Package ‘tcpl’

October 6, 2023

Title ToxCast Data Analysis Pipeline

Version 3.1.0

Description A set of tools for processing and modeling high-throughput and high-content chemical screening data. The package was developed for the the chemical screening data generated by the US EPA ToxCast program, but can be used for diverse chemical screening efforts.

URL https://github.com/USEPA/CompTox-ToxCast-tcpl

Depends R (>= 3.5.0)

Imports data.table (>= 1.9.4), DBI, RMariaDB, numDeriv, RColorBrewer, utils, stats, methods, graphics, grDevices, sqldf, dplyr, tidyr, plotly, tcplfit2, ggplot2, gridExtra, stringr

Suggests roxygen2, knitr, prettydoc, rmarkdown, htmlTable, testthat (>= 3.0.0), reshape2, viridis, kableExtra, colorspace, magrittr, vdiffr

License MIT + file LICENSE

LazyData true

RoxygenNote 7.2.3

VignetteBuilder knitr

Encoding UTF-8

Config/testthat/edition 3

NeedsCompilation no

Author Richard S Judson [ctb, ths],
Dayne L Filer [aut],
Jason Brown [cre],
Sarah E Davidson-Fritz [ctb] (<https://orcid.org/0000-0002-2891-9380>),
Madison Feshuk [ctb] (<https://orcid.org/0000-0002-1390-6405>),
Lori Kolaczkowski [ctb],
Kurt Dunham [ctb],
Carter Thunes [ctb],
Ashley Ko [ctb],
Todd Zurlinden [ctb],

Parth Kothiya [ctb],
Woodrow R Setzer [ctb],
Matthew T Martin [ctb, ths],
Katie Paul Friedman [ctb] (<https://orcid.org/0000-0002-2710-1691>)

Maintainer Jason Brown <brown.jason@epa.gov>

Repository CRAN

Date/Publication 2023-10-06 19:50:02 UTC

R topics documented:

`.buildAssayQ` *Generate query for assay information*

Description

`.buildAssayQ` generates a query string to load assay information

Usage

```
.buildAssayQ(out, tblo, fld = NULL, val = NULL, add.fld = NULL)
```

Arguments

`out`	Character, the default fields to include
`tblo`	Integer, the order to send the fields to prepOutput
`fld`	Character, the field(s) to query/subset on
`val`	List, vectors of values for each field to query/subset on. Must be in the same order as 'fld'.
`add.fld`	Character, additional field(s) to include, but not query/ subset on

Value

A character containing the query to send to tcplQuery

`.convertNames` *Convert assay names to their abbreviations*

Description

`.convertNames` converts the assay names as they appear in the tcpl database to their respective abbreviations

Usage

```
.convertNames(names)
```

Arguments

`names`	Character, strings to convert

Value

The same character vector given with any name strings converted to the abbreviated version

`.load6DR` *Load data for tcpl6*

Description

`.load6DR` loads dose-response data for tcpl6.

Usage

```
.load6DR(ae)
```

Arguments

`ae`	String aeid to query on

`.plateHeat` *Plot plate heatmap*

Description

Plot plate heatmap, to be used with tcplPlotPlate

Usage

```
.plateHeat(vals, rowi, coli, wllt, wllq, rown, coln, main, arng)
```

Arguments

`vals`	Numeric, the well values
`rowi`	Integer, the row index
`coli`	Integer, the column index
`wllt`	Character, the well type
`wllq`	Logical, the well quality
`rown`	Integer, the number of rows on the plate
`coln`	Integer, the number of columns on the plate
`main`	Character of length 1, the title/main
`arng`	Numeric of length 2, the minimum and maximum values to constrain the color scale

Note

Optimized for an output with height = 20/3, width = 10, and pointsize = 10

`.prepField` *Paste appropriate table name to field name*

Description

Paste appropriate table name to field name

Usage

```
.prepField(fld, tbl, db)
```

Arguments

`fld`	Character, the table fields
`tbl`	Character, the possible tables
`db`	Character, the database containing the tables

Details

The function loops through the given tables, and for each field i it assigns the last table containing i to i. ORDER OF FLD MATTERS!!

`blineShift` *Shift the baseline to 0*

Description

`blineShift` Takes in dose-response data and shifts the baseline to 0 based on the window.

Usage

```
blineShift(resp, logc, wndw)
```

Arguments

`resp`	Numeric, the response values
`logc`	Numeric, the log10 concentration values
`wndw`	Numeric, the threshold window

Value

A numeric vector containing the shifted response values

Note

This function is not exported and is not intended to be used by the user.

See Also

`mc3_mthds`, `mc3`

`chdat` *Chemical library of tested chemicals in the example datasets with the corresponding sample IDs.*

Description

Chemical library of tested chemicals in the example datasets with the corresponding sample IDs.

Usage

```
chdat
```

Format

A data frame with 6 rows and 6 variables:

spid	sample ID
casn	Chemical Abstract Service(CAS) number
chnm	chemical name
dsstox_substance_id	chemical-specific DTXSID
code	CAS number compressed into numeric string
chid	unique chemical ID number for tcpl

Source

ToxCast database

`check_tcpl_db_schema` *Function that checks if the most recent v3 table schema is used in the database schema*

Description

Function that checks if the most recent v3 table schema is used in the database schema

Usage

```
check_tcpl_db_schema()
```

Value

boolean TRUE if param tables are listed in schema FALSE otherwise

Examples

```
## Not run:
#connect to database first with tcplConf
tcplConf(user=user,
 pass= pass,
 db=dbname,
 drvr='MySQL',
 host=hostname)

#check if it is part of the new schema
new_schema <- check_tcpl_db_schema()

## End(Not run)
```

`Configure functions` *Functions for configuring the tcpl package*

Description

These functions are used to configure the tcpl settings.

Usage

```
tcplConf(drvr = NULL, user = NULL, pass = NULL, host = NULL, db = NULL, ...)

tcplConfDefault()

tcplConfExample()

tcplConfList(show.pass = FALSE)

tcplConfLoad(list.new = TRUE)

tcplConfReset()

tcplConfSave()
```

Arguments

`drvr`	Character of length 1, which database driver to use
`user`	Character of length 1, the database server username
`pass`	Character of length 1, the database server password
`host`	Character of length 1, the database server
`db`	Character of length 1, the name of the tcpl database
`...`	Additional arguments that should be passed to dbConnect function
`show.pass`	Logical, should the password be returned
`list.new`	Logical of length 1, should the new settings be printed?

Details

Currently, the tcpl package only supports the "MySQL" and "tcplLite" database drivers.

The settings can be stored in a configuration file to make the using the package more user-friendly. To create the configuration file, the user must first create a system environment variable ('TCPL_CONF') that points to to the file. There is more information about system environment variables in `Startup` and `Sys.getenv`. Briefly, the user needs to modify the '.Renviron' file in their home directory. If the file does not exist, create it, and add the following line:

TCPL_CONF=path/to/confFile.conf

Here 'path/to/confFile.conf' can be any path to a file. One suggestion would be to include .tcplConf in the home directory, e.g. TCPL_CONF=~/.tcplConf. Note, '~' may not indicate the home directory on every operating system. Once the environment variable is added, the user can change the settings using `tcplConf`, then save the settings to the file given by the TCPL_CONF environment variable running `tcplConfSave()`.

`tcplConf` changes `options` to set the tcpl-specific options, most importantly to configure the connection to the tcpl databases. `tcplConf` will only change non-null values, and can be used to change a single value if needed.

`tcplConfSave` modifies the configuration file to reflect the current tcpl settings.

`tcplConfList` lists the values assigned to the tcpl global options.

`tcplConfLoad` updates the tcpl settings to reflect the current configuration file.

`tcplConfDefault` changes the `options` to reflect the default settings for the example tcplLite database, i.e. local directory, but does not alter the configuration file.

`tcplConfReset` is used to generate the initial configuration script, and can be used to reset or regenerate the configuration script by the user.

`flareFunc` *Calculate the weighted mean of a square to detect plate flares*

Description

`flareFunc` calculates the weighted mean of square regions to detect plate flares.

Usage

```
flareFunc(val, coli, rowi, apid, r)
```

Arguments

`val`	Numeric, the well values
`coli`	Integer, the well column index
`rowi`	Integer, the well row index
`apid`	Character, the assay plate id
`r`	Integer, the number of wells from the center well (in one direction) to make the square

See Also

MC6_Methods, Method functions, mc6

Hill model utilites *Functions to solve the Hill model*

Description

These functions solve for Hill model parameters.

Usage

```
tcplHillACXX(XX, tp, ga, gw, bt = 0)

tcplHillConc(val, tp, ga, gw, bt = 0)

tcplHillVal(logc, tp, ga, gw, bt = 0)
```

Arguments

XX	Numeric, the activity level (percentage of the top value)
tp	Numeric, the top value from the Hill model
ga	Numeric, the logAC50 value from the Hill model
gw	Numeric, the Hill coefficient from the Hill model
bt	Numeric, the bottom value from the Hill model
val	Numeric, the activity value
logc	Numeric, the log concentration

Details

tcplHillVal computes the value of the Hill model for a given log concentration.

tcplHillACXX computes the activity concentration for a Hill model for a given activity level.

tcplHillConc computes the Hill model concentration for a given value.

Examples

```
## The following code gives examples for a Hill model with a top of 50,
## bottom of 0, AC50 of 1 and Hill coefficient of 1.
## tcplHillVal calculates activity value given a concentration. tcplHillVal
## will return the tp/2 when logc equals ga:
tcplHillVal(logc = 1, tp = 50, ga = 1, gw = 1, bt = 0)

## Here, tcplHillConc returns the concentration where the value equals 20
tcplHillConc(val = 20, tp = 50, ga = 1, gw = 1, bt = 0)

## Note how this differs from tcplHillACXX:
```

```
tcplHillACXX(XX = 20, tp = 50, ga = 1, gw = 1, bt = 0)

## tcplHillACXX is based on the top value and allows the user to calculate
## specifc activity concentrations based on a percentage of the top value

## For example, we can calculate the value for the concentration 0.25, then
## use that value to check the other two functions.

value <- tcplHillVal(logc = 0.25, tp = 50, ga = 1, gw = 1, bt = 0)
c1 <- tcplHillConc(val = value, tp = 50, ga = 1, gw = 1, bt = 0)
c2 <- tcplHillACXX(XX = value/50*100, tp = 50, ga = 1, gw = 1, bt = 0)
all.equal(0.25, c1, c2)

## Notice, the value had to be transformed to a percentage of the top value
## when using tcplHillACXX
```

`interlaceFunc` *Calculate the weighted mean of a square to detect interlace effect*

Description

`interlaceFunc` calculates the distance weighted mean of square regions from a 384-well plate that is interlaced onto a 1536 well plate to detect non-random signals coming from the source plate

Usage

```
interlaceFunc(val, intq, coli, rowi, apid, r)
```

Arguments

`val`	Numeric, the well values
`intq`	Numeric, interlace quadrant
`coli`	Integer, the well column index
`rowi`	Integer, the well row index
`apid`	Character, the assay plate id
`r`	Integer, the number of wells from the center well (in one direction) to make the square

See Also

`MC6_Methods`, `Method functions`, `mc6`

`invitrodb_dd` *Short descriptions of fields for different tables are stored in a data dictionary.*

Description

Short descriptions of fields for different tables are stored in a data dictionary.

Usage

```
invitrodb_dd
```

Format

A data frame with 44 rows and 3 variables:

invitrodb_table Table of the data dictionary

invitrodb_field Field of the data dictionary

description Description

Source

ToxCast database

`is.odd` *Check for odd numbers*

Description

`is.odd` takes an integer vector, `x`, and returns TRUE for odd integers.

Usage

```
is.odd(x)
```

Arguments

`x`	An integer

Value

`TRUE` for odd integers and `FALSE` for even integers.

See Also

Other tcpl abbreviations: `lu()`, `lw()`, `sink.reset()`

`Load assay information` *Functions for loading assay information*

Description

These functions query the tcpl databases and returns a data.table with assay ID and name information. More information about the assay hierarchy is available in the overview vignette.

Usage

```
tcplLoadAcid(fld = NULL, val = NULL, add.fld = NULL)

tcplLoadAeid(fld = NULL, val = NULL, add.fld = NULL)

tcplLoadAid(fld = NULL, val = NULL, add.fld = NULL)

tcplLoadAsid(fld = NULL, val = NULL, add.fld = NULL)
```

Arguments

`fld`	Character, the field(s) to query/subset on
`val`	List, vectors of values for each field to query/subset on. Must be in the same order as 'fld'.
`add.fld`	Character, additional field(s) to include, but not query/ subset on

Details

Each element in the assay hierarchy has its own function, loading the ID and name for the given assay element. For example, `tcplLoadAsid` will return the assay source ID (asid) and assay source name (asnm).

Value

A data.table containing the ID, name, and any additional fields.

Examples

```
## Store the current config settings, so they can be reloaded at the end
## of the examples
conf_store <- tcplConfList()
TCPLlite <- file.path(system.file(package = "tcpl"), "example")
tcplConf(db = TCPLlite, user = NA, host = NA, drvr = "tcplLite")

## The load assay functions can be used without any parameters to list the
## full list of registered assay elements:
tcplLoadAsid()
tcplLoadAeid()
```

```
## Similarly, the user can add fields without doing any element selection:
tcplLoadAeid(add.fld = c("asid", "aid", "acid"))

## Or, the user can look only at a subset:
tcplLoadAeid(fld = "aeid", val = 1, add.fld = "asid")

## The field can be any value in one of the corresponding assay element
## tables, but the functions also recognize the abbreviated version of
## the name fields.
tcplListFlds("assay")
a1 <- tcplLoadAeid(fld = "anm", val = "Steroidogenesis")
a2 <- tcplLoadAeid(fld = "assay_name", val = "Steroidogenesis")
identical(a1, a2)

## Reset configuration
options(conf_store)
```

---

lu — *Abbreviation for* `length(unique(x))`

Description

`lu` takes a logical vector, `x`, and returns `length(unique(x))`.

`lu` takes a logical vector, `x`, and returns `length(unique(x))`.

Usage

```
lu(x)

lu(x)
```

Arguments

`x`	A logical

Value

The unique of the `TRUE` values in `x`

The unique of the `TRUE` values in `x`

See Also

`unique`, `which`

`unique`, `which`

Other tcpl abbreviations: `is.odd()`, `lw()`, `sink.reset()`

Other tcpl abbreviations: `is.odd()`, `lw()`, `sink.reset()`

`lw` *Abbreviation for* `length(which(x))`

Description

`lw` takes a logical vector, `x`, and returns `length(which(x))`.

`lw` takes a logical vector, `x`, and returns `length(which(x))`.

Usage

```
lw(x)
```

```
lw(x)
```

Arguments

`x`	A logical

Value

The length of the `TRUE` values in `x`

The length of the `TRUE` values in `x`

See Also

`length`, `which`

`length`, `which`

Other tcpl abbreviations: `is.odd()`, `lu()`, `sink.reset()`

Other tcpl abbreviations: `is.odd()`, `lu()`, `sink.reset()`

`mc1` *Perform level 1 multiple-concentration processing*

Description

`mc1` loads level 0 data from the tcpl database for the given id and performs level 1 multiple-concentration processing. The processed data is then loaded into the mc1 table and all subsequent data is deleted with `tcplCascade`. See details for more information.

The individual processing functions are no longer exported, as it is typically more convenient and suggested to use the `tcplRun` wrapper function.

Usage

```
mc1(ac, wr = FALSE)
```

Arguments

`ac`	Integer of length 1, assay component id (acid) for processing.
`wr`	Logical, whether the processed data should be written to the tcpl database

Details

Level 1 processing includes defining the concentration and replicate index, cndx and repi, respectively.

Value

A boolean of length 1, indicating the success of the processing, or when 'wr' is FALSE, a list where the first element is a boolean indicating the success of processing and the second element is a data.table containing the processed data

See Also

Other multiple-concentration: `mc2()`, `mc3()`, `mc4()`, `mc5()`, `mc6()`

`mc2` *Perform level 2 multiple-concentration processing*

Description

`mc2` loads level 1 data from the tcpl database for the given id and performs level 2 multiple-concentration processing. The processed data is then loaded into the mc2 table and all subsequent data is deleted with `tcplCascade`. See details for more information.

The individual processing functions are no longer exported, as it is typically more convenient and suggested to use the `tcplRun` wrapper function.

Usage

```
mc2(ac, wr = FALSE)
```

Arguments

`ac`	Integer of length 1, assay component id (acid) for processing.
`wr`	Logical, whether the processed data should be written to the tcpl database

Details

Level 2 multiple-concentration processing includes defining the corrected value, cval, based on the correction methods listed in the mc2_acid and mc2_methods tables.

Value

A boolean of length 1, indicating the success of the processing, or when 'wr' is FALSE, a list where the first element is a boolean indicating the success of processing and the second element is a data.table containing the processed data

See Also

`Method functions`, `MC2_Methods`

Other multiple-concentration: `mc1()`, `mc3()`, `mc4()`, `mc5()`, `mc6()`

`MC2_Methods` *List of level 2 multiple-concentration correction functions*

Description

`mc2_mthds` returns a list of correction/transformation functions to be used during level 2 multiple-concentration processing.

Usage

```
mc2_mthds()
```

Details

The functions contained in the list returned by `mc2_mthds` return a list of expressions to be executed in the `mc2` (not exported) function environment. The functions are described here for reference purposes, The `mc2_mthds` function is not exported, nor is it intended for use.

All available methods are described in the Available Methods section, listed by the function/method name.

Value

A list functions

Available Methods

More information about the level 2 multiple-concentration processing is available in the package vignette, "Data_processing."

Correction Methods:

log2 Transform the corrected response value (cval) to log-scale (base 2).

log10 Transform the corrected response value (cval) to log-scale (base 10).

rmneg Exclude wells with negative corrected response values (cval) and downgrading their well quality (wllq); if $cval < 0, wllq = 0$.

rmzero Exclude wells with corrected response values (cval) equal to zero and downgrading their well quality (wllq); if $cval = 0, wllq = 0$.

mult25 Multiply corrected response value (cval) by 25; $25 * cval$.

mult100 Multiply corrected response value (cval) by 100; $100 * cval$.

negshift Shift corrected response values (cval) by subtracting the minimum cval and adding 1, such that the new minimum is 1; $cval - min + 1$.

mult2.5 Multiply corrected response value (cval) by 2.5; $2.5 * cval$.

mult3 Multiply corrected response value (cval) by 3; $3 * cval$.

mult6 Multiply corrected response value (cval) by 6; $6 * cval$.

sub100 Center data around zero by subtracting the corrected response value (cval) from 100; $100 - cval$. Typically used if data was pre-normalized around 100 with responses decreasing to 0.

zscore.npwlls Convert the corrected response value (cval) to an absolute Z-Score based on the neutral and positive control wells (wllts = n and p), by assay plate ID (apid); $cval = |(cval - mean(cval\ for\ wllt\ =\ n\ and\ p)/sd(cval\ for\ wllt\ =\ n\ and\ p)|$.

sub1 Center data around zero by subtracting the corrected response value (cval) from 1; $1 - cval$. Typically used if data was pre-normalized around 1 with responses decreasing to 0.

Aggregation Methods:

agg.mean.rep.apid Aggregate technical test replicates (wllt=t) by taking the plate-wise mean per sample id (spid), assay plate (apid), and concentration index (cndx).

agg.median.rep.apid Aggregate technical test replicates (wllt=t) by taking the plate-wise median per sample id (spid), assay plate (apid), and concentration index (cndx).

agg.percent.rep.spid Use for binary data. Aggregate technical replicates as percentage by taking the sum of hits relative to total replicates per sample id (spid) and concentration index (cndx); cval = (sum(rval)/.N)*100.

agg.percent.rep.spid.min1 Use for binary data. Aggregate technical replicates as percentage by taking the sum of hits relative to total replicates per per sample id (spid) and concentration index (cndx), where there is more than one replicate; cval = (sum(rval)/.N)*100, where .N>1.

agg.mean.rep.apid Aggregate technical replicates by taking the plate-wise mean per sample id (spid), assay plate (apid), and concentration index (cndx).

agg.median.rep.apid Aggregate technical replicates by taking the plate-wise median per sample id (spid), assay plate (apid), and concentration index (cndx).

agg.percent.rep.spid Use for binary data. Aggregate technical replicates as percentage by taking the sum of hits relative to total replicates per sample id (spid) and concentration index (cndx); $cval = (sum(rval)/.N) * 100$.

agg.percent.rep.spid.min1 Use for binary data. Aggregate technical replicates as percentage by taking the sum of hits relative to total replicates per per sample id (spid) and concentration index (cndx), where there is more than one replicate; $cval = (sum(rval)/.N) * 100,\ where\ .N > 1$.

Note

This function is not exported and is not intended to be used by the user.

See Also

`mc2`, `Method functions` to query what methods get applied to each acid

mc3 — *Perform level 3 multiple-concentration processing*

Description

`mc3` loads level 2 data from the tcpl database for the given id and performs level 3 multiple-concentration processing. The processed data is then loaded into the mc3 table and all subsequent data is deleted with `tcplCascade`. See details for more information.

The individual processing functions are no longer exported, as it is typically more convenient and suggested to use the `tcplRun` wrapper function.

Usage

```
mc3(ac, wr = FALSE)
```

Arguments

`ac`	Integer of length 1, assay component id (acid) for processing.
`wr`	Logical, whether the processed data should be written to the tcpl database

Details

Level 3 multiple-concentration processing includes mapping assay component to assay endpoint, duplicating the data when the assay component has multiple assay endpoints, and any normalization of the data. Data normalization based on methods listed in mc3_aeid and mc3_methods tables.

Value

A boolean of length 1, indicating the success of the processing, or when 'wr' is FALSE, a list where the first element is a boolean indicating the success of processing and the second element is a data.table containing the processed data

See Also

`Method functions`, `MC3_Methods`

Other multiple-concentration: `mc1()`, `mc2()`, `mc4()`, `mc5()`, `mc6()`

MC3_Methods — *List of level 3 multiple-concentration normalization methods*

Description

`mc3_mthds` returns a list of normalization methods to be used during level 3 multiple-concentration processing.

Usage

```
mc3_mthds()
```

Details

The functions contained in the list returned by `mc3_mthds` take `aeids` (a numeric vector of aeid values) and returns a list of expressions to be executed in the `mc3` (not exported) function environment. The functions are described here for reference purposes, The `mc3_mthds` function is not exported, nor is it intended for use.

All available methods are described in the Available Methods section, listed by the type of function and the function/method name.

Value

A list of functions

Available Methods

The methods are broken into three types, based on what fields they define. Different methods are used to define "bval" (the baseline value), "pval" (the positive control value), and "resp" (the final response value).

Although it does not say so specifically in each description, all methods are applied by aeid.

More information about the level 3 multiple-concentration processing is available in the package vignette, "Data_processing."

bval Methods:

bval.apid.nwlls.med Calculate the baseline value (bval) as the plate-wise median, by assay plate ID (apid), of the corrected values (cval) for neutral control wells (wllt = n).

bval.apid.lowconc.med Calculate the baseline value (bval) as the plate-wise median, by assay plate ID (apid), of the corrected values (cval) for test compound wells (wllt = t) with a concentration index (cndx) of 1 or 2.

bval.apid.twlls.med Calculate the baseline value (bval) as the plate-wise median, by assay plate ID (apid), of the corrected values (cval) of test compound wells (wllt = t).

bval.apid.tn.med Calculate the baseline value (bval) as the plate-wise median, by assay plate ID (apid), of the corrected values (cval) for test compound wells (wllt = t) and neutral control wells (wllt = n).

bval.apid.nwllslowconc.med Calculate the baseline value (bval) as the plate-wise median, by assay plate ID (apid), of the corrected values (cval) of test compound wells (wllt = t) with a concentration index (cndx) of 1 or 2 or neutral control wells (wllt = n).

bval.spid.lowconc.med Calculate the baseline value (bval) as the sample-wise median, by sample ID (spid), of the corrected values (cval) of the three lowest concentration test compound wells (wllt = t and cndx = 1, 2, & 3).

bval.apid.nwllstcwllslowconc.med Calculate the baseline value (bval) as the plate-wise median, by assay plate ID (apid), of the corrected values (cval) for neutral control wells (wllt = n) or wells with a concentration index (cndx) of 1 or 2 and well type of test compound (wllt = t) or gain-of-signal control in multiple concentrations (wllt = c).

bval.aeid.nwlls.med Calculate the baseline value (bval) as the endpoint-wise median, by assay component endpoint ID (aeid), corrected value (cval) for neutral control wells (wllt = n).

pval Methods:

pval.apid.pwlls.med Calculate the positive control value (pval) as the plate-wise median, by assay plate ID (apid), of the corrected values (cval) for single-concentration gain-of-signal positive control wells (wllt = p).

pval.apid.mwlls.med Calculate the positive control value (pval) as the plate-wise median, by assay plate ID (apid), of the corrected values (cval) for multiple-concentration loss-of-signal negative control wells (wllt = m).

pval.apid.medpcbyconc.max Calculate the positive control value (pval) as the plate-wise maximum, by assay plate ID (apid), of the medians of the corrected values (cval) for gain-of-signal single- or multiple-concentration negative control wells (wllt = m or o) by apid, well type, and concentration.

pval.apid.medpcbyconc.min Calculate the positive control value (pval) as the plate-wise minimum, by assay plate ID (apid), of the medians of corrected value (cval) of gain-of-signal single- or multiple-concentration positive control wells (wllt = p or c) by apid, well type, and concentration.

pval.apid.medncbyconc.min Calculate the positive control value (pval) as the plate-wise minimum, by assay plate ID (apid), of the medians of the corrected values (cval) for gain-of-signal single- or multiple-concentration negative control wells (wllt = m or o) by apid, well type, and concentration.

pval.apid.pmv.min Calculate the positive control value (pval) as the plate-wise minimum, by assay plate ID (apid), of the medians of the corrected values (cval) for single-concentration gain-of-signal, multiple-concentration loss-of-signal, or viability control wells (wllt = p, m, or v) by apid, well type, and concentration.

pval.apid.pmv.max Calculate the positive control value (pval) as the plate-wise maximum, by assay plate ID (apid), of the medians of the corrected values (cval) for single-concentration gain-of-signal, multiple-concentration loss-of-signal, or viability control wells (wllt = p, m, or v) by apid, well type, and concentration.

pval.apid.f.max Calculate the positive control value (pval) as the plate-wise maximum, by assay plate ID (apid), of the medians of important reference wells (wllt = f) values by apid and concentration.

pval.apid.f.min Calculate the positive control value (pval) as the plate-wise minimum, by assay plate ID (apid), of the medians of important reference wells (wllt = f) values by apid and concentration.

pval.apid.p.max Calculate the positive control value (pval) as the plate-wise maximum, by assay plate ID (apid), of the medians of the corrected values (cval) for single-concentration gain-of-signal control wells (wllt = p) by apid.

pval.apid.p.min Calculate the positive control value (pval) as the plate-wise minimum, by assay plate ID (apid), of the medians of corrected values (cval) for single-concentration gain-of-signal control wells (wllt = p) by apid.

pval.apid.v.min Calculate the positive control value (pval) as the plate-wise minimum, by assay plate ID (apid), of the medians of the corrected values (cval) for viability control wells (wllt = v) by apid and concentration.

pval.zero Set the positive control value (pval) to 0; $pval = 0$.

pval.apid.owlls.med Calculate the positive control value (pval) as the plate-wise median, by assay plate ID (apid), of the corrected values (cval) for single-concentration negative control wells (wllt = o).

pval.2bval Calculate the positive control value (pval) as the plate-wise median, by assay plate ID (apid), of the corrected values (cval) for neutral control wells (wllt = n) multiplied by 2.

pval.maxp Calculate the positive control value (pval) as the endpoint-wise maximum, by assay component ID (aeid), of the corrected values for single-concentration gain-of-signal wells (wllt = p).

pval.apid.bwlls.med Calculate the positive control value (pval) as the plate-wise median, by assay plate ID (apid), of the corrected values (cval) for blank wells (wllt= b).

pval.twlls.99pct Calculate positive control value (pval) as the 99th percentile of all corrected value (cvals) of the test compound wells (wllt = t).

pval.neg.100 Calculate positive control value (pval) as -100 for endpoints in the down direction; $pval = -100$.

resp Methods:

resp.pc Calculate the normalized response (resp) as a percent of control, i.e. the ratio of the difference between the corrected (cval) and baseline (bval) values divided the difference between the positive control (pval) and baseline (bval) values multiplied by 100; $resp = (cval - bval)/(pval - bval) * 100$.

resp.pc.pval.cor Calculate the normalized response (resp) as a percent of control, i.e. the ratio of the difference between the corrected (cval) and baseline (bval) values divided the positive control (pval) value multiplied by 100; $resp = (cval - bval)/pval * 100$.

resp.fc Calculate the normalized response (resp) as the fold change, i.e. the ratio of the corrected (cval) and baseline (bval) values; $resp = cval/bal$.

resp.logfc Calculate the normalized response (resp) as the fold change of logged, i.e. the difference between corrected (cval) and baseline (bval) log-scale values.

resp.log2 Transform the response values to log-scale (base 2).

resp.mult25 Multiply the normalized response value (resp) by 25; $25 * resp$.

resp.scale.mad.log2fc Scale the normalized response value (resp) by the ratio of log2(1.2) and 3 multiplied by the baseline median absolute deviation (bmad) of the unscaled normalized response values (resp); $(\log_2 1.2)/3 * bmad * resp$.

resp.scale.quant.log2fc Scale the normalized response value (resp). First, determine the maximum difference (md) by finding the maximum between the absolute difference of the 1st percentile minus the 50th percentile and the absolute difference of the 99th percentile minus the 50th percentile. Then multiply resp by log2(1.2) divided by 20 percent of md; $(\log_2 1.2)/0.2 * md * resp$.

resp.multneg1 Multiply the normalized response value (resp) by -1; $-1 * resp$.

resp.shiftneg.3bmad Shift all the normalized response values (resp) less than -3 multiplied by the baseline median absolute deviation (bmad) to 0; if $resp < -3 * bmad,\ resp = 0$.

resp.shiftneg.6bmad Shift all the normalized response values (resp) less than -6 multiplied by the baseline median absolute deviation (bmad) to 0; if $resp < -6 * bmad,\ resp = 0$.

resp.shiftneg.10bmad Shift all the normalized response values (resp) less than 10 multiplied by the baseline median absolute deviation (bmad) to 0; if $resp < -10 * bmad,\ resp = 0$.

resp.blineshift.3bmad.repi Shift the normalized response value (resp) with a baseline correction, by replicate index (repi), with a window of 3 multiplied by the baseline median absolute deviation (bmad).

resp.blineshift.50.repi Shift the normalized response value (resp) with a baseline correction, by replicate index (repi), with a window of 50.

resp.blineshift.3bmad.spid Shift the normalized response value (resp) with a baseline correction, by sample ID (spid), with a window of 3 multiplied by the baseline median absolute deviation (bmad).

resp.blineshift.50.spid Shift the normalized response value (resp) with a baseline correction, by sample ID (spid), with a window of 50.

none Set the corrected response value (cval) as the normalized response value (resp); $cval = resp$. No additional mc3 methods needed for endpoint-specific normalization.

resp.zerocenter.fc Calculate the normalized response (resp) as a zero center fold change, i.e. 1 minus the ratio of corrected (cval) and baseline (bval) values; $resp = 1 - cval/bval$. Typically used for increasing responses.

resp.incr.zerocenter.fc Calculate the normalized response (resp) as a zero center fold change, i.e. the ratio of the the corrected (cval) and baseline (bval) values minus 1; $resp = cval/bval - 1$. Typically used for increasing responses.

resp.mult100 Multiply the normalized response value (resp) by 100; $100 * resp$.

Note

This function is not exported and is not intended to be used by the user.

See Also

`mc3`, `Method functions` to query what methods get applied to each aeid

`mc4` *Perform level 4 multiple-concentration processing*

Description

`mc4` loads level 3 data from the tcpl database for the given id and performs level 4 multiple-concentration processing. The processed data is then loaded into the mc4 table and all subsequent data is deleted with `tcplCascade`. See details for more information.

The individual processing functions are no longer exported, as it is typically more convenient and suggested to use the `tcplRun` wrapper function.

Usage

```
mc4(ae, wr = FALSE)
```

Arguments

ae	Integer of length 1, assay endpoint id (aeid) for processing.
wr	Logical, whether the processed data should be written to the tcpl database

Details

Level 4 multiple-concentration modeling takes the dose-response data for chemical-assay pairs, and fits three models to the data: constant, hill, and gain-loss. For more information about the models see `Models`. When a chemical has more than one sample, the function fits each sample separately.

Value

A boolean of length 1, indicating the success of the processing, or when 'wr' is FALSE, a list where the first element is a boolean indicating the success of processing and the second element is a data.table containing the processed data

See Also

`tcplFit`, `Models`

Other multiple-concentration: `mc1()`, `mc2()`, `mc3()`, `mc5()`, `mc6()`

`MC4_Methods` *List of level 4 multiple-concentration methods for calculating bmad*

Description

`mc4_mthds` returns a list of methods to be used during level 4 multiple-concentration processing for calculating bmad

Usage

```
mc4_mthds()
```

Details

The functions contained in the list returned by `mc4_mthds` take `aeids` (a numeric vector of aeid values) and returns a list of expressions to be executed in the `mc4` (not exported) function environment. The functions are described here for reference purposes, The `mc4_mthds` function is not exported, nor is it intended for use.

All available methods are described in the Available Methods section, listed by the type of function and the function/method name.

Value

A list of functions

Available Methods

Although it does not say so specifically in each description, all methods are applied by aeid.

More information about the level 4 multiple-concentration processing is available in the package vignette, "Data_processing."

bmad.aeid.lowconc.twells Calculate the baseline median absolute value (bmad) as the median absolute deviation of normalized response values (rep) for test compound wells (wllt = t) with concentration index (cndx) equal to 1 or 2.

bmad.aeid.lowconc.nwells Calculate the baseline median absolute value (bmad) as the median absolute deviation of normalized response values (resp) for neutral control wells (wllt = n).

onesd.aeid.lowconc.twells Calculate one standard deviation of the normalized response for test compound wells (wllt = t) with a concentration index (cndx) of 1 or 2; $onesd = \sqrt{\sum (resp - mean(resp))^2/(n-1)}$. Used to establish BMR and therefore required for tcplfit2 processing.

bidirectional.false Limits bidirectional fitting and processes data in positive analysis direction only. Use for gain-of-signal or inverted data.

bmad5.onesd16.static Replace baseline median absolute deviation (bmad) with 5 and one standard deviation (osd) of the normalized response for test compound wells (wllt = t) with a concentration index (cndx) of 1 or 2 with 16. Typically used for binary data where values would otherwise be 0; non-zero values are required for tcplfit2 processing.

Note

This function is not exported and is not intended to be used by the user.

See Also

`mc4`, `Method functions` to query what methods get applied to each aeid

`mc5` *Perform level 5 multiple-concentration processing*

Description

`mc5` loads level 4 data from the tcpl database for the given id and performs level 5 multiple-concentration processing. The processed data is then loaded into the mc5 table and all subsequent data is deleted with `tcplCascade`. See details for more information.

The individual processing functions are no longer exported, as it is typically more convenient and suggested to use the `tcplRun` wrapper function.

Arguments

`ae`	Integer of length 1, assay endpoint id (aeid) for processing.
`wr`	Logical, whether the processed data should be written to the tcpl database

Details

Level 5 multiple-concentration hit-calling uses the fit parameters and the activity cutoff methods from mc5_aeid and mc5_methods to make an activity call and identify the winning model for each fit.

Value

A boolean of length 1, indicating the success of the processing, or when 'wr' is FALSE, a list where the first element is a boolean indicating the success of processing and the second element is a data.table containing the processed data

See Also

`Method functions`, `MC5_Methods`

Other multiple-concentration: `mc1()`, `mc2()`, `mc3()`, `mc4()`, `mc6()`

`MC5_Methods` *Load list of level 5 multiple-concentration cutoff methods*

Description

`mc5_mthds` returns a list of additional activity cutoff methods to be used during level 5 multiple-concentration processing.

Usage

```
mc5_mthds(ae)
```

Arguments

`ae`	Integer of length 1, the assay endpoint id

Details

The functions contained in the list returned by `mc5_mthds` take `aeids` (a numeric vector of aeid values) and returns a list of expressions to be executed in the `mc5` (not exported) function environment. The functions are described here for reference purposes, The `mc5_mthds` function is not exported, nor is it intended for use.

All available methods are described in the "Available Methods" section, listed by the cutoff type in ascending order of cutoff value.

Value

A list of functions

Available Methods

The methods are broken down into five categories based on the type of cutoff they assign. Different methods are used to define cutoffs for "bmad" (baseline median absolute value), "fc" (fold change), "log" ($\log_2$ or $\log_{10}$), "pc" (percent of control), and "other" (uncategorized cutoffs).

All methods are applied by aeid.

Although there are method exceptions (notably within the "other" category), only highest calculated cutoff value based on assigned methods will be selected for hitcalling. Therefore, only the largest cutoff method per method type should be assigned.

More information about the level 5 multiple-concentration processing is available in the package vignette, "Data_processing."

BMAD Methods:

bmad1 Add a cutoff value of 1 multiplied by baseline median absolute value (bmad). By default, bmad is calculated using test compound wells (wllt = t) for the endpoint.

bmad2 Add a cutoff value of 2 multiplied by the baseline median absolute deviation (bmad). By default, bmad is calculated using test compound wells (wllt = t) for the endpoint.

bmad3 Add a cutoff value of 3 multiplied by the baseline median absolute deviation (bmad) as defined at Level 4.

bmad4 Add a cutoff value of 4 multiplied the baseline median absolute deviation (bmad). By default, bmad is calculated using test compound wells (wllt = t) for the endpoint.

bmad5 Add a cutoff value of 5 multiplied the baseline median absolute deviation (bmad). By default, bmad is calculated using test compound wells (wllt = t) for the endpoint.

bmad6 Add a cutoff value of 6 multiplied by the baseline median absolute deviation (bmad). By default, bmad is calculated using test compound wells (wllt = t) for the endpoint.

bmad10 Add a cutoff value of 10 multiplied by the baseline median absolute deviation (bmad). By default, bmad is calculated using test compound wells (wllt = t) for the endpoint.

Fold Change Methods:

fc0.2 Add a cutoff value of 0.2. Typically for zero centered fold change data.

fc0.3 Add a cutoff value of 0.3. Typically for zero centered fold change data.

Log Methods: Log Base 2

neglog2_0.88 Add a cutoff value of $-\log_2 0.88$.

log2_1.2 Add a cutoff value of $\log_2 1.2$. Typically for fold change data.

log2_2 Add a cutoff value $\log_2 2$. Typically for fold change data.

Log Base 10

log10_1.2 Add a cutoff value of $\log_{10} 1.2$. Typically for fold change data.

log10_2 Add a cutoff value of $\log_{10} 2$. Typically for fold change data.

Percent of Control Methods:

pc05 Add a cutoff value of 5. Typically for percent of control data.

pc10 Add a cutoff value of 10. Typically for percent of control data.

pc20 Add a cutoff value of 20. Typically for percent of control data.

pc25 Add a cutoff value of 25. Typically for percent of control data.

pc30 Add a cutoff value of 30. Typically for percent of control data.

pc50 Add a cutoff value of 50. Typically for percent of control data.

pc70 Add a cutoff value of 70. Typically for percent of control data.

pc95 Add a cutoff value of 95. Typically for percent of control data.

Other Methods:

maxmed20pct Add a cutoff value of 20 percent of the maximum of all endpoint maximal average response values (max_med).

coff_2.32 Add a cutoff value of 2.32.

loec.coff Method not yet updated for tcpl implementation. Identify the lowest observed effective concentration (loec) compared to baseline.

Note

This function is not exported and is not intended to be used by the user.

See Also

`mc5`, `Method functions` to query what methods get applied to each aeid.

`mc6` *Perform level 6 multiple-concentration processing*

Description

`mc6` loads level 5 data from the tcpl database for the given id and performs level 6 multiple-concentration processing. The processed data is then loaded into the mc6 table and all subsequent data is deleted with `tcplCascade`. See details for more information.

The individual processing functions are no longer exported, as it is typically more convenient and suggested to use the `tcplRun` wrapper function.

Usage

```
mc6(ae, wr = FALSE)
```

Arguments

`ae`	Integer of length 1, assay endpoint id (aeid) for processing.
`wr`	Logical, whether the processed data should be written to the tcpl database

Details

Level 6 multiple-concentration flagging uses both the plate level concentration-response data and the modeled parameters to flag potential false positives and false negative results.

Value

A boolean of length 1, indicating the success of the processing, or when 'wr' is FALSE, a list where the first element is a boolean indicating the success of processing and the second element is a data.table containing the processed data

See Also

`Method functions`, `MC6_Methods`

Other multiple-concentration: `mc1()`, `mc2()`, `mc3()`, `mc4()`, `mc5()`

`MC6_Methods` *Load list of level 6 multiple-concentration flag methods*

Description

`mc6_mthds` returns a list of flag methods to be used during level 6 multiple-concentration processing.

Usage

```
mc6_mthds()
```

Value

A list functions

Available Methods

More information about the level 6 multiple-concentration processing is available in the package vignette, "Data_processing."

modl.directionality.fail Flag series if model directionality is questionable, i.e. if the winning model direction was opposite, more responses (resp) would have exceeded the cutoff (coff). If loss was winning directionality (top < 0), flag if $count(resp < -1 * coff) < 2 * count(resp > coff)$. If gain was winning directionality (top > 0), flag if $count(resp > coff) < 2 * count(resp < -1 * coff)$.

low.nrep Flag series if the average number of replicates per concentration is less than 2; $nrep < 2$.

low.nconc Flag series if 4 concentrations or less were tested; $nconc <= 4$.

bmd.high Flag series if modeled benchmark dose (BMD) is greater than AC50 (concentration at 50 percent maximal response). This is indicates high variability in baseline response in excess of more than half of the maximal response.

singlept.hit.high Flag single-point hit that's only at the highest conc tested, where series is an active hit call (hitc >= 0.9) with the median response observed above baseline occurring only at the highest tested concentration tested.

singlept.hit.mid Flag single-point hit that's not at the highest conc tested, where series is an active hit call (hitc >= 0.9) with the median response observed above baseline occurring only at one concentration and not the highest concentration tested.

multipoint.neg Flag multi-point miss, where series is an inactive hit call (hitc < 0.9) with multiple median responses observed above baseline.

gnls.lowconc Flag series where winning model is gain-loss (gnls) and the gain AC50 is less than the minimum tested concentration, and the loss AC50 is less than the mean tested concentration.

noise Flag series as noisy if the quality of fit as calculated by the root mean square error (rmse) for the series is greater than the cutoff (coff); $rmse > coff$.

border Flag series if borderline activity is suspected based on modeled top parameter (top) relative to cutoff (coff); $|top| <= 1.2 * coff\ or\ |top| >= 0.8 * coff$.

overfit.hit Method not yet updated for tcpl implementation. Flag hit-calls that would get changed after doing the small N correction to the aic values.

efficacy.50 Flag low efficacy hits if series has an active hit call (hitc >= 0.9) and efficacy values (e.g. top and maximum median response) less than 50 percent; intended for biochemical assays. If $hitc >= 0.9$ and $coff >= 5$, then flag when $top < 50$ or $maxmed < 50$. If $hitc >= 0.9$ and $coff < 5$, then flag when $top < \log_2 1.5$ or $maxmed < \log_2 1.5$.

ac50.lowconc Flag series with an active hit call (hitc >= 0.9) if AC50 (concentration at 50 percent maximal response) is less than the lowest concentration tested; if $hitc >= 0.9$ and $ac50 < 10^{log_c min}$, then flag.

viability.gnls Flag series with an active hit call (hitc >= 0.9) if denoted as cell viability assay with winning model is gain-loss (gnls); if hitc >= 0.9, modl = "gnls" and cell_viability_assay = 1, then flag.

no.med.gt.3bmad Flag series where no median response values are greater than baseline as defined by 3 times the baseline median absolute deviation (bmad); nmed_gtbl = 0, where nmed_gtbl is the number of medians greater than 3 * bmad.

Note

This function is not exported and is not intended to be used by the user.

See Also

`mc6`, `Method functions` to query what methods get applied to each aeid.

`mcdat` *A subset of ToxCast data showing changes in the activity of the intracellular estrogen receptor.*

Description

The example dataset is used to illustrate how the user can pipeline multiple-concentration data from chemical screening using tcplLite.

Usage

```
mcdat
```

Format

A data frame with 14183 rows and 10 variables:

spid sample ID

apid assay plate ID

rowi well-plate row number

coli well-plate column number

wllt well type

wllq well quality

conc concentration in micromolar

rval raw assay component readout value

srcf source file containing the data

acsn assay component source name

Source

ToxCast database

`mc_vignette` *List with multi-concentration data for the vignette*

Description

This dataset is a list with 6 data.tables (mc0,mc1,mc2,mc3,mc4,mc5).

Usage

```
mc_vignette
```

Format

1. **mc0** A data frame with 78 rows and 18 columns containing level 0 formatted raw data.
   spid Sample ID
   chid Unique chemical ID number for tcpl
   casn Chemical Abstract Service(CAS) number
   chnm Chemical name
   dsstox_substance_id Chemical-specific DTXSID
   code CAS number compressed into numeric string
   acid Assay Component ID

acnm Assay Component Name

m0id Level 0 (mc0) ID

apid Assay plate ID

rowi Row Index

coli Column Index

wllt Well Type

wllq Well Quality (0 or 1)

conc Concentration in micromolar

rval Raw assay component readout value

srcf Source file containing the raw data

conc_unit Concentration Units

2. **mc1** A data frame with 78 rows and 21 columns containing level 1 replicate and concentration level indicated data.

spid Sample ID

chid Unique chemical ID number for tcpl

casn Chemical Abstract Service(CAS) number

chnm Chemical name

dsstox_substance_id Chemical-specific DTXSID

code CAS number compressed into numeric string

acid Assay Component ID

acnm Assay Component Name

m0id Level 0 (mc0) ID

m1id Level 1 (mc1) ID

apid Assay plate ID

rowi Row Index

coli Column Index

wllt Well Type

wllq Well Quality (0 or 1)

conc Concentration in micromolar

rval Raw assay component readout value

cndx Concentration index defined by ranking the unique concentrations, with the lowest concentration starting at 1.

repi Temporary replicate ID is defined, the data are scanned from top to bottom and increment the replicate index every time a replicate ID is duplicated

srcf Source file containing the raw data

conc_unit Concentration Units

3. **mc2** A data frame with 78 rows and 20 columns containing level 2 assay component-specific corrections.

spid Sample ID

chid Unique chemical ID number for tcpl

casn Chemical Abstract Service(CAS) number

chnm Chemical name

dsstox_substance_id Chemical-specific DTXSID
code CAS number compressed into numeric string
acid Assay Component ID
acnm Assay Component Name
m0id Level 0 (mc0) ID
m1id Level 1 (mc1) ID
m2id Level 2 (mc2) ID
apid Assay plate ID
rowi Row Index
coli Column Index
wllt Well Type
conc Concentration in micromolar
cval Corrected Value
cndx Concentration index defined by ranking the unique concentrations, with the lowest concentration starting at 1.
repi Temporary replicate ID is defined, the data are scanned from top to bottom and increment the replicate index every time a replicate ID is duplicated
conc_unit Concentration Units

4. **mc3** A data frame with 78 rows and 22 columns containing level 3 assay endpoint normalized data.

spid Sample ID
chid Unique chemical ID number for tcpl
casn Chemical Abstract Service(CAS) number
chnm Chemical name
dsstox_substance_id Chemical-specific DTXSID
code CAS number compressed into numeric string
aeid Assay Component Endpoint ID
aenm Assay endpoint name (i.e., assay_component_endpoint_name)
m0id Level 0 (mc0) ID
m1id Level 1 (mc1) ID
m2id Level 2 (mc2) ID
m3id Level 3 (mc3) ID
logc Log base 10 concentration
resp Normalized response value
cndx Concentration index defined by ranking the unique concentrations, with the lowest concentration starting at 1.
wllt Well Type
apid Assay plate ID
rowi Row Index
coli Column Index
repi Temporary replicate ID is defined, the data are scanned from top to bottom and increment the replicate index every time a replicate ID is duplicated

resp_unit Response Units

conc_unit Concentration Units

5. **mc4** A data frame with 5 rows and 149 columns containing level 4 concentration-response fitting data (all fits).

spid Sample ID

chid Unique chemical ID number for tcpl

casn Chemical Abstract Service(CAS) number

chnm Chemical name

dsstox_substance_id Chemical-specific DTXSID

code CAS number compressed into numeric string

aeid Assay Component Endpoint ID

aenm Assay endpoint name (i.e., assay_component_endpoint_name)

m4id Level 4 (mc4) ID

bmad The median absolute deviation of all treatment wells (default option) or blank wells

resp_max Maximum observed response

resp_min Minimum observed response

max_mean Maximum mean response

max_mean_conc Concentration of the maximum mean response

max_med Maximum median response

max_med_conc Concentration of the maximum median response

logc_max Maximum concentration on the log scale

logc_min Minimum concentration on the log scale

nconc The total number of concentration groups

npts Total number of observed responses (i.e. data points in the concentration series)

nrep Number of replicates in concentration groups

nmed_gtbl The number of median responses greater than 3BMAD

cnst_success Success indicator for the Constant model; 1 if the optimization was successful, otherwise 0

cnst_aic Akaike Information Criteria (AIC) for the Constant model

cnst_rme Root mean square error for the Constant model

cnst_er Error term for the Constant model

hill_success Success indicator for the Hill model; 1 if the optimization was successful, otherwise 0

hill_aic Akaike Information Criteria (AIC) for the Hill model

hill_cov Success indicator for the Hill model covariance calculation; 1 if the Hessian matrix inversion is successful, otherwise 0

hill_rme Root mean square erro for the Hill model

hill_tp The top parameter indicating the maximal estimated response

hill_ga The gain parameter for the Hill model, gain AC50

hill_p The power parameter for the Hill model

hill_er Error term for the Hill model

hill_tp_sd Standard deviation of the Hill model top parameter

hill_ga_sd Standard deviation of the Hill model gain parameter

hill_p_sd Standard deviation of the Hill model power parameter

hill_er_sd Standard deviation of the Hill model error term

hill_top The maximal response on the resulting Hill model fit

hill_ac50 Concentration at 50% of the maximal response on the Hill model fit

gnls_success Success indicator for the Gain-loss model; 1 if the optimization was successful, otherwise 0

gnls_aic Akaike Information Criteria (AIC) for the Gain-loss model

gnls_cov Success indicator for the Gain-loss model covariance calculation; 1 if the Hessian matrix inversion is successful, otherwise 0

gnls_rme Root mean square erro for the Gain-loss model

gnls_tp The top parameter indicating the maximal estimated response

gnls_ga The gain parameter for the Gain-loss model, gain AC50

gnls_p The gain power parameter for the Gain-loss model

gnls_la The loss parameter for the Gain-loss model, loss AC50

gnls_q The loss power parameter for the Gain-loss model

gnls_er Error term for the Gain-loss model

gnls_tp_sd Standard deviation of the Gain-loss model top parameter

gnls_ga_sd Standard deviation of the Gain-loss model gain parameter

gnls_p_sd Standard deviation of the Gain-loss model gain power parameter

gnls_la_sd Standard deviation of the Gain-loss model loss parameter

gnls_q_sd Standard deviation of the Gain-loss model loss power parameter

gnls_er_sd Standard deviation of the Gain-loss model error term

gnls_top The maximal response on the resulting Gain-loss model fit

gnls_ac50 Concentration at 50% of the maximal response on the Gain-loss model fit, gain AC50

gnls_ac50_loss Concentration at 50% of the maximal response on the Gain-loss model fit, loss AC50

poly1_success Success indicator for the Polynomial 1 model; 1 if the optimization was successful, otherwise 0

poly1_aic Akaike Information Criteria (AIC) for the Polynomial 1 model

poly1_cov Success indicator for the Polynomial 1 model covariance calculation; 1 if the Hessian matrix inversion is successful, otherwise 0

poly1_rme Root mean square erro for the Polynomial 1 model

poly1_a The y-scale parameter for the Polynomial 1 model

poly1_er Error term for the Polynomial 1 model

poly1_a_sd Standard deviation of the Polynomial 1 model y-scale parameter

poly1_er_sd Standard deviation of the Polynomial 1 model error term

poly1_top The maximal response on the resulting Polynomial 1 model fit

poly1_ac50 Concentration at 50% of the maximal response on the Polynomial 1 model fit

poly2_success Success indicator for the Polynomial 2 model; 1 if the optimization was successful, otherwise 0

poly2_aic Akaike Information Criteria (AIC) for the Polynomial 2 model

poly2_cov Success indicator for the Polynomial 2 model covariance calculation; 1 if the Hessian matrix inversion is successful, otherwise 0

poly2_rme Root mean square erro for the Polynomial 2 model

poly2_a The y-scale parameter for the Polynomial 2 model

poly2_b The x-scale parameter for the Polynomial 2 model

poly2_er Error term for the Polynomial 2 model

poly2_a_sd Standard deviation of the Polynomial 2 model y-scale parameter

poly2_b_sd Standard deviation of the Polynomial 2 model x-scale parameter

poly2_er_sd Standard deviation of the Polynomial 2 model error term

poly2_top The maximal response on the resulting Polynomial 2 model fit

poly2_ac50 Concentration at 50% of the maximal response on the Polynomial 2 model fit

pow_success Success indicator for the Power model; 1 if the optimization was successful, otherwise 0

pow_aic Akaike Information Criteria (AIC) for the Power model

pow_cov Success indicator for the Power model covariance calculation; 1 if the Hessian matrix inversion is successful, otherwise 0

pow_rme Root mean square erro for the Power model

pow_a The y-scale parameter for the Power model

pow_p The power parameter for the Power model

pow_er Error term for the Power model

pow_a_sd Standard deviation of the Power model y-scale parameter

pow_p_sd Standard deviation of the Power model power parameter

pow_er_sd Standard deviation of the Power model error term

pow_top The maximal response on the resulting Power model fit

pow_ac50 Concentration at 50% of the maximal response on the Power model fit

exp2_success Success indicator for the Exponential 2 model; 1 if the optimization was successful, otherwise 0

exp2_aic Akaike Information Criteria (AIC) for the Exponential 2 model

exp2_cov Success indicator for the Exponential 2 model covariance calculation; 1 if the Hessian matrix inversion is successful, otherwise 0

exp2_rme Root mean square erro for the Exponential 2 model

exp2_a The y-scale parameter for the Exponential 2 model

exp2_b The x-scale parameter for the Exponential 2 model

exp2_er Error term for the Exponential 2 model

exp2_a_sd Standard deviation of the Exponential 2 model y-scale parameter

exp2_b_sd Standard deviation of the Exponential 2 model x-scale parameter

exp2_er_sd Standard deviation of the Exponential 2 model error term

exp2_top The maximal response on the resulting Exponential 2 model fit

exp2_ac50 Concentration at 50% of the maximal response on the Exponential 2 model fit

exp3_success Success indicator for the Exponential 3 model; 1 if the optimization was successful, otherwise 0

exp3_aic Akaike Information Criteria (AIC) for the Exponential 3 model

exp3_cov Success indicator for the Exponential 3 model covariance calculation; 1 if the Hessian matrix inversion is successful, otherwise 0

exp3_rme Root mean square erro for the Exponential 3 model

exp3_a The y-scale parameter for the Exponential 3 model

exp3_b The x-scale parameter for the Exponential 3 model

exp3_p The power parameter for the Exponential 3 model

exp3_er Error term for the Exponential 3 model

exp3_a_sd Standard deviation of the Exponential 3 model y-scale parameter

exp3_b_sd Standard deviation of the Exponential 3 model x-scale parameter

exp3_p_sd Standard deviation of the Exponential 3 model power parameter

exp3_er_sd Standard deviation of the Exponential 3 model error term

exp3_top The maximal response on the resulting Exponential 3 model fit

exp3_ac50 Concentration at 50% of the maximal response on the Exponential 3 model fit

exp4_success Success indicator for the Exponential 4 model; 1 if the optimization was successful, otherwise 0

exp4_aic Akaike Information Criteria (AIC) for the Exponential 4 model

exp4_cov Success indicator for the Exponential 4 model covariance calculation; 1 if the Hessian matrix inversion is successful, otherwise 0

exp4_rme Root mean square erro for the Exponential 4 model

exp4_tp The top parameter indicating the maximal estimated response

exp4_ga The gain parameter for the Exponential 4 model, gain AC50

exp4_er Error term for the Exponential 4 model

exp4_tp_sd Standard deviation of the Exponential 4 model top parameter

exp4_ga_sd Standard deviation of the Exponential 4 model gain parameter

exp4_er_sd Standard deviation of the Exponential 4 model error term

exp4_top The maximal response on the resulting Exponential 4 model fit

exp4_ac50 Concentration at 50% of the maximal response on the Exponential 4 model fit

exp5_success Success indicator for the Exponential 5 model; 1 if the optimization was successful, otherwise 0

exp5_aic Akaike Information Criteria (AIC) for the Exponential 5 model

exp5_cov Success indicator for the Exponential 5 model covariance calculation; 1 if the Hessian matrix inversion is successful, otherwise 0

exp5_rme Root mean square erro for the Exponential 5 model

exp5_tp The top parameter indicating the maximal estimated response

exp5_ga The gain parameter for the Exponential 5 model, gain AC50

exp5_p The power parameter for the Exponential 5 model

exp5_er Error term for the Exponential 5 model

exp5_tp_sd Standard deviation of the Exponential 5 model top parameter

exp5_ga_sd Standard deviation of the Exponential 5 model gain parameter

exp5_p_sd Standard deviation of the Exponential 5 model power parameter

exp5_er_sd Standard deviation of the Exponential 5 model error term

exp5_top The maximal response on the resulting Exponential 5 model fit

exp5_ac50 Concentration at 50% of the maximal response on the Exponential 5 model fit

all_onesd Standard deviation of the baseline response for all models
all_bmed Median noise estimation of the baseline response for all models
resp_unit Response Units
conc_unit Concentration Units

6. **mc5** A data frame with 5 rows and 54 columns containing level 5 best curve-fit and hitcall data.

spid Sample ID
chid Unique chemical ID number for tcpl
casn Chemical Abstract Service(CAS) number
chnm Chemical name
dsstox_substance_id Chemical-specific DTXSID
code CAS number compressed into numeric string
aeid Assay Component Endpoint ID
aenm Assay endpoint name (i.e., assay_component_endpoint_name)
m5id Level 5 (mc5) ID
m4id Level 4 (mc4) ID
bmad The median absolute deviation of all treatment wells (default option) or blank wells
resp_max Maximum observed response
resp_min Minimum observed response
max_mean Maximum mean response
max_mean_conc Concentration of the maximum mean response
max_med Maximum median response
max_med_conc Concentration of the maximum median response
logc_max Maximum concentration on the log scale
logc_min Minimum concentration on the log scale
nconc The total number of concentration groups
npts Total number of observed responses (i.e. data points in the concentration series)
nrep Number of replicates in concentration groups
nmed_gtbl The number of median responses greater than 3BMAD
hitc Hitcall
modl Best model fit from tcplFit2 curve-fitting
fitc Fit category
coff Cutoff
top_over_cutoff Ratio of the top of the best model fit curve and the cutoff
rmse Root mean squared error
a The y-scale parameter for poly1, poly2, pow, exp2, or exp3 model
er Error term
bmr Benchmark response
bmdl Lower 95% confidence bound on the benchmark dose/concentration estimate
caikwt Akaike Information Criteria weight of constant model relative to the best model fit
mll Maximum log-likelihood of the best model fit
hitcall Continuous hitcall

ac50 Concentration where 50% of the maximal response occurs - if 'modl' is the Hill or Gain-loss model this is for the "gain" side of the response

top The maximal response on the best model curve fit - i.e. top of the curve fit

ac5 Concentration where 5% of the maximal response occurs

ac10 Concentration where 10% of the maximal response occurs

ac20 Concentration where 20% of the maximal response occurs

acc Concentration where the efficacy cutoff response occurs

ac1sd Concentration where one standard deviation of the background response occurs

bmd Benchmark response/concentration estimate - concentration where the benchmark response occurs

bmdu Upper 95% confidence bound on the benchmark dose/concentration estimate

tp The top curve parameter for the exp4, exp5, hill, or gnls model

ga The gain parameter for the hill or gnls model - gain AC50

p The power parameter for the pow, exp3, exp5, gnls, or hill model - for gnls this is the gain power parameter

q The loss power parameter for the gnls model

la The loss parameter for the gnls model, loss AC50

ac50_loss Concentration where 50% of the maximal response occurs - if 'modl' is the Hill or Gain-loss model this is for the "loss" side of the response

b The x-scale parameter for poly2, exp2, or exp3 model

resp_unit Response Units

conc_unit Concentration Units

`Method functions` *Functions for managing processing methods*

Description

These functions are used to manage which methods are used to process data. They include methods for assigning, clearing, and loading the assigned methods. Also, `tcplMthdList` lists the available methods.

Usage

```
tcplMthdAssign(lvl, id, mthd_id, ordr = NULL, type)

tcplMthdClear(lvl, id, mthd_id = NULL, type)

tcplMthdList(lvl, type = "mc")

tcplMthdLoad(lvl, id = NULL, type = "mc")
```

Arguments

`lvl`	Integer of length 1, the method level
`id`	Integer, the assay component or assay endpoint id(s)
`mthd_id`	Integer, the method id(s)
`ordr`	Integer, the order in which to execute the analysis methods, must be the same length as mthd_id, does not apply to levels 5 or 6
`type`	Character of length 1, the data type, "sc" or "mc"

Details

`tcplMthdLoad` loads the assigned methods for the given level and ID(s). Similarly, `tcplMthdList` displays the available methods for the given level. These two functions do not make any changes to the database.

Unlike the `-Load` and `-List` functions, the `-Assign` and `-Clear` functions alter the database and trigger a delete cascade. `tcplMthdAssign` assigns methods to the given ID(s), and `tcplMthdClear` removes methods. In addition to the method ID ('mthd_id'), assigning methods at some levels require an order ('ordr'). The 'ordr' parameter is necessary to allow progression of methods at level one for single-concentration processing, and levels two and three for multiple-concentration processing. More information about method assignments and the delete cascade are available in the package vignette.

Examples

```
## Store the current config settings, so they can be reloaded at the end
## of the examples
conf_store <- tcplConfList()
tcplConfDefault()

## tcplListMthd allows the user to display the available methods for
## a given level and data type
head(tcplMthdList(lvl = 2, type = "mc"))

## tcplLoadMthd shows which methods are assigned for the given ID, level,
## and data type. Here we will show how to register, load, and clear methods
## using an acid not in the example database. Note: There is no check for
## whether an ID exists before assigning/clearing methods.
tcplMthdLoad(lvl = 2, id = 55, type = "mc")

## Not run:
## ACID 55 does not have any methods. Assign methods from the list above.
tcplMthdAssign(lvl = 2,
               id = 55,
               mthd_id = c(3, 4, 2),
               ordr = 1:3,
               type = "mc")
## Method assignment can be done for multiple assays, too.
tcplMthdAssign(lvl = 2,
               id = 53:54,
               mthd_id = c(3, 4, 2),
                ordr = 1:3,
                type = "mc")

## Cleanup example method assigments
tcplMthdClear(lvl = 2, id = 53:55, type = "mc")

## End(Not run)
## Reset configuration
options(conf_store)
```

`Models` *Model objective functions*

Description

These functions take in the dose-response data and the model parameters, and return a likelihood value. They are intended to be optimized using `constrOptim` in the `tcplFit` function.

Usage

```
tcplObjCnst(p, resp)

tcplObjGnls(p, lconc, resp)

tcplObjHill(p, lconc, resp)

tcplObjCnst(p, resp)

tcplObjGnls(p, lconc, resp)

tcplObjHill(p, lconc, resp)
```

Arguments

`p`	Numeric, the parameter values. See details for more information.
`resp`	Numeric, the response values
`lconc`	Numeric, the log10 concentration values

Details

These functions produce an estimated value based on the model and given parameters for each observation. Those estimated values are then used with the observed values and a scale term to calculate the log-likelihood.

Let $t(z, \nu)$ be the Student's t-distribution with ν degrees of freedom, y_i be the observed response at the i^{th} observation, and μ_i be the estimated response at the i^{th} observation. We calculate z_i as:

$$z_i = \frac{y_i - \mu_i}{e^{\sigma}}$$

where σ is the scale term. Then the log-likelihood is:

$$\sum_{i=1}^{n}[ln(t(z_i, 4)) - \sigma]$$

Where n is the number of observations.

Value

The log-likelihood.

Constant Model (cnst)

`tcplObjCnst` calculates the likelyhood for a constant model at 0. The only parameter passed to `tcplObjCnst` by `p` is the scale term σ. The constant model value μ_i for the i^{th} observation is given by:

$$\mu_i = 0$$

`tcplObjCnst` calculates the likelyhood for a constant model at 0. The only parameter passed to `tcplObjCnst` by `p` is the scale term σ. The constant model value μ_i for the i^{th} observation is given by:

$$\mu_i = 0$$

Gain-Loss Model (gnls)

`tcplObjGnls` calculates the likelyhood for a 5 parameter model as the product of two Hill models with the same top and both bottoms equal to 0. The parameters passed to `tcplObjGnls` by `p` are (in order) top (tp), gain log AC50 (ga), gain hill coefficient (gw), loss log AC50 la, loss hill coefficient lw, and the scale term (σ). The gain-loss model value μ_i for the i^{th} observation is given by:

$$g_i = \frac{1}{1 + 10^{(ga - x_i)gw}}$$

$$l_i = \frac{1}{1 + 10^{(x_i - la)lw}}$$

$$\mu_i = tp(g_i)(l_i)$$

where x_i is the log concentration for the i^{th} observation.

`tcplObjGnls` calculates the likelyhood for a 5 parameter model as the product of two Hill models with the same top and both bottoms equal to 0. The parameters passed to `tcplObjGnls` by `p` are (in order) top (tp), gain log AC50 (ga), gain hill coefficient (gw), loss log AC50 la, loss hill coefficient lw, and the scale term (σ). The gain-loss model value μ_i for the i^{th} observation is given by:

$$g_i = \frac{1}{1 + 10^{(ga - x_i)gw}}$$

$$l_i = \frac{1}{1 + 10^{(x_i - la)lw}}$$

$$\mu_i = tp(g_i)(l_i)$$

where x_i is the log concentration for the i^{th} observation.

Hill Model (hill)

`tcplObjHill` calculates the likelyhood for a 3 parameter Hill model with the bottom equal to 0. The parameters passed to `tcplObjHill` by `p` are (in order) top (tp), log AC50 (ga), hill coefficient (gw), and the scale term (σ). The hill model value μ_i for the i^{th} observation is given by:

$$\mu_i = \frac{tp}{1 + 10^{(ga - x_i)gw}}$$

where x_i is the log concentration for the i^{th} observation.

`tcplObjHill` calculates the likelyhood for a 3 parameter Hill model with the bottom equal to 0. The parameters passed to `tcplObjHill` by `p` are (in order) top (tp), log AC50 (ga), hill coefficient (gw), and the scale term (σ). The hill model value μ_i for the i^{th} observation is given by:

$$\mu_i = \frac{tp}{1 + 10^{(ga - x_i)gw}}$$

where x_i is the log concentration for the i^{th} observation.

`Query functions` *Wrappers for sending queries and fetching results*

Description

These functions send a query to the given database, and are the access point for all tcpl functions that query or update the tcpl database.

Usage

```
tcplQuery(
  query,
  db = getOption("TCPL_DB"),
  drvr = getOption("TCPL_DRVR"),
  tbl = NULL
)

tcplSendQuery(
  query,
  db = getOption("TCPL_DB"),
  drvr = getOption("TCPL_DRVR"),
  tbl = NULL,
  delete = F
)
```

Arguments

`query`	Character of length 1, the query string
`db`	Character of length 1, the name of the tcpl database
`drvr`	Character of length 1, which database driver to use
`tbl`	Tables to be read queried
`delete`	Logical of length 1, execute delete on queried table

Details

Currently, the tcpl package only supports the "MySQL" and "tcplLite" database drivers.

`tcplQuery` returns a data.table object with the query results. `tcplSendQuery` sends a query, but does not fetch any results, and returns 'TRUE' or the error message given by the database.

Examples

```
## Store the current config settings, so they can be reloaded at the end
## of the examples
conf_store <- tcplConfList()
TCPLlite <- file.path(system.file(package = "tcpl"), "example")
tcplConf(db = TCPLlite, user = NA, host = NA, drvr = "tcplLite")

tcplQuery("SELECT 'Hello World';")

## When using tcplLite, name of table must be passed into tcplQuery
if (conf_store$TCPL_DRVR == 'MySQL') {
  tcplQuery("SELECT * FROM assay;")
} else {
  tcplQuery("SELECT * FROM assay;", tbl='assay')
}

## Reset configuration
options(conf_store)
```

`Register/update annotation` *Functions for registering & updating annotation information*

Description

These functions are used to register and update the chemical and assay annotation information.

Usage

```
tcplRegister(what, flds)

tcplUpdate(what, id, flds)
```

Arguments

`what`	Character of length 1, the name of the ID to register or update
`flds`	Named list, the other fields and their values
`id`	Integer, the ID value(s) to update

Details

These functions are used to populate the tcpl database with the necessary annotation information to complete the processing. As shown in the package vignette, the package requires some information about the samples and assays before data can be loaded into the tcpl database.

Depending on what is being registered, different information is required. The following table lists the fields that can be registered/updated by these functions, and the minimal fields required for registering a new ID. (The database table affected is in parentheses.)

- asid (assay_source): assay_source_name
- aid (assay): asid, assay_name, assay_footprint
- acid (assay_component): aid, assay_component_name
- aeid (assay_component_endpoint): acid, assay_component_endpoint_name, normalized_data_type
- acsn (assay_component_map): acid, acsn
- spid (sample): spid, chid
- chid (chemical): chid, casn
- clib (chemical_library): chid, clib

Note: The functions accept the abbreviated forms of the names, ie. "aenm" rather than the full "assay_component_endpoint_name." More information about the registration process and all of the fields is available in the vignette.

Examples

```
## Not run:
## Store the current config settings, so they can be reloaded at the end
## of the examples
conf_store <- tcplConfList()
tcplConfDefault()

## Load current ASID information
tcplLoadAsid()

## Register a new assay source
tcplRegister(what = "asid", flds = list(asnm = "example_asid"))

## Show the newly registered ASID
tcplLoadAsid(add.fld = "assay_source_desc")

## Notice that the newly created ASID does not have an assay_source_desc.
## The field could have been defined during the registration process, but
## can also be updated using tcplUpdate
i1 <- tcplLoadAsid()[asnm == "example_asid", asid]
tcplUpdate(what = "asid",
           id = i1,
           flds = list(assay_source_desc = "example asid description"))
tcplLoadAsid(add.fld = "assay_source_desc")
```

```
## Remove the created ASID. Note: Manually deleting primary keys can cause
## serious database problems and should not generally be done.

## If using the tcplLite DRVR, must specify table name
if (conf_store$TCPL_DRVR == 'MySQL') {
  tcplSendQuery(paste0("DELETE FROM assay_source WHERE asid = ", i1, ";"))
} else {
  qy <- paste0("SELECT * FROM assay_source WHERE NOT asid = ", i1, ";")
  tcplSendQuery(qy, tbl='assay_source', delete=TRUE)
}

## Reset configuration
options(conf_store)

## End(Not run)
```

registerMthd *Add a new analysis method*

Description

`registerMthd` registers a new analysis method to the tcpl databases.

Usage

```
registerMthd(lvl, mthd, desc, nddr = 0L, type)
```

Arguments

`lvl`	Integer of length 1, the level for the analysis method
`mthd`	Character, the name of the method
`desc`	Character, same length as mthd, the method description
`nddr`	Integer, 0 or 1, 1 if the method requires loading the dose- response data
`type`	Character of length 1, the data type, "sc" or "mc"

Details

'mthd' must match a corresponding function name in the functions that load the methods, ie. `mc2_mthds`. 'nddr' only applies to level 6 methods.

`sc1` *Perform level 1 single-concentration processing*

Description

`sc1` loads level 0 data from the tcpl database for the given id and performs level 1 single-concentration processing. The processed data is then loaded into the sc1 table and all subsequent data is deleted with `tcplCascade`. See details for more information.

The individual processing functions are no longer exported, as it is typically more convenient and suggested to use the `tcplRun` wrapper function.

Usage

```
sc1(ac, wr = FALSE)
```

Arguments

`ac`	Integer of length 1, assay component id (acid) for processing.
`wr`	Logical, whether the processed data should be written to the tcpl database

Details

Level 1 single-concentration processing includes mapping assay component to assay endpoint, duplicating the data when the assay component has multiple assay endpoints, and any normalization of the data. Data normalization based on methods listed in sc1_aeid and sc1_methods tables.

Value

A boolean of length 1, indicating the success of the processing, or when 'wr' is FALSE, a list where the first element is a boolean indicating the success of processing and the second element is a data.table containing the processed data

See Also

`Method functions`, `SC1_Methods`

Other single-concentration: `sc2()`

`SC1_Methods` *List of level 1 single-concentration normalization functions*

Description

`sc1_mthds` returns a list of functions to be used during level 1 single-concentration processing.

Usage

```
sc1_mthds()
```

Details

The functions contained in the list returned by `sc1_mthds` return a list of expressions to be executed in the `sc2` (not exported) function environment. The functions are described here for reference purposes, The `sc1_mthds` function is not exported, nor is it intended for use.

All available methods are described in the Available Methods section, listed by the function/method name.

Value

A list functions

Available Methods

The methods are broken into three types, based on what fields they define. Different methods are used to define "bval" (the baseline value), "pval" (the positive control value), and "resp" (the final response value).

Although it does not say so specifically in each description, all methods are applied by acid.

More information about the level 3 single-concentration processing is available in the package vignette, "Data_processing."

bval Methods:

bval.apid.nwlls.med Calculate the baseline value (bval) as the plate-wise median, by assay plate ID (apid), of the raw values (rval) for neutral control wells (wllt = n).

bval.apid.twlls.med Calculate the baseline value (bval) as the plate-wise median, by assay plate ID (apid), of the raw values (rval) for test compound wells (wllt = t).

bval.apid.tn.med Calculate the baseline value (bval) as the plate-wise median, by assay plate ID (apid), of the raw values (rval) for test compound wells (wllt = t) and neutral control wells (wllt = n).

pval Methods:

pval.apid.pwlls.med Calculate the positive control value (pval) as the plate-wise median, by assay plate ID (apid), of the raw values (rval) for single-concentration gain-of-signal positive control wells (wllt = p).

pval.apid.mwlls.med Calculate the positive control value (pval) as the plate-wise median, by assay plate ID (apid), of the raw values (rval) for multiple-concentration loss-of-signal negative control wells (wllt = m).

pval.apid.medpcbyconc.max Calculate the positive control value (pval) as the plate-wise maximum, by assay plate ID (apid), of the medians of the raw values (rval) for gain-of-signal single- or multiple-concentration positive control wells (wllt = p or c) by apid, well type, and concentration.

pval.apid.medpcbyconc.min Calculate the positive control value (pval) as the plate-wise minimum, by assay plate ID (apid), of the medians of the raw values (rval) for gain-of-signal single- or multiple-concentration positive control wells (wllt = p or c) by apid, well type, and concentration.

pval.apid.medncbyconc.min Calculate the positive control value (pval) as the plate-wise minimum, by assay plate ID (apid), of the medians of the raw values (rval) for gain-of-signal single- or multiple-concentration negative control wells (wllt = m or o) by apid, well type, and concentration.

pval.zero Set the positive control value (pval) to 0; pval = 0.

pval.apid.or.aeid.pwlls.med Calculate the positive control value (pval) as the plate-wise median, by assay plate ID (apid), of the raw values (rval) for single-concentration gain-of-signal positive control wells (wllt = p). For plates without p wells, set the pval as the median pval calculated from all plates.

resp Methods:

resp.pc Calculate the normalized response (resp) as a percent of control, i.e. the ratio of the difference between the raw (rval) and baseline (bval) values divided by the difference between positive control (pval) and baseline (bval) values multiplied by 100; $resp = (rval - bval)/(pval - bval) * 100$.

resp.fc Calculate the normalized response (resp) as fold change, i.e. the ratio of the raw (rval) and baseline (bval) values; $resp = rval/bval$.

resp.logfc Calculate the normalized response (resp) as the fold change of logged, i.e. the difference between raw (rval) and baseline (bval) log-scale values.

resp.log2 Transform the response values to log-scale (base 2).

resp.multneg1 Multiply the normalized response value (resp) by -1; $-1 * resp$.

none Use raw value (rval) as is. This may be necessary for additional endpoint-specific adjustments, or where no additional sc1 methods are needed.

resp.incr.zerocenter.fc Calculate the normalized response (resp) as a zero center fold change, i.e. the ratio of the raw (rval) and baseline (bval) values minus 1; $resp = rval/bval - 1$. Typically used for increasing responses.

Note

This function is not exported and is not intended to be used by the user.

See Also

`sc1`, `Method functions` to query what methods get applied to each acid `sc2` *Perform level 2 single-concentration processing*

Description

`sc2` loads level 1 data from the tcpl database for the given id and performs level 2 single-concentration processing. The processed data is then loaded into the sc2 table and all subsequent data is deleted with `tcplCascade`. See details for more information.

The individual processing functions are no longer exported, as it is typically more convenient and suggested to use the `tcplRun` wrapper function.

Usage

```
sc2(ae, wr = FALSE)
```

Arguments

`ae`	Integer of length 1, assay endpoint id (aeid) for processing.
`wr`	Logical, whether the processed data should be written to the tcpl database

Details

Level 2 single-concentration processing defines the bmad value, and uses the activity cutoff methods from sc2_aeid and sc2_methods to make an activity call.

Value

A boolean of length 1, indicating the success of the processing, or when 'wr' is FALSE, a list where the first element is a boolean indicating the success of processing and the second element is a data.table containing the processed data

See Also

`Method functions`, `SC2_Methods`

Other single-concentration: `sc1()`

`SC2_Methods` *List of level 2 single-concentration hit-call functions*

Description

`sc2_mthds` returns a list of functions to be used during level 2 single-concentration processing.

Usage

```
sc2_mthds()
```

Details

The functions contained in the list returned by `sc2_mthds` return a list of expressions to be executed in the `sc2` (not exported) function environment. The functions are described here for reference purposes, The `sc2_mthds` function is not exported, nor is it intended for use.

All available methods are described in the Available Methods section, listed by the function/method name.

Value

A list functions

Available Methods

The methods are broken down into four categories based on the type of cutoff they assign. Different methods are used to define cutoffs for "bmad" (baseline median absolute value), "pc" (percent of control), "pc or bmad", "log" ($\log_2$ or $\log_{10}$), and "other" (uncategorized methods).

All methods are applied by aeid.

Although there are method exceptions (notably within the “other” category), only highest calculated cutoff value based on assigned methods will be selected for hitcalling. Therefore, only the largest cutoff method per method type should be assigned.

More information about the level 2 single-concentration processing is available in the package vignette, "Data_processing."

BMAD Methods:

bmad1 Add a cutoff value of 1 multiplied by baseline median absolute deviation (bmad). By default, bmad is calculated using test compound wells (wllt = t) for the endpoint.

bmad1.5 Add a cutoff value of 1.5 multiplied by the baseline median absolute deviation (bmad). By default, bmad is calculated using test compound wells (wllt = t) for the endpoint.

bmad2 Add a cutoff value of 2 multiplied by the baseline median absolute deviation (bmad). By default, bmad is calculated using test compound wells (wllt = t) for the endpoint.

bmad3 Add a cutoff value of 3 multiplied by the baseline median absolute deviation (bmad). By default, bmad is calculated using test compound wells (wllt = t) for the endpoint.

bmad5 Add a cutoff value of 5 multiplied the baseline median absolute deviation (bmad). By default, bmad is calculated using test compound wells (wllt = t) for the endpoint.

bmad6 Add a cutoff value of 6 multiplied by the baseline median absolute deviation (bmad). By default, bmad is calculated using test compound wells (wllt = t) for the endpoint.

bmad10 Add a cutoff value of 10 multiplied by the baseline median absolute deviation (bmad). By default, bmad is calculated using test compound wells (wllt = t) for the endpoint.

Percent of Control Methods:

pc0.88 Add a cutoff value of 0.88. Typically for percent of control data.

pc20 Add a cutoff value of 20. Typically for percent of control data.

pc25 Add a cutoff value of 25. Typically for percent of control data.

pc30 Add a cutoff value of 30. Typically for percent of control data.

Percent of Control or BMAD Methods:

pc30orbmad3 Add a cutoff value of either 30 or 3 multiplied by the baseline median absolute deviation (bmad), whichever is less. By default, bmad is calculated using test compound wells (wllt = t) for the endpoint.

Log Methods: Log Base 2

log2_0.76 Add a cutoff value of 0.76 for log2-transformed data. This was a custom threshold value set for endpoint id 1690 (formerly aeid 1691).

log2_1.2 Add a cutoff value of $log_2 1.2$. Typically for fold change data.

log2_1.5 Add a cutoff value of $log_2 1.5$. Typically for fold change data.

Log Base 10

log10_1.2 Add a cutoff value of $log_{10} 1.2$. Typically for fold change data.

Other Methods:

ow_bmad_nwells Overwrite the default baseline median absolute value (bmad) with a bmad calculated using neutral control wells (wllt = n).

ow_bidirectional_false Overwrite the max_med and max_tmp values, which were calculated using absolute value, to a calculation not using absolute value for non-bidirectional data.

Note

This function is not exported and is not intended to be used by the user.

See Also

`sc2`, `Method functions` to query what methods get applied to each acid

scdat — *A subset of ToxCast data showing changes in transcription factor activity for multiple targets.*

Description

The example dataset is used to illustrate how the user can pipeline single-concentration data from chemical screening using tcplLite.

Usage

```
scdat
```

Format

A data frame with 320 rows and 10 variables:

spid sample ID

apid assay plate ID

rowi well-plate row number (N/A)

coli well-plate column number (N/A)

wllt well type (N/A)

wllq well quality (N/A)

conc concentration in micromolar

rval raw assay component readout value

srcf source file containing the data

acsn assay component source name

Source

ToxCast database

sc_vignette — *List with single-concentration data for the vignette*

Description

This dataset is a list with 3 data.tables (sc0,sc1,sc2).

Usage

```
sc_vignette
```

Format

1. **sc0** A data frame with 10 rows and 18 columns containing level 0 formatted raw data.

   spid Sample ID

   chid Unique chemical ID number for tcpl

   casn Chemical Abstract Service(CAS) number

   chnm Chemical name

   dsstox_substance_id Chemical-specific DTXSID

   code CAS number compressed into numeric string

   acid Assay Component ID

   acnm Assay Component Name

   s0id Level 0 (sc0) ID

   apid Assay plate ID

   rowi Row Index

   coli Column Index

   wllt Well Type

   wllq Well Quality (0 or 1)

   conc Concentration in micromolar

   rval Raw assay component readout value

   srcf Source file containing the raw data

   conc_unit Concentration Units

2. **sc1** A data frame with 10 rows and 20 columns containing level 1 normalized data.

   spid Sample ID

   chid Unique chemical ID number for tcpl

   casn Chemical Abstract Service(CAS) number

   chnm Chemical name

   dsstox_substance_id Chemical-specific DTXSID

   code CAS number compressed into numeric string

   aeid Assay Component Endpoint ID

   aenm Assay endpoint name (i.e., assay_component_endpoint_name)

   acid Assay Component ID

   acnm Assay Component Name

   s0id Level 0 (sc0) ID

   s1id Level 1 (sc1) ID

   apid Assay plate ID

   rowi Row Index

   coli Column Index

   wllt Well Type

   logc Log base 10 concentration

   resp Normalized response value

   resp_unit Response Units

   conc_unit Concentration Units

3. **sc2** A data frame with 10 rows and 15 columns containing level 2 efficacy/hit designation data.

spid	Sample ID
chid	Unique chemical ID number for tcpl
casn	Chemical Abstract Service(CAS) number
chnm	Chemical name
dsstox_substance_id	Chemical-specific DTXSID
code	CAS number compressed into numeric string
aeid	Assay Component Endpoint ID
aenm	Assay endpoint name (i.e., assay_component_endpoint_name)
s2id	Level 2 (sc2) ID
bmad	The median absolute deviation of all treatment wells (default option) or blank wells
max_med	Maximum median response
hitc	Hitcall
coff	Cutoff
resp_unit	Response Units
conc_unit	Concentration Units

sink.reset *Reset all sinks*

Description

`sink.reset` resets all sinks and returns all output to the console.

Usage

```
sink.reset()
```

Details

`sink.reset` identifies all sinks with `sink.number` then returns all output and messages back to the console.

See Also

`sink`, `sink.number`

Other tcpl abbreviations: `is.odd()`, `lu()`, `lw()`

tcplAddModel — *Draw a tcpl Model onto an existing plot*

Description

tcplAddModel draws a a line for one of the tcpl Models (see Models for more information) onto an existing plot.

Usage

```
tcplAddModel(pars, modl = NULL, adj = NULL, ...)
```

Arguments

pars	List of parameters from level 4 or 5 output
modl	Character of length 1, the model to plot: 'cnst,' 'hill,' or 'gnls'
adj	Numeric of length 1, an adjustment factor, see details for more information
...	Additional arguments passed to curve

Details

tcplAddModel draws the model line assuming the x-axis represents log base 10 concentration.

If modl is NULL, the function checks pars$modl and will return an error if pars$modl is also NULL.

adj is intended to scale the models, so that models with different response units can be visualized on a single plot. The recommended value for adl is 1/(3*bmad) for level 4 data and 1/coff for level 5 data. If adj is NULL the function will check pars$adj and set adj to 1 if pars$adj is also NULL.

See Also

Models, tcplPlotFits

Examples

```
## Create some dummy data to plot
logc <- 1:10
r1 <- sapply(logc, tcplHillVal, ga = 5, tp = 50, gw = 0.5)
r2 <- log2(sapply(logc, tcplHillVal, ga = 4, tp = 30, gw = 0.5))
p1 <- tcplFit(logc = logc, resp = r1, bmad = 10)
p2 <- tcplFit(logc = logc, resp = r2, bmad = log2(1.5))

## In the dummy data above, the two plots are on very different scales
plot(r1 ~ logc, pch = 16, ylab = "raw response")
tcplAddModel(pars = p1, modl = "hill")
points(r2 ~ logc)
tcplAddModel(pars = p2, modl = "hill", lty = "dashed")
```

```
## To visualize the two curves on the same plot for comparison, we can
## scale the values to the bmad, such that a scaled response of 1 will equal
## the bmad for each curve.
plot(r1/10 ~ logc, pch = 16, ylab = "scaled response")
tcplAddModel(pars = p1, modl = "hill", adj = 1/10)
points(r2/log2(5) ~ logc)
tcplAddModel(pars = p2, modl = "hill", adj = 1/log2(5), lty = "dashed")
```

`tcplAICProb` *Calculate the AIC probabilities*

Description

`tcplAICProb` Calculates the probability that the model best represents the data based on the AIC value for each model.

Usage

```
tcplAICProb(...)
```

Arguments

`...`	Numeric vectors of AIC values

Details

The function takes vectors of AIC values. Each vector represents the model AIC values for multiple observation sets. Each vector must contain the same number and order of observation sets. The calculation assumes every possible model is accounted for, and the results should be interpreted accordingly.

Value

A vector of probability values for each model given, as a list.

See Also

`tcplFit`, `AIC` for more information about AIC values.

Examples

```
## Returns the probability for each model, given models with AIC values
## ranging from 80 to 100
tcplAICProb(80, 85, 90, 95, 100)

## Also works for vectors
m1 <- c(95, 195, 300) ## model 1 for three different observations
```

```
m2 <- c(100, 200, 295) ## model 2 for three different observations
tcplAICProb(m1, m2)
```

---

tcplAppend *Append rows to a table*

Description

`tcplAppend` takes a data.table (dat) and appends the data.table into a database table.

Usage

```
tcplAppend(dat, tbl, db, lvl = NULL)
```

Arguments

`dat`	data.table, the data to append to a table
`tbl`	Character of length 1, the table to append to
`db`	Character of length 1, the database containing `tbl`
`lvl`	Usually Integer to indicate what level to auto-increment

Note

This function is not exported and not intended to be used by the user.

---

tcplCascade *Do a cascading delete on tcpl screening data*

Description

`tcplCascade` deletes the data for the given id(s) starting at the processing level given. The delete will cascade through all subsequent tables.

Usage

```
tcplCascade(lvl, type, id)
```

Arguments

`lvl`	Integer of length 1, the first level to delete from
`type`	Character of length 1, the data type, "sc" or "mc"
`id`	Integer, the id(s) to delete. See details for more information.

Details

The data type can be either 'mc' for multiple concentration data, or 'sc' for single concentration data. Multiple concentration data will be loaded into the level tables, whereas the single concentration will be loaded into the single tables.

If lvl is less than 3, id is interpreted as acid(s) and if lvl is greater than or equal to 3, id is interpreted as aeid(s).

Note

This function is not exported and not intended to be used by the user.

`tcplCode2CASN` *Convert chemical code to CAS Registry Number*

Description

`tcplCode2CASN` takes a code and converts it CAS Registry Number.

Usage

```
tcplCode2CASN(code)
```

Arguments

`code`	Character of length 1, a chemical code

Details

The function checks for the validity of the CAS Registry Number. Also, the ToxCast data includes chemicals for which there is no CASRN. The convention for these chemicals is to give them a CASRN as NOCAS_chid; the code for these compounds is CNOCASchid. The function handles the NOCAS compounds as they are stored in the database, as shown in the example below.

Value

A CAS Registry Number.

Examples

```
tcplCode2CASN("C80057")
tcplCode2CASN("C09812420") ## Invalid CASRN will give a warning
tcplCode2CASN("CNOCAS0015") ## The underscore is reinserted for NOCAS codes
```

tcplCytoPt — *Calculate the cytotoxicity point based on the "burst" endpoints*

Description

`tcplCytoPt` calculates the cytotoxicity point and average cytotoxicity distribution based on the activity in the "burst" assay endpoints.

Usage

```
tcplCytoPt(
  chid = NULL,
  aeid = NULL,
  flag = TRUE,
  min.test = TRUE,
  default.pt = 3
)
```

Arguments

`chid`	Integer, chemical ID values to subset on
`aeid`	Integer, assay endpoint ID values to override the "burst assay" definitions
`flag`	Integer, mc6_mthd_id values to be passed to `tcplSubsetChid`
`min.test`	Integer or Boolean, the number of tested assay endpoints required for a chemical to be used in calculating the "global MAD."
`default.pt`	Numeric of length 1, the default cytotoxicity point value

Details

`tcplCytoPt` provides estimates for chemical-specific cytotoxicity distributions (more information available in the vignette.) Before calculating the cytotoxicity distributions, the level 5 data is subsetted by the `tcplSubsetChid` function.

The 'chid' parameter specifies a subset of chemicals to use in the calculations, given by chemical ID (chid). The 'aeid' parameter specifies which assays to use in calculating the cytotoxicity point and distribution. By default `tcplCytoPt` will use all available chemicals and the assay endpoints defined by the 'burst_assay' field in the "assay_component_endpoint" table. The examples show how to identify the "burst" endpoints.

`tcplCytoPt` returns the cytotoxicity point (the AC50 values of the active "burst" endpoints), the corresponding MAD, and the global MAD (median of the calculated MAD values). Not every chemical must be tested in every "burst" endpoint. The 'min.test' parameter allows the user to specify a minimum number of tested assay endpoints as a requirement for MAD values to be included in the global MAD calculation. For example, suppose the user supplies 10 "burst" assays. The user can choose to require a chemical to be tested in at least 5 of those assays for it's MAD value to be included in the global MAD calculation. Having chemicals with many less "burst" endpoints tested may inflate or deflate the global MAD calculation. By default (values of `TRUE` or `NULL`), `tcplCytoPt` requires a chemical to be tested in at least 80% of the given "burst" assays. The user can also provide 'min.test' values of `FALSE` (indicating to include all MAD values), or a number (indicating a specific number of endpoints).

Chemicals without at least 2 active "burst" assays do not have a MAD value, and the cytotoxicity point is defined by the 'default.pt' parameter. The default value for 'default.pt' is 3.

The resulting data.table has the following fields:

1. "chid" – The chemical ID.
2. "code" – The chemical code.
3. "chnm" – The chemical name.
4. "casn" – The chemical CASRN.
5. "med" – The median of the "burst" endpoint log(AC50)
6. "mad" – The MAD of the "burst" endpoint log(AC50) values.
7. "ntst" – The number of "burst" endpoints tested.
8. "nhit" – The number of active "burst" endpoints.
9. "used_in_global_mad_calc" – TRUE/FALSE, whether the mad value was used in the global MAD calculation.
10. "global_mad" – The median of the "mad" values where "used_in_global_mad_calc" is TRUE.
11. "cyto_pt" – The cytotoxicity point, or the value in "med" when "nhit" is at least 2.
12. "cyto_pt_um" – 10^{cyto_pt}
13. "lower_bnd_um" – $10^{cyto\_pt - 3global\_mad}$

Value

A data.table with the cytotoxicity distribution for each chemical. The definition of the field names are listed under "details."

Examples

```
## Store the current config settings, so they can be reloaded at the end
## of the examples
conf_store <- tcplConfList()
tcplConfDefault()

## Can only calculate the cytotox burst if using the MySQL database and
## TCPL_DRVR == 'MySQL'

if (getOption("TCPL_DRVR") == "MySQL") {

## Load the "burst" endpoints -- none are defined in the example dataset
tcplLoadAeid(fld = "burst_assay", val = 1)

## Calculate the cytotoxicity distributions using both example endpoints
tcplCytoPt(aeid = 1:2)

## The above example does not calculate a global MAD, because no chemical
## hit both endpoints. (This makes sense, because both endpoints are
## derived from one component, where one endpoint is acitivity in the
## up direction, and the other is activity in the down direction.)
## Note, the cyto_pt is also 3 for all chemicals, because the function
## requires at least two endpoints to calculate a cytotoxicity point. If
## the user wishes to use one assay, this function is not necessary.

## Changing 'default.pt' will change cyto_pt in the resulting data.table
tcplCytoPt(aeid = 1:2, default.pt = 6)
}

## Reset configuration
options(conf_store)
```

`tcpldbStats` *Get summary statistics for the database*

Description

`tcpldbStats` takes a string(type) and an optional parameter(val) to return the summary statistics on the entire tcplLite database When type = "all" the val is ignored. the function returns the number of distinct spid and aeids in the database at each level When type = "aeid", the val parameter has to be a valid aeid in the database. The function returns a table consisting of the number of distinct spids at each level of processing for the aeid given in 'val' When type = "spid", the val parameter has to be a valid spid in the database. The function returns a table consisting of the number of distinct aeids at each level of processing for the given spid in 'val'

Usage

```
tcpldbStats(type = "all", val = NULL)
```

Arguments

`type`	String either "all", "aeid" or "spid"
`val`	integer if type = "aeid" , string if type = "spid"

`tcplDefine` *Load data dictionary descriptions*

Description

`tcplDefine` queries the tcpl databases and returns field descriptions from the data dictionary.

Usage

```
tcplDefine(val = NULL)
```

Arguments

`val`	The values to query on. Can be any combination of table names (to return all of its field descriptions) and field names

Details

Short descriptions of fields for different tables are stored in a data dictionary. Query by table name to retrieve descriptions of each field in the given table, and/or query by field name to retrieve descriptions on every field with the given name, regardless of which table they belong to.

Value

A data.table with the data dictionary information for the given parameters.

Examples

```
## Store the current config settings, so they can be reloaded at the end
## of the examples
conf_store <- tcplConfList()
tcplConf(drvr = "example")

## Passing no parameters returns all of the fields described in the data
## dictionary
tcplDefine()

## Specifying table names of 'chemical' and 'sample' yields all of the
## fields from the 'chemical' and 'sample' tables
tcplDefine(c("chemical", "sample"))

## Specifying a field of 'wllt' yields all of the fields from any table that
## contains 'wllt' as a field
tcplDefine("wllt")

## Specifying a combination of table and field names results in all of the
## fields which are contained in the given tables and all of the given fields
## found in any table
tcplDefine(c("chemical", "spid", "wllt"))

## Reset configuration
options(conf_store)
```

`tcplDelete` *Delete rows from tcpl databases*

Description

`tcplDelete` deletes rows from the given table and database.

Usage

```
tcplDelete(tbl, fld, val, db)
```

Arguments

`tbl`	Character, length 1, the table to delete from
`fld`	Character, the field(s) to query on
`val`	List, vectors of values for each field to query on. Must be in the same order as 'fld'.
`db`	Character, the database containing the table

Note

This function is not exported and not intended to be used by the user.

See Also

`tcplSendQuery`

`tcplFit` *Fit the data with the constant, hill, and gain-loss models*

Description

`tcplFit` fits the constant, hill, and gain-loss models to the given data and returns some summary statistics and the fit parameters in a list.

Usage

```
tcplFit(
  logc,
  resp,
  bmad,
  force.fit = FALSE,
  bidirectional = FALSE,
  verbose = FALSE,
  ...
)
```

Arguments

`logc`	Numeric, log concentration values
`resp`	Numeric, normalized response values
`bmad`	Numeric, the baseline median absolute deviation for the entire assay
`force.fit`	Logical, TRUE indicates to attempt fitting every concentration series

`bidirectional`	Boolean If TRUE, bidirectional negative data before fitting (default=FALSE) The original version of the code required the data to start at small values and rise, so that negative curves had to be bidirectionalped outside the function, and TOP was always positive. Setting bidirectional to TRUE allows both rising and falling curves
`verbose`	Boolean If TRUE print warning messages
`...`	Any other data to be included in list output.

Details

when at least one median value is greater than 3*bmad.

Value

List of summary values and fit parameters for the given data.

See Also

`tcplObjCnst`, `tcplObjHill`, `tcplObjGnls`, `constrOptim`

Examples

```
logc <- 1:10
resp <- sapply(1:10, tcplHillVal, ga = 5, tp = 50, gw = 0.5)
params <- tcplFit(logc = logc, resp = resp, bmad = 10)
plot(resp ~ logc)
tcplAddModel(pars = params, modl = "hill")
```

`tcplFit2` *tcpl Wrapper for tcplfit2_core including additional calculations to fit into new schema*

Description

tcpl Wrapper for tcplfit2_core including additional calculations to fit into new schema

Usage

```
tcplFit2(
  dat,
 fitmodels = c("cnst", "hill", "gnls", "poly1", "poly2", "pow", "exp2", "exp3", "exp4",
    "exp5"),
  bmed = NULL,
  bidirectional = TRUE
)
```

Arguments

`dat`	output from level 3 processing
`fitmodels`	list of the models that should be fit with the data
`bmed`	baseline value, typically should be 0
`bidirectional`	boolean, default is TRUE (bidirectional fitting)

Value

Data.table with an additional column fitparams that includes all of the fitting parameters

---

`tcplFit2_nest` *Nest dataframe into a list that is readable by tcplfit2*

Description

Nest dataframe into a list that is readable by tcplfit2

Usage

```
tcplFit2_nest(dat)
```

Arguments

`dat`	a dataframe that has all of the fitting parameters in the style of tcplloaddata

Value

a list of fitting parameters that can be consumed by tcplfit2

`tcplFit2_unnest` *Unnest tcplfit2 parameters into a dataframe*

Description

Unnest tcplfit2 parameters into a dataframe

Usage

```
tcplFit2_unnest(output)
```

Arguments

`output`	list of output from tcplfit2

Value

list of parameters unnested and compiled into a dataframe

tcplGetAeid *get Aeid for endpoint name*

Description

`tcplGetAeid` takes a string(name) and finds the assay component endpoint names that match the string and the aeids associated with those names.The function performs a regular expression like matching for strings in the assay component endpoint name column in the assay component endpoint table.

Usage

```
tcplGetAeid(name)
```

Arguments

`name`	A string that will be matched to the assay component endpoint name

Examples

```
## Store the current config settings, so they can be reloaded at the end
## of the examples
conf_store <- tcplConfList()
tcplConfExample()

## Search for aenm (assay name) case insensitive
tcplGetAeid("TOX21")
tcplGetAeid("tox21")

## Reset configuration
options(conf_store)
```

tcplggplot *tcplggplot*

Description

tcplggplot

Usage

```
tcplggplot(dat, lvl = 5, verbose = FALSE)
```

Arguments

`dat`	data table with all required conc/resp data
`lvl`	integer level of data that should be plotted level 4 - all fit models level 5 - all fit models and winning model with hitcall level 6 - include all flags
`verbose`	boolean should plotting include table of values next to the plot

Value

A ggplot object or grob with accompanied table depending on verbose option

---

`tcplHit2` *Hitcalling with tcplfit2*

Description

Hitcalling with tcplfit2

Usage

```
tcplHit2(mc4, coff)
```

Arguments

`mc4`	data.table with level 4 data
`coff`	cutoff value for hitcalling

Value

Data.table with key value pairs of hitcalling parameters

`tcplListFlds` *Load the field names for a table*

Description

`tcplListFlds` loads the column names for the given table and database.

Usage

```
tcplListFlds(tbl, db = getOption("TCPL_DB"))
```

Arguments

`tbl`	Character of length 1, the tcpl database table
`db`	Character of length 1, the tcpl database

Details

This function can be particularly useful in defining the ’fld’ param in the tcplLoad- functions.

Value

A string of field names for the given table.

Examples

```
## Gives the fields in the mc1 table
tcplListFlds("mc1")
```

`tcplLoadChem` *Load sample/chemical information*

Description

`tcplLoadChem` queries the tcpl database and returns the chemical information for the given field and values.

Usage

```
tcplLoadChem(field = NULL, val = NULL, exact = TRUE, include.spid = TRUE)
```

Arguments

`field`	Character of length 1, the field to query on
`val`	Vector of values to subset on
`exact`	Logical, should chemical names be considered exact?
`include.spid`	Logical, should spid be included?

Details

The ’field’ parameter is named differently from the ’fld’ parameter seen in other functions because it only takes one input.

In the MySQL environment the user should be able to give partial chemical name strings, to find chemicals with similar names. For example, setting ’val’ to "phenol" when ’field’ is "chnm" and ’exact’ is `FALSE` might pull up the chemicals "Bisphenol A" and "4-Butylphenol". More technically, setting ’exact’ to `FALSE` passes the string in ’val’ to an RLIKE statement within the MySQL query.

Value

A data.table with the chemical information for the given parameters

Examples

```
## Store the current config settings, so they can be reloaded at the end
## of the examples
conf_store <- tcplConfList()
tcplConfExample()

## Passing no parameters gives all of the registered chemicals with their
## sample IDs
tcplLoadChem()

## Or the user can exclude spid and get a unique list of chemicals
tcplLoadChem(include.spid = FALSE)

## In addition, the user can retrieve only the registered chemicals from the chemical table
tcplLoadChem(field = 'chem.only')

## Other examples:
tcplLoadChem(field = "chnm", val = "Bisphenol A")
tcplLoadChem(field = "chid", val = 20182)

## Reset configuration
options(conf_store)
```

---

tcplLoadClib *Load chemical library information*

Description

`tcplLoadClib` queries the tcpl databases and returns information about the chemical library.

Usage

```
tcplLoadClib(field = NULL, val = NULL)
```

Arguments

`field`	Character of length 1, `'chid'` or `'clib'`, whether to search by chemical id (chid), or chemical library (clib)
`val`	The values to query on

Details

Chemicals are stored in different libraries by chemical ID. Therefore, it is not possible to delineate samples with the same chemical ID into two distinct chemical libraries. However, it is possible for a chemical ID to belong to more than one (or no) chemical libraries.

When chemicals belong to more than one library, the chemical is listed multiple times (one for each distinct library).

Value

A data.table with the chemical library information for the given parameters.

Examples

```
## Store the current config settings, so they can be reloaded at the end
## of the examples
conf_store <- tcplConfList()
tcplConfExample()

## Passing no parameters gives all of the chemical ISs that have a chemical
## library registered
clib <- tcplLoadClib()

## Notice there are more rows in tcplLoadClib than in tcplLoadChem,
## indicating some chemicals must belong to more than library.
chem <- tcplLoadChem(include.spid = FALSE)
nrow(chem)
nrow(clib)

## It is possible that some chemicals do not have a chemical library
## registered, although this is not the case in the example data.
all(chem$chid %in% clib$chid)

## Show the unique chemical libraries
clib[ , unique(clib)]

## Specifying a chemical library will not show what other libraries a
## chemical might belong to.
tcplLoadClib(field = "clib", val = "TOXCAST")
tcplLoadClib(field = "chid", val = 20182)

## Reset configuration
options(conf_store)
```

`tcplLoadConcUnit` *Load concentration units for assay endpoints*

Description

`tcplLoadUnit` queries the tcpl databases and returns a data.table with the concentration units for the given assay endpoint ids (spid).

Usage

```
tcplLoadConcUnit(spid)
```

Arguments

`spid`	Integer, assay endpoint ids

Value

A data.table containing level 3 correction methods for the given spids.

See Also

`tcplQuery`, `data.table`

`tcplLoadData` *Load tcpl data*

Description

`tcplLoadData` queries the tcpl databases and returns a data.table with data for the given level and data type.

Usage

```
tcplLoadData(lvl, fld = NULL, val = NULL, type = "mc", add.fld = TRUE)
```

Arguments

`lvl`	Integer of length 1, the level of data to load
`fld`	Character, the field(s) to query on
`val`	List, vectors of values for each field to query on. Must be in the same order as 'fld'.
`type`	Character of length 1, the data type, "sc" or "mc"
`add.fld`	Boolean if true we want to return the additional parameters fit with tcplfit2

Details

The data type can be either 'mc' for mutliple concentration data, or 'sc' for single concentration data. Multiple concentration data will be loaded into the 'mc' tables, whereas the single concentration will be loaded into the 'sc' tables.

Setting 'lvl' to "agg" will return an aggregate table containing the m4id with the concentration-response data and m3id to map back to well-level information.

Leaving `fld` NULL will return all data.

Valid `fld` inputs are based on the data level and type:

type	lvl	Queried tables
sc	0	sc0
sc	1	sc0, sc1

sc	agg	sc1, sc2_agg
sc	2	sc2
mc	0	mc0
mc	1	mc0, mc1
mc	2	mc0, mc1, mc2
mc	3	mc0, mc1, mc3
mc	agg	mc3, mc4_agg
mc	4	mc4
mc	5	mc4, mc5
mc	6	mc4, mc6
mc	7	mc4, mc7

Value

A data.table containing data for the given fields.

See Also

`tcplQuery`, `data.table`

Examples

```
## Store the current config settings, so they can be reloaded at the end
## of the examples
conf_store <- tcplConfList()
tcplConfExample()

## Load all of level 0 for multiple-concentration data, note 'mc' is the
## default value for type
tcplLoadData(lvl = 0)

## Load all of level 1 for single-concentration
tcplLoadData(lvl = 1, type = "sc")

## List the fields available for level 1, coming from tables mc0 and mc1
tcplListFlds(tbl = "mc0")
tcplListFlds(tbl = "mc1")

## Load level 0 data where the well type is "t" and the concentration
## index is 3 or 4
tcplLoadData(lvl = 1, fld = c("wllt", "cndx"), val = list("t", c(3:4)))

## Reset configuration
options(conf_store)
```

`tcplLoadUnit` *Load response units for assay endpoints*

Description

`tcplLoadUnit` queries the tcpl databases and returns a data.table with the response units for the given assay endpoint ids (aeid).

Usage

```
tcplLoadUnit(aeid)
```

Arguments

`aeid`	Integer, assay endpoint ids

Value

A data.table containing level 3 correction methods for the given aeids.

See Also

`tcplQuery`, `data.table`

`tcplLvlCount` *Load tcpl level counts*

Description

`tcplLvlCount` queries the tcpl databases and returns a data frame with count totals for the given levels and data type.

Usage

```
tcplLvlCount(lvls = NULL, type = "mc")
```

Arguments

`lvls`	Integer or list of Integers, The levels of data to load
`type`	Character of length 1, the data type, "sc" or "mc"

Details

The data type can be either 'mc' for mutliple concentration data, or 'sc' for single concentration data.

Leaving `lvls` NULL will return all data.

Value

A data.table containing data for the given fields.

See Also

`tcplQuery`, `data.table`

Examples

```
## Store the current config settings, so they can be reloaded at the end
## of the examples
conf_store <- tcplConfList()
TCPLlite <- file.path(system.file(package = "tcpl"), "example")
tcplConf(db = TCPLlite, user = NA, host = NA, drvr = "tcplLite")

## Get all counts for level 1 for multiple-concentration
tcplLvlCount(lvls = 1)

## Not run:
## Get all counts for levels 4 through 7 for multiple-concentration
tcplLvlCount(lvls = 4:7)

## Get all counts for multiple-concentration data, note 'mc' is the
## default value for type
tcplLvlCount()

## End(Not run)

## Reset configuration
options(conf_store)
```

`tcplMakeAeidMultiPlts` *Create a .pdf with all dose-response plots for a given aeid, 6 per page*

Description

`tcplMakeAeidMultiPlts` Create a .pdf with all dose-response plots for a given aeid

Usage

```
tcplMakeAeidMultiPlts(
  aeid,
  lvl = 4L,
  fname = NULL,
  odir = getwd(),
  clib = NULL,
  hitc.all = TRUE
)
```

Arguments

`aeid`	Integer of length 1, the assay endpoint id
`lvl`	Integer of length 1, the data level to use (4-7)
`fname`	Character, the filename
`odir`	The directory to save the .pdf file in
`clib`	Character, the chemical library to subset on, see `tcplLoadClib` for more information.
`hitc.all`	If FALSE, only plots with hitc==1 will be displayed

Details

`tcplMakeAeidMultiPlts` provides a wrapper for `tcplMultiplot`, allowing the user to produce PDFs with the curve plots without having to separately load all of the data and establish the PDF device.

If 'fname' is `NULL`, a default name is given by concatenating together assay information.

`tcplMakeAeidPlts` *Create a .pdf with dose-response plots*

Description

`tcplMakeAeidPlts` creates a .pdf file with the dose-response plots for the given aeid.

Usage

```
tcplMakeAeidPlts(
  aeid,
  compare = F,
  lvl = 4L,
  fname = NULL,
  odir = getwd(),
  ordr.fitc = TRUE,
  clib = NULL,
  cnst = NULL
)
```

Arguments

`aeid`	Integer of length 1 or 2, the assay endpoint id
`compare`	Boolean to for comparison of aeids if length(aeid)>1
`lvl`	Integer of length 1, the data level to use (4-7). Only level 5-6 valid for compare aeids.
`fname`	Character, the filename
`odir`	The directory to save the .pdf file in

`ordr.fitc`	Logical, should the fits be ordered by fit category?
`clib`	Character, the chemical library to subset on, see
`cnst`	Constant hline to draw on plot `tcplLoadClib` for more information.

Details

`tcplMakeAeidPlts` provides a wrapper for `tcplPlotFits`, allowing the user to produce PDFs with the curve plots without having to separately load all of the data and establish the PDF device.

If 'fname' is `NULL`, a default name is given by concatenating together assay information.

Note, the default value for ordr.fitc is `TRUE` in `tcplMakeAeidPlts`, but `FALSE` in `tcplPlotFits`

Note, only level 5 or level 6 is valid for comparing 2 aeids.

Examples

```
## Not run:
## Will produce the same result as the example for tcplPlotFits
tcplMakeAeidPlts(aeid = 1, lvl = 6, ordr.fitc = FALSE)

## End(Not run)

## Not run:
## Compare two aeids on same plots
tcplMakeAeidPlts(aeid = c(1,2), compare=T, lvl = 6)

## End(Not run)
```

`tcplMakeChidMultiPlts` *Create a .pdf with all dose-response plots for a given chid, 6 per page*

Description

`tcplMakeChidMultiPlts` Create a .pdf with all dose-response plots for a given chid

Usage

```
tcplMakeChidMultiPlts(
  chid,
  lvl = 4L,
  fname = NULL,
  odir = getwd(),
  clib = NULL,
  hitc.all = TRUE
)
```

Arguments

`chid`	Integer of length 1, the chemical id
`lvl`	Integer of length 1, the data level to use (4-7)
`fname`	Character, the filename
`odir`	The directory to save the .pdf file in
`clib`	Character, the chemical library to subset on, see `tcplLoadClib` for more information.
`hitc.all`	If FALSE, only plots with hitc==1 will be displayed

Details

`tcplMakeChidMultiPlts` provides a wrapper for `tcplMultiplot`, allowing the user to produce PDFs with the curve plots without having to separately load all of the data and establish the PDF device.

If 'fname' is `NULL`, a default name is given by concatenating together assay information.

`tcplMultiplot` *Plot summary fits based on fit and dose-response data*

Description

`tcplMultiplot` takes the dose-response and fit data and produces summary plot figures.

Usage

```
tcplMultiplot(dat, agg, flg = NULL, boot = NULL, browse = FALSE, hitc.all)
```

Arguments

`dat`	data.table, level 4 or level 5 data, see details.
`agg`	data.table, concentration-response aggregate data, see details.
`flg`	data.table, level 6 data, see details.
`boot`	data.table, level 7 data, see details.
`browse`	Logical, should `browser()` be called after every plot?
`hitc.all`	Logical, if FALSE, only plots with hitc==1 will be displayed

Details

The data for 'dat', 'agg', and 'flg' should be loaded using the `tcplLoadData` function with the appropriate 'lvl' parameter. See help page for `tcplLoadData` for more information.

If dat contains only one aeid, plots will be ordered by chemical name (chnm). Otherwise, plots are ordered by assay endpoint name (aenm). ## While it is most likely the user will want to just save all of the plots ## to view in a PDF, the 'browse' parameter can be used to quickly view ## some plots.

tcplPlot *#————————————————————- Generic Plotting Function for tcpl*

Description

`tcplLoadData` queries the tcpl databases and returns a plot for the given level and data type.

Usage

```
tcplPlot(
  lvl = 5,
  fld = "m4id",
  val = NULL,
  type = "mc",
  by = NULL,
  output = c("console", "pdf", "png", "jpg", "svg", "tiff"),
  fileprefix = paste0("tcplPlot_", Sys.Date()),
  multi = NULL,
  verbose = FALSE,
  nrow = NULL,
  ncol = NULL,
  dpi = 600
)
```

Arguments

`lvl`	Integer of length 1, the level of data to load.
`fld`	Character, the field(s) to query on.
`val`	List, vectors of values for each field to query on. Must be in the same order as 'fld'.
`type`	Character of length 1, the data type, "sc" or "mc".
`by`	Parameter to divide files into e.g. "aeid".
`output`	How should the plot be presented. To view the plot in application, use "console", or to save as a file type, use "pdf", "jpg", "png", "svg", or "tiff".
`fileprefix`	Prefix of file when saving.
`multi`	Boolean, by default TRUE for "pdf". If multi is TRUE, output by default 4 plots per page for 'verbose' = TRUE and 6 plots per page for 'verbose' = FALSE.
`verbose`	Boolean, by default FALSE. If TRUE, a table with fitting parameters is included with the plot.
`nrow`	Integer, number of rows in multiplot. By default 2.
`ncol`	Integer, number of columns in multiplot. By default 3, 2 if verbose.
`dpi`	Integer, image print resolution. By default 600.

Details

The data type can be either 'mc' for mutliple concentration data, or 'sc' for single concentration data. Multiple concentration data will be loaded into the 'mc' tables, whereas the single concentration will be loaded into the 'sc' tables.

Setting 'lvl' to "agg" will return an aggregate table containing the m4id with the concentration-response data and m3id to map back to well-level information.

Leaving `fld` NULL will return all data.

Examples

```
## Store the current config settings, so they can be reloaded at the end
## of the examples
conf_store <- tcplConfList()
tcplConfExample()

tcplPlot(lvl = 4, fld = "m4id", val = c(18609966)) ## Create a level 4 plot

## Reset configuration
options(conf_store)
```

`tcplPlotFitc` *Plot the fit category tree*

Description

`tcplPlotFitc` makes a plot showing the level 5 fit categories.

Usage

```
tcplPlotFitc(fitc = NULL, main = NULL, fitc_sub = NULL)
```

Arguments

`fitc`	Integer, the fit categories
`main`	Character of length 1, the title (optional)
`fitc_sub,`	Integer, a subset of fit categories to plot

Note

Suggested device size (inches): width = 10, height = 7.5, pointsize = 9

Examples

```
## Not run:
## Plot visualization of fit categories for all level 5 data
tcplPlotFitc(fitc = tcplLoadData(5)$fitc)

## End(Not run)
```

`tcplPlotFits` *Plot summary fits based on fit and dose-response data*

Description

`tcplPlotFits` takes the dose-response and fit data and produces summary plot figures.

Usage

```
tcplPlotFits(
  dat,
  agg,
  flg = NULL,
  boot = NULL,
  ordr.fitc = FALSE,
  browse = FALSE,
  cnst = NULL,
  orig.aeid = NULL,
  compare = F
)
```

Arguments

`dat`	data.table, level 4 or level 5 data, see details.
`agg`	data.table, concentration-response aggregate data, see details.
`flg`	data.table, level 6 data, see details.
`boot`	data.table, level 7 data, see details.
`ordr.fitc`	Logical, should the fits be ordered by fit category?
`browse`	Logical, should `browser()` be called after every plot?
`cnst`	Constant hline to draw on plot
`orig.aeid`	Original aeid list from tcplMakeAeidPlts to maintain order
`compare`	boolean to determine if aeids should be compared on same plot

Details

The data for 'dat', 'agg', and 'flg' should be loaded using the `tcplLoadData` function with the appropriate 'lvl' parameter. See help page for `tcplLoadData` for more information.

Supplying level 4 data for the 'dat' parameter will result in level 4 plots. Similarly, supp

If fits are not ordered by fit category, they will be ordered by chemical ID. Inputs with multiple assay endpoints will first be ordered by assay endpoint ID.

Examples

```
## Store the current config settings, so they can be reloaded at the end
## of the examples
conf_store <- tcplConfList()
tcplConfDefault()

## tcplPlotFits needs data.tables supplying the concentration/response
## data stored in mc4_agg, as well as the fit information from mc4 or mc5.
## Additionally, tcplPlotFits can take level 6 data from mc6 and add the
## flag information to the plots. The following shows how to make level 5
## plots. Adding the 'flg' parameter would result in level 6 plots, and
## loading level 4, rather than level 5 data, would result in level 4 plots.

l5 <- tcplLoadData(lvl = 5, fld = "m4id", val = 18609966)
l4_agg <- tcplLoadData(lvl = "agg", fld = "m4id", val = 18609966)

## Not run:
pdf(file = "tcplPlotFits.pdf", height = 6, width = 10, pointsize = 10)
tcplPlotFits(dat = l5, agg = l4_agg)
graphics.off()

## End(Not run)

## While it is most likely the user will want to just save all of the plots
## to view in a PDF, the 'browse' parameter can be used to quickly view
## some plots.

## Start by identifying some sample IDs to plot, then call tcplPlotFits with
## a subset of the data. This browse function is admittedly clunky.
bpa <- tcplLoadChem(field = "chnm", val = "Bisphenol A")[ , spid]
l5_sub <- l5[spid %in% bpa]
## Not run:
tcplPlotFits(dat = l5_sub,
             agg = l4_agg[m4id %in% l5_sub$m4id],
             browse = TRUE)

## End(Not run)

## Reset configuration
options(conf_store)
```

`tcplPlotlyPlot` *tcplPlotlyPlot*

Description

tcplPlotlyPlot

Usage

```
tcplPlotlyPlot(dat, lvl = 5)
```

Arguments

`dat`	data table with all required conc/resp data
`lvl`	integer level of data that should be plotted level 4 - all fit models level 5 - all fit models and winning model with hitcall level 6 - include all flags

Value

A plotly plot

`tcplPlotM4ID` *Plot fit summary plot by m4id*

Description

`tcplPlotM4ID` creates a summary plots for the given m4id(s) by loading the appropriate data from the tcpl databases and sending it to `tcplPlotFits`

Usage

```
tcplPlotM4ID(m4id, lvl = 4L)
```

Arguments

`m4id`	Integer, m4id(s) to plot
`lvl`	Integer, the level of data to plot

Details

A level 4 plot ('lvl' = 4) will plot the concentration series and the applicable curves, without an indication of the activity call or the winning model. Level 4 plots can be created without having done subsequent processing.

Level 5 plots include the level 4 information with the activity call and model selection. The winning model will be highlighted red in the side panel containing the summary statistics. Level 6 plots, in addition the all of the level 4 and 5 information, include the positive flag IDs. If the flag has an associated value, the value will be in parentheses following the flag ID. Level 7 plots in addition to all of the level 4, 5, and 6 information, include the AC50 confidence interval and hit percentage information from bootstrapping.

See Also

`tcplPlotFits`, `tcplMakeAeidPlts`

Examples

```
## Store the current config settings, so they can be reloaded at the end
## of the examples
conf_store <- tcplConfList()
tcplConfExample()

tcplPlotM4ID(m4id = 18609966, lvl = 4) ## Create a level 4 plot
tcplPlotM4ID(m4id = 18609966, lvl = 5) ## Create a level 5 plot
tcplPlotM4ID(m4id = 18609966, lvl = 6) ## Create a level 6 plot

#' ## Reset configuration
options(conf_store)
```

`tcplPlotPlate` *Plot plate heatmap*

Description

`tcplPlotPlate` generates a heatmap of assay plate data

Usage

```
tcplPlotPlate(dat, apid, id = NULL, quant = c(0.001, 0.999))
```

Arguments

`dat`	data.table containing tcpl data
`apid`	Character of length 1, the apid to plot
`id`	Integer of length 1, the assay component id (acid) or assay endpoint id (aeid), depending on level. Only need to specify for multiplexed assays when more than one acid/aeid share an apid.
`quant`	Numeric vector, the range of data to include in the legend

Details

The legend represents the range of the data supplied to dat, for the applicable ID. The additional horizontal lines on the legend indicate the range of the plotted plate, to show the relation of the plate to the assay as a whole. A plot with a legend specific for the given apid can be created by only supplying the data for the apid of interest to 'dat'.

The quant parameter, by default including 99.8 allows for extreme outliers without losing resolution. Outliers in either direction will be highlighted with a dark ring, as seen in the example. A NULL value for 'quant' will not restrict the data at all, and will use the full range for the legend.

Wells with a well quality of 0 (only applicable for level 1 plots), will have an "X" through their center.

Note

For the optimal output size, use width = 10, height = 10*(2/3), pointsize = 10, units = "in"

Examples

```
## Store the current config settings, so they can be reloaded at the end
## of the examples
conf_store <- tcplConfList()
tcplConfDefault()

d1 <- tcplLoadData(lvl = 1, fld = "acid", val = 1)
## Not run:
tcplPlotPlate(dat = d1, apid = "09Apr2014.Plate.17")

## End(Not run)

## Reset configuration
options(conf_store)
```

`tcplPrepOtpt` *Map assay/chemical ID values to annotation information*

Description

`tcplPrepOtpt` queries the chemical and assay information from the tcpl database, and maps the annotation information to the given data.

Usage

```
tcplPrepOtpt(dat, ids = NULL)
```

Arguments

`dat`	data.table, output from `tcplLoadData`
`ids`	Character, (optional) a subset of ID fields to map

Details

tcplPrepOtpt is used to map chemical and assay identifiers to their respective names and annotation information to create a human-readable table that is more suitable for an export/output.

By default the function will map sample ID (spid), assay component id (acid), and assay endpoint ID (aeid) values. However, if 'ids' is not null, the function will only attempt to map the ID fields given by 'ids.'

Value

The given data.table with chemical and assay information mapped

Examples

```
## Store the current config settings, so they can be reloaded at the end
## of the examples
conf_store <- tcplConfList()
tcplConfExample()

## Load some example data
d1 <- tcplLoadData(1)

## Check for chemical name in 'dat'
"chnm" %in% names(d1) ## FALSE

#' ## Map all annotations
d2 <- tcplPrepOtpt(d1) ##
"chnm" %in% names(d2) ## TRUE
"acnm" %in% names(d2) ## TRUE

## Map chemical annotation only
d3 <- tcplPrepOtpt(d1, ids = "spid")
"chnm" %in% names(d3) ## TRUE
"acnm" %in% names(d3) ## FALSE

## Reset configuration
options(conf_store)
```

---

tcplRun *Perform data processing*

Description

tcplRun is the function for performing the data processing, for both single-concentration and multiple-concentration formats.

Usage

```
tcplRun(
  asid = NULL,
  slvl,
  elvl,
  id = NULL,
  type = "mc",
  mc.cores = NULL,
  outfile = NULL,
  runname = NULL
)
```

Arguments

`asid`	Integer, assay source id
`slvl`	Integer of length 1, the starting level to process
`elvl`	Integer of length 1, the ending level to process
`id`	Integer, rather than assay source id, the specific assay component or assay endpoint id(s) (optional)
`type`	Character of length 1, the data type, "sc" or "mc"
`mc.cores`	Integer of length 1, the number of cores to use, set to 1 when using Windows operating system
`outfile`	Character of length 1, the name of the log file (optional)
`runname`	Character of length 1, the name of the run to be used in the outfile (optional)

Details

The `tcplRun` function is the core processing function within the package. The function acts as a wrapper for individual processing functions, (ie. `mc1`, `sc1`, etc.) that are not exported. If possible, the processing is done in parallel by 'id' by utilizing the `mclapply` function within the parallel package.

If slvl is less than 4, 'id' is interpreted as acid and if slvl is 4 or greater 'id' is interpreted as aeid. Must give either 'asid' or 'id'. If an id fails no results get loaded into the database, and the id does not get placed into the cue for subsequent level processing.

The 'type' parameter specifies what type of processing to complete: "mc" for multiple-concentration processing, and "sc" for single-concentration processing.

Value

A list containing the results from each level of processing. Each level processed will return a named logical vector, indicating the success of the processing for the id.

`tcplSubsetChid` *Subset level 5 data to a single sample per chemical*

Description

`tcplSubsetChid` subsets level 5 data to a single tested sample per chemical. In other words, if a chemical is tested more than once (a chid has more than one spid) for a given assay endpoint, the function uses a series of logic to select a single "representative" sample.

Usage

```
tcplSubsetChid(dat, flag = TRUE, type = "mc", export_ready = TRUE)
```

Arguments

`dat`	data.table, a data.table with level 5 data
`flag`	Integer, the mc6_mthd_id values to go into the flag count, see details for more information
`type`	Character of length 1, the data type, "sc" or "mc"
`export_ready`	Boolean, default TRUE, should only export ready 1 values be included in calculation

Details

`tcplSubsetChid` is intended to work with level 5 data that has chemical and assay information mapped with `tcplPrepOtpt`.

To select a single sample, first a "consensus hit-call" is made by majority rule, with ties defaulting to active. After the chemical-wise hit call is made, the samples corresponding to to chemical-wise hit call are logically ordered using the fit category, the number of the flags, and the modl_ga, then the first sample for every chemical is selected.

The `flag` param can be used to specify a subset of flags to be used in the flag count. Leaving `flag` TRUE utilize all the available flags. Setting `flag` to `FALSE` will do the subsetting without considering any flags.

Value

A data.table with a single sample for every given chemical-assay pair.

See Also

`tcplPrepOtpt`

Examples

```
## Store the current config settings, so they can be reloaded at the end
## of the examples
conf_store <- tcplConfList()
tcplConfExample()

## Load the example level 5 data
d1 <- tcplLoadData(lvl = 5, fld = "aeid", val = 797)
d1 <- tcplPrepOtpt(d1)

## Subset to an example of a duplicated chid
d2 <- d1[chid == 20182]
d2[, list(m4id, hitc, fitc, modl_ga)]

## Here the consensus hit-call is 1 (active), and the fit categories are
## all equal. Therefore, if the flags are ignored, the selected sample will
## be the sample with the lowest modl_ga.
tcplSubsetChid(dat = d2, flag = FALSE)[, list(m4id, modl_ga)]

## Reset configuration
options(conf_store)
```

tcplVarMat *Create chemical by assay matrices*

Description

`tcplVarMat` creates chemical by assay matrices.

Usage

```
tcplVarMat(
  chid = NULL,
  aeid = NULL,
  add.vars = NULL,
  row.id = "code",
  flag = TRUE,
  cyto.pars = list(),
  include.na.chid = FALSE,
  odir = NULL,
  file.prefix = NULL
)
```

Arguments

chid	Integer, chemical ID values to subset on
aeid	Integer, assay endpoint ID values to subset on

`add.vars`	Character, mc4 or mc5 field(s) not included in the standard list to add additional matrices
`row.id`	Character, the chemical identifier to use in the output
`flag`	Integer or Logical of length 1, passed to `tcplSubsetChid`
`cyto.pars`	List, named list of arguments passed to `tcplCytoPt`
`include.na.chid`	Logical of length 1, whether to include the chemicals not listed in the tcpl databases (ie. controls)
`odir`	Directory to write comma separated file(s)
`file.prefix`	Character of length 1, prefix to the file name when odir is not NULL

Details

The `tcplVarMat` function is used to create chemical by assay matrices for different parameters. The standard list of matrices returned includes:

1. "modl_ga" – The logAC50 (in the gain direction) for the winning model.
2. "hitc" – The hit-call for the winning model.
3. "m4id" – The m4id, listing the concentration series selected by `tcplSubsetChid`.
4. "zscore" – The z-score based on the output from `tcplCytoPt`. The formula used for calculating the z-score is $-(modl\_ga - cyto\_pt)/global\_mad$
5. "tested" – 1 or 0, 1 indicating the chemical/assay pair was tested in either the single- or multiple-concentration format
6. "tested_sc" – 1 or 0, 1 indicating the chemical/assay pair was tested in the single-concentration format
7. "tested_mc" – 1 or 0, 1 indicating the chemical/assay pair was tested in the multiple-concentration format
8. "ac50" – a modified AC50 table (in non-log units) where assay/chemical pairs that were not tested, or tested and had a hitcall of 0 or -1 have the value 1e6.
9. "neglogac50" – -log(AC50/1e6) where assay/chemical pairs that were not tested, or tested and had a hitcall of 0 or -1 have the value 0.

To add additional matrices, the 'add.vars' parameter can be used to specify the fields from the mc4 or mc5 tables to create matrices for.

When more than one sample is included for a chemical/assay pair, `tcplVarMat` aggregates multiple samples to a chemical level call utilizing `tcplSubsetChid`.

By setting `odir` the function will write out a csv with, naming the file with the convention: "var_Matrix_date.csv" where 'var' is the name of the matrix. A prefix can be added to the output files using the 'file.prefix' parameter.

When a concentration series has a sample id not listed in the `tcpl` database, and 'include.na.chid' is TRUE, the rowname for that series will be the concatenation of "SPID_" and the spid. Note, if the user gives a subset of chid values to the 'chid' parameter, 'include.na.chid' will be set to FALSE with a warning.

The tcplVarMat function calls both `tcplSubsetChid` and `tcplCytoPt` (which separately calls `tcplSubsetChid`). The input for the `tcplVarMat` 'flag' parameter is passed to the `tcplSubsetChid` call used to parse down the data to create the matrices. The `tcplSubsetChid` called within `tcplCytoPt` (to parse down the cytotoxicity data used to define the "zscore" matrix) can be modified by passing a separate 'flag' element in the list defined by the 'cyto.pars' parameter.

Value

A list of chemical by assay matrices where the rownames are given by the 'row.id' parameter, and the colnames are given by assay endpoint name (aenm).

See Also

`tcplSubsetChid`

Examples

```
## Store the current config settings, so they can be reloaded at the end
## of the examples
conf_store <- tcplConfList()
TCPLlite <- file.path(system.file(package = "tcpl"), "example")
tcplConf(db = TCPLlite, user = NA, host = NA, drvr = "tcplLite")
## Not run:
## Demonstrate the returned values. Note with no "burst" assays defined in
## the example database, the user must provide which aeid values to use
## in calculating the cytotoxicity distributions for the 'zscore' matrix.
tcplVarMat(chid = 1:5, cyto.pars = list(aeid = 1:2))

## Other changes can be made
tcplVarMat(chid = 1:5, row.id = "chnm", cyto.pars = list(aeid = 1:2))
tcplVarMat(chid = 1:5, add.vars = "max_med", cyto.pars = list(aeid = 1:2))

## End(Not run)
## Reset configuration
options(conf_store)
```

`tcplWriteData` *Write screening data into the tcpl databases*

Description

`tcplWriteData` takes a data.table with screening data and writes the data into the given level table in the tcpl databases.

Usage

```
tcplWriteData(dat, lvl, type)
```

Arguments

`dat`	data.table, the screening data to load
`lvl`	Integer of length 1, the data processing level
`type`	Character of length 1, the data type, "sc" or "mc"

Details

This function appends data onto the existing table. It also deletes all the data for any acids or aeids dat contains from the given and all downstream tables.

The data type can be either 'mc' for mutliple concentration data, or 'sc' for single concentration data. Multiple concentration data will be loaded into the level tables, whereas the single concentration will be loaded into the single tables.

Note

This function is not exported and is not intended to be used by the user. The user should only write level 0 data, which is written with `tcplWriteLvl0`.

See Also

`tcplCascade`, `tcplAppend`, `tcplWriteLvl0`

`tcplWriteLvl0` *Write level 0 screening data into the tcpl databases*

Description

`tcplWriteLvl0` takes a data.table with level 0 screening data and writes the data into the level 0 tables in the tcpl databases.

Usage

```
tcplWriteLvl0(dat, type)
```

Arguments

`dat`	data.table, the screening data to load
`type`	Character of length 1, the data type, "sc" or "mc"

Details

This function appends data onto the existing table. It also deletes all the data for any acids or aeids dat contains from the given and all downstream tables.

Before writing any data the function maps the assay component source name(s) (acsn) to assay component id (acid), ensures the proper class on each field and checks for every test compound sample id (spid where wllt == "t") in the tcpl chemical database. If field types get changed a warning is given listing the affected fields and they type they were coerced to. If the acsn(s) or spid(s) do not map to the tcpl databases the function will return an error and the data will not be written.

The data type can be either 'mc' for mutliple concentration data, or 'sc' for single concentration data. Multiple concentration data will be loaded into the level tables, whereas the single concentration will be loaded into the single tables.

Note

This function should only be used to load level 0 data.

See Also

`tcplCascade`, `tcplAppend`

`write_lvl_4` *Write level 4 with updated schema*

Description

Write level 4 with updated schema

Usage

```
write_lvl_4(dat)
```

Arguments

`dat`	output of tcplfit2 that has been unnested into a data.table

Index