compatibility with generic <code>as.data.frame()</code> )
<code>out_format</code>	Should the output be 'long' (default) or 'wide'?
<code>filter_requested</code>	Only return rows with parameters that were specifically requested?
<code>filter_excluded</code>	Should excluded values be removed?
<code>out.format</code>	Deprecated in favor of <code>out_format</code>

### Value

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

---

`assert_aucmethod`      *Assert that a value is a valid AUC method*

---

### 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            *Verify that concentration measurements are valid*

---

**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

A data.frame with columns named "conc" and "time" or an informative error

---

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_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())). 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

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

---

### 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

*Assert that a lambda.z value is valid*

---

### 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 <a href="#">vname</a> .

**Value**

lambda.z or an informative error

---

assert\_number\_between *Confirm that a value is greater than another value*

---

**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 <code>assert_numeric_between()</code>
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 <a href="#">vname</a> .

**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 checkmate::assert_numeric()
.var.name	[character(1)] Name of the checked object to print in assertions. Defaults to the heuristic implemented in <a href="#">vname</a> .

### Value

x

---

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)
```

**Arguments**

object	The PKNCAdata object
--------	----------------------

**Value**

The PKNCAdata object (confirmed to be usable)

---

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.

---

`as_PKNCAconc`*Convert an object into a PKNCAconc object*

---

**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

---

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

---

**Description**

Generate a sparse\_pk object

**Usage**

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

**Arguments**

conc	Measured concentrations
time	Time of the measurement of the concentrations
subject	Subject identifiers (may be any class; may not be null)

**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. 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.sparse\\_auc\(\)](#), [sparse\\_auc\\_weight\\_linear\(\)](#), [sparse\\_mean\(\)](#)

---

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).

---

business.mean	<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.deps`     *Take in a single row of an interval specification and return that row updated with any additional calculations that must be done to fulfill all dependencies.*

---

### Description

Take in a single row of an interval specification and return that row updated with any additional calculations that must be done to fulfill all dependencies.

### Usage

```
check.interval.deps(x)
```

### Arguments

x                      A data frame with one or more rows of the PKNCA interval

### Value

The interval specification with additional calculations added where requested outputs require them.

### See Also

Other Interval specifications: [add.interval.col\(\)](#), [check.interval.specification\(\)](#), [choose.auc.intervals\(\)](#), [get.interval.cols\(\)](#), [get.parameter.deps\(\)](#)

---

`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)
```

### Arguments

x                      The data frame specifying what to calculate during each time 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.

**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\(\)](#), [check.interval.deps\(\)](#), [choose.auc.intervals\(\)](#), [get.interval.cols\(\)](#), [get.parameter.deps\(\)](#)

---

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.

**See Also**

[addProvenance\(\)](#)

---

choose.auc.intervals *Choose intervals to compute AUCs from time and dosing information*

---

## Description

Intervals for AUC are selected by the following metrics:

1. If only one dose is administered, use the `PKNCA.options("single.dose.aucs")`
2. If more than one dose is administered, estimate the AUC between any two doses that have PK taken at both of the dosing times and at least one time between the doses.
3. For the final dose of multiple doses, try to determine the dosing interval ( $\tau$ ) and estimate the AUC in that interval if multiple samples are taken in the interval.
4. If there are samples  $> \tau$  after the last dose, calculate the half life after the last dose.

## Usage

```
choose.auc.intervals(
  time.conc,
  time.dosing,
  options = list(),
  single.dose.aucs = 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.

## Value

A data frame with columns for `start`, `end`, `auc.type`, and `half.life`. See [check.interval.specification\(\)](#) for column definitions. The data frame may have zero rows if no intervals could be found.

## See Also

[pk.calc.auc\(\)](#), [pk.calc.aumc\(\)](#), [pk.calc.half.life\(\)](#), [PKNCA.options\(\)](#)

Other Interval specifications: [add.interval.col\(\)](#), [check.interval.deps\(\)](#), [check.interval.specification\(\)](#), [get.interval.cols\(\)](#), [get.parameter.deps\(\)](#)

Other Interval determination: [find.tau\(\)](#)

---

 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

conc	Measured concentrations
time	Time of the measurement of the concentrations
tlast	Time of last concentration above the limit of quantification (will be calculated, if not provided)
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'.
options	List of changes to the default PKNCA options (see PKNCA.options())

### 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 <a href="#">clean.conc.na()</a> )
check	Run <a href="#">assert_conc_time()</a> ?

**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 <a href="#">PKNCA.options()</a> )
conc.na	How to handle NA concentrations? Either 'drop' or a number to impute.
check	Run <a href="#">assert_conc_time()</a> ?

**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

---

 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](#)

**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()

```

### 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 `PKNCA.options("min.hl.r.squared")` 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

### Functions

- `exclude_nca_span.ratio()`: Exclude based on span.ratio
- `exclude_nca_max.aucinf.pext()`: Exclude based on AUC percent extrapolated (both observed and predicted)
- `exclude_nca_count_conc_measured()`: Exclude AUC measurements based on count of concentrations measured and not below the lower limit of quantification
- `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`)

### See Also

Other Result exclusions: [exclude\(\)](#)

**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 NCA Results Based on 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.

**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**

dplyr filtering for PKNCA

**Usage**

```

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

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

## S3 method for class 'PKNCAdose'
filter(.data, ..., .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>	<a href="#">&lt;data-masking&gt;</a> Expressions that return a logical value, and are defined in terms of the variables in <code>.data</code> . If multiple expressions are included, they are combined with the <code>&amp;</code> operator. Only rows for which all conditions evaluate to <code>TRUE</code> are kept.
<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.

**See Also**

Other dplyr verbs: [group\\_by.PKNCAresults\(\)](#), [inner\\_join.PKNCAresults\(\)](#), [mutate.PKNCAresults\(\)](#)

---

find.tau

*Find the repeating interval within a vector of doses*

---

**Description**

This is intended to find the interval over which x repeats by the rule `unique(mod(x, interval))` is minimized.

**Usage**

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

**Arguments**

x	the vector to find the interval within
na.action	What to do with NAs in x
options	List of changes to the default PKNCA options (see <code>PKNCA.options()</code> )
tau.choices	the intervals to look for if the doses are not all equally spaced.

**Value**

A scalar indicating the repeating interval with the most repetition.

1. If all values are NA then NA is returned.
2. If all values are the same, then 0 is returned.
3. If all values are equally spaced, then that spacing is returned.
4. If one of the choices can minimize the number of unique values, then that is returned.
5. If none of the choices can minimize the number of unique values, then -1 is returned.

**See Also**

Other Interval determination: [choose.auc.intervals\(\)](#)

---

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, conc_units)
```

**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.
conc_units	NULL or the units to set for concentration measures

**Value**

A data.frame with one row and columns named "r.squared", "adj.r.squared", "PROB", "lambda.z", "clast.pred", "lambda.z.n.points", "half.life", "span.ratio"

**See Also**

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

---

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`*Compute the geometric mean, sd, and CV*

---

**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**

<code>x</code>	A vector to compute the geometric mean of
<code>na.rm</code>	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)
```

---

<code>get.best.model</code>	<i>Extract the best model from a list of models using the AIC.</i>
-----------------------------	--

---

**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>	Parameters passed to AIC.list

**Value**

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

---

<code>get.first.model</code>	<i>Get the first model from a list of models</i>
------------------------------	--

---

**Description**

Get the first model from a list of models

**Usage**

```
get.first.model(object)
```

**Arguments**

<code>object</code>	the list of (lists of, ...) models
---------------------	------------------------------------

**Value**

The first item in the object that is not a list or NA. If NA is passed in or the list (of lists) is all NA, then NA is returned.

---

get.interval.cols      *Get the columns that can be used in an interval specification*

---

**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.deps\(\)](#), [check.interval.specification\(\)](#), [choose.auc.intervals\(\)](#), [get.parameter.deps\(\)](#)

**Examples**

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

---

get.parameter.deps      *Get all columns that depend on a parameter*

---

**Description**

Get all columns that depend on a parameter

**Usage**

```
get.parameter.deps(x)
```

**Arguments**

x                      The parameter name (as a character string)

**Value**

A character vector of parameter names that depend on the parameter x. If none depend on x, then the result will be an empty vector.

**See Also**

Other Interval specifications: [add.interval.col\(\)](#), [check.interval.deps\(\)](#), [check.interval.specification\(\)](#), [choose.auc.intervals\(\)](#), [get.interval.cols\(\)](#)

---

getAttributeColumn      *Retrieve the value of an attribute column.*

---

**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      *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.*

---

**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(object$data$conc),
  level,
  data = object$result,
  sep
)

```

### Arguments

object	The object to extract the data from
form	The formula to extract the data from (defaults to the formula from object)
level	optional. If included, this specifies the level(s) of the groups to include. If a numeric scalar, include the first level 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.
data	The data to extract the groups from (defaults to the data from object)
sep	Unused (kept for compatibility with the nlme package)
...	Arguments passed to other getGroups 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\(\)](#)

---

get\_halflife\_points     *Determine which concentrations were used for half-life calculation*

---

**Description**

Determine which concentrations were used for half-life calculation

**Usage**

```
get_halflife_points(object)
```

**Arguments**

object            A PKNCAresults 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_halflife_points(o_nca)
```

---

get_impute_method	<i>Get the impute function from either the intervals column or from the method</i>
-------------------	--

---

**Description**

Get the impute function from either the intervals column or from the method

**Usage**

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

**Arguments**

intervals	the data.frame of intervals
impute	the imputation definition

**Value**

The imputation function vector

---

```
group_by.PKNCAresults dplyr grouping for PKNCA
```

---

**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.
...	In <code>group_by()</code> , 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 <code>mutate()</code> step before the <code>group_by()</code> . Computations are not allowed in <code>nest_by()</code> . In <code>ungroup()</code> , variables to remove from the grouping.
.add	When FALSE, the default, <code>group_by()</code> will override existing groups. To add to the existing groups, use <code>.add = TRUE</code> . This argument was previously called <code>add</code> , but that prevented creating a new grouping variable called <code>add</code> , and conflicts with our naming conventions.
.drop	Drop groups formed by factor levels that don't appear in the data? The default is TRUE except when <code>.data</code> has been previously grouped with <code>.drop = FALSE</code> . See <code>group_by_drop_default()</code> for details.
x	A <code>tbl()</code>

**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 <code>dbplyr</code> or <code>dtplyr</code> ). See <i>Methods</i> , below, for more details.
by	<p>A join specification created with <code>join_by()</code>, or a character vector of variables to join by.</p> <p>If <code>NULL</code>, the default, <code>*_join()</code> will perform a natural join, using all variables in common across <code>x</code> and <code>y</code>. A message lists the variables so that you can check they're correct; suppress the message by supplying <code>by</code> explicitly.</p> <p>To join on different variables between <code>x</code> and <code>y</code>, use a <code>join_by()</code> specification. For example, <code>join_by(a == b)</code> will match <code>x\$a</code> to <code>y\$b</code>.</p> <p>To join by multiple variables, use a <code>join_by()</code> 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 <code>x</code> and <code>y</code>, you can shorten this by listing only the variable names, like <code>join_by(a, c)</code>.</p> <p><code>join_by()</code> can also be used to perform inequality, rolling, and overlap joins. See the documentation at <a href="#">?join_by</a> for details on these types of joins.</p> <p>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 <code>x</code> and <code>y</code>, use a named character vector like <code>by = c("x_a" = "y_a", "x_b" = "y_b")</code>.</p> <p>To perform a cross-join, generating all combinations of <code>x</code> and <code>y</code>, see <code>cross_join()</code>.</p>
copy	If <code>x</code> and <code>y</code> are not from the same data source, and <code>copy</code> is <code>TRUE</code> , then <code>y</code> will be copied into the same <code>src</code> as <code>x</code> . This allows you to join tables across <code>srcs</code> , but it is a potentially expensive operation so you must opt into it.
suffix	If there are non-joined duplicate variables in <code>x</code> and <code>y</code> , 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	<p>Should the join keys from both <code>x</code> and <code>y</code> be preserved in the output?</p> <ul style="list-style-type: none"> <li>• If <code>NULL</code>, the default, joins on equality retain only the keys from <code>x</code>, while joins on inequality retain the keys from both inputs.</li> <li>• If <code>TRUE</code>, all keys from both inputs are retained.</li> <li>• If <code>FALSE</code>, only keys from <code>x</code> are retained. For right and full joins, the data in key columns corresponding to rows that only exist in <code>y</code> are merged into the key columns from <code>x</code>. Can't be used when joining on inequality conditions.</li> </ul>

**See Also**

Other dplyr verbs: [filter.PKNCAResults\(\)](#), [group\\_by.PKNCAResults\(\)](#), [mutate.PKNCAResults\(\)](#)

---

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,  
  extrapol.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

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

...	Additional arguments passed to <code>interpolate.conc()</code> or <code>extrapolate.conc()</code> .
<code>conc.blq</code>	How to handle BLQ values. (See <code>clean.conc.blq()</code> for usage instructions.)
<code>conc.na</code>	How to handle NA concentrations. (See <code>clean.conc.na()</code> )
<code>check</code>	Run <code>assert_conc_time()</code> , <code>clean.conc.blq()</code> , and <code>clean.conc.na()</code> ?
<code>conc.origin</code>	The concentration before the first measurement. <code>conc.origin</code> 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.
<code>time.dose</code>	Time of the dose
<code>route.dose</code>	What is the route of administration ("intravascular" or "extravascular"). See the details for how this parameter is used.
<code>duration.dose</code>	What is the duration of administration? See the details for how this parameter is used.
<code>out.after</code>	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

---

`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)

---

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

---

**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.

---

mutate.PKNCAresults    *dplyr mutate-based modification for PKNCA*

---

**Description**

dplyr mutate-based modification for PKNCA

**Usage**

```
## S3 method for class 'PKNCAresults'
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.PKNCAresults\(\)](#), [group\\_by.PKNCAresults\(\)](#), [inner\\_join.PKNCAresults\(\)](#)

---

normalize\_exclude      *Normalize the exclude column by setting blanks to NA*

---

**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	<i>Calculate amount excreted (typically in urine or feces)</i>
------------	--

---

### 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  $\text{sum}(\text{conc} \times \text{volume})$ .

The units for the concentration and volume should match such that  $\text{sum}(\text{conc} \times \text{volume})$  has units of mass or moles.

**Value**

The amount excreted during the interval

**See Also**

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

---

pk.calc.aucabove      *Calculate the AUC above a given concentration*

---

**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 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> )

**Value**

The AUC of the concentration above the limit

---

pk.calc.aucint	<i>Calculate the AUC over an interval with interpolation and/or extrapolation of concentrations for the beginning and end of the interval.</i>
----------------	--

---

**Description**

Calculate the AUC over an interval with interpolation and/or extrapolation of concentrations for the beginning and end of the interval.

**Usage**

```
pk.calc.aucint(  
  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,  
  method = NULL,  
  auc.type = "AUClast",  
  conc.blq = NULL,  
  conc.na = NULL,  
  check = TRUE,  
  ...,  
  options = list()  
)
```

```
pk.calc.aucint.last(  
  conc,  
  time,  
  start = NULL,  
  end = NULL,  
  time.dose,  
  ...,  
  options = list()  
)
```

```
pk.calc.aucint.all(  
  conc,  
  time,  
  start = NULL,  
  end = NULL,  
  time.dose,
```

```

    ...,
    options = list()
)

pk.calc.aucint.inf.obs(
  conc,
  time,
  start = NULL,
  end = NULL,
  time.dose,
  lambda.z,
  clast.obs,
  ...,
  options = list()
)

pk.calc.aucint.inf.pred(
  conc,
  time,
  start = NULL,
  end = NULL,
  time.dose,
  lambda.z,
  clast.pred,
  ...,
  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
start	The start time of the interval
end	The end time of the interval
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
time.dose, route, duration.dose	The time of doses, route of administration, and duration of dose used with interpolation and extrapolation of concentration data (see <a href="#">interp.extrap.conc.dose()</a> ). If NULL, <a href="#">interp.extrap.conc()</a> will be used instead (assuming that no doses affecting concentrations are in the interval).
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'.

conc.blq	How to handle BLQ values in between the first and last above LOQ concentrations. (See <a href="#">clean.conc.blq()</a> for usage instructions.)
conc.na	How to handle missing concentration values. (See <a href="#">clean.conc.na()</a> for usage instructions.)
check	Run <a href="#">assert_conc_time()</a> , <a href="#">clean.conc.blq()</a> , and <a href="#">clean.conc.na()</a> ?
...	Additional arguments passed to <a href="#">pk.calc.auxc</a> and <a href="#">interp.extrap.conc</a>
options	List of changes to the default PKNCA options (see <a href="#">PKNCA.options()</a> )

### Details

When [pk.calc.aucint\(\)](#) needs to extrapolate using `lambda.z` (in other words, using the half-life), it will always extrapolate using the logarithmic trapezoidal rule to align with using a half-life calculation for the extrapolation.

### Value

The AUC for an interval of time as a number

### Functions

- [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

### See Also

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

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

---

pk.calc.auciv

*Calculate AUC for intravenous dosing*

---

### Description

Calculate AUC for intravenous dosing

### Usage

```
pk.calc.auciv(conc, time, c0, auc, ..., options = list(), check = TRUE)
```

```
pk.calc.auciv_pbext(auc, auciv)
```

**Arguments**

conc	Measured concentrations
time	Time of the measurement of the concentrations
c0	The concentration at time 0, typically calculated using <code>pk.calc.c0()</code>
auc	The AUC calculated using conc and time without c0 (it may be calculated using any method)
...	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> )
check	Run <code>assert_conc_time()</code> , <code>clean.conc.blq()</code> , and <code>clean.conc.na()</code> ?
auciv	The AUC calculated using c0

**Details**

The AUC for intravenous (IV) dosing extrapolates the AUC back from the first measurement to time 0 using c0 and the AUC calculated by another method (for example the `auclast`).

The calculation method takes the following steps:

- `time = 0` must be present in the data with a measured concentration.
- The AUC between `time = 0` and the next time point is calculated (`auc_first`).
- The AUC between `time = 0` with `c0` and the next time point is calculated (`auc_second`).
- The final AUC is the initial AUC plus the difference between the two AUCs (`auc_final <- auc + auc_second - auc_first`).

The calculation for back-extrapolation is  $100 \times (1 - \text{auc}/\text{auciv})$ .

**Value**

`pk.calc.auciv`: The AUC calculated using c0

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

**Functions**

- `pk.calc.auciv_pbext()`: Calculate the percent back-extrapolated AUC for IV administration

---

pk.calc.aucpext      *Calculate the AUC percent extrapolated*

---

### 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.

---

pk.calc.auxc      *A 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 BLQ values in between the first and last above LOQ concentrations. (See <code>clean.conc.blq()</code> for usage instructions.)
conc.na	How to handle missing concentration values. (See <code>clean.conc.na()</code> for usage instructions.)
check	Run <code>assert_conc_time()</code> , <code>clean.conc.blq()</code> , and <code>clean.conc.na()</code> ?
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 <code>pk.calc.auxc</code> , these values are passed to <code>pk.calc.auxc</code>

### 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 AU`Clast`.
- `pk.calc.auc.inf()`: Compute the AUC`inf`
- `pk.calc.auc.inf.obs()`: Compute the AUC`inf` with the observed `Clast`.
- `pk.calc.auc.inf.pred()`: Compute the AUC`inf` with the predicted `Clast`.
- `pk.calc.auc.all()`: Compute the AU`Call`.
- `pk.calc.aumc()`: Compute the area under the moment curve
- `pk.calc.aumc.last()`: Compute the AUM`Clast`.
- `pk.calc.aumc.inf()`: Compute the AUM`Cinf`
- `pk.calc.aumc.inf.obs()`: Compute the AUM`Cinf` with the observed `Clast`.
- `pk.calc.aumc.inf.pred()`: Compute the AUM`Cinf` with the predicted `Clast`.
- `pk.calc.aumc.all()`: Compute the AUM`Call`.

## 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.aucint\(\)](#)

## 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.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	Check the conc and time inputs

**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)

---

pk.calc.cav                    *Calculate the average concentration during an interval.*

---

**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  $auc / (end - start)$ .

**Value**

The Cav (average concentration during the interval)

---

pk.calc.ceoi                    *Determine the concentration at the end of infusion*

---

**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 <code>assert_conc_time()</code> ?

**Value**

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

---

`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.

---

`pk.calc.clast.obs`*Determine the last observed concentration above the limit of quantification (LOQ).*

---

**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.clr(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.cmax\(\)](#), [pk.calc.count\\_conc\(\)](#), [pk.calc.cstart\(\)](#), [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  $\text{sum}(ae)/auc$ .

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\(\)](#)

---

pk.calc.cmax	<i>Determine maximum observed PK concentration</i>
--------------	--

---

### 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 <a href="#">assert_conc()</a> ?

### 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.clast.obs\(\)](#), [pk.calc.count\\_conc\(\)](#), [pk.calc.cstart\(\)](#), [pk.calc.ctrough\(\)](#)

Other NCA parameters for concentrations during the intervals: [pk.calc.clast.obs\(\)](#), [pk.calc.count\\_conc\(\)](#), [pk.calc.cstart\(\)](#), [pk.calc.ctrough\(\)](#)

### Examples

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

---

pk.calc.count_conc	<i>Count the number of concentration measurements in an interval</i>
--------------------	--

---

**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 <a href="#">assert_conc()</a> ?

**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)

**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

**See Also**

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

---

pk.calc.cstart	<i>Determine the concentration at the beginning of the interval</i>
----------------	---

---

**Description**

Determine the concentration at the beginning of the interval

**Usage**

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

**Arguments**

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

**Value**

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

**See Also**

Other NCA parameters for concentrations during the intervals: [pk.calc.clast.obs\(\)](#), [pk.calc.cmax\(\)](#), [pk.calc.count\\_conc\(\)](#), [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)
```

**Arguments**

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

**Value**

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

**See Also**

Other NCA parameters for concentrations during the intervals: [pk.calc.clast.obs\(\)](#), [pk.calc.cmax\(\)](#), [pk.calc.count\\_conc\(\)](#), [pk.calc.cstart\(\)](#)

---

pk.calc.deg.fluc      *Determine the degree of fluctuation*

---

**Description**

Determine the degree of fluctuation

**Usage**

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

**Arguments**

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

**Details**

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

**Value**

The degree of fluctuation around the average concentration.

---

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.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\(\)](#)

---

pk.calc.half.life      *Compute the half-life and associated parameters*

---

**Description**

The terminal elimination half-life is estimated from the final points in the concentration-time curve using semi-log regression ( $\log(\text{conc}) \sim \text{time}$ ) 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,  
  manually.selected.points = FALSE,  
  options = list(),  
  min.hl.points = NULL,  
  adj.r.squared.factor = NULL,  
  conc.blq = NULL,  
  conc.na = NULL,  
  first.tmax = NULL,  
  allow.tmax.in.half.life = NULL,  
  check = TRUE  
)
```

**Arguments**

conc	Measured concentrations
time	Time of the measurement of the concentrations
tmax	Time of maximum concentration (will be calculated and included in the return data frame if not given)
tlast	Time of last concentration above the limit of quantification (will be calculated and included in the return data frame if not given)
time.dose	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>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>min.hl.points</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
<code>adj.r.squared.factor</code>	The allowance in adjusted r-squared for adding another point.
<code>conc.blq</code>	See <code>clean.conc.blq()</code>
<code>conc.na</code>	See <code>clean.conc.na()</code>
<code>first.tmax</code>	See <code>pk.calc.tmax()</code> .
<code>allow.tmax.in.half.life</code>	Allow the concentration point for <code>tmax</code> to be included in the half-life slope calculation.
<code>check</code>	Run <code>assert_conc_time()</code> , <code>clean.conc.blq()</code> , and <code>clean.conc.na()</code> ?

### Details

See the "Half-Life Calculation" vignette for more details on the calculation methods used.

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`). The best half-life is chosen by the following rules in order:

- At least `min.hl.points` points included
- A  $\lambda.z > 0$  and at the same time the best adjusted r-squared (within `adj.r.squared.factor`)
- The one with the most points included

If `manually.selected.points` is `TRUE`, the `conc` and `time` data are used as-is without any form of selection for the best-fit half-life.

### Value

A data frame with one row and columns for

**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)

**r.squared** coefficient of determination

**adj.r.squared** adjusted coefficient of determination

**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  
**clast.pred** Concentration at tlast as predicted by the half-life line  
**half.life** half-life  
**span.ratio** span ratio [ratio of half-life to time used for half-life calculation]

## 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.

---

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

---

pk.calc.mrt

*Calculate the mean residence time (MRT) for single-dose data or linear multiple-dose data.*

---

## 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)

**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)

**Details**

mrt is  $\text{aumc}/\text{auc} - \text{duration.dose}/2$  where  $\text{duration.dose} = 0$  for oral administration.

**Value**

the numeric value of the mean residence time

**Functions**

- `pk.calc.mrt.iv()`: MRT for an IV infusion

**See Also**

[pk.calc.mrt.md\(\)](#)

---

<code>pk.calc.mrt.md</code>	<i>Calculate the mean residence time (MRT) for multiple-dose data with nonlinear kinetics.</i>
-----------------------------	--

---

**Description**

Calculate the mean residence time (MRT) for multiple-dose data with nonlinear kinetics.

**Usage**

```
pk.calc.mrt.md(auctau, aumctau, aucinf, tau)
```

**Arguments**

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.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.

**See Also**[pk.calc.mrt\(\)](#)

---

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)

---

pk.calc.sparse_auc	<i>Calculate AUC and related parameters using sparse NCA methods</i>
--------------------	--

---

**Description**

The AUC is calculated as:

**Usage**

```
pk.calc.sparse_auc(  
  conc,  
  time,  
  subject,  
  method = NULL,  
  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)
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\(\)](#), [sparse\\_auc\\_weight\\_linear\(\)](#), [sparse\\_mean\(\)](#)

---

pk.calc.swing

*Determine the PK swing*

---

**Description**

Determine the PK swing

**Usage**

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

**Arguments**

cmax	The maximum observed concentration
cmin	The minimum observed concentration

**Details**

swing is  $100 * (c_{max} - c_{min}) / c_{min}$ .

**Value**

The swing above the minimum concentration. If  $c_{min}$  is zero, then the result is infinity.

---

pk.calc.thalf.eff      *Calculate the effective half-life*

---

**Description**

Calculate the effective half-life

**Usage**

```
pk.calc.thalf.eff(mrt)
```

**Arguments**

mrt                      the mean residence time to infinity

**Details**

thalf.eff is  $\log(2) * mrt$ .

**Value**

the numeric value of the effective half-life

---

pk.calc.time\_above      *Determine time at or above a set value*

---

**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 PKNCA.options())
check	Run <code>assert_conc_time()</code> , <code>clean.conc.blq()</code> , and <code>clean.conc.na()</code> ?

**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

---

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 <code>assert_conc_time()</code> ?

**Value**

The time of the last observed concentration measurement

**Functions**

- `pk.calc.tfirst()`: Determine the first concentration above the limit of quantification.

---

pk.calc.tmax	<i>Determine time of maximum observed PK concentration</i>
--------------	--

---

**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**

conc	Measured concentrations
time	Time of the measurement of the concentrations
options	List of changes to the default PKNCA options (see PKNCA.options())
first.tmax	If there is more than time that matches the maximum concentration, should the first be considered as Tmax? If not, then the last is considered Tmax.
check	Run <code>assert_conc_time()</code> ?

**Value**

The time of the maximum concentration

**Examples**

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

---

pk.calc.totdose      *Extract the dose used for calculations*

---

**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.vss                      *Calculate the steady-state volume of distribution (Vss)*

---

**Description**

Calculate the steady-state volume of distribution (Vss)

**Usage**

pk.calc.vss(cl, mrt)

**Arguments**

cl                      the clearance  
mrt                     the mean residence time

**Details**

vss is  $cl \cdot mrt$ .

**Value**

the volume of distribution at steady-state

---

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)

**Arguments**

cl                      the clearance (or apparent observed clearance)  
lambda.z                The elimination rate (in units of inverse time) for extrapolation

**Details**

vz is  $cl / \lambda.z$ .

---

pk.nca                      *Compute NCA parameters for each interval for each subject.*

---

### 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`.

### 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,
  impute_method = NA_character_,
  include_half.life = NULL,
  exclude_half.life = NULL,
  subject,
  sparse,
  interval,
  options = list()
)
```

**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)
impute_method	The method to use for imputation as a character string
include_half.life	An optional boolean vector of the concentration measurements to include in the half-life calculation. If given, no half-life point selection will occur.
exclude_half.life	An optional boolean vector of the concentration measurements to exclude from the half-life calculation.
subject	Subject identifiers (used for sparse calculations)
sparse	Should only sparse calculations be performed (TRUE) or only dense calculations (FALSE)?
interval	One row of an interval definition (see <a href="#">check.interval.specification()</a> for how to define the interval.
options	List of changes to the default PKNCA options (see <code>PKNCA.options()</code> )

**Value**

A data frame with the start and end time along with all PK parameters for the interval

**See Also**

[check.interval.specification\(\)](#)

---

pk.nca.intervals      *Compute NCA for multiple intervals*

---

**Description**

Compute NCA for multiple intervals

**Usage**

```
pk.nca.intervals(
  data_conc,
  data_dose,
  data_intervals,
  sparse,
  options,
  impute,
  verbose = FALSE
)
```

**Arguments**

data_conc	A data.frame or tibble with standardized column names as output from <code>prepare_PKNCAconc()</code>
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>
sparse	Should only sparse calculations be performed (TRUE) or only dense calculations (FALSE)?
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
verbose	Indicate, by <code>message()</code> , the current state of calculation.

**Value**

A data.frame with all NCA results

---

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 <code>pk.tss.monoexponential()</code> or <code>pk.tss.stepwise.linear()</code> .
type	The type of Tss to calculate, either <code>stepwise.linear</code> or <code>monoexponential</code>
check	See <code>pk.tss.data.prep()</code>

**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	See <a href="#">clean.conc.blq()</a>
conc.na	See <a href="#">clean.conc.na()</a>
check	Run <a href="#">assert_conc_time()</a> ?
...	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

...	See <a href="#">pk.tss.data.prep()</a>
tss.fraction	The fraction of steady-state required for calling steady-state
output	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.
check	See <a href="#">pk.tss.data.prep()</a> .
verbose	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

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.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 <code>tss.constant</code> .
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 <a href="#">pk.tss.data.prep()</a>
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 <a href="#">pk.tss.data.prep()</a>

### 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\(\)](#)

---

PKNCA

*Compute noncompartmental pharmacokinetics*

---

### 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:

- Clare Buckeridge <clare.buckeridge@pfizer.com>
- Gerardo Jose Rodriguez <gerardo.jrac@gmail.com> ([ORCID](#))

Other contributors:

- Sridhar Duvvuri [contributor]

**See Also**

Useful links:

- <https://humanpred.github.io/pknca/>
- <https://github.com/humanpred/pknca>
- <http://humanpred.github.io/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`     *Choose either the value from an option list or the current set value for an option.*

---

### 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

<code>name</code>	The option name requested.
<code>value</code>	A value to check for the option (NULL to choose not to check the value).
<code>options</code>	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`     *Set default options for PKNCA functions*

---

### 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, ).

**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-PKNCA", 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\(\)](#)

**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  
)
```

**Arguments**

name	The parameter name or a vector of parameter names. It must have already been defined (see <a href="#">add.interval.col()</a> ).
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

**Value**

All current summary settings (invisibly)

**See Also**

[summary.PKNCAresults\(\)](#)

Other PKNCA calculation and summary settings: [PKNCA.choose.option\(\)](#), [PKNCA.options\(\)](#)

**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)
```

---

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,
  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 is typically assumed to be the subject; see the documentation for the subject argument for exceptions to this assumption.
subject	The column indicating the subject number. If not provided, this defaults to the beginning of the inner groups: For example with concentration~time Study+Subject/Analyte, the inner groups start with the first grouping variable before a /, Subject. If there is only one grouping variable, it is assumed to be the subject (e.g. concentration~time Subject), and if there are multiple grouping variables without a /, subject is assumed to be the last one. For single-subject data, it is assigned as NULL.

<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 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.
<code>duration</code>	(optional) The duration of collection as is typically used for concentration measurements in urine or feces.
<code>volume</code>	(optional) The volume (or mass) of collection as is typically used for urine or feces measurements.
<code>exclude_half.life, include_half.life</code>	A character scalar for the column name in the dataset of the points to exclude from the half-life calculation (still using normal curve-stripping selection rules for the other points) or to include for the half-life (using specifically those points and bypassing automatic curve-stripping point selection). See the "Half-Life Calculation" vignette for more details on the use of these arguments.
<code>sparse</code>	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)?
<code>concu, amountu, timeu</code>	Either unit values (e.g. "ng/mL") or column names within the data where units are provided.
<code>concu_pref, amountu_pref, timeu_pref</code>	Preferred units for reporting (not column names)

**Value**

A PKNCAconc object that can be used for automated NCA.

**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()
)

```

**Arguments**

<code>data.conc</code>	Concentration data as a PKNCAconc object or a data frame
<code>data.dose</code>	Dosing data as a PKNCAdose object (see details)
<code>...</code>	arguments passed to <code>PKNCAdata.default</code>
<code>formula.conc</code>	Formula for making a PKNCAconc 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 PKNCAconc object.
<code>formula.dose</code>	Formula for making a PKNCAdose 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 PKNCAdose object.
<code>impute</code>	Methods for imputation. NA 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> . 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)
<code>units</code>	A data.frame 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> )

**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.

**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).

**See Also**

[choose.auc.intervals\(\)](#), [pk.nca\(\)](#), [pknca\\_units\\_table\(\)](#)

Other PKNCA objects: [PKNCAconc\(\)](#), [PKNCAdose\(\)](#), [PKNCAresults\(\)](#)

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**

data	A data frame with time and the groups defined in formula.
...	Ignored.
formula	The formula defining the dose.amount~time groups where time is the time of the dosing and dose.amount is the amount administered at that time (see Details).

route	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".
rate, duration	(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 (duration=0). If rate is given, then the dose amount must be given (the left hand side of the formula).
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.
doseu	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

### Value

A PKNCAconc object that can be used for automated NCA.

### See Also

Other PKNCA objects: [PKNCAconc\(\)](#), [PKNCAdata\(\)](#), [PKNCAresults\(\)](#)

---

PKNCAresults

*Generate a PKNCAresults object*

---

### 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\(\)](#)

---

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 <code>add.interval.col</code>
-----------	---

**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**

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 functions (without the `PKNCA_impute_method_` part)

**Details**

This function is not for use by users of PKNCA.

**Value**

A list of character vectors of functions to run.

---

`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()
  )

```

### Arguments

conc	Measured concentrations
time	Time of the measurement of the concentrations
start	The start time of the interval
...	ignored
options	List of changes to the default PKNCA options (see PKNCA.options())
end	The end time of the interval
conc.group	All concentrations measured for the group
time.group	Time of all concentrations measured for the group
max_shift	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()`: Add a new concentration of 0 at the start time, even if a nonzero concentration exists at that time (usually used with single-dose data)
- `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)

---

`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\_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,
  doseu,
  amountu,
  timeu,
  concu_pref = NULL,
  doseu_pref = NULL,
  amountu_pref = NULL,
  timeu_pref = NULL,
  conversions = data.frame()
)
```

**Arguments**

concu, doseu, amountu, timeu        Units for concentration, dose, amount, and time in the source data

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.

**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"
)
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"
)
```

---

*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

---

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.

---

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 <a href="#">head()</a> )
summarize	Summarize the nested number of groups
...	Arguments passed to <code>print.formula</code> and <code>print.data.frame</code>
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 <a href="#">print.PKNCAconc()</a> and <a href="#">print.PKNCAdose()</a>

---

```
print.provenance      Print the summary of a provenance object
```

---

**Description**

Print the summary of a provenance object

**Usage**

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

**Arguments**

x	The object to be printed
...	Ignored

**Value**

invisible text of the printed information

---

```
print.summary_PKNCAResults
      Print the results summary
```

---

**Description**

Print the results summary

**Usage**

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

**Arguments**

x	A <code>summary_PKNCAResults</code> object
...	passed to <code>print.data.frame</code> ( <code>row.names</code> is always set to <code>FALSE</code> )

**Value**

x invisibly

**See Also**

[summary\\_PKNCAResults\(\)](#)

---

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\(\)](#)

---

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	<i>Set the exclude parameter on an object</i>
------------------	---

---

### 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

---

set_intervals	<i>Set Intervals</i>
---------------	----------------------

---

**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

**Value**

The data object with the intervals set.

---

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 $\text{abs}(\log_{10}(x)) > \text{sci\_range}$ .
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\(\)](#)

---

sort.interval.cols     *Sort the interval columns by dependencies.*

---

**Description**

Columns are always to the right of columns that they depend on.

**Usage**

```
## S3 method for class 'interval.cols'
sort()
```

---

`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.sparse_auc()`, `sparse_mean()`

---

sparse_mean	<i>Calculate the mean concentration at all time points for use in sparse NCA calculations</i>
-------------	---

---

### 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 <a href="#">as_sparse_pk()</a>
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  $\geq 50\%$  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.sparse\\_auc\(\)](#), [sparse\\_auc\\_weight\\_linear\(\)](#)

---

sparse_pk_attribute	<i>Set or get a sparse_pk object attribute</i>
---------------------	--

---

### 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      *Extract the mean concentration-time profile as a data.frame*

---

**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      *Summarize a PKNCAdata object showing important details about the concentration, dosing, and interval information.*

---

**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 *Summarize PKNCA results*

---

### Description

Summarize PKNCA results

### Usage

```
## S3 method for class 'PKNCAresults'
summary(
  object,
  ...,
  drop_group = object$data$conc$columns$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
)
```

### 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.
drop.group, not.calculated.string	summarize.n.per.group, not.requested.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.

**Details**

Excluded results will not be included in the summary.

**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")
```

---

superposition

*Compute noncompartmental superposition for repeated dosing*

---

**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**

conc	Measured concentrations
...	Additional arguments passed to the <code>half.life</code> function if required to compute <code>lambda.z</code> .
time	Time of the measurement of the concentrations
dose.input	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.
tau	The dosing interval
dose.times	The time of dosing within the dosing interval. The <code>min(dose.times)</code> must be $\geq 0$ , and the <code>max(dose.times)</code> must be $< \tau$ . There may be more than one dose times given as a vector.
dose.amount	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> .
n.tau	The number of <code>tau</code> dosing intervals to simulate or <code>Inf</code> for steady-state.
options	List of changes to the default PKNCA options (see <code>PKNCA.options()</code> )
lambda.z	The elimination rate (in units of inverse time) for extrapolation

clast.pred	To use predicted as opposed to observed Clast, either give the value for clast.pred here or set it to true (for automatic calculation from the half-life).
tlast	The time of last observed concentration above the limit of quantification. This is calculated if not provided.
additional.times	Times to include in the final outputs in addition to the standard times (see details). All <code>min(additional.times)</code> must be $\geq 0$ , and the <code>max(additional.times)</code> must be $\leq \tau$ .
check.blq	Must the first concentration measurement be below the limit of quantification?
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'.
steady.state.tol	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`.

### Value

A data frame with columns named "conc" and "time".

### See Also

[interp.extrap.conc\(\)](#)

---

time_calc	<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 <code>base::as.numeric.difftime()</code>

**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

---

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 calculated as described in equation 6 of the same paper.

**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

# Index

- \* **AUC calculations**
  - pk.calc.aucint, 58
  - pk.calc.auxc, 62
- \* **Data cleaners**
  - clean.conc.blq, 23
  - clean.conc.na, 25
- \* **Formula parsing**
  - findOperator, 33
  - parse\_formula\_to\_cols, 55
- \* **Internal**
  - any\_sparse\_dense\_in\_interval, 8
  - assert\_intervaltime\_single, 12
  - auc\_integrate, 17
  - choose\_interval\_method, 23
  - cov\_holder, 26
  - getDataName.PKNCAconc, 39
  - interp\_extrap\_conc\_method, 52
  - parse\_formula\_to\_cols, 55
  - pk\_nca\_result\_to\_df, 110
  - pknca\_find\_units\_param, 105
  - PKNCA\_impute\_fun\_list, 106
  - pknca\_unit\_conversion, 109
  - pknca\_units\_add\_paren, 107
  - sparse\_pk\_attribute, 120
  - sparse\_to\_dense\_pk, 121
- \* **Interval determination**
  - choose.auc.intervals, 22
  - find.tau, 32
- \* **Interval specifications**
  - add.interval.col, 5
  - check.interval.deps, 20
  - check.interval.specification, 20
  - choose.auc.intervals, 22
  - get.interval.cols, 37
  - get.parameter.deps, 37
- \* **NCA parameter calculations**
  - pk.calc.half.life, 75
- \* **NCA parameters for concentrations during the intervals**
  - pk.calc.clast.obs, 68
  - pk.calc.cmax, 70
  - pk.calc.count\_conc, 71
  - pk.calc.cstart, 71
  - pk.calc.ctrough, 72
- \* **PKNCA calculation and summary settings**
  - PKNCA.choose.option, 96
  - PKNCA.options, 96
  - PKNCA.set.summary, 98
- \* **PKNCA object extractors**
  - getDataName.PKNCAconc, 39
  - getDepVar, 40
  - getIndepVar, 41
- \* **PKNCA objects**
  - PKNCAconc, 99
  - PKNCAdata, 101
  - PKNCAdose, 103
  - PKNCAresults, 104
- \* **Result exclusions**
  - exclude, 27
  - exclude\_nca, 28
- \* **Sparse Methods**
  - as\_sparse\_pk, 17
  - pk.calc.sparse\_auc, 79
  - sparse\_auc\_weight\_linear, 119
  - sparse\_mean, 120
- \* **Time to steady-state calculations**
  - pk.tss, 89
  - pk.tss.monoexponential, 91
  - pk.tss.stepwise.linear, 93
- \* **dplyr verbs**
  - filter.PKNCAresults, 31
  - group\_by.PKNCAresults, 43
  - inner\_join.PKNCAresults, 45
  - mutate.PKNCAresults, 54
  - ?join\_by, 48
- add.interval.col, 5, 20–22, 37, 38
- add.interval.col(), 99
- addProvenance, 7

- addProvenance(), 21
- adj.r.squared, 8
- any\_sparse\_dense\_in\_interval, 8
- as.data.frame(), 9
- as.data.frame.PKNCAResults, 9
- as.data.frame.PKNCAResults(), 86
- as\_PKNCAConc, 16
- as\_PKNCAdata (as\_PKNCAConc), 16
- as\_PKNCAdose (as\_PKNCAConc), 16
- as\_PKNCAResults (as\_PKNCAConc), 16
- as\_sparse\_pk, 17, 80, 119, 120
- as\_sparse\_pk(), 26, 119–121, 127
- assert\_aucmethod, 9
- assert\_conc, 10
- assert\_conc(), 70, 71
- assert\_conc\_time (assert\_conc), 10
- assert\_conc\_time(), 24, 25, 51, 60, 61, 64, 67, 69, 76, 82–84, 90
- assert\_dosetau, 11
- assert\_intervals, 11
- assert\_intervaltime\_single, 12
- assert\_lambda\_z, 12
- assert\_number\_between, 13
- assert\_numeric\_between, 14
- assert\_PKNCAdata, 15
- assert\_time (assert\_conc), 10
- assert\_unit (assert\_unit\_col), 15
- assert\_unit\_col, 15
- assert\_unit\_value (assert\_unit\_col), 15
- auc\_integrate, 17
  
- business.cv (business.mean), 18
- business.geocv (business.mean), 18
- business.geomean (business.mean), 18
- business.max (business.mean), 18
- business.mean, 18
- business.median (business.mean), 18
- business.min (business.mean), 18
- business.range (business.mean), 18
- business.sd (business.mean), 18
  
- check.conc.time (defunct), 27
- check.conversion, 19
- check.interval.deps, 7, 20, 21, 22, 37, 38
- check.interval.specification, 7, 20, 20, 22, 37, 38
- check.interval.specification(), 22, 37, 88, 102
- checkProvenance, 21
  
- checkProvenance(), 7
- choose.auc.intervals, 7, 20, 21, 22, 32, 37, 38
- choose.auc.intervals(), 102, 103
- choose\_interval\_method, 23
- clean.conc.blq, 23, 25
- clean.conc.blq(), 51, 60, 61, 64, 65, 76, 82, 90
- clean.conc.na, 25, 25
- clean.conc.na(), 24, 51, 60, 61, 64, 76, 82, 90
- cov\_holder, 26
- cross\_join(), 48
  
- defunct, 27
  
- exclude, 27, 29
- exclude(), 86, 116
- exclude\_nca, 28, 28
- exclude\_nca\_by\_param, 30
- exclude\_nca\_count\_conc\_measured (exclude\_nca), 28
- exclude\_nca\_max.aucinf.pext (exclude\_nca), 28
- exclude\_nca\_min.hl.adj.r.squared (exclude\_nca), 28
- exclude\_nca\_min.hl.r.squared (exclude\_nca), 28
- exclude\_nca\_span.ratio (exclude\_nca), 28
- exclude\_nca\_tmax\_0 (exclude\_nca), 28
- exclude\_nca\_tmax\_early (exclude\_nca), 28
- extrapolate.conc (interp.extrap.conc), 49
- extrapolate\_conc\_lambda\_z (interp\_extrap\_conc\_method), 52
  
- filter.PKNCAConc (filter.PKNCAResults), 31
- filter.PKNCAdose (filter.PKNCAResults), 31
- filter.PKNCAResults, 31, 44, 49, 54
- find.tau, 22, 32
- findOperator, 33, 55
- fit\_half\_life, 33
- formula.PKNCAConc, 34
- formula.PKNCAdose (formula.PKNCAConc), 34
- full\_join.PKNCAConc (inner\_join.PKNCAResults), 45

- full\_join.PKNCAdose  
(inner\_join.PKNCAResults), 45
- full\_join.PKNCAResults  
(inner\_join.PKNCAResults), 45
- geocv (geomean), 35
- geomean, 35
- geosd (geomean), 35
- get.best.model, 36
- get.first.model, 36
- get.interval.cols, 7, 20–22, 37, 38
- get.parameter.deps, 7, 20–22, 37, 37
- get\_half-life\_points, 42
- get\_impute\_method, 43
- getAttributeColumn, 38
- getAttributeColumn(), 115
- getColumnValueOrNot, 38
- getDataName (getDataName.PKNCAConc), 39
- getDataName.PKNCAConc, 39, 40, 42
- getDepVar, 39, 40, 42
- getGroups.PKNCAConc, 40
- getGroups.PKNCAdata  
(getGroups.PKNCAConc), 40
- getGroups.PKNCAdose  
(getGroups.PKNCAConc), 40
- getGroups.PKNCAResults  
(getGroups.PKNCAConc), 40
- getIndepVar, 39, 40, 41
- group\_by.PKNCAConc  
(group\_by.PKNCAResults), 43
- group\_by.PKNCAdose  
(group\_by.PKNCAResults), 43
- group\_by.PKNCAResults, 32, 43, 49, 54
- group\_by\_drop\_default(), 44
- group\_vars.PKNCAConc, 44
- group\_vars.PKNCAdata  
(group\_vars.PKNCAConc), 44
- group\_vars.PKNCAdose  
(group\_vars.PKNCAConc), 44
- group\_vars.PKNCAResults  
(group\_vars.PKNCAConc), 44
- head(), 111
- inner\_join.PKNCAConc  
(inner\_join.PKNCAResults), 45
- inner\_join.PKNCAdose  
(inner\_join.PKNCAResults), 45
- inner\_join.PKNCAResults, 32, 44, 45, 54
- interp.extrap.conc, 49
- interp.extrap.conc(), 59, 125
- interp.extrap.conc.dose(), 59, 60
- interp\_extrap\_conc\_method, 52
- interpolate.conc (interp.extrap.conc),  
49
- interpolate\_conc\_linear  
(interp\_extrap\_conc\_method), 52
- interpolate\_conc\_log  
(interp\_extrap\_conc\_method), 52
- is\_sparse\_pk (is\_sparse\_pk.PKNCAConc),  
53
- is\_sparse\_pk.PKNCAConc, 53
- join\_by(), 48
- left\_join.PKNCAConc  
(inner\_join.PKNCAResults), 45
- left\_join.PKNCAdose  
(inner\_join.PKNCAResults), 45
- left\_join.PKNCAResults  
(inner\_join.PKNCAResults), 45
- model.frame.PKNCAConc, 53
- model.frame.PKNCAdose  
(model.frame.PKNCAConc), 53
- mutate.PKNCAConc (mutate.PKNCAResults),  
54
- mutate.PKNCAdose (mutate.PKNCAResults),  
54
- mutate.PKNCAResults, 32, 44, 49, 54
- normalize\_exclude, 55
- options(), 96
- parse\_formula\_to\_cols, 33, 55
- pk.business, 56
- pk.business(), 19
- pk.calc.ae, 56
- pk.calc.ae(), 69, 75
- pk.calc.auc (pk.calc.auxc), 62
- pk.calc.auc(), 22
- pk.calc.aucabove, 57
- pk.calc.aucint, 58, 65
- pk.calc.auciv, 60
- pk.calc.auciv\_pbext (pk.calc.auciv), 60
- pk.calc.aucpext, 62
- pk.calc.aumc (pk.calc.auxc), 62
- pk.calc.aumc(), 22

- pk.calc.auxc, 60, 62
- pk.calc.c0, 65
- pk.calc.c0(), 51, 52
- pk.calc.cav, 67
- pk.calc.ceoi, 67
- pk.calc.cl, 68
- pk.calc.clast.obs, 68, 70–72
- pk.calc.clast.obs(), 50, 52
- pk.calc.clr, 69
- pk.calc.clr(), 57, 75
- pk.calc.cmax, 69, 70, 71, 72
- pk.calc.cmin(pk.calc.cmax), 70
- pk.calc.count\_conc, 69, 70, 71, 72
- pk.calc.count\_conc\_measured  
(pk.calc.count\_conc), 71
- pk.calc.cstart, 69–71, 71, 72
- pk.calc.ctrough, 69–72, 72
- pk.calc.deg.fluc, 73
- pk.calc.dn, 73
- pk.calc.f, 74
- pk.calc.fe, 74
- pk.calc.fe(), 57, 69
- pk.calc.half.life, 75
- pk.calc.half.life(), 22, 34, 52
- pk.calc.kel, 77
- pk.calc.mrt, 77
- pk.calc.mrt(), 79
- pk.calc.mrt.md, 78
- pk.calc.mrt.md(), 78
- pk.calc.ptr, 79
- pk.calc.sparse\_auc, 17, 79, 119, 120
- pk.calc.sparse\_auc(), 119
- pk.calc.sparse\_auclast  
(pk.calc.sparse\_auc), 79
- pk.calc.swing, 80
- pk.calc.tfirfirst(pk.calc.tlast), 83
- pk.calc.thalf.eff, 81
- pk.calc.time\_above, 81
- pk.calc.tlag, 82
- pk.calc.tlast, 83
- pk.calc.tmax, 83
- pk.calc.tmax(), 76
- pk.calc.todose, 84
- pk.calc.vss, 85
- pk.calc.vz, 85
- pk.nca, 86
- pk.nca(), 103
- pk.nca.interval, 86
- pk.nca.intervals, 88
- pk.tss, 89, 91, 94
- pk.tss.data.prep, 90
- pk.tss.data.prep(), 89, 91–94
- pk.tss.monoexponential, 89, 91, 94
- pk.tss.monoexponential(), 89
- pk.tss.monoexponential.individual, 92
- pk.tss.monoexponential.population, 92
- pk.tss.stepwise.linear, 89, 91, 93
- pk.tss.stepwise.linear(), 89
- pk\_nca\_result\_to\_df, 110
- PKNCA, 94
- PKNCA-package (PKNCA), 94
- PKNCA.choose.option, 96, 97, 99
- PKNCA.options, 96, 96, 99
- PKNCA.options(), 22, 60, 86, 98
- PKNCA.options.describe, 98
- PKNCA.options.describe(), 97
- PKNCA.set.summary, 96, 97, 98
- PKNCA.set.summary(), 113, 123
- pk\_nca\_find\_units\_param, 105
- PKNCA\_impute\_fun\_list, 106
- PKNCA\_impute\_method, 106
- PKNCA\_impute\_method\_start\_cmin  
(PKNCA\_impute\_method), 106
- PKNCA\_impute\_method\_start\_conc0  
(PKNCA\_impute\_method), 106
- PKNCA\_impute\_method\_start\_predose  
(PKNCA\_impute\_method), 106
- pk\_nca\_unit\_conversion, 109
- pk\_nca\_units\_add\_paren, 107
- pk\_nca\_units\_table, 108
- pk\_nca\_units\_table(), 102, 103
- PKNCAconc, 99, 103–105
- PKNCAdata, 101, 101, 104, 105
- PKNCAdata(), 86, 108, 109
- PKNCAdose, 101, 103, 103, 105
- PKNCAresults, 101, 103, 104, 104
- print.PKNCAconc, 110
- print.PKNCAconc(), 111
- print.PKNCAdata, 111
- print.PKNCAdata(), 121
- print.PKNCAdose (print.PKNCAconc), 110
- print.PKNCAdose(), 111
- print.provenance, 112
- print.summary\_PKNCAresults, 112
- print.summary\_PKNCAresults(), 123
- right\_join.PKNCAconc

- (inner\_join.PKNCAResults), 45
- right\_join.PKNCAdose
  - (inner\_join.PKNCAResults), 45
- right\_join.PKNCAResults
  - (inner\_join.PKNCAResults), 45
- round(), 114
- roundingSummarize, 113
- roundString, 113
- roundString(), 118
  
- set\_intervals, 117
- setAttributeColumn, 114
- setDuration (setDuration.PKNCAConc), 115
- setDuration.PKNCAConc, 115
- setExcludeColumn, 116
- setRoute, 116
- signif(), 118
- signifString, 117
- signifString(), 113, 114
- sort.interval.cols, 118
- sparse\_auc\_weight\_linear, 17, 80, 119, 120
- sparse\_auc\_weight\_linear(), 80
- sparse\_mean, 17, 80, 119, 120
- sparse\_pk\_attribute, 120
- sparse\_to\_dense\_pk, 121
- summary.PKNCAConc (print.PKNCAConc), 110
- summary.PKNCAdata, 121
- summary.PKNCAdose (print.PKNCAConc), 110
- summary.PKNCAResults, 122
- summary.PKNCAResults(), 86, 99, 112
- superposition, 123
  
- tbl(), 44
- time\_calc, 125
- tss.monoexponential.generate.formula, 126
  
- ungroup.PKNCAConc
  - (group\_by.PKNCAResults), 43
- ungroup.PKNCAdose
  - (group\_by.PKNCAResults), 43
- ungroup.PKNCAResults
  - (group\_by.PKNCAResults), 43
  
- var\_sparse\_auc, 127
- vname, 12–14