Package 'tidywater'

November 5, 2024

Type Package

Title Water Quality Models for Drinking Water Treatment Processes

Version 0.6.2

URL https://github.com/BrownandCaldwell-Public/tidywater

BugReports https://github.com/BrownandCaldwell-Public/tidywater/issues

Description Provides multiple water chemistry-
based models and published empirical models in one standard format.
Functions can be chained together to model a complete treatment process and are designed to work in a 'tidyverse' workflow.
Models are primarily based on these sources:
Benjamin, M. M. (2002, ISBN:147862308X),
Crittenden, J. C., Trussell, R., Hand, D., Howe, J. K., & Tchobanoglous, G., Borchardt, J. H. (2012, ISBN:9781118131473),
USEPA. (2001) <https://www.epa.gov/sites/default/files/2017-03/documents/wtp_model_v._2.0_manual_508.pdf>.

License Apache License (>= 2) | MIT + file LICENSE

Encoding UTF-8

LazyData true

Imports dplyr, tidyr, knitr, ggplot2, ggrepel, magrittr, purrr, furrr, methods, rlang

RoxygenNote 7.3.2

Depends R (>= 2.10)

Suggests rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

Date 2024-11-01

NeedsCompilation no

Author Sierra Johnson [aut, cre],
Libby McKenna [aut],
Riley Mulhern [aut] (<https://orcid.org/0000-0001-6293-3672>),

Chris Corwin [aut] (<https://orcid.org/0000-0002-9462-0352>),
Rachel Merrifield [ctb],
Mayuri Namasivayam [ctb],
USEPA [cph] (Copyright holder of included TELSS fragments (dissolve_pb function)),
Brown and Caldwell [fnd, cph]

Maintainer Sierra Johnson <sjohnson2@brwncald.com>

Repository CRAN

Date/Publication 2024-11-05 07:50:02 UTC

Contents

`balance_ions` *Add Na, K, Cl, or SO4 to balance overall charge in a water*

Description

This function takes a water defined by `define_water` and balances charge.

Usage

```
balance_ions(water)
```

Arguments

`water`	Water created with define_water, which may have some ions set to 0 when unknown

Details

If more cations are needed, sodium will be added, unless a number for sodium is already provided and potassium is 0, then it will add potassium. Similarly, anions are added using chloride, unless sulfate is 0. If calcium and magnesium are not specified when defining a water with `define_water`, they will default to 0 and not be changed by this function. This function is purely mathematical. User should always check the outputs to make sure values are reasonable for the input source water.

Value

A water class object with updated ions to balance water charge.

Examples

```
water_defined <- define_water(7, 20, 50, 100, 80, 10, 10, 10, 10, tot_po4 = 1) %>%
  balance_ions()
```

`balance_ions_chain` *Apply 'balance_ions' within a dataframe and output a column of 'water' class to be chained to other tidywater functions*

Description

This function allows `balance_ions` to be added to a piped data frame. Its output is a 'water' class, and can therefore be used with "downstream" tidywater functions.

Usage

```
balance_ions_chain(
  df,
  input_water = "defined_water",
  output_water = "balanced_water"
)
```

Arguments

`df`	a data frame containing a water class column, which has already been computed using `define_water_chain`
`input_water`	name of the column of water class data to be used as the input for this function. Default is "defined_water".
`output_water`	name of the output column storing updated parameters with the class, water. Default is "balanced_water".

Details

For large datasets, using ‘fn_once‘ or ‘fn_chain‘ may take many minutes to run. These types of functions use the furrr package for the option to use parallel processing and speed things up. To initialize parallel processing, use ‘plan(multisession)‘ or ‘plan(multicore)‘ (depending on your operating system) prior to your piped code with the ‘fn_once‘ or ‘fn_chain‘ functions. Note, parallel processing is best used when your code block takes more than a minute to run, shorter run times will not benefit from parallel processing.

Value

A data frame containing a water class column with updated ions to balance water charge.

See Also

balance_ions

Examples

```
library(purrr)
library(furrr)
library(tidyr)
library(dplyr)

example_df <- water_df %>%
  define_water_chain() %>%
  balance_ions_chain() %>%
  chemdose_ph_chain(naoh = 5)

example_df <- water_df %>%
  define_water_chain() %>%
  balance_ions_chain(output_water = "balanced ions, balanced life") %>%
  chemdose_ph_chain(input_water = "balanced ions, balanced life", naoh = 5)

# Initialize parallel processing
plan(multisession, workers = 2) # Remove the workers argument to use all available compute
example_df <- water_df %>%
  define_water_chain() %>%
  balance_ions_chain() %>%
  chemdose_ph_chain(naoh = 5)

# Optional: explicitly close multisession processing
plan(sequential)
```

---

balance_ions_once *Apply ‘balance_ions‘ function and output a dataframe*

Description

This function allows balance_ions to be added to a piped data frame. tidywater functions cannot be added after this function because they require a ‘water‘ class input.

Usage

```
balance_ions_once(df, input_water = "defined_water")
```

Arguments

df	a data frame containing a water class column, which has already been computed using define_water_chain
input_water	name of the column of water class data to be used as the input for this function. Default is "defined_water".

Details

For large datasets, using ‘fn_once‘ or ‘fn_chain‘ may take many minutes to run. These types of functions use the furrr package for the option to use parallel processing and speed things up. To initialize parallel processing, use ‘plan(multisession)‘ or ‘plan(multicore)‘ (depending on your operating system) prior to your piped code with the ‘fn_once‘ or ‘fn_chain‘ functions. Note, parallel processing is best used when your code block takes more than a minute to run, shorter run times will not benefit from parallel processing.

Value

A dataframe with updated ions to balance water charge

See Also

balance_ions

Examples

```
library(purrr)
library(furrr)
library(tidyr)
library(dplyr)

example_df <- water_df %>%
  define_water_chain() %>%
  balance_ions_once()

example_df <- water_df %>%
  define_water_chain(output_water = "Different_defined_water_column") %>%
  balance_ions_once(input_water = "Different_defined_water_column")

# Initialize parallel processing
plan(multisession, workers = 2) # Remove the workers argument to use all available compute
example_df <- water_df %>%
```

```
  define_water_chain() %>%
  balance_ions_once()

# Optional: explicitly close multisession processing
plan(sequential)
```

`biofilter_toc` *Determine TOC removal from biofiltration using Terry & Summers BDOC model*

Description

This function applies the Terry model to a water created by `define_water` to determine biofiltered DOC (mg/L).

Usage

```
biofilter_toc(water, ebct, ozonated = TRUE)
```

Arguments

`water`	Source water object of class "water" created by `define_water`.
`ebct`	The empty bed contact time (min) used for the biofilter
`ozonated`	Logical; TRUE if the water is ozonated (default), FALSE otherwise

Value

A water class object with modeled DOC removal from biofiltration.

Source

Terry and Summers 2018

Examples

```
library(tidywater)
water <- define_water(ph = 7, temp = 25, alk = 100, toc = 5.0, doc = 4.0, uv254 = .1) %>%
  biofilter_toc(ebct = 10, ozonated = FALSE)
```

blend_waters — *Determine blended water quality from multiple waters based on mass balance and acid/base equilibrium*

Description

This function takes a vector of waters defined by `define_water` and a vector of ratios and outputs a new water object with updated ions and pH.

Usage

```
blend_waters(waters, ratios)
```

Arguments

`waters`	Vector of source waters created by `define_water`
`ratios`	Vector of ratios in the same order as waters. (Blend ratios must sum to 1)

Value

A water class object with blended water quality parameters.

See Also

`define_water`

Examples

```
water1 <- define_water(7, 20, 50)
water2 <- define_water(7.5, 20, 100, tot_nh3 = 2)
blend_waters(c(water1, water2), c(.4, .6))
```

blend_waters_chain — *Apply 'blend_waters' within a dataframe and output a column of 'water' class to be chained to other tidywater functions*

Description

This function allows `blend_waters` to be added to a piped data frame.

Usage

```
blend_waters_chain(df, waters, ratios, output_water = "blended_water")
```

Arguments

df	a data frame containing a water class column, which has already been computed using define_water_chain,
waters	List of column names containing a water class to be blended
ratios	List of column names or vector of blend ratios in the same order as waters. (Blend ratios must sum to 1)
output_water	name of output column storing updated parameters with the class, water. Default is "blended_water".

Details

The data input comes from a 'water' class column, initialized in define_water or balance_ions. The 'water' class columns to use in the function are specified as function arguments. Ratios may be input as columns with varied ratios (in this case, input column names in the function arguments), OR input as numbers directly.

For large datasets, using 'fn_once' or 'fn_chain' may take many minutes to run. These types of functions use the furrr package for the option to use parallel processing and speed things up. To initialize parallel processing, use 'plan(multisession)' or 'plan(multicore)' (depending on your operating system) prior to your piped code with the 'fn_once' or 'fn_chain' functions. Note, parallel processing is best used when your code block takes more than a minute to run, shorter run times will not benefit from parallel processing.

Value

A data frame with a water class column containing updated ions and pH.

See Also

blend_waters

Examples

```
library(purrr)
library(furrr)
library(tidyr)
library(dplyr)

example_df <- water_df %>%
  define_water_chain() %>%
  balance_ions_chain() %>%
  chemdose_ph_chain(naoh = 22) %>%
  mutate(
    ratios1 = .4,
    ratios2 = .6
  ) %>%
  blend_waters_chain(
    waters = c("defined_water", "dosed_chem_water"),
    ratios = c("ratios1", "ratios2"), output_water = "Blending_after_chemicals"
  )
```

```
example_df <- water_df %>%
  define_water_chain() %>%
  balance_ions_chain() %>%
  chemdose_ph_chain(naoh = 22, output_water = "dosed") %>%
 blend_waters_chain(waters = c("defined_water", "dosed", "balanced_water"), ratios = c(.2, .3, .5))


# Initialize parallel processing
plan(multisession, workers = 2) # Remove the workers argument to use all available compute
example_df <- water_df %>%
  define_water_chain() %>%
  balance_ions_chain() %>%
  chemdose_ph_chain(naoh = 22, output_water = "dosed") %>%
 blend_waters_chain(waters = c("defined_water", "dosed", "balanced_water"), ratios = c(.2, .3, .5))

# Optional: explicitly close multisession processing
plan(sequential)
```

`blend_waters_once` *Apply 'blend_waters' to a dataframe and output 'water' slots as a dataframe*

Description

This function allows `blend_waters` to be added to a piped data frame.

Usage

```
blend_waters_once(df, waters, ratios)
```

Arguments

`df`	a data frame containing a water class column, which has already been computed using `define_water_chain`
`waters`	List of column names containing a water class to be blended
`ratios`	List of column names or vector of blend ratios in the same order as waters. (Blend ratios must sum to 1)

Details

The data input comes from a 'water' class column, initialized in `define_water` or `balance_ions`. The 'water' class columns to use in the function are specified as function arguments. Ratios may be input as columns with varied ratios (in this case, input column names in the function arguments), OR input as numbers directly.

tidywater functions cannot be added after this function because they require a ‘water‘ class input.

For large datasets, using ‘fn_once‘ or ‘fn_chain‘ may take many minutes to run. These types of functions use the furrr package for the option to use parallel processing and speed things up. To initialize parallel processing, use ‘plan(multisession)‘ or ‘plan(multicore)‘ (depending on your operating system) prior to your piped code with the ‘fn_once‘ or ‘fn_chain‘ functions. Note, parallel processing is best used when your code block takes more than a minute to run, shorter run times will not benefit from parallel processing.

Value

A data frame with blended water quality parameters.

See Also

blend_waters

Examples

```
library(purrr)
library(furrr)
library(tidyr)
library(dplyr)

example_df <- water_df %>%
  define_water_chain() %>%
  balance_ions_chain() %>%
  chemdose_ph_chain(naoh = 22, output_water = "dosed") %>%
  mutate(
    ratios1 = .4,
    ratios2 = .6
  ) %>%
 blend_waters_once(waters = c("defined_water", "dosed"), ratios = c("ratios1", "ratios2"))

example_df <- water_df %>%
  define_water_chain() %>%
  balance_ions_chain() %>%
  chemdose_ph_chain(naoh = 22, output_water = "dosed") %>%
 blend_waters_once(waters = c("defined_water", "dosed", "balanced_water"), ratios = c(.2, .3, .5))


# Initialize parallel processing
plan(multisession, workers = 2) # Remove the workers argument to use all available compute
example_df <- water_df %>%
  define_water_chain() %>%
  balance_ions_chain() %>%
  chemdose_ph_chain(naoh = 22, output_water = "dosed") %>%
 blend_waters_once(waters = c("defined_water", "dosed", "balanced_water"), ratios = c(.2, .3, .5))

# Optional: explicitly close multisession processing
plan(sequential)
```

`bromatecoeffs` *Data frame of bromate coefficients for predicting bromate formation during ozonation*

Description

A dataset containing coefficients for calculating ozone formation

Usage

```
bromatecoeffs
```

Format

A dataframe with 30 rows and 10 columns

model First author of source model

ammonia Either T or F, depending on whether the model applies to waters with ammonia present.

A First coefficient in bromate model

a Exponent in bromate model, associated with Br-

b Exponent in bromate model, associated with DOC

c Exponent in bromate model, associated with UVA

d Exponent in bromate model, associated with pH

e Exponent in bromate model, associated with Alkalinity

f Exponent in bromate model, associated with ozone dose

g Exponent in bromate model, associated with reaction time

h Exponent in bromate model, associated with ammonia (NH4+)

i Exponent in bromate model, associated with temperature

I Coefficient in bromate model, associated with temperature in the exponent. Either i or I are used, not both.

Source

Ozekin (1994), Sohn et al (2004), Song et al (1996), Galey et al (1997), Siddiqui et al (1994)

See references list at: https://github.com/BrownandCaldwell-Public/tidywater/wiki/References

`calculate_corrosion` *Calculate six corrosion and scaling indices (AI, RI, LSI, LI, CSMR, CCPP)*

Description

`calculate_corrosion` takes an object of class "water" created by `define_water` and calculates corrosion and scaling indices.

Usage

```
calculate_corrosion(
  water,
  index = c("aggressive", "ryznar", "langelier", "ccpp", "larsonskold", "csmr"),
  form = "calcite"
)
```

Arguments

`water`	Source water of class "water" created by `define_water`
`index`	The indices to be calculated. Default calculates all six indices: "aggressive", "ryznar", "langelier", "ccpp", "larsonskold", "csmr" CCPP may not be able to be calculated sometimes, so it may be advantageous to leave this out of the function to avoid errors
`form`	Form of calcium carbonate mineral to use for modelling solubility: "calcite" (default), "aragonite", or "vaterite"

Details

Aggressiveness Index (AI), unitless - the corrosive tendency of water and its effect on asbestos cement pipe.

Ryznar Index (RI), unitless - a measure of scaling potential.

Langelier Saturation Index (LSI), unitless - describes the potential for calcium carbonate scale formation. Equations use empirical calcium carbonate solubilities from Plummer and Busenberg (1982) and Crittenden et al. (2012) rather than calculated from the concentrations of calcium and carbonate in the water.

Larson-skold Index (LI), unitless - describes the corrosivity towards mild steel.

Chloride-to-sulfate mass ratio (CSMR), mg Cl/mg SO4 - indicator of galvanic corrosion for lead solder pipe joints.

Calcium carbonate precipitation potential (CCPP), mg/L as CaCO3 - a prediction of the mass of calcium carbonate that will precipitate at equilibrium. A positive CCPP value indicates the amount of CaCO3 (mg/L as CaCO3) that will precipitate. A negative CCPP indicates how much CaCO3 can be dissolved in the water.

Value

A water class object with updated corrosion and scaling index slots.

Source

AWWA (1977)

Crittenden et al. (2012)

Langelier (1936)

Larson and Skold (1958)

Merrill and Sanks (1977a)

Merrill and Sanks (1977b)

Merrill and Sanks (1978)

Nguyen et al. (2011)

Plummer and Busenberg (1982)

Ryznar (1946)

Schock (1984)

Trussell (1998)

U.S. EPA (1980)

See reference list at https://github.com/BrownandCaldwell-Public/tidywater/wiki/References

See Also

define_water

Examples

```
water <- define_water(
  ph = 8, temp = 25, alk = 200, tot_hard = 200,
  tds = 576, cl = 150, so4 = 200
) %>%
  calculate_corrosion()

water <- define_water(ph = 8, temp = 25, alk = 100, tot_hard = 50, tds = 200) %>%
  calculate_corrosion(index = c("aggressive", "ccpp"))
```

`calculate_corrosion_chain` *Apply 'calculate_corrosion' to a dataframe and output a column of 'water' class to be chained to other tidywater functions.*

Description

This function allows `calculate_corrosion` to be added to a piped data frame. Up to six additional columns will be added to the output 'water' class column depending on what corrosion/scaling indices are selected: Aggressive index (AI), Ryznar index (RI), Langelier saturation index (LSI), Larson-Skold index (LI), chloride-to-sulfate mass ratio (CSMR) & calcium carbonate precipitation potential (CCPP).

Usage

```
calculate_corrosion_chain(
  df,
  input_water = "defined_water",
  output_water = "corrosion_indices",
  index = c("aggressive", "ryznar", "langelier", "ccpp", "larsonskold", "csmr"),
  form = "calcite"
)
```

Arguments

`df`	a data frame containing a column, defined_water, which has already been computed using `define_water`, and a column named for each of the chemicals being dosed
`input_water`	name of the column of water class data to be used as the input. Default is "defined_water".
`output_water`	name of output column storing updated indices with the class, water. Default is "corrosion_indices".
`index`	The indices to be calculated. Default calculates all six indices: "aggressive", "ryznar", "langelier", "ccpp", "larsonskold", "csmr" CCPP may not be able to be calculated sometimes, so it may be advantageous to leave this out of the function to avoid errors
`form`	Form of calcium carbonate mineral to use for modelling solubility: "calcite" (default), "aragonite", or "vaterite"

Details

The data input comes from a 'water' class column, initialized in `define_water` or `balance_ions`. The 'water' class column to use in the function is specified in the 'input_water' argument (default input 'water' is "defined_water". The name of the output 'water' class column defaults to "corrosion_indices", but may be altered using the 'output_water' argument.

For large datasets, using ‘fn_once‘ or ‘fn_chain‘ may take many minutes to run. These types of functions use the furrr package for the option to use parallel processing and speed things up. To initialize parallel processing, use ‘plan(multisession)‘ or ‘plan(multicore)‘ (depending on your operating system) prior to your piped code with the ‘fn_once‘ or ‘fn_chain‘ functions. Note, parallel processing is best used when your code block takes more than a minute to run, shorter run times will not benefit from parallel processing.

Value

A data frame containing a water class column with updated corrosion and scaling index slots.

See Also

calculate_corrosion

Examples

```
library(purrr)
library(furrr)
library(tidyr)
library(dplyr)

example_df <- water_df %>%
  slice_head(n = 2) %>% # used to make example run faster
  define_water_chain() %>%
  calculate_corrosion_chain()

example_df <- water_df %>%
  slice_head(n = 2) %>% # used to make example run faster
  define_water_chain() %>%
  calculate_corrosion_chain(index = c("aggressive", "ccpp"))


# Initialize parallel processing
plan(multisession, workers = 2) # Remove the workers argument to use all available compute
example_df <- water_df %>%
  define_water_chain() %>%
  calculate_corrosion_chain(index = c("aggressive", "ccpp"))

# Optional: explicitly close multisession processing
plan(sequential)
```

---

calculate_corrosion_once

Apply ‘calculate_corrosion‘ to a dataframe and create new columns with up to 6 corrosion indices

---

Description

This function allows calculate_corrosion to be added to a piped data frame. Up to six additional columns will be added to the dataframe depending on what corrosion/scaling indices are selected: Aggressive index (AI), Ryznar index (RI), Langelier saturation index (LSI), Larson-Skold index (LI), chloride-to-sulfate mass ratio (CSMR) & calcium carbonate precipitation potential (CCPP).

Usage

```
calculate_corrosion_once(
  df,
  input_water = "defined_water",
  index = c("aggressive", "ryznar", "langelier", "ccpp", "larsonskold", "csmr"),
  form = "calcite"
)
```

Arguments

df	a data frame containing a water class column, created using define_water
input_water	name of the column of water class data to be used as the input. Default is "defined_water".
index	The indices to be calculated. Default calculates all six indices: "aggressive", "ryznar", "langelier", "ccpp", "larsonskold", "csmr". CCPP may not be able to be calculated sometimes, so it may be advantageous to leave this out of the function to avoid errors
form	Form of calcium carbonate mineral to use for modelling solubility: "calcite" (default), "aragonite", or "vaterite"

Details

The data input comes from a ‘water‘ class column, initialized in define_water or balance_ions.

For large datasets, using ‘fn_once‘ or ‘fn_chain‘ may take many minutes to run. These types of functions use the furrr package for the option to use parallel processing and speed things up. To initialize parallel processing, use ‘plan(multisession)‘ or ‘plan(multicore)‘ (depending on your operating system) prior to your piped code with the ‘fn_once‘ or ‘fn_chain‘ functions. Note, parallel processing is best used when your code block takes more than a minute to run, shorter run times will not benefit from parallel processing.

Value

A data frame containing specified corrosion and scaling indices.

See Also

calculate_corrosion

Examples

```
library(purrr)
library(furrr)
library(tidyr)
library(dplyr)

example_df <- water_df %>%
  slice_head(n = 2) %>% # used to make example run faster
  define_water_chain() %>%
  calculate_corrosion_once()

example_df <- water_df %>%
  slice_head(n = 2) %>% # used to make example run faster
  define_water_chain() %>%
  calculate_corrosion_once(index = c("aggressive", "ccpp"))


# Initialize parallel processing
plan(multisession, workers = 2) # Remove the workers argument to use all available compute
example_df <- water_df %>%
  define_water_chain() %>%
  calculate_corrosion_once(index = c("aggressive", "ccpp"))

# Optional: explicitly close multisession processing
plan(sequential)
```

---

calculate_dic *Calculate dissolved inorganic carbon (DIC) from total carbonate*

Description

This function takes a water class object defined by define_water and outputs a DIC (mg/L).

Usage

```
calculate_dic(water)
```

Arguments

water	a water class object containing columns with all the parameters listed in define_water

Value

A numeric value for the calculated DIC.

See Also

define_water

Examples

```
example_dic <- define_water(8, 15, 200) %>%
  calculate_dic()
```

`calculate_hardness` *Calculate hardness from calcium and magnesium*

Description

This function takes Ca and Mg in mg/L and returns hardness in mg/L as CaCO3

Usage

```
calculate_hardness(ca, mg, type = "total", startunit = "mg/L")
```

Arguments

`ca`	Calcium concentration in mg/L as Ca
`mg`	Magnesium concentration in mg/L as Mg
`type`	"total" returns total hardness, "ca" returns calcium hardness. Defaults to "total"
`startunit`	Units of Ca and Mg. Defaults to mg/L

Value

A numeric value for the total hardness in mg/L as CaCO3.

Examples

```
calculate_hardness(50, 10)

water_defined <- define_water(7, 20, 50, 100, 80, 10, 10, 10, 10, tot_po4 = 1)
calculate_hardness(water_defined@ca, water_defined@mg, "total", "M")
```

`chemdose_ct` *Determine disinfection credit from chlorine.*

Description

This function takes a water defined by `define_water` and other disinfection parameters and outputs a data frame of the required CT (‘ct_required‘), actual CT (‘ct_actual‘), and giardia log removal (‘glog_removal‘).

Usage

```
chemdose_ct(water, time, residual, baffle)
```

Arguments

`water`	Source water object of class "water" created by `define_water`. Water must include ph and temp
`time`	Retention time of disinfection segment in minutes.
`residual`	Minimum chlorine residual in disinfection segment in mg/L as Cl2.
`baffle`	Baffle factor - unitless value between 0 and 1.

Details

CT actual is a function of time, chlorine residual, and baffle factor, whereas CT required is a function of pH, temperature, chlorine residual, and the standard 0.5 log removal of giardia requirement. CT required is an empirical regression equation developed by Smith et al. (1995) to provide conservative estimates for CT tables in USEPA Disinfection Profiling Guidance. Log removal is a rearrangement of the CT equations.

Value

A data frame of the required CT, actual CT, and giardia log removal.

Source

Smith et al. (1995)

USEPA (2020)

See references list at: https://github.com/BrownandCaldwell-Public/tidywater/wiki/References

See Also

`define_water`

Examples

```
example_ct <- define_water(ph = 7.5, temp = 25) %>%
  chemdose_ct(time = 30, residual = 1, baffle = 0.7)
```

chemdose_dbp *Calculate DBP formation*

Description

chemdose_dbp calculates disinfection byproduct (DBP) formation based on the U.S. EPA's Water Treatment Plant Model (U.S. EPA, 2001). Required arguments include an object of class "water" created by define_water chlorine dose, type, reaction time, and treatment applied (if any). The function also requires additional water quality parameters defined in define_water including bromide, TOC, UV254, temperature, and pH.

Usage

```
chemdose_dbp(
  water,
  cl2,
  time,
  treatment = "raw",
  cl_type = "chorine",
  location = "plant"
)
```

Arguments

water	Source water object of class "water" created by define_water
cl2	Applied chlorine dose (mg/L as Cl2). Model results are valid for doses between 1.51 and 33.55 mg/L.
time	Reaction time (hours). Model results are valid for reaction times between 2 and 168 hours.
treatment	Type of treatment applied to the water. Options include "raw" for no treatment (default), "coag" for water that has been coagulated or softened, and "gac" for water that has been treated by granular activated carbon (GAC). GAC treatment has also been used for estimating formation after membrane treatment with good results.
cl_type	Type of chlorination applied, either "chlorine" (default) or "chloramine".
location	Location for DBP formation, either in the "plant" (default), or in the distributions system, "ds".

Details

The function will calculate haloacetic acids (HAA) as HAA5, and total trihalomethanes (TTHM). Use summarise_wq to quickly tabulate the results.

Value

A water class object with predicted DBP concentrations.

Source

TTHMs, raw: U.S. EPA (2001) equation 5-131

HAAs, raw: U.S. EPA (2001) equation 5-134

TTHMs, treated: U.S. EPA (2001) equation 5-139

HAAs, treated: U.S. EPA (2001) equation 5-142

See references list at: https://github.com/BrownandCaldwell-Public/tidywater/wiki/References

Examples

```
example_dbp <- suppressWarnings(define_water(8, 20, 66, toc = 4, uv254 = .2, br = 50)) %>%
  chemdose_dbp(cl2 = 2, time = 8)
example_dbp <- suppressWarnings(define_water(7.5, 20, 66, toc = 4, uv254 = .2, br = 50)) %>%
  chemdose_dbp(cl2 = 3, time = 168, treatment = "coag", location = "ds")
```

---

chemdose_dbp_chain *Apply 'chemdose_dbp' within a data frame and output a column of 'water' class to be chained to other tidywater functions*

Description

DBP = disinfection byproduct

Usage

```
chemdose_dbp_chain(
  df,
  input_water = "defined_water",
  output_water = "disinfected_water",
  cl2 = 0,
  time = 0,
  treatment = "raw",
  cl_type = "chlorine",
  location = "plant"
)
```

Arguments

df	a data frame containing a water class column, which has already been computed using define_water_chain. The df may include a column named for the applied chlorine dose (cl2), and a column for time.
input_water	name of the column of water class data to be used as the input for this function. Default is "defined_water".
output_water	name of the output column storing updated parameters with the class, water. Default is "disinfected_water".

cl2	Applied chlorine dose (mg/L as Cl2). Model results are valid for doses between 1.51 and 33.55 mg/L.
time	Reaction time (hours). Model results are valid for reaction times between 2 and 168 hours.
treatment	Type of treatment applied to the water. Options include "raw" for no treatment (default), "coag" for water that has been coagulated or softened, and "gac" for water that has been treated by granular activated carbon (GAC). GAC treatment has also been used for estimating formation after membrane treatment with good results.
cl_type	Type of chlorination applied, either "chlorine" (default) or "chloramine".
location	Location for DBP formation, either in the "plant" (default), or in the distribution system, "ds".

Details

This function allows chemdose_dbp to be added to a piped data frame. Its output is a ‘water‘ class, and can therefore be used with "downstream" tidywater functions. TTHM, HAA5, and individual DBP species will be updated based on the applied chlorine dose, the reaction time, treatment type, chlorine type, and DBP formation location.

The data input comes from a ‘water‘ class column, as initialized in define_water or balance_ions.

If the input data frame has a chlorine dose column (cl2) or time column (time), the function will use those columns. Note: The function can only take cl2 and time inputs as EITHER a column or from the function arguments, not both.

For large datasets, using ‘fn_once‘ or ‘fn_chain‘ may take many minutes to run. These types of functions use the furrr package for the option to use parallel processing and speed things up. To initialize parallel processing, use ‘plan(multisession)‘ or ‘plan(multicore)‘ (depending on your operating system) prior to your piped code with the ‘fn_once‘ or ‘fn_chain‘ functions. Note, parallel processing is best used when your code block takes more than a minute to run, shorter run times will not benefit from parallel processing.

Value

A data frame containing a water class column with predicted DBP concentrations.

See Also

chemdose_dbp

Examples

```
library(purrr)
library(furrr)
library(tidyr)
library(dplyr)

example_df <- water_df %>%
  mutate(br = 50) %>%
  define_water_chain() %>%
```

```
  balance_ions_chain() %>%
  chemdose_dbp_chain(input_water = "balanced_water", cl2 = 4, time = 8)

example_df <- water_df %>%
  mutate(br = 50) %>%
  define_water_chain() %>%
  balance_ions_chain() %>%
  mutate(
    cl2 = seq(2, 24, 2),
    time = 30
  ) %>%
  chemdose_dbp_chain(input_water = "balanced_water")

example_df <- water_df %>%
  mutate(br = 80) %>%
  define_water_chain() %>%
  balance_ions_chain() %>%
  mutate(time = 8) %>%
  chemdose_dbp_chain(
    input_water = "balanced_water", cl = 6, treatment = "coag",
    location = "ds", cl_type = "chloramine"
  )


# Initialize parallel processing
plan(multisession, workers = 2) # Remove the workers argument to use all available compute
example_df <- water_df %>%
  mutate(br = 50) %>%
  define_water_chain() %>%
  balance_ions_chain() %>%
  chemdose_dbp_chain(input_water = "balanced_water", cl2 = 4, time = 8)

# Optional: explicitly close multisession processing
plan(sequential)
```

---

chemdose_dbp_once *Apply 'chemdose_dbp'function within a data frame and output a data frame*

Description

DBP = disinfection byproduct

Usage

```
chemdose_dbp_once(
  df,
  input_water = "defined_water",
```

```
  cl2 = 0,
  time = 0,
  treatment = "raw",
  cl_type = "chlorine",
  location = "plant"
)
```

Arguments

df	a data frame containing a water class column, which has already been computed using define_water_once. The df may include a column named for the applied chlorine dose (cl2), and a column for time.
input_water	name of the column of water class data to be used as the input for this function. Default is "defined_water".
cl2	Applied chlorine dose (mg/L as Cl2). Model results are valid for doses between 1.51 and 33.55 mg/L.
time	Reaction time (hours). Model results are valid for reaction times between 2 and 168 hours.
treatment	Type of treatment applied to the water. Options include "raw" for no treatment (default), "coag" for water that has been coagulated or softened, and "gac" for water that has been treated by granular activated carbon (GAC). GAC treatment has also been used for estimating formation after membrane treatment with good results.
cl_type	Type of chlorination applied, either "chlorine" (default) or "chloramine".
location	Location for DBP formation, either in the "plant" (default), or in the distribution system, "ds".

Details

This function allows chemdose_dbp to be added to a piped data frame. Its output is a data frame containing columns for TTHM, HAA5, and individual DBP species. DBPs are estimated based on the applied chlorine dose, the reaction time, treatment type, chlorine type, and DBP formation location.

The data input comes from a ‘water‘ class column, as initialized in define_water or balance_ions.

If the input data frame has a chlorine dose column (cl2) or time column (time), the function will use those columns. Note: The function can only take cl2 and time inputs as EITHER a column or from the function arguments, not both.

tidywater functions cannot be added after this function because they require a ‘water‘ class input.

For large datasets, using ‘fn_once‘ or ‘fn_chain‘ may take many minutes to run. These types of functions use the furrr package for the option to use parallel processing and speed things up. To initialize parallel processing, use ‘plan(multisession)‘ or ‘plan(multicore)‘ (depending on your operating system) prior to your piped code with the ‘fn_once‘ or ‘fn_chain‘ functions. Note, parallel processing is best used when your code block takes more than a minute to run, shorter run times will not benefit from parallel processing.

Value

A data frame with predicted DBP concentrations.

See Also

chemdose_dbp

Examples

```
library(purrr)
library(furrr)
library(tidyr)
library(dplyr)

example_df <- water_df %>%
  mutate(br = 50) %>%
  define_water_chain() %>%
  balance_ions_chain() %>%
  chemdose_dbp_once(input_water = "balanced_water", cl2 = 4, time = 8)

example_df <- water_df %>%
  mutate(br = 50) %>%
  define_water_chain() %>%
  balance_ions_chain() %>%
  mutate(
    cl2 = seq(2, 24, 2),
    time = 30
  ) %>%
  chemdose_dbp_once(input_water = "balanced_water")

example_df <- water_df %>%
  mutate(br = 80) %>%
  define_water_chain() %>%
  balance_ions_chain() %>%
  mutate(time = 8) %>%
  chemdose_dbp_once(
    input_water = "balanced_water", cl = 6, treatment = "coag",
    location = "ds", cl_type = "chloramine"
  )

# Initialize parallel processing
plan(multisession, workers = 2) # Remove the workers argument to use all available compute
example_df <- water_df %>%
  mutate(br = 50) %>%
  define_water_chain() %>%
  balance_ions_chain() %>%
  chemdose_dbp_once(input_water = "balanced_water", cl2 = 4, time = 8)

# Optional: explicitly close multisession processing
plan(sequential)
```

chemdose_f *Calculate new fluoride concentration after dosing alum.*

Description

Applies equation of the form: raw_f - A*alum^a*ph ^ b * raw_f^c. There is no published model, so it is recommended to fit the coefficients with experimental data. When fitting, the following units must be used: Alum in mg/L as chemical, Fluoride in mg/L, pH in SU. Default coefficients are fit from Sollo et al (1978). This function outputs a water class object with an updated fluoride concentration (which will be in M, per standard water units).

Usage

```
chemdose_f(water, alum, coeff = c(1.11, 0.628, -2.07, 0.861))
```

Arguments

water	Source water object of class "water" created by define_water
alum	Amount of hydrated aluminum sulfate added in mg/L: Al2(SO4)3*14H2O + 6HCO3 -> 2Al(OH)3(am) +3SO4 + 14H2O + 6CO2
coeff	Model coefficients to use as vector of numbers.

Value

A water class object with an updated fluoride concentration.

Examples

```
dosed_water <- define_water(ph = 7, temp = 25, alk = 50, f = 4) %>%
  chemdose_ph(alum = 50) %>%
  chemdose_f(alum = 50)

convert_units(dosed_water@f, "f", "M", "mg/L")
```

chemdose_ph *Calculate new pH and ion balance after chemical addition*

Description

chemdose_ph calculates the new pH, alkalinity, and ion balance of a water based on different chemical additions.

Usage

```
chemdose_ph(
  water,
  hcl = 0,
  h2so4 = 0,
  h3po4 = 0,
  co2 = 0,
  naoh = 0,
  caoh2 = 0,
  mgoh2 = 0,
  na2co3 = 0,
  nahco3 = 0,
  caco3 = 0,
  cacl2 = 0,
  cl2 = 0,
  naocl = 0,
  nh4oh = 0,
  nh42so4 = 0,
  alum = 0,
  ferricchloride = 0,
  ferricsulfate = 0,
  ach = 0,
  softening_correction = FALSE
)
```

Arguments

water	Source water object of class "water" created by define_water
hcl	Amount of hydrochloric acid added in mg/L: HCl -> H + Cl
h2so4	Amount of sulfuric acid added in mg/L: H2SO4 -> 2H + SO4
h3po4	Amount of phosphoric acid added in mg/L: H3PO4 -> 3H + PO4
co2	Amount of carbon dioxide added in mg/L: CO2 (gas) + H2O -> H2CO3*
naoh	Amount of caustic added in mg/L: NaOH -> Na + OH
caoh2	Amount of lime added in mg/L: Ca(OH)2 -> Ca + 2OH
mgoh2	Amount of magneisum hydroxide added in mg/L: Mg(OH)2 -> Mg + 2OH
na2co3	Amount of soda ash added in mg/L: Na2CO3 -> 2Na + CO3
nahco3	Amount of sodium bicarbonate added in mg/L: NaHCO3 -> Na + H + CO3
caco3	Amount of calcium carbonate added (or removed) in mg/L: CaCO3 -> Ca + CO3
cacl2	Amount of calcium chloride added in mg/L: CaCl2 -> Ca2+ + 2Cl-
cl2	Amount of chlorine gas added in mg/L as Cl2: Cl2(g) + H2O -> HOCl + H + Cl
naocl	Amount of sodium hypochlorite added in mg/L as Cl2: NaOCl -> Na + OCl
nh4oh	Amount of ammonium hydroxide added in mg/L as N: NH4OH -> NH4 + OH
nh42so4	Amount of ammonium sulfate added in mg/L as N: (NH4)2SO4 -> 2NH4 + SO4

`alum` Amount of hydrated aluminum sulfate added in mg/L: Al2(SO4)3*14H2O + 6HCO3 -> 2Al(OH)3(am) +3SO4 + 14H2O + 6CO2

`ferricchloride` Amount of ferric Chloride added in mg/L: FeCl3 + 3HCO3 -> Fe(OH)3(am) + 3Cl + 3CO2

`ferricsulfate` Amount of ferric sulfate added in mg/L: Fe2(SO4)3*8.8H2O + 6HCO3 -> 2Fe(OH)3(am) + 3SO4 + 8.8H2O + 6CO2

`ach` Amount of aluminum chlorohydrate added in mg/L: Al2(OH)5Cl*2H2O + HCO3 -> 2Al(OH)3(am) + Cl + 2H2O + CO2

`softening_correction` Set to TRUE to correct post-softening pH (caco3 must be < 0). Default is FALSE. Based on WTP model equation 5-62

Details

The function takes an object of class "water" created by `define_water` and user-specified chemical additions and returns a new object of class "water" with updated water quality. Units of all chemical additions are in mg/L as chemical (not as product).

`chemdose_ph` works by evaluating all the user-specified chemical additions and solving for what the new pH must be using `uniroot` to satisfy the principle of electroneutrality in pure water while correcting for the existing alkalinity of the water that the chemical is added to. Multiple chemicals can be added simultaneously or each addition can be modeled independently through sequential doses.

Value

A water class object with updated pH, alkalinity, and ions post-chemical addition.

See Also

`define_water`, `convert_units`

Examples

```
water <- define_water(ph = 7, temp = 25, alk = 10)
# Dose 1 mg/L of hydrochloric acid
dosed_water <- chemdose_ph(water, hcl = 1)
dosed_water@ph

# Dose 1 mg/L of hydrochloric acid and 5 mg/L of alum simultaneously
dosed_water <- chemdose_ph(water, hcl = 1, alum = 5)
dosed_water@ph

# Dose 1 mg/L of hydrochloric acid and 5 mg/L of alum sequentially
dosed_water1 <- chemdose_ph(water, hcl = 1)
dosed_water1@ph
dosed_water2 <- chemdose_ph(dosed_water1, alum = 5)
dosed_water2@ph

# Softening:
```

```
water2 <- define_water(ph = 7, temp = 25, alk = 100, tot_hard = 350)
dosed_water1 <- chemdose_ph(water2, caco3 = -100)
dosed_water1@ph
dosed_water2 <- chemdose_ph(water2, caco3 = -100, softening_correction = TRUE)
dosed_water2@ph
```

---

chemdose_ph_chain *Apply 'chemdose_ph' within a dataframe and output a column of 'water' class to be chained to other tidywater functions*

Description

This function allows chemdose_ph to be added to a piped data frame. Its output is a 'water' class, and can therefore be used with "downstream" tidywater functions. Ions and pH will be updated based on input chemical doses.

Usage

```
chemdose_ph_chain(
  df,
  input_water = "defined_water",
  output_water = "dosed_chem_water",
  hcl = 0,
  h2so4 = 0,
  h3po4 = 0,
  co2 = 0,
  naoh = 0,
  na2co3 = 0,
  nahco3 = 0,
  caoh2 = 0,
  mgoh2 = 0,
  cl2 = 0,
  naocl = 0,
  nh4oh = 0,
  nh42so4 = 0,
  alum = 0,
  ferricchloride = 0,
  ferricsulfate = 0,
  ach = 0,
  caco3 = 0
)
```

Arguments

df	a data frame containing a water class column, which has already been computed using define_water_chain. The df may include columns named for the chemical(s) being dosed.

input_water	name of the column of water class data to be used as the input for this function. Default is "defined_water".
output_water	name of the output column storing updated parameters with the class, water. Default is "dosed_chem_water".
hcl	Hydrochloric acid: HCl -> H + Cl
h2so4	Sulfuric acid: H2SO4 -> 2H + SO4
h3po4	Phosphoric acid: H3PO4 -> 3H + PO4
co2	Carbon Dioxide CO2 (gas) + H2O -> H2CO3*
naoh	Caustic: NaOH -> Na + OH
na2co3	Soda ash: Na2CO3 -> 2Na + CO3
nahco3	Sodium bicarbonate: NaHCO3 -> Na + H + CO3
caoh2	Lime: Ca(OH)2 -> Ca + 2OH
mgoh2	Magneisum hydroxide: Mg(OH)2 -> Mg + 2OH
cl2	Chlorine gas: Cl2(g) + H2O -> HOCl + H + Cl
naocl	Sodium hypochlorite: NaOCl -> Na + OCl
nh4oh	Amount of ammonium hydroxide added in mg/L as N: NH4OH -> NH4 + OH
nh42so4	Amount of ammonium sulfate added in mg/L as N: (NH4)2SO4 -> 2NH4 + SO4
alum	Hydrated aluminum sulfate Al2(SO4)3*14H2O + 6HCO3 -> 2Al(OH)3(am) +3SO4 + 14H2O + 6CO2
ferricchloride	Ferric Chloride FeCl3 + 3HCO3 -> Fe(OH)3(am) + 3Cl + 3CO2
ferricsulfate	Amount of ferric sulfate added in mg/L: Fe2(SO4)3*8.8H2O + 6HCO3 -> 2Fe(OH)3(am) + 3SO4 + 8.8H2O + 6CO2
ach	Amount of aluminum chlorohydrate added in mg/L: Al2(OH)5Cl*2H2O + HCO3 -> 2Al(OH)3(am) + Cl + 2H2O + CO2
caco3	Amount of calcium carbonate added (or removed) in mg/L: CaCO3 -> Ca + CO3

Details

The data input comes from a ‘water‘ class column, as initialized in `define_water` or `balance_ions`.

If the input data frame has a column(s) name matching a valid chemical(s), the function will dose that chemical(s) in addition to the ones specified in the function's arguments. The column names must match the chemical names as displayed in `chemdose_ph`. To see which chemicals can be passed into the function, see `chemdose_ph`.

For large datasets, using ‘fn_once‘ or ‘fn_chain‘ may take many minutes to run. These types of functions use the furrr package for the option to use parallel processing and speed things up. To initialize parallel processing, use ‘plan(multisession)‘ or ‘plan(multicore)‘ (depending on your operating system) prior to your piped code with the ‘fn_once‘ or ‘fn_chain‘ functions. Note, parallel processing is best used when your code block takes more than a minute to run, shorter run times will not benefit from parallel processing.

Value

A data frame containing a water class column with updated pH, alkalinity, and ions post-chemical addition.

See Also

chemdose_ph

Examples

```
library(purrr)
library(furrr)
library(tidyr)
library(dplyr)

example_df <- water_df %>%
  define_water_chain() %>%
  balance_ions_chain() %>%
  chemdose_ph_chain(input_water = "balanced_water", naoh = 5)

example_df <- water_df %>%
  define_water_chain() %>%
  balance_ions_chain() %>%
  mutate(
    hcl = seq(1, 12, 1),
    naoh = 20
  ) %>%
  chemdose_ph_chain(input_water = "balanced_water", mgoh2 = 55, co2 = 4)

# Initialize parallel processing
plan(multisession, workers = 2) # Remove the workers argument to use all available compute
example_df <- water_df %>%
  define_water_chain() %>%
  balance_ions_chain() %>%
  chemdose_ph_chain(input_water = "balanced_water", naoh = 5)

# Optional: explicitly close multisession processing
plan(sequential)
```

---

chemdose_ph_once *Apply 'chemdose_ph' function and output a dataframe*

Description

This function allows chemdose_ph to be added to a piped data frame. Its output is a data frame with updated ions and pH.

Usage

```
chemdose_ph_once(
  df,
  input_water = "defined_water",
  hcl = 0,
  h2so4 = 0,
  h3po4 = 0,
  co2 = 0,
  naoh = 0,
  na2co3 = 0,
  nahco3 = 0,
  caoh2 = 0,
  mgoh2 = 0,
  cl2 = 0,
  naocl = 0,
  nh4oh = 0,
  nh42so4 = 0,
  alum = 0,
  ferricchloride = 0,
  ferricsulfate = 0,
  ach = 0,
  caco3 = 0
)
```

Arguments

df	a data frame containing a water class column, which has already been computed using define_water_chain. The df may include columns named for the chemical(s) being dosed.
input_water	name of the column of water class data to be used as the input for this function. Default is "defined_water".
hcl	Hydrochloric acid: HCl -> H + Cl
h2so4	Sulfuric acid: H2SO4 -> 2H + SO4
h3po4	Phosphoric acid: H3PO4 -> 3H + PO4
co2	Carbon Dioxide CO2 (gas) + H2O -> H2CO3*
naoh	Caustic: NaOH -> Na + OH
na2co3	Soda ash: Na2CO3 -> 2Na + CO3
nahco3	Sodium bicarbonate: NaHCO3 -> Na + H + CO3
caoh2	Lime: Ca(OH)2 -> Ca + 2OH
mgoh2	Magneisum hydroxide: Mg(OH)2 -> Mg + 2OH
cl2	Chlorine gas: Cl2(g) + H2O -> HOCl + H + Cl
naocl	Sodium hypochlorite: NaOCl -> Na + OCl
nh4oh	Amount of ammonium hydroxide added in mg/L as N: NH4OH -> NH4 + OH
nh42so4	Amount of ammonium sulfate added in mg/L as N: (NH4)2SO4 -> 2NH4 + SO4

alum	Hydrated aluminum sulfate Al2(SO4)3*14H2O + 6HCO3 -> 2Al(OH)3(am) +3SO4 + 14H2O + 6CO2
ferricchloride	Ferric Chloride FeCl3 + 3HCO3 -> Fe(OH)3(am) + 3Cl + 3CO2
ferricsulfate	Amount of ferric sulfate added in mg/L: Fe2(SO4)3*8.8H2O + 6HCO3 -> 2Fe(OH)3(am) + 3SO4 + 8.8H2O + 6CO2
ach	Amount of aluminum chlorohydrate added in mg/L: Al2(OH)5Cl*2H2O + HCO3 -> 2Al(OH)3(am) + Cl + 2H2O + CO2
caco3	Amount of calcium carbonate added (or removed) in mg/L: CaCO3 -> Ca + CO3

Details

The data input comes from a 'water' class column, as initialized in `define_water` or `balance_ions`.

If the input data frame has a column(s) name matching a valid chemical(s), the function will dose that chemical(s) in addition to the ones specified in the function's arguments. The column names must match the chemical names as displayed in `chemdose_ph`. To see which chemicals can be passed into the function, see `chemdose_ph`.

tidywater functions cannot be added after this function because they require a 'water' class input.

For large datasets, using 'fn_once' or 'fn_chain' may take many minutes to run. These types of functions use the furrr package for the option to use parallel processing and speed things up. To initialize parallel processing, use 'plan(multisession)' or 'plan(multicore)' (depending on your operating system) prior to your piped code with the 'fn_once' or 'fn_chain' functions. Note, parallel processing is best used when your code block takes more than a minute to run, shorter run times will not benefit from parallel processing.

Value

A data frame with updated pH, alkalinity, and ions post-chemical addition.

See Also

`chemdose_ph`

Examples

```
library(purrr)
library(furrr)
library(tidyr)
library(dplyr)

example_df <- water_df %>%
  define_water_chain() %>%
  balance_ions_chain() %>%
  chemdose_ph_once(input_water = "balanced_water", naoh = 5)

example_df <- water_df %>%
  define_water_chain() %>%
  balance_ions_chain() %>%
```

```
  mutate(
    hcl = seq(1, 12, 1),
    naoh = 20
  ) %>%
  chemdose_ph_once(input_water = "balanced_water", mgoh2 = 55, co2 = 4)

# Initialize parallel processing
plan(multisession, workers = 2) # Remove the workers argument to use all available compute
example_df <- water_df %>%
  define_water_chain() %>%
  balance_ions_chain() %>%
  chemdose_ph_once(input_water = "balanced_water", naoh = 5)

# Optional: explicitly close multisession processing
plan(sequential)
```

---

chemdose_toc *Determine TOC removal from coagulation*

Description

This function applies the Edwards (1997) model to a water created by define_water to determine coagulated DOC. Coagulated UVA is from U.S. EPA (2001) equation 5-80. Note that the models rely on pH of coagulation. If only raw water pH is known, utilize chemdose_ph first.

Usage

```
chemdose_toc(
  water,
  alum = 0,
  ferricchloride = 0,
  ferricsulfate = 0,
  coeff = "Alum"
)
```

Arguments

water	Source water object of class "water" created by define_water. Water must include ph, doc, and uv254
alum	Amount of hydrated aluminum sulfate added in mg/L: Al2(SO4)3*14H2O + 6HCO3 -> 2Al(OH)3(am) +3SO4 + 14H2O + 6CO2
ferricchloride	Amount of ferric chloride added in mg/L: FeCl3 + 3HCO3 -> Fe(OH)3(am) + 3Cl + 3CO2
ferricsulfate	Amount of ferric sulfate added in mg/L: Fe2(SO4)3*8.8H2O + 6HCO3 -> 2Fe(OH)3(am) + 3SO4 + 8.8H2O + 6CO2
coeff	String specifying the Edwards coefficients to be used from "Alum", "Ferric", "General Alum", "General Ferric", or "Low DOC" or named vector of coefficients, which must include: k1, k2, x1, x2, x3, b

Value

A water class object with an updated DOC, TOC, and UV254 concentration.

Source

Edwards (1997)

U.S. EPA (2001)

See reference list at: https://github.com/BrownandCaldwell-Public/tidywater/wiki/References

See Also

chemdose_ph

Examples

```
water <- define_water(ph = 7, temp = 25, alk = 100, toc = 3.7, doc = 3.5, uv254 = .1)
dosed_water <- chemdose_ph(water, alum = 30) %>%
  chemdose_toc(alum = 30, coeff = "Alum")

dosed_water <- chemdose_ph(water, ferricsulfate = 30) %>%
  chemdose_toc(ferricsulfate = 30, coeff = "Ferric")

dosed_water <- chemdose_ph(water, alum = 10, h2so4 = 10) %>%
  chemdose_toc(alum = 10, coeff = c(
    "x1" = 280, "x2" = -73.9, "x3" = 4.96,
    "k1" = -0.028, "k2" = 0.23, "b" = 0.068
  ))
```

---

chemdose_toc_chain *Apply 'chemdose_toc' within a dataframe and output a column of 'water' class to be chained to other tidywater functions*

Description

This function allows chemdose_toc to be added to a piped data frame. Its output is a 'water' class, and can therefore be used with "downstream" tidywater functions. TOC, DOC, and UV254 will be updated based on input chemical doses.

Usage

```
chemdose_toc_chain(
  df,
  input_water = "defined_water",
  output_water = "coagulated_water",
  alum = 0,
  ferricchloride = 0,
```

```
  ferricsulfate = 0,
  coeff = "Alum"
)
```

Arguments

df	a data frame containing a water class column, which has already been computed using define_water_chain. The df may include a column named for the coagulant being dosed, and a column named for the set of coefficients to use.
input_water	name of the column of Water class data to be used as the input for this function. Default is "defined_water".
output_water	name of the output column storing updated parameters with the class, Water. Default is "coagulated_water".
alum	Hydrated aluminum sulfate Al2(SO4)3*14H2O + 6HCO3 -> 2Al(OH)3(am) +3SO4 + 14H2O + 6CO2
ferricchloride	Ferric Chloride FeCl3 + 3HCO3 -> Fe(OH)3(am) + 3Cl + 3CO2
ferricsulfate	Amount of ferric sulfate added in mg/L: Fe2(SO4)3*8.8H2O + 6HCO3 -> 2Fe(OH)3(am) + 3SO4 + 8.8H2O + 6CO2
coeff	String specifying the Edwards coefficients to be used from "Alum", "Ferric", "General Alum", "General Ferric", or "Low DOC" or named vector of coefficients, which must include: k1, k2, x1, x2, x3, b

Details

The data input comes from a ‘water‘ class column, as initialized in define_water or balance_ions.

If the input data frame has a coagulant(s) name matching a valid coagulant(s), the function will dose that coagulant(s). Note: The function can only dose a coagulant either a column or from the function arguments, not both.

The column names must match the chemical names as displayed in chemdose_toc. To see which chemicals can be passed into the function, see chemdose_toc.

For large datasets, using ‘fn_once‘ or ‘fn_chain‘ may take many minutes to run. These types of functions use the furrr package for the option to use parallel processing and speed things up. To initialize parallel processing, use ‘plan(multisession)‘ or ‘plan(multicore)‘ (depending on your operating system) prior to your piped code with the ‘fn_once‘ or ‘fn_chain‘ functions. Note, parallel processing is best used when your code block takes more than a minute to run, shorter run times will not benefit from parallel processing.

Value

A data frame containing a water class column with updated DOC, TOC, and UV254 concentrations.

See Also

chemdose_toc

Examples

```
library(purrr)
library(furrr)
library(tidyr)
library(dplyr)

example_df <- water_df %>%
  define_water_chain() %>%
  balance_ions_chain() %>%
  chemdose_ph_chain(alum = 30) %>%
  chemdose_toc_chain(input_water = "dosed_chem_water")

example_df <- water_df %>%
  define_water_chain() %>%
  balance_ions_chain() %>%
  mutate(
    ferricchloride = seq(1, 12, 1),
    coeff = "Ferric"
  ) %>%
  chemdose_toc_chain(input_water = "balanced_water")

example_df <- water_df %>%
  define_water_chain() %>%
  balance_ions_chain() %>%
  chemdose_toc_chain(input_water = "balanced_water", alum = 40, coeff = "General Alum")

# Initialize parallel processing
plan(multisession, workers = 2) # Remove the workers argument to use all available compute
example_df <- water_df %>%
  define_water_chain() %>%
  balance_ions_chain() %>%
  mutate(ferricchloride = seq(1, 12, 1)) %>%
  chemdose_toc_chain(input_water = "balanced_water", coeff = "Ferric")

# Optional: explicitly close multisession processing
plan(sequential)
```

---

chemdose_toc_once *Apply 'chemdose_toc' function and output a data frame*

Description

This function allows chemdose_toc to be added to a piped data frame. Its output is a data frame with updated TOC, DOC, and UV254.

Usage

```
chemdose_toc_once(
```

```
  df,
  input_water = "defined_water",
  alum = 0,
  ferricchloride = 0,
  ferricsulfate = 0,
  coeff = "Alum"
)
```

Arguments

df	a data frame containing a water class column, which has already been computed using define_water_chain. The df may include a column named for the coagulant being dosed, and a column named for the set of coefficients to use.
input_water	name of the column of Water class data to be used as the input for this function. Default is "defined_water".
alum	Hydrated aluminum sulfate Al2(SO4)3*14H2O + 6HCO3 -> 2Al(OH)3(am) +3SO4 + 14H2O + 6CO2
ferricchloride	Ferric Chloride FeCl3 + 3HCO3 -> Fe(OH)3(am) + 3Cl + 3CO2
ferricsulfate	Amount of ferric sulfate added in mg/L: Fe2(SO4)3*8.8H2O + 6HCO3 -> 2Fe(OH)3(am) + 3SO4 + 8.8H2O + 6CO2
coeff	String specifying the Edwards coefficients to be used from "Alum", "Ferric", "General Alum", "General Ferric", or "Low DOC" or named vector of coefficients, which must include: k1, k2, x1, x2, x3, b

Details

The data input comes from a ‘water‘ class column, as initialized in define_water or balance_ions.

If the input data frame has a column(s) name matching a valid coagulant(s), the function will dose that coagulant(s). Note: The function can only dose a coagulant as either a column or from the function arguments, not both.

The column names must match the coagulant names as displayed in chemdose_toc. To see which coagulants can be passed into the function, see chemdose_toc.

tidywater functions cannot be added after this function because they require a ‘water‘ class input.

For large datasets, using ‘fn_once‘ or ‘fn_chain‘ may take many minutes to run. These types of functions use the furrr package for the option to use parallel processing and speed things up. To initialize parallel processing, use ‘plan(multisession)‘ or ‘plan(multicore)‘ (depending on your operating system) prior to your piped code with the ‘fn_once‘ or ‘fn_chain‘ functions. Note, parallel processing is best used when your code block takes more than a minute to run, shorter run times will not benefit from parallel processing.

Value

A data frame with an updated DOC, TOC, and UV254 concentration.

See Also

chemdose_toc

Examples

```
library(purrr)
library(furrr)
library(tidyr)
library(dplyr)

example_df <- water_df %>%
  define_water_chain() %>%
  balance_ions_chain() %>%
  chemdose_ph_chain(alum = 30) %>%
  chemdose_toc_once(input_water = "dosed_chem_water")

example_df <- water_df %>%
  define_water_chain() %>%
  balance_ions_chain() %>%
  mutate(
    ferricchloride = seq(1, 12, 1),
    coeff = "Ferric"
  ) %>%
  chemdose_toc_once(input_water = "balanced_water")

example_df <- water_df %>%
  define_water_chain() %>%
  balance_ions_chain() %>%
  chemdose_toc_once(input_water = "balanced_water", alum = 40, coeff = "General Alum")

# Initialize parallel processing
plan(multisession, workers = 2) # Remove the workers argument to use all available compute
example_df <- water_df %>%
  define_water_chain() %>%
  balance_ions_chain() %>%
  mutate(ferricchloride = seq(1, 12, 1)) %>%
  chemdose_toc_once(input_water = "balanced_water", coeff = "Ferric")

# Optional: explicitly close multisession processing
plan(sequential)
```

---

chloramine_conv *Data frame of conversion factors for estimating DBP formation from chloramines*

Description

A dataset containing conversion factors for calculating DBP formation

Usage

```
chloramine_conv
```

Format

A dataframe with 17 rows and 3 columns

ID abbreviation of dbp species

alias full name of dbp species

percent specifies the percent of DBP formation predicted from chloramines compared to chlorine, assuming the same chlorine dose applied

Source

U.S. EPA (2001), Table 5-10

See references list at: https://github.com/BrownandCaldwell-Public/tidywater/wiki/References

---

convert_units *Calculate unit conversions for common compounds*

Description

This function takes a value and converts units based on compound name.

Usage

```
convert_units(value, formula, startunit = "mg/L", endunit = "M")
```

Arguments

value	Value to be converted
formula	Chemical formula of compound. Accepts compounds in mweights for conversions between g and mol or eq
startunit	Units of current value, currently accepts g/L; g/L CaCO3; g/L N; M; eq/L; and the same units with "m", "u", "n" prefixes
endunit	Desired units, currently accepts same as start units

Value

A numeric value for the converted parameter.

Examples

```
convert_units(50, "ca") # converts from mg/L to M by default
convert_units(50, "ca", "mg/L", "mg/L CaCO3")
convert_units(50, "ca", startunit = "mg/L", endunit = "eq/L")
```

convert_water *Convert 'water' class object to a dataframe*

Description

This converts a 'water' class to a dataframe with individual columns for each slot (water quality parameter) in the 'water'. This is useful for one-off checks and is applied in all 'fn_once' tidywater functions. For typical applications, there may be a 'fn_once' tidywater function that provides a more efficient solution.

Usage

```
convert_water(water)
```

Arguments

water	A water class object

Value

A data frame containing columns for all non-NA water slots.

See Also

define_water

Examples

```
library(dplyr)
library(tidyr)

# Generates 1 row dataframe
example_df <- define_water(ph = 7, temp = 20, alk = 100) %>%
  convert_water()

example_df <- water_df %>%
  define_water_chain() %>%
  mutate(to_dataframe = map(defined_water, convert_water)) %>%
  unnest(to_dataframe) %>%
  select(-defined_water)
```

convert_watermg — *Convert a 'water' class object to a dataframe with ions in mg/L or ug/L*

Description

This function is the same as convert_water except it converts the units of following slots from M to mg/L: na, ca, mg, k, cl, so4, hco3, co3, h2po4, hpo4, po4, ocl, bro3, f, fe, al. These slots are converted to ug/L: br, mn. All other values remain unchanged.

Usage

```
convert_watermg(water)
```

Arguments

water	A water class object

Value

A data frame containing columns for all non-NA water slots with ions in mg/L.

Examples

```
water_defined <- define_water(7, 20, 50, 100, 80, 10, 10, 10, 10, tot_po4 = 1) %>%
  convert_watermg()
```

dbpcoeffs — *Data frame of DBP coefficients for predicting DBP formation*

Description

A dataset containing coefficients for calculating DBP formation

Usage

```
dbpcoeffs
```

Format

A dataframe with 30 rows and 10 columns

ID abbreviation of dbp species

alias full name of dbp species

water_type specifies which model the constants apply to, either treated or untreated water

A First coefficient in DBP model

a Second coefficient in DBP model, associated with TOC or DOC

b Third coefficient in DBP model, associated with Cl2

c Fourth coefficient in DBP model, associated with Br-

d Fifth coefficient in DBP model, associated with temperature

e Sixth coefficient in DBP model, associated with pH

f Seventh coefficient in DBP model, associated with reaction time

Source

U.S. EPA (2001)

See references list at: https://github.com/BrownandCaldwell-Public/tidywater/wiki/References

---

`dbp_correction` *Data frame of correction factors for estimating DBP formation as a function of location*

Description

A dataset containing correction factors for calculating DBP formation

Usage

```
dbp_correction
```

Format

A dataframe with 17 rows and 4 columns

ID abbreviation of dbp species

alias full name of dbp species

plant specifies the correction factor for modelling DBP formation within a treatment plant

ds specifies the correction factor for modelling DBP formation within the distribution system

Source

U.S. EPA (2001), Table 5-7

See references list at: https://github.com/BrownandCaldwell-Public/tidywater/wiki/References

`define_water` *Create a water class object given water quality parameters*

Description

This function takes user-defined water quality parameters and creates an S4 "water" class object that forms the input and output of all tidywater models.

Usage

```
define_water(
  ph,
  temp = 25,
  alk,
  tot_hard,
  ca,
  mg,
  na,
  k,
  cl,
  so4,
  tot_ocl = 0,
  tot_po4 = 0,
  tot_nh3 = 0,
  tds,
  cond,
  toc,
  doc,
  uv254,
  br,
  f,
  fe,
  al,
  mn
)
```

Arguments

`ph`	water pH
`temp`	Temperature in degree C
`alk`	Alkalinity in mg/L as CaCO3
`tot_hard`	Total hardness in mg/L as CaCO3
`ca`	Calcium in mg/L Ca2+
`mg`	Magnesium in mg/L Mg2+
`na`	Sodium in mg/L Na+

k	Potassium in mg/L K+
cl	Chloride in mg/L Cl-
so4	Sulfate in mg/L SO42-
tot_ocl	Chlorine in mg/L as Cl2. Used when a starting water has a chlorine residual.
tot_po4	Phosphate in mg/L as PO4 3-. Used when a starting water has a phosphate residual.
tot_nh3	Total ammonia in mg/L as N
tds	Total Dissolved Solids in mg/L (optional if ions are known)
cond	Electrical conductivity in uS/cm (optional if ions are known)
toc	Total organic carbon (TOC) in mg/L
doc	Dissolved organic carbon (DOC) in mg/L
uv254	UV absorbance at 254 nm (cm-1)
br	Bromide in ug/L Br-
f	Fluoride in mg/L F-
fe	Iron in mg/L Fe3+
al	Aluminum in mg/L Al3+
mn	Manganese in ug/L Mn2+

Details

Carbonate balance is calculated and units are converted to mol/L. Ionic strength is determined from ions, TDS, or conductivity. Missing values are handled by defaulting to 0 or NA. Calcium hardness defaults to 65 manually specify all ions in the define_water arguments. The following equations are used to determine ionic strength: Ionic strength (if TDS provided): Crittenden et al. (2012) equation 5-38 Ionic strength (if electrical conductivity provided): Snoeyink & Jenkins (1980) Ionic strength (from ion concentrations): Lewis and Randall (1921), Crittenden et al. (2012) equation 5-37 Temperature correction of dielectric constant (relative permittivity): Harned and Owen (1958), Crittenden et al. (2012) equation 5-45.

Value

A water class object where slots are filled or calculated based on input parameters.

Examples

```
water_missingions <- define_water(ph = 7, temp = 15, alk = 100, tds = 10)
water_defined <- define_water(7, 20, 50, 100, 80, 10, 10, 10, 10, tot_po4 = 1)
```

`define_water_chain` *Apply 'define_water' within a dataframe and output a column of 'water' class to be chained to other tidywater functions*

Description

This function allows `define_water` to be added to a piped data frame. Its output is a 'water' class, and can therefore be chained with "downstream" tidywater functions.

Usage

```
define_water_chain(df, output_water = "defined_water")
```

Arguments

`df`	a data frame containing columns with all the parameters listed in `define_water`
`output_water`	name of the output column storing updated parameters with the class, water. Default is "defined_water".

Details

For large datasets, using 'fn_once' or 'fn_chain' may take many minutes to run. These types of functions use the furrr package for the option to use parallel processing and speed things up. To initialize parallel processing, use 'plan(multisession)' or 'plan(multicore)' (depending on your operating system) prior to your piped code with the 'fn_once' or 'fn_chain' functions. Note, parallel processing is best used when your code block takes more than a minute to run, shorter run times will not benefit from parallel processing.

Value

A data frame containing a water class column.

See Also

`define_water`

Examples

```
library(purrr)
library(furrr)
library(tidyr)
library(dplyr)

example_df <- water_df %>%
  define_water_chain() %>%
  balance_ions_once()

example_df <- water_df %>%
```

```
  define_water_chain(output_water = "This is a column of water") %>%
  balance_ions_once(input_water = "This is a column of water")

# Initialize parallel processing
plan(multisession, workers = 2) # Remove the workers argument to use all available compute
example_df <- water_df %>%
  define_water_chain() %>%
  balance_ions_once()

#' #Optional: explicitly close multisession processing
plan(sequential)
```

`define_water_once` *Apply 'define_water' and output a dataframe*

Description

This function allows `define_water` to be added to a piped data frame. It outputs all carbonate calculations and other parameters in a data frame. tidywater functions cannot be added after this function because they require a 'water' class input.

Usage

```
define_water_once(df)
```

Arguments

`df`	a data frame containing columns with all the parameters listed in `define_water`

Details

For large datasets, using 'fn_once' or 'fn_chain' may take many minutes to run. These types of functions use the furrr package for the option to use parallel processing and speed things up. To initialize parallel processing, use 'plan(multisession)' or 'plan(multicore)' (depending on your operating system) prior to your piped code with the 'fn_once' or 'fn_chain' functions. Note, parallel processing is best used when your code block takes more than a minute to run, shorter run times will not benefit from parallel processing.

Value

A data frame containing columns that were filled or calculated based on define_water.

See Also

`define_water`

Examples

```
library(purrr)
library(furrr)
library(tidyr)
library(dplyr)

example_df <- water_df %>% define_water_once()

# Initialize parallel processing
plan(multisession, workers = 2) # Remove the workers argument to use all available compute
example_df <- water_df %>% define_water_once()

# Optional: explicitly close multisession processing
plan(sequential)
```

---

discons *Dissociation constants and standard enthalpy for weak acids/bases*

Description

Equilibrium constants (k) and corresponding standard enthalpy of reaction values (deltah) for significant acids in water influencing pH at equilibrium. Includes carbonate, sulfate, phosphate, and hypochlorite. Standard enthalpy of reaction is calculated by taking the sum of the enthalpy of formation of each individual component minus the enthalpy of formation of the final product. e.g., the standard enthalpy of reaction for water can be calculated as: deltah_h2o = deltah_f_oh + deltah_f_h - deltah_f_h2o = -230 + 0 - (-285.83) = 55.83 kJ/mol. See MWH (2012) example 5-5 and Benjamin (2002) eq. 2.96.

Usage

```
discons
```

Format

A dataframe with 8 rows and 3 columns

ID Coefficient type

k Equilibrium constant

deltah Standard enthalpy in J/mol

Source

Benjamin (2015) Appendix A.1 and A.2.

See references list at: https://github.com/BrownandCaldwell-Public/tidywater/wiki/References

`dissolve_pb` *Simulate contributions of various lead solids to total soluble lead*

Description

This function takes a water data frame defined by `define_water` and outputs a dataframe of the controlling lead solid and total lead solubility. Lead solid solubility is calculated based on controlling solid. Total dissolved lead species (tot_dissolved_pb, M) are calculated based on lead complex calculations. Some lead solids have two k-constant options. The function will default to the EPA's default constants. The user may change the constants to hydroxypyromorphite = "Zhu" or pyromorphite = "Xie" or laurionite = "Lothenbach"

Usage

```
dissolve_pb(
  water,
  hydroxypyromorphite = "Schock",
  pyromorphite = "Topolska",
  laurionite = "Nasanen"
)
```

Arguments

`water`	Source water object of class "water" created by `define_water`. Water must include alk and is. If po4, cl, and so4 are known, those should also be included.
`hydroxypyromorphite`	defaults to "Schock", the constant, K, developed by Schock et al (1996). Can also use "Zhu".
`pyromorphite`	defaults to "Topolska", the constant, K, developed by Topolska et al (2016). Can also use "Xie".
`laurionite`	defaults to "Nasanen", the constant, K, developed by Nasanen & Lindell (1976). Can also use "Lothenbach".

Details

The solid with lowest solubility will form the lead scale (controlling lead solid).

Make sure that total dissolved solids, conductivity, or ca, na, cl, so4 are used in 'define_water' so that an ionic strength is calculated.

Value

A data frame containing only the controlling lead solid and modeled dissolved lead concentration.

Source

Code is from EPA's TELSS lead solubility dashboard https://github.com/USEPA/TELSS which is licensed under MIT License: Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

Wahman et al. (2021)

See references list at: https://github.com/BrownandCaldwell-Public/tidywater/wiki/References

See Also

define_water

Examples

```
example_pb <- define_water(
  ph = 7.5, temp = 25, alk = 93, cl = 240,
  tot_po4 = 0, so4 = 150, tds = 200
) %>%
  dissolve_pb()
example_pb <- define_water(
  ph = 7.5, temp = 25, alk = 93, cl = 240,
  tot_po4 = 0, so4 = 150, tds = 200
) %>%
  dissolve_pb(pyromorphite = "Xie")
```

---

dissolve_pb_once *Apply 'dissolve_pb' to a dataframe and create a new column with numeric dose*

Description

This function allows dissolve_pb to be added to a piped data frame. Two additional columns will be added to the dataframe; the name of the controlling lead solid, and total dissolved lead (M).

Usage

```
dissolve_pb_once(
  df,
  input_water = "defined_water",
  output_col_solid = "controlling_solid",
  output_col_result = "pb",
  hydroxypyromorphite = "Schock",
  pyromorphite = "Topolska",
```

```
  laurionite = "Nasanen",
  water_prefix = TRUE
)
```

Arguments

df	a data frame containing a water class column, which has already been computed using define_water_chain
input_water	name of the column of water class data to be used as the input. Default is "defined_water".
output_col_solid	name of the output column storing the controlling lead solid. Default is "controlling_solid".
output_col_result	name of the output column storing dissolved lead in M. Default is "pb".
hydroxypyromorphite	defaults to "Schock", the constant, K, developed by Schock et al (1996). Can also use "Zhu".
pyromorphite	defaults to "Topolska", the constant, K, developed by Topolska et al (2016). Can also use "Xie".
laurionite	defaults to "Nasanen", the constant, K, developed by Nasanen & Lindell (1976). Can also use "Lothenbach".
water_prefix	name of the input water used for the calculation, appended to the start of output columns. Default is TRUE. Chenge to FALSE to remove the water prefix from output column names.

Details

The data input comes from a ‘water‘ class column, initialized in define_water or balance_ions. Use the ‘output_col_solid‘ and ‘output_col_result‘ arguments to name the ouput columns for the controlling lead solid and total dissolved lead, respectively. The input ‘water‘ used for the calculation will be appended to the start of these output columns. Omit the input ‘water‘ in the output columns, set ‘water_prefix‘ to FALSE (default is TRUE).

For large datasets, using ‘fn_once‘ or ‘fn_chain‘ may take many minutes to run. These types of functions use the furrr package for the option to use parallel processing and speed things up. To initialize parallel processing, use ‘plan(multisession)‘ or ‘plan(multicore)‘ (depending on your operating system) prior to your piped code with the ‘fn_once‘ or ‘fn_chain‘ functions. Note, parallel processing is best used when your code block takes more than a minute to run, shorter run times will not benefit from parallel processing.

Value

A data frame containing the controlling lead solid and modeled dissolved lead concentration as new columns.

See Also

dissolve_pb

Examples

```
library(purrr)
library(furrr)
library(tidyr)
library(dplyr)

example_df <- water_df %>%
  define_water_chain() %>%
  balance_ions_chain() %>%
  dissolve_pb_once(input_water = "balanced_water")

example_df <- water_df %>%
  define_water_chain() %>%
  dissolve_pb_once(output_col_result = "dissolved_lead", pyromorphite = "Xie")

# Initialize parallel processing
plan(multisession, workers = 2) # Remove the workers argument to use all available compute
example_df <- water_df %>%
  define_water_chain() %>%
  dissolve_pb_once(output_col_result = "dissolved_lead", laurionite = "Lothenbach")

# Optional: explicitly close multisession processing
plan(sequential)
```

---

edwardscoeff *Data frame of Edwards model coefficients*

Description

A dataset containing coefficients from the Edwards (1997) model for coagulation TOC removal.

Usage

```
edwardscoeff
```

Format

A dataframe with 5 rows and 7 columns:

ID Coefficient type

x3 x3 parameter

x2 x2 parameter

x1 x1 parameter

k1 k1 parameter

k2 k2 parameter

b b parameter

Source

Edwards (1997) Table 2.

See references list at: https://github.com/BrownandCaldwell-Public/tidywater/wiki/References

---

`leadsol_constants` *Data frame of equilibrium constants for lead and copper solubility*

Description

A dataset containing equilibrium constants for lead solubility

Usage

```
leadsol_constants
```

Format

A dataframe with 38 rows and 3 columns

Solids:

species_name Name of lead solid or complex with possible _letter to cite different references

constant_name Reference ID for constants

log_value Equilibrium constant log value

source Source for equilibrium constant value

Source

Benjamin (2010)

Lothenbach et al. (1999)

Nasanen & Lindell (1976)

Powell et al. (2009)

Powell et al. (2005)

Schock et al. (1996)

Topolska et al. (2016)

Xie & Giammar (2007)

Zhu et al. (2015)

Wahman et al. (2021)

See references list at: https://github.com/BrownandCaldwell-Public/tidywater/wiki/References

mweights *Molar weights of relevant compounds*

Description

A dataset containing the molar weights of several compounds in g/mol. Column names are lowercase chemical formulas (with no charge), with the exception of the following coagulants: alum = Al2(SO4)3*14H2O, ferricchloride = FeCl3, ferricsulfate = Fe2(SO4)3*8.8H2O,

Usage

```
mweights
```

Format

A dataframe with one row and one column per compound

ozonate_bromate *Calculate bromate formation*

Description

Calculates bromate (BrO3-, ug/L) formation based on selected model. Required arguments include an object of class "water" created by `define_water` ozone dose, reaction time, and desired model. The function also requires additional water quality parameters defined in `define_water` including bromide, DOC or UV254 (depending on the model), pH, alkalinity (depending on the model), and optionally, ammonia (added when defining water using the 'tot_nh3' argument.)

Usage

```
ozonate_bromate(water, dose, time, model = "Ozekin")
```

Arguments

`water`	Source water object of class "water" created by `define_water`
`dose`	Applied ozone dose (mg/L as O3). Results typically valid for 1-10 mg/L, but varies depending on model.
`time`	Reaction time (minutes). Results typically valid for 1-120 minutes, but varies depending on model.
`model`	Model to apply. One of c("Ozekin", "Sohn", "Song", "Galey", "Siddiqui")

Value

A water class object with calculated bromate (ug/L).

Source

Ozekin (1994), Sohn et al (2004), Song et al (1996), Galey et al (1997), Siddiqui et al (1994)

See references list at: https://github.com/BrownandCaldwell-Public/tidywater/wiki/References

Examples

```
example_dbp <- suppressWarnings(define_water(8, 20, 66, toc = 4, uv254 = .2, br = 50)) %>%
  ozonate_bromate(dose = 1.5, time = 5, model = "Ozekin")
example_dbp <- suppressWarnings(define_water(7.5, 20, 66, toc = 4, uv254 = .2, br = 50)) %>%
  ozonate_bromate(dose = 3, time = 15, model = "Sohn")
```

---

ozonate_ct *Determine disinfection credit from ozone.*

Description

This function takes a water defined by define_water and the first order decay curve parameters from an ozone dose and outputs a dataframe of acutal CT, and log removal for giardia, virus, and crypto

Usage

```
ozonate_ct(water, time, dose, kd, baffle)
```

Arguments

water	Source water object of class "water" created by define_water. Water must include ph and temp
time	Retention time of disinfection segment in minutes.
dose	Ozone dose in mg/L. This value can also be the y intercept of the decay curve (often slightly lower than ozone dose.)
kd	First order decay constant. This parameter is optional. If not specified, the default ozone decay equations will be used.
baffle	Baffle factor - unitless value between 0 and 1.

Details

First order decay curve for ozone has the form: ‘residual = dose * exp(kd*time)‘. kd should be a negative number. Actual CT is an integration of the first order curve. The first 30 seconds are removed from the integral to account for instantaneous demand.

Value

A data frame containing actual CT, giardia log removal, virus log removal, and crypto log removal.

Source

USEPA (2020) Equation 4-4 through 4-7 https://www.epa.gov/system/files/documents/2022-02/disprof_bench_3rules_final_

See references list at: https://github.com/BrownandCaldwell-Public/tidywater/wiki/References

See Also

define_water

Examples

```
# Use kd from experimental data (recommended):
define_water(ph = 7.5, temp = 25) %>%
  ozonate_ct(time = 10, dose = 2, kd = -0.5, baffle = 0.9)
define_water(ph = 7.5, alk = 100, doc = 2, uv254 = .02, br = 50) %>%
  ozonate_ct(time = 10, dose = 2, baffle = 0.5)
```

---

pac_toc *Calculate DOC Concentration in PAC system*

Description

Calculates DOC concentration multiple linear regression model found in 2-METHYLISOBORNEOL AND NATURAL ORGANIC MATTER ADSORPTION BY POWDERED ACTIVATED CARBON by HYUKJIN CHO (2007) Required arguments include an object of class "water" created by define_water initial DOC concentration, amount of PAC added to system, contact time with PAC, type of PAC

water must contain DOC or TOC value.

Usage

```
pac_toc(water, dose, time, type = "bituminous")
```

Arguments

water	Source water object of class "water" created by define_water
dose	Applied PAC dose (mg/L). Model results are valid for doses concentrations between 5 and 30 mg/L.
time	Contact time (minutes). Model results are valid for reaction times between 10 and 1440 minutes
type	Type of PAC applied, either "bituminous", "lignite", "wood".

Details

The function will calculate DOC concentration by PAC adsorption in drinking water treatment. UV254 concentrations are predicted based on a linear relationship with DOC.

Value

A water class object with post-PAC predicted DOC and UV254.

Source

See references list at: https://github.com/BrownandCaldwell-Public/tidywater/wiki/References

CHO(2007)

Examples

```
water <- define_water(toc = 2.5, uv254 = .05, doc = 1.5) %>%
  pac_toc(dose = 15, time = 50, type = "wood")
```

`pac_toc_chain` *Apply 'pac_toc' within a data frame and output a column of 'water' class to be chained to other tidywater functions PAC = powdered activated carbon*

Description

This function allows `pac_toc` to be added to a piped data frame. Its output is a 'water' class, and can therefore be used with "downstream" tidywater functions.

Usage

```
pac_toc_chain(
  df,
  input_water = "defined_water",
  output_water = "pac_water",
  dose = 0,
  time = 0,
  type = "bituminous"
)
```

Arguments

`df`	a data frame containing a water class column, which has already been computed using `define_water_chain`. The df may include columns named for the dose, time, and type
`input_water`	name of the column of water class data to be used as the input for this function. Default is "defined_water".
`output_water`	name of the output column storing updated parameters with the class, water. Default is "disinfected_water".
`dose`	Applied PAC dose (mg/L). Model results are valid for doses concentrations between 5 and 30 mg/L.

time	Contact time (minutes). Model results are valid for reaction times between 10 and 1440 minutes
type	Type of PAC applied, either "bituminous", "lignite", "wood".

Details

The data input comes from a ‘water‘ class column, as initialized in define_water.

If the input data frame has a dose, time or type column, the function will use those columns. Note: The function can only take dose, time, and type inputs as EITHER a column or from the function arguments, not both.

tidywater functions cannot be added after this function because they require a ‘water‘ class input.

For large datasets, using ‘fn_once‘ or ‘fn_chain‘ may take many minutes to run. These types of functions use the furrr package for the option to use parallel processing and speed things up. To initialize parallel processing, use ‘plan(multisession)‘ or ‘plan(multicore)‘ (depending on your operating system) prior to your piped code with the ‘fn_once‘ or ‘fn_chain‘ functions. Note, parallel processing is best used when your code block takes more than a minute to run, shorter run times will not benefit from parallel processing.

Value

A data frame containing a water class column with updated DOC, TOC, and UV254 concentrations.

Source

See references list at: https://github.com/BrownandCaldwell-Public/tidywater/wiki/References

CHO(2007)

See Also

pac_toc

Examples

```
library(purrr)
library(furrr)
library(tidyr)
library(dplyr)

example_df <- water_df %>%
  define_water_chain("raw") %>%
  pac_toc_chain(input_water = "raw", dose = 10, time = 20)

example_df <- water_df %>%
  define_water_chain("raw") %>%
  mutate(dose = seq(11, 22, 1), time = 30) %>%
  pac_toc_chain(input_water = "raw")

example_df <- water_df %>%
  define_water_chain("raw") %>%
```

```
  mutate(time = 8) %>%
  pac_toc_chain(
    input_water = "raw", dose = 6, type = "wood"
  )

# Initialize parallel processing
plan(multisession, workers = 2) # Remove the workers argument to use all available compute
example_df <- water_df %>%
  define_water_chain("raw") %>%
  pac_toc_chain(input_water = "raw", dose = 4, time = 8)

# Optional: explicitly close multisession processing
plan(sequential)
```

---

pac_toc_once *Apply 'pac_toc'function within a data frame and output a data frame*

Description

PAC = powdered activated carbon

Usage

```
pac_toc_once(
  df,
  input_water = "defined_water",
  dose = 0,
  time = 0,
  type = "bituminous"
)
```

Arguments

df	a data frame containing a water class column, which has already been computed using define_water_chain. The df may include columns named for the dose, time, and type
input_water	name of the column of water class data to be used as the input for this function. Default is "defined_water".
dose	Applied PAC dose (mg/L). Model results are valid for doses concentrations between 5 and 30 mg/L.
time	Contact time (minutes). Model results are valid for reaction times between 10 and 1440 minutes
type	Type of PAC applied, either "bituminous", "lignite", "wood".

Details

This function allows pac_toc to be added to a piped data frame. Its output is a data frame containing a water with updated TOC, DOC, and UV254.

The data input comes from a 'water' class column, as initialized in define_water.

If the input data frame has a dose, time or type column, the function will use those columns. Note: The function can only take dose, time, and type inputs as EITHER a column or from the function arguments, not both.

tidywater functions cannot be added after this function because they require a 'water' class input.

For large datasets, using 'fn_once' or 'fn_chain' may take many minutes to run. These types of functions use the furrr package for the option to use parallel processing and speed things up. To initialize parallel processing, use 'plan(multisession)' or 'plan(multicore)' (depending on your operating system) prior to your piped code with the 'fn_once' or 'fn_chain' functions. Note, parallel processing is best used when your code block takes more than a minute to run, shorter run times will not benefit from parallel processing.

Value

A data frame with an updated DOC, TOC, and UV254 concentration.

Source

See references list at: https://github.com/BrownandCaldwell-Public/tidywater/wiki/References

CHO(2007)

See Also

pac_toc

Examples

```
library(purrr)
library(furrr)
library(tidyr)
library(dplyr)

example_df <- water_df %>%
  define_water_chain("raw") %>%
  pac_toc_once(input_water = "raw", dose = 10, time = 20)

example_df <- water_df %>%
  define_water_chain("raw") %>%
  mutate(dose = seq(5, 60, 5), time = 30) %>%
  pac_toc_once(input_water = "raw")

example_df <- water_df %>%
  define_water_chain("raw") %>%
  mutate(time = 8) %>%
  pac_toc_once(
    input_water = "raw", dose = 6, type = "wood"
```

```
  )

# Initialize parallel processing
plan(multisession, workers = 2) # Remove the workers argument to use all available compute
example_df <- water_df %>%
  define_water_chain("raw") %>%
  pac_toc_once(input_water = "raw", dose = 4, time = 8)

# Optional: explicitly close multisession processing
plan(sequential)
```

`plot_ions` *Create summary plot of ions from water class*

Description

This function takes a water data frame defined by `define_water` and outputs an ion balance plot.

Usage

```
plot_ions(water)
```

Arguments

`water`	Source water vector created by link function here

Value

A ggplot object displaying the water's ion balance.

Examples

```
water_defined <- define_water(7, 20, 50, 100, 80, 10, 10, 10, 10, tot_po4 = 1)
plot_ions(water_defined)
```

`pluck_water` *Pluck out a single parameter from a 'water' class object*

Description

This function plucks one or more selected parameters from selected columns of 'water' class objects. The names of the output columns will follow the form 'water_parameter' To view all slots as columns, please use one of the 'fn_once' functions or `convert_water`.

Usage

```
pluck_water(df, input_waters = c("defined_water"), parameter)
```

Arguments

df	a data frame containing a water class column, which has already been computed using define_water
input_waters	vector of names of the columns of water class data to be used as the input for this function.
parameter	vector of water class parameters to view outside the water column

Value

A data frame containing columns of selected parameters from a list of water class objects.

See Also

convert_water

Examples

```
library(dplyr)
library(furrr)
library(purrr)
library(tidyr)

pluck_example <- water_df %>%
  define_water_chain() %>%
  pluck_water(parameter = "tot_co3")

pluck_example <- water_df %>%
  define_water_chain() %>%
  balance_ions_chain() %>%
 pluck_water(input_waters = c("defined_water", "balanced_water"), parameter = c("na", "cl"))

plan(multisession, workers = 2) # Remove the workers argument to use all available compute
pluck_example <- water_df %>%
  define_water_chain() %>%
  pluck_water(parameter = c("ph", "alk"))

# Optional: explicitly close multisession processing
plan(sequential)
```

solvecost_chem *Determine chemical cost*

Description

This function takes a chemical dose in mg/L, plant flow, chemical strength, and $/lb and calculates cost.

Usage

```
solvecost_chem(dose, flow, strength = 100, cost, time = "day")
```

Arguments

dose	Chemical dose in mg/L as chemical
flow	Plant flow in MGD
strength	Chemical product strength in percent. Defaults to 100 percent.
cost	Chemical product cost in $/lb
time	Desired output units, one of c("day", "month", "year"). Defaults to "day".

Value

A numeric value for chemical cost, $/time.

Examples

```
alum_cost <- solvecost_chem(dose = 20, flow = 10, strength = 49, cost = .22)
```

solvecost_labor *Determine labor cost*

Description

This function takes number of FTE and annual $/FTE and determines labor cost

Usage

```
solvecost_labor(fte, cost, time = "day")
```

Arguments

fte	Number of FTEs. Can be decimal.
cost	$/year per FTE
time	Desired output units, one of c("day", "month", "year"). Defaults to "day".

Value

A numeric value for labor $/time.

Examples

```
laborcost <- solvecost_labor(1.5, 50000)
```

`solvecost_power` *Determine power cost*

Description

This function takes kW,

Usage

```
solvecost_power(power, utilization = 100, cost, time = "day")
```

Arguments

`power`	Power consumed in kW
`utilization`	Amount of time equipment is running in percent. Defaults to continuous.
`cost`	Power cost in $/kWhr
`time`	Desired output units, one of c("day", "month", "year"). Defaults to "day".

Value

A numeric value for power, $/time.

Examples

```
powercost <- solvecost_power(50, 100, .08)
```

`solvecost_solids` *Determine solids disposal cost*

Description

This function takes coagulant doses in mg/L as chemical, removed turbidity, and cost ($/lb) to determine disposal cost.

Usage

```
solvecost_solids(
  alum = 0,
  ferricchloride = 0,
  ferricsulfate = 0,
  flow,
  turb,
  b = 1.5,
  cost,
  time = "day"
)
```

Arguments

`alum`	Hydrated aluminum sulfate Al2(SO4)3*14H2O + 6HCO3 -> 2Al(OH)3(am) +3SO4 + 14H2O + 6CO2
`ferricchloride`	Ferric Chloride FeCl3 + 3HCO3 -> Fe(OH)3(am) + 3Cl + 3CO2
`ferricsulfate`	Amount of ferric sulfate added in mg/L: Fe2(SO4)3*8.8H2O + 6HCO3 -> 2Fe(OH)3(am) + 3SO4 + 8.8H2O + 6CO2
`flow`	Plant flow in MGD
`turb`	Turbidity removed in NTU
`b`	Correlation factor from turbidity to suspended solids. Defaults to 1.5.
`cost`	Disposal cost in $/lb
`time`	Desired output units, one of c("day", "month", "year"). Defaults to "day".

Value

A numeric value for disposal costs, $/time.

Source

https://water.mecc.edu/courses/ENV295Residuals/lesson3b.htm#:~:text=From

Examples

```
alum_solidscost <- solvecost_solids(alum = 50, flow = 10, turb = 2, cost = 0.05)
```

`solvedose_alk` *Calculate a desired chemical dose for a target alkalinity*

Description

This function calculates the required amount of a chemical to dose based on a target alkalinity and existing water quality. Returns numeric value for dose in mg/L. Uses uniroot on the chemdose_ph function.

Usage

```
solvedose_alk(water, target_alk, chemical)
```

Arguments

`water`	Source water of class "water" created by `define_water`
`target_alk`	The final alkalinity in mg/L as CaCO3 to be achieved after the specified chemical is added.
`chemical`	The chemical to be added. Current supported chemicals include: acids: "hcl", "h2so4", "h3po4", "co2", bases: "naoh", "na2co3", "nahco3", "caoh2", "mgoh2"

Value

A numeric value for the required chemical dose.

See Also

`define_water`

Examples

```
dose_required <- define_water(ph = 7.9, temp = 22, alk = 100, 80, 50) %>%
  solvedose_alk(target_alk = 150, "naoh")
```

`solvedose_alk_once` *Apply 'solvedose_alk' to a dataframe and create a new column with numeric dose*

Description

This function allows `solvedose_alk` to be added to a piped data frame. Its output is a chemical dose in mg/L.

Usage

```
solvedose_alk_once(
  df,
  input_water = "defined_water",
  output_column = "dose_required",
  target_alk = NULL,
  chemical = NULL
)
```

Arguments

df	a data frame containing a water class column, which has already been computed using define_water_chain. The df may include a column with names for each of the chemicals being dosed.
input_water	name of the column of water class data to be used as the input. Default is "defined_water".
output_column	name of the output column storing doses in mg/L. Default is "dose_required".
target_alk	set a goal for alkalinity using the function argument or a data frame column
chemical	select the chemical to be used to reach the desired alkalinity using function argument or data frame column

Details

The data input comes from a ‘water‘ class column, initialized in define_water or balance_ions.

If the input data frame has column(s) named "target_alk" or "chemical", the function will use the column(s) as function argument(s). If these columns aren't present, specify "target_alk" or "chemical" as function arguments. The chemical names must match the chemical names as displayed in solvedose_alk. To see which chemicals can be dosed, see solvedose_alk.

For large datasets, using ‘fn_once‘ or ‘fn_chain‘ may take many minutes to run. These types of functions use the furrr package for the option to use parallel processing and speed things up. To initialize parallel processing, use ‘plan(multisession)‘ or ‘plan(multicore)‘ (depending on your operating system) prior to your piped code with the ‘fn_once‘ or ‘fn_chain‘ functions. Note, parallel processing is best used when your code block takes more than a minute to run, shorter run times will not benefit from parallel processing.

Value

A data frame containing the original data frame and columns for target alkalinity, chemical dosed, and required chemical dose.

See Also

solvedose_alk

Examples

```
library(purrr)
library(furrr)
library(tidyr)
library(dplyr)

example_df <- water_df %>%
  define_water_chain() %>%
  balance_ions_chain() %>%
  mutate(
    target_alk = 300,
    chemical = rep(c("naoh", "na2co3"), 6)
  ) %>%
  solvedose_alk_once()

# When the selected chemical can't raise the alkalinity, the dose_required will be NA
# Eg,soda ash can't bring the alkalinity to 100 when the water's alkalinity is already at 200.

example_df <- water_df %>%
  define_water_chain() %>%
 solvedose_alk_once(input_water = "defined_water", target_alk = 100, chemical = "na2co3")


example_df <- water_df %>%
  define_water_chain() %>%
  mutate(target_alk = seq(100, 210, 10)) %>%
  solvedose_alk_once(chemical = "na2co3")


# Initialize parallel processing
plan(multisession, workers = 2) # Remove the workers argument to use all available compute
example_df <- water_df %>%
  define_water_chain() %>%
  mutate(target_alk = seq(100, 210, 10)) %>%
  solvedose_alk_once(chemical = "na2co3")

# Optional: explicitly close multisession processing
plan(sequential)
```

---

solvedose_ph *Calculate a desired chemical dose for a target pH*

Description

solvedose_ph calculates the required amount of a chemical to dose based on a target pH and existing water quality. The function takes an object of class "water" created by define_water, and user-specified chemical and target pH and returns a numeric value for the required dose in mg/L.

solvedose_ph uses uniroot on chemdose_ph to match the required dose for the requested pH target.

Usage

```
solvedose_ph(water, target_ph, chemical)
```

Arguments

water	Source water of class "water" created by define_water
target_ph	The final pH to be achieved after the specified chemical is added.
chemical	The chemical to be added. Current supported chemicals include: acids: "hcl", "h2so4", "h3po4", "co2"; bases: "naoh", "na2co3", "nahco3", "caoh2", "mgoh2"

Value

A numeric value for the required chemical dose.

See Also

define_water, chemdose_ph

Examples

```
water <- define_water(ph = 7, temp = 25, alk = 10)

# Calculate required dose of lime to reach pH 8
solvedose_ph(water, target_ph = 8, chemical = "caoh2")
```

---

solvedose_ph_once *Apply 'solvedose_ph' to a dataframe and create a new column with numeric dose*

Description

This function allows solvedose_ph to be added to a piped data frame. Its output is a chemical dose in mg/L.

Usage

```
solvedose_ph_once(
  df,
  input_water = "defined_water",
  output_column = "dose_required",
  target_ph = NULL,
  chemical = NULL
)
```

Arguments

df	a data frame containing a water class column, which has already been computed using define_water_chain. The df may include a column with names for each of the chemicals being dosed.
input_water	name of the column of water class data to be used as the input. Default is "defined_water".
output_column	name of the output column storing doses in mg/L. Default is "dose_required".
target_ph	set a goal for pH using the function argument or a data frame column
chemical	select the chemical to be used to reach the desired pH using function argument or data frame column

Details

The data input comes from a ‘water‘ class column, initialized in define_water or balance_ions.

If the input data frame has column(s) named "target_ph" or "chemical", the function will use the column(s) as function argument(s). If these columns aren’t present, specify "target_ph" or "chemical" as function arguments. The chemical names must match the chemical names as displayed in solvedose_ph. To see which chemicals can be dosed, see solvedose_ph.

For large datasets, using ‘fn_once‘ or ‘fn_chain‘ may take many minutes to run. These types of functions use the furrr package for the option to use parallel processing and speed things up. To initialize parallel processing, use ‘plan(multisession)‘ or ‘plan(multicore)‘ (depending on your operating system) prior to your piped code with the ‘fn_once‘ or ‘fn_chain‘ functions. Note, parallel processing is best used when your code block takes more than a minute to run, shorter run times will not benefit from parallel processing.

Value

A data frame containing the original data frame and columns for target pH, chemical dosed, and required chemical dose.

See Also

solvedose_ph

Examples

```
library(purrr)
library(furrr)
library(tidyr)
library(dplyr)

example_df <- water_df %>%
  define_water_chain() %>%
  mutate(
    target_ph = 10,
    chemical = rep(c("naoh", "mgoh2"), 6)
  ) %>%
  solvedose_ph_once(input_water = "defined_water")
```

```
example_df <- water_df %>%
  define_water_chain() %>%
  solvedose_ph_once(input_water = "defined_water", target_ph = 8.8, chemical = "naoh")


example_df <- water_df %>%
  define_water_chain() %>%
  mutate(target_ph = seq(9, 10.1, .1)) %>%
  solvedose_ph_once(chemical = "naoh")


# Initialize parallel processing
plan(multisession, workers = 2) # Remove the workers argument to use all available compute
example_df <- water_df %>%
  define_water_chain() %>%
  mutate(target_ph = seq(9, 10.1, .1)) %>%
  solvedose_ph_once(chemical = "naoh")

# Optional: explicitly close multisession processing
plan(sequential)
```

`solvemass_chem` *Convert mg/L of chemical to lb/day*

Description

This function takes a chemical dose in mg/L, plant flow, and chemical strength and calculates lb/day of product

Usage

```
solvemass_chem(dose, flow, strength = 100)
```

Arguments

`dose`	Chemical dose in mg/L as chemical
`flow`	Plant flow in MGD
`strength`	Chemical product strength in percent. Defaults to 100 percent.

Value

A numeric value for the chemical mass in lb/day.

Examples

```
alum_mass <- solvemass_chem(dose = 20, flow = 10, strength = 49)
```

solveresid_o3 *Determine ozone decay*

Description

This function applies the ozone decay model to a water created by `define_water` from U.S. EPA (2001) equation 5-128.

Usage

```
solveresid_o3(water, dose, time)
```

Arguments

`water`	Source water object of class "water" created by `define_water`.
`dose`	Applied ozone dose in mg/L
`time`	Ozone contact time in minutes

Value

A numeric value for the resiudal ozone.

Source

U.S. EPA (2001)

See reference list at: https://github.com/BrownandCaldwell-Public/tidywater/wiki/References

Examples

```
ozone_resid <- define_water(7, 20, 100, doc = 2, toc = 2.2, uv254 = .02, br = 50) %>%
  solveresid_o3(dose = 2, time = 10)
```

solveresid_o3_once *Apply 'solveresid_o3' to a data frame and create a new column with residual ozone dose*

Description

This function allows `solveresid_o3` to be added to a piped data frame. Once additional column will be added to the data frame; the residual ozone dose (mg/L)

Usage

```
solveresid_o3_once(df, input_water = "defined_water", dose = 0, time = 0)
```

Arguments

df	a data frame containing a water class column, which has already been computed using define_water_chain
input_water	name of the column of Water class data to be used as the input for this function. Default is "defined_water".
dose	Applied ozone dose in mg/L
time	Ozone contact time in minutes

Details

The data input comes from a ‘water‘ class column, initialized in define_water or balance_ions.

For large datasets, using ‘fn_once‘ or ‘fn_chain‘ may take many minutes to run. These types of functions use the furrr package for the option to use parallel processing and speed things up. To initialize parallel processing, use ‘plan(multisession)‘ or ‘plan(multicore)‘ (depending on your operating system) prior to your piped code with the ‘fn_once‘ or ‘fn_chain‘ functions. Note, parallel processing is best used when your code block takes more than a minute to run, shorter run times will not benefit from parallel processing.

Value

A data frame containing the original data frame and columns for ozone dosed, time, and ozone residual.

Examples

```
library(dplyr)
ozone_resid <- water_df %>%
  mutate(br = 50) %>%
  define_water_chain() %>%
  solveresid_o3_once(dose = 2, time = 10)

ozone_resid <- water_df %>%
  mutate(br = 50) %>%
  define_water_chain() %>%
  mutate(
    dose = seq(1, 12, 1),
    time = seq(2, 24, 2)
  ) %>%
  solveresid_o3_once()
```

---

summarize_wq *Create summary table from water class*

Description

This function takes a water data frame defined by `define_water` and outputs a formatted summary table of specified water quality parameters.

`summarise_wq()` and `summarize_wq()` are synonyms.

Usage

```
summarize_wq(water, params = c("general"))

summarise_wq(water, params = c("general"))
```

Arguments

`water`	Source water vector created by `define_water`.
`params`	List of water quality parameters to be summarized. Options include "general", "ions", "corrosion", and "dbps". Defaults to "general" only.

Details

Use `calculate_corrosion` for corrosivity indicators and `chemdose_dbp` for modeled DBP concentrations.

Value

A knitr_kable table of specified water quality parameters.

Examples

```
# Summarize general parameters
water_defined <- define_water(7, 20, 50, 100, 80, 10, 10, 10, 10, tot_po4 = 1)
summarize_wq(water_defined)

# Summarize major cations and anions
summarize_wq(water_defined, params = list("ions"))
```

`water_df` *Data frame of water quality parameters*

Description

A dataset containing fabricated water quality to use as tidywater inputs. Parameters are set to reasonable water quality ranges. Parameters are as follows:

Usage

```
water_df
```

Format

A dataframe with 12 rows and 11 columns:

ph pH in standard units (SU)

temp Temperature in degree C

alk Alkalinity in mg/L as CaCO3

tot_hard Total hardness in mg/L as CaCO3

ca_hard Calcium hardness in mg/L as CaCO3

na Sodium in mg/L Na+

k Potassium in mg/L K+

cl Chloride in mg/L Cl-

so4 Sulfate in mg/L SO42-

tot_ocl Total chlorine in mg/L as Cl2

tot_po4 Total phosphate in mg/L as PO42-

Source

Fabricated for use in examples.

Index