Package ‘tidywater’

May 17, 2025

Type Package

Title Water Quality Models for Drinking Water Treatment Processes

Version 0.8.2

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 de-
signed 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., Bor-
chardt, 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, forcats, knitr, ggplot2, ggrepel, magrittr,
purrr, furrr, methods, rlang, deSolve

RoxygenNote 7.3.2

Depends R (>= 2.10)

Suggests rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

Date 2025-05-16

NeedsCompilation no

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

Chris Corwin [aut] (ORCID: <https://orcid.org/0000-0002-9462-0352>),
Rachel Merrifield [ctb],
Mayuri Namasivayam [ctb],
Phoebe Chen [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 2025-05-17 07:50:02 UTC

Contents

`balance_ions` *Add an ion to balance overall charge in a water*

Description

This function takes a water defined by define_water and balances charge. For a single water use `balance_ions`; for a dataframe use `balance_ions_chain`. Use pluck_water to get values from the output water as new dataframe columns.

Usage

```
balance_ions(water, anion = "cl", cation = "na")

balance_ions_chain(
  df,
  input_water = "defined_water",
  output_water = "balanced_water",
  anion = "cl",
  cation = "na"
)
```

Arguments

`water`	Water created with define_water, which may have some ions set to 0 when unknown
`anion`	Selected anion to use to for ion balance when more cations are present. Defaults to "cl". Choose one of c("cl", "so4").
`cation`	Selected cation to use to for ion balance when more anions are present. Defaults to "na". Choose one of c("na", "k", "ca", or "mg").
`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 water classes. Default is "balanced_water".

Details

If more cations are needed, sodium will be added. User may specify which cation ("na", "k", "ca", or "mg") to use for balancing. 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 unless specified in the cation argument. Anions are added by default with chloride. User may specify which anion ("cl", "so4") to use for balancing. This function is purely mathematical. User should always check the outputs to make sure values are reasonable for the input source water.

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

`balance_ions` returns a single water class object with updated ions to balance water charge.

`balance_ions_chain` returns a dataframe with a new column with the ion balanced water

Examples

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

water_defined <- define_water(7, 20, 50, tot_hard = 150) %>%
  balance_ions(anion = "so4")

example_df <- water_df %>%
  define_water_chain() %>%
  balance_ions_chain(anion = "so4", cation = "ca")


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

# 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). For a single water use `biofilter_toc`; for a dataframe use `biofilter_toc_chain`. Use pluck_water to get values from the output water as new dataframe columns. For most arguments in the `_chain` helper "use_col" default looks for a column of the same name in the dataframe. The argument can be specified directly in the function instead or an unquoted column name can be provided.

Usage

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

biofilter_toc_chain(
  df,
  input_water = "defined_water",
  output_water = "biofiltered_water",
  ebct = "use_col",
  ozonated = "use_col"
)
```

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.
`df`	a data frame containing a water class column, which has already been computed using define_water_chain. The df may include a column indicating the EBCT or whether the water is ozonated.
`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 "biofiltered_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

`biofilter_toc` returns water class object with modeled DOC removal from biofiltration.

`biofilter_toc_chain` returns a data frame containing a water class column with updated DOC, TOC, and UV254 water slots.

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)


library(purrr)
library(tidyr)
library(dplyr)

example_df <- water_df %>%
  define_water_chain() %>%
  biofilter_toc_chain(input_water = "defined_water", ebct = 10, ozonated = FALSE)

example_df <- water_df %>%
  define_water_chain() %>%
  mutate(
    BiofEBCT = c(10, 10, 10, 15, 15, 15, 20, 20, 20, 25, 25, 25),
    ozonated = c(rep(TRUE, 6), rep(FALSE, 6))
  ) %>%
  biofilter_toc_chain(input_water = "defined_water", ebct = BiofEBCT)


# Initialize parallel processing
library(furrr)
plan(multisession, workers = 2) # Remove the workers argument to use all available compute
example_df <- water_df %>%
  define_water_chain() %>%
  biofilter_toc_chain(input_water = "defined_water", ebct = c(10, 20))

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

`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. For a single blend use `blend_waters`; for a dataframe use `blend_waters_chain`. Use pluck_water to get values from the output water as new dataframe columns.

Usage

```
blend_waters(waters, ratios)

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

Arguments

`waters`	Vector of source waters created by define_water. For `chain` function, this can include quoted column names and/or existing single water objects unquoted.
`ratios`	Vector of ratios in the same order as waters. (Blend ratios must sum to 1). For `chain` function, this can also be a list of quoted column names.
`df`	a data frame containing a water class column, which has already been computed using define_water_chain
`output_water`	name of output column storing updated parameters with the class, water. Default is "blended_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

`blend_waters` returns a water class object with blended water quality parameters.

`blend_waters_chain` returns a data frame with a water class column containing blended water quality

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))


library(dplyr)

example_df <- water_df %>%
  slice_head(n = 3) %>%
  define_water_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"
  )


waterA <- define_water(7, 20, 100, tds = 100)
example_df <- water_df %>%
  slice_head(n = 3) %>%
  define_water_chain() %>%
  blend_waters_chain(waters = c("defined_water", waterA), ratios = c(.8, .2))

# Initialize parallel processing
library(furrr)
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)
```

---

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

This function takes an object created by define_water and calculates corrosion and scaling indices. For a single water, use `calculate_corrosion`; to apply the calculations to a dataframe, use `calculate_corrosion_once`.

Usage

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

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

calculate_corrosion_chain(
  df,
  input_water = "defined_water",
  output_water = "corrosion_indices",
  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"
`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".
`output_water`	name of output column storing updated indices with the class, water. Default is "corrosion_indices".

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.

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

`calculate_corrosion` returns a single water class object with updated corrosion and scaling index slots.

`calculate_corrosion_once` returns a data frame containing specified corrosion and scaling indices as columns.

`calculate_corrosion_chain` returns a data frame containing a water class column 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`

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"))


library(dplyr)

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

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

# Initialize parallel processing
library(furrr)
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_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_chloramine — *Calculate chlorine and chloramine Concentrations with the breakpoint cblorination approach*

Description

`chemdose_chloramine`, adopted from the U.S. EPA's Chlorine Breakpoint Curve Simulator, calculates chlorine and chlorinamine concentrations based on the two papers Jafvert & Valentine (Environ. Sci. Technol., 1992, 26 (3), pp 577-586) and Vikesland et al. (Water Res., 2001, 35 (7), pp 1766-1776). Required arguments include an object of class "water" created by `define_water`, chlorine dose, and reaction time. The function also requires additional water quality parameters defined in `define_water` including temperature, pH, and alkalinity.

Usage

```
chemdose_chloramine(
  water,
  time,
  cl2 = 0,
  nh3 = 0,
  use_free_cl_slot = FALSE,
  use_tot_nh3_slot = FALSE
)

chemdose_chloramine_chain(
  df,
  input_water = "defined_water",
  output_water = "chlorinated_water",
  time = "use_col",
  cl2 = "use_col",
  nh3 = "use_col",
  use_free_cl_slot = "use_col",
  use_tot_nh3_slot = "use_col"
)
```

Arguments

`water`	Source water object of class "water" created by `define_water`
`time`	Reaction time (minutes). Time defined needs to be greater or equal to 1 minute.
`cl2`	Applied chlorine dose (mg/L as Cl2), defaults to 0.If not specified, use free_chlorine slot in water.
`nh3`	Applied ammonia dose (mg/L as N), defaults to 0. If not specified, use tot_nh3 slot in water.
`use_free_cl_slot`	Defaults to FALSE. If TRUE, uses free_chlorine slot in water. If TRUE AND there is a cl2 input, both the free_chlorine water slot and chlorine dose will be used.
`use_tot_nh3_slot`	Defaults to FALSE. If TRUE, uses tot_nh3 slot in water. If TRUE AND there is a nh3 input, both the tot_nh3 water slot and ammonia dose will be used.
`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_dose), and a column for time in hours.
`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 "chlorinated_water".

Details

The function will calculate the chlorine and chloramine concentrations and update the "water" class object proceed to the next steps of the treatment chain.

Value

`chemdose_chloramine` returns a water class object with predicted chlorine and chloramine concentrations.

`chemdose_chloramine_chain` returns a data frame containing water class column with updated chlorine residuals.

Source

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

Examples

```
breakpoint <- define_water(7.5, 20, 65, free_chlorine = 5, tot_nh3 = 1) %>%
  chemdose_chloramine(time = 40, cl2 = 2, nh3 = 1, use_free_cl_slot = TRUE)


library(dplyr)

breakpoint <- water_df %>%
  mutate(free_chlorine = 5, tot_nh3 = 1) %>%
  slice_head(n = 3) %>%
  define_water_chain() %>%
  mutate(
    time = 8,
    cl2dose = c(2, 3, 4)
  ) %>%
  chemdose_chloramine_chain(
    input_water = "defined_water",
    cl2 = cl2dose,
    use_free_cl_slot = TRUE,
    use_tot_nh3_slot = TRUE
  )


# Initialize parallel processing
library(furrr)
plan(multisession, workers = 2) # Remove the workers argument to use all available compute

example_df <- water_df %>%
  define_water_chain() %>%
  chemdose_chloramine_chain(
    input_water = "defined_water", cl2 = c(2, 4), nh3 = 2, time = 8
  )

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

chemdose_chlordecay *Calculate chlorine decay*

Description

calculates the decay of chlorine or chloramine based on the U.S. EPA's Water Treatment Plant Model (U.S. EPA, 2001). For a single water use `chemdose_chlordecay`; for a dataframe use `chemdose_chlordecay_chain`. For most arguments in the `_chain` helper "use_col" default looks for a column of the same name in the dataframe. The argument can be specified directly in the function instead or an unquoted column name can be provided.

Usage

```
chemdose_chlordecay(
  water,
  cl2_dose,
  time,
  treatment = "raw",
  cl_type = "chlorine",
  use_chlorine_slot = FALSE
)

chemdose_chlordecay_chain(
  df,
  input_water = "defined_water",
  output_water = "disinfected_water",
  cl2_dose = "use_col",
  time = "use_col",
  treatment = "use_col",
  cl_type = "use_col",
  use_chlorine_slot = "use_col"
)
```

Arguments

`water`	Source water object of class "water" created by `define_water`
`cl2_dose`	Applied chlorine or chloramine dose (mg/L as cl2). Model results are valid for doses between 0.995 and 41.7 mg/L for raw water, and for doses between 1.11 and 24.7 mg/L for coagulated water.
`time`	Reaction time (hours). Chlorine decay model results are valid for reaction times between 0.25 and 120 hours.Chloramine decay model does not have specified boundary conditions.
`treatment`	Type of treatment applied to the water. Options include "raw" for no treatment (default), "coag" for water that has been coagulated or softened.
`cl_type`	Type of chlorination applied, either "chlorine" (default) or "chloramine".

`use_chlorine_slot`	Defaults to FALSE. When TRUE, uses either free_chlorine or combined_chlorine slot in water (depending on cl_type). If 'cl2_dose' argument, not specified, chlorine slot will be used. If 'cl2_dose' specified and use_chlorine_slot is TRUE, all chlorine will be summed.
`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 in hours.
`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".

Details

Required arguments include an object of class "water" created by define_water, applied chlorine/chloramine dose, type, reaction time, and treatment applied (options include "raw" for no treatment, or "coag" for coagulated water). The function also requires additional water quality parameters defined in define_water including TOC and UV254. The output is a new "water" class with the calculated total chlorine value stored in the 'free_chlorine' or 'combined_chlorine' slot, depending on what type of chlorine is dosed. When modeling residual concentrations through a unit process, the U.S. EPA Water Treatment Plant Model applies a correction factor based on the influent and effluent residual concentrations (see U.S. EPA (2001) equation 5-118) that may need to be applied manually by the user based on the output.

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

`chemdose_chlordecay` returns an updated disinfectant residual in the free_chlorine or combined_chlorine water slot in units of M. Use convert_units to convert to mg/L.

`chemdose_chlordecay_chain` returns a data frame containing a water class column with updated chlorine residuals.

Source

U.S. EPA (2001)

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

Examples

```
example_cl2 <- define_water(8, 20, 66, toc = 4, uv254 = 0.2) %>%
  chemdose_chlordecay(cl2_dose = 2, time = 8)
```

```
example_cl2 <- define_water(8, 20, 66, toc = 4, uv254 = 0.2, free_chlorine = 3) %>%
  chemdose_chlordecay(cl2_dose = 2, time = 8, use_chlorine_slot = TRUE)


library(dplyr)

example_df <- water_df %>%
  mutate(br = 50) %>%
  define_water_chain() %>%
  chemdose_chlordecay_chain(input_water = "defined_water", cl2_dose = 4, time = 8)

example_df <- water_df %>%
  mutate(
    br = 50,
    free_chlorine = 2
  ) %>%
  define_water_chain() %>%
  mutate(
    cl2_dose = seq(2, 24, 2),
    ClTime = 30
  ) %>%
  chemdose_chlordecay_chain(
    time = ClTime,
    use_chlorine_slot = TRUE,
    treatment = "coag",
    cl_type = "chloramine"
  )

# Initialize parallel processing
library(furrr)
plan(multisession, workers = 2) # Remove the workers argument to use all available compute
example_df <- water_df %>%
  mutate(br = 50) %>%
  define_water_chain() %>%
  chemdose_chlordecay_chain(cl2_dose = 4, time = 8)

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

chemdose_dbp *Calculate DBP formation*

Description

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. For a single water use `chemdose_dbp`; for a dataframe use `chemdose_dbp_chain`. For most arguments in the `_chain` helper "use_col" default looks for a column of the same name in the dataframe. The argument can be specified directly in the function instead or an unquoted column name can be provided.

Usage

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

chemdose_dbp_chain(
  df,
  input_water = "defined_water",
  output_water = "disinfected_water",
  cl2 = "use_col",
  time = "use_col",
  treatment = "use_col",
  cl_type = "use_col",
  location = "use_col"
)
```

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".
`df`	a data frame containing a water class column, which has already been computed using define_water. The df may include columns for the other function arguments.
`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".

Details

The function will calculate haloacetic acids (HAA) as HAA5, and total trihalomethanes (TTHM). Use `summarize_wq(water, params = c("dbps"))` to quickly tabulate the results.

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

`chemdose_dbp` returns a single water class object with predicted DBP concentrations.

`chemdose_dbp_chain` returns a data frame containing a water class column 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 <- define_water(8, 20, 66, toc = 4, uv254 = .2, br = 50) %>%
  chemdose_dbp(cl2 = 2, time = 8)
example_dbp <- define_water(7.5, 20, 66, toc = 4, uv254 = .2, br = 50) %>%
  chemdose_dbp(cl2 = 3, time = 168, treatment = "coag", location = "ds")


library(dplyr)

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

example_df <- water_df %>%
  mutate(br = 50) %>%
  slice_sample(n = 3) %>%
  define_water_chain() %>%
  mutate(
    cl2_dose = c(2, 3, 4),
    time = 30
  ) %>%
  chemdose_dbp_chain(cl2 = cl2_dose, treatment = "coag", location = "ds", cl_type = "chloramine")

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

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

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

Description

Calculates the new pH, alkalinity, and ion balance of a water based on different chemical additions. For a single water use `chemdose_ph`; for a dataframe use `chemdose_ph_chain`. Use pluck_water to get values from the output water as new dataframe columns. For most arguments in the `_chain` helper "use_col" default looks for a column of the same name in the dataframe. The argument can be specified directly in the function instead or an unquoted column name can be provided.

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
)

chemdose_ph_chain(
  df,
  input_water = "defined_water",
  output_water = "dosed_chem_water",
  hcl = "use_col",
  h2so4 = "use_col",
  h3po4 = "use_col",
  co2 = "use_col",
  naoh = "use_col",
  na2co3 = "use_col",
  nahco3 = "use_col",
  caoh2 = "use_col",
  mgoh2 = "use_col",
  cacl2 = "use_col",
  cl2 = "use_col",
  naocl = "use_col",
  nh4oh = "use_col",
  nh42so4 = "use_col",
  alum = "use_col",
  ferricchloride = "use_col",
  ferricsulfate = "use_col",
  ach = "use_col",
  caco3 = "use_col",
  softening_correction = "use_col"
)
```

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
`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".

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.

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

`chemdose_ph` returns a water class object with updated pH, alkalinity, and ions post-chemical addition.

chemdose_ph_chain returns a data frame containing a water class column 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


library(dplyr)

example_df <- water_df %>%
  slice_head(n = 3) %>%
  define_water_chain() %>%
  chemdose_ph_chain(input_water = "defined_water", naoh = 5)

example_df <- water_df %>%
  slice_head(n = 3) %>%
  define_water_chain() %>%
  mutate(
    hcl = c(2, 4, 6),
    Caustic = 20
  ) %>%
  chemdose_ph_chain(mgoh2 = c(20, 55), co2 = 4, naoh = Caustic)


# Initialize parallel processing
library(furrr)
plan(multisession, workers = 2) # Remove the workers argument to use all available compute
example_df <- water_df %>%
    define_water_chain() %>%
    chemdose_ph_chain(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. For a single water use `chemdose_toc`; for a dataframe use `chemdose_toc_chain`. Use pluck_water to get values from the output water as new dataframe columns. For most arguments in the `_chain` helper "use_col" default looks for a column of the same name in the dataframe. The argument can be specified directly in the function instead or an unquoted column name can be provided.

Usage

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

chemdose_toc_chain(
  df,
  input_water = "defined_water",
  output_water = "coagulated_water",
  alum = "use_col",
  ferricchloride = "use_col",
  ferricsulfate = "use_col",
  coeff = "use_col"
)
```

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
`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".

Value

`chemdose_toc` returns a single water class object with an updated DOC, TOC, and UV254 concentration.

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

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
  ))
```

```
library(dplyr)

example_df <- water_df %>%
  define_water_chain() %>%
  chemdose_toc_chain(input_water = "defined_water", alum = 30)

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


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

`cl2coeffs` *Data frame of Cl2 decay coefficients*

Description

A dataset containing coefficients for calculating Cl2 decay

Usage

```
cl2coeffs
```

Format

A dataframe with 3 rows and 4 columns

treatment Specifies the treatment applied to the water

a Coefficient in chlorine decay model, associated with chlorine dose and time

b Coefficient in chlorine decay model, associated with chlorine dose & organics

c Exponent in chlorine decay model, associated with chlorine dose & organics

Source

U.S. EPA (2001)

`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.

Use convert_water to keep all slots in the same units as the water.

Use convert_watermg to convert to more typical units. Converts the following slots from M to mg/L: na, ca, mg, k, cl, so4, hco3, co3, h2po4, hpo4, po4, ocl, bro3, f, fe, al. Converts these slots to ug/L: br, mn. All other values remain unchanged.

Usage

```
convert_water(water)

convert_watermg(water)
```

Arguments

`water`	A water class object

Value

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

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

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)

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,
  free_chlorine = 0,
  combined_chlorine = 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-
`free_chlorine`	Free chlorine in mg/L as Cl2. Used when a starting water has a free chlorine residual.
`combined_chlorine`	Combined chlorine (chloramines) in mg/L as Cl2. Used when a starting water has a chloramine 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 defaults to 65 percent of the total hardness when not specified. DOC defaults to 95 percent of TOC.

Value

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

Source

Crittenden et al. (2012) equation 5-38 - ionic strength from TDS

Snoeyink & Jenkins (1980) - ionic strength from conductivity

Lewis and Randall (1921), Crittenden et al. (2012) equation 5-37 - ionic strength from ion concentrations

Harned and Owen (1958), Crittenden et al. (2012) equation 5-45 - Temperature correction of dielectric constant (relative permittivity)

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

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

example_df <- water_df %>%
  define_water_chain(output_water = "This is a column of water") %>%
  balance_ions_chain(input_water = "This is a column of water")

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

#' #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

```
example_df <- water_df %>%
  define_water_once()
```

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. For a single water, use `dissolve_pb`; to apply the model to a dataframe, use `dissolve_pb_once`. For most arguments, the `_chain` and `_once` helpers "use_col" default looks for a column of the same name in the dataframe. The argument can be specified directly in the function instead or an unquoted column name can be provided.

Usage

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

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

`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".
`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".
`water_prefix`	name of the input water used for the calculation, appended to the start of output columns. Default is TRUE. Change to FALSE to remove the water prefix from output column names.

Details

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

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

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

`dissolve_pb` returns a one row data frame containing only the controlling lead solid and modeled dissolved lead concentration.

`dissolve_pb_once` returns a data frame containing the controlling lead solid and modeled dissolved lead concentration as new columns.

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`

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")


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

```
# Initialize parallel processing
library(furrr)
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 lower-case 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.) For a single water use `ozonate_bromate`; for a dataframe use `ozonate_bromate_chain`. Use pluck_water to get values from the output water as new dataframe columns. For most arguments in the `_chain` helper "use_col" default looks for a column of the same name in the dataframe. The argument can be specified directly in the function instead or an unquoted column name can be provided.

Usage

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

ozonate_bromate_chain(
  df,
  input_water = "defined_water",
  output_water = "ozonated_water",
  dose = "use_col",
  time = "use_col",
  model = "use_col"
)
```

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")
`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 in minutes.
`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 "ozonated_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

`ozonate_bromate` returns a single water class object with calculated bromate (ug/L).

`ozonate_bromate_chain` returns a data frame containing a water class column with updated bro3.

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


library(dplyr)

example_df <- water_df %>%
  slice_head(n = 6) %>%
  mutate(br = 50) %>%
  define_water_chain() %>%
  mutate(
    dose = c(seq(.5, 3, .5)),
    OzoneTime = 30
  ) %>%
  ozonate_bromate_chain(time = OzoneTime, model = "Sohn")


# Initialize parallel processing
library(furrr)
plan(multisession, workers = 2) # Remove the workers argument to use all available compute
example_df <- water_df %>%
  mutate(br = 50) %>%
  define_water_chain() %>%
  ozonate_bromate_chain(dose = 4, time = 8)

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

`pactoccoeffs` *Data frame of PAC TOC model coefficients*

Description

A dataset containing coefficients for calculating PAC TOC removal

Usage

```
pactoccoeffs
```

Format

A dataframe with 4 rows and 3 columns

pactype Specifies PAC type

A Constant in the PAC model

a Coefficient in PAC model, associated with DOC0

b Coefficient in PAC model, associated with dose

c Coefficient in PAC model, associated with time

Source

Cho (2007)

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) For a single water use `pac_toc`; for a dataframe use `pac_toc_chain`. Use pluck_water to get values from the output water as new dataframe columns. For most arguments in the `_chain` helper "use_col" default looks for a column of the same name in the dataframe. The argument can be specified directly in the function instead or an unquoted column name can be provided.

water must contain DOC or TOC value.

Usage

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

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

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".
`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 "pac_water".

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.

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

`pac_toc` returns a water class object with updated DOC, TOC, and UV254 slots.

`pac_toc_chain` returns a data frame containing a water class column with updated DOC, TOC, and UV254 slots

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")


library(dplyr)

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


# Initialize parallel processing
library(furrr)
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)
```

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

`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`

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. Can also specify "all" to get all non-NA water slots.

Value

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

See Also

convert_water

Examples

```
pluck_example <- water_df %>%
  define_water_chain("raw") %>%
  pluck_water(input_waters = c("raw"), parameter = c("hco3", "doc"))


library(furrr)
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)

library(dplyr)
cost_data <- tibble(
  dose = seq(10, 50, 10),
  flow = 10
) %>%
  mutate(costs = solvecost_chem(dose = dose, flow = flow, 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)

library(dplyr)
cost_data <- tibble(
  fte = seq(1, 10, 1)
) %>%
  mutate(costs = solvecost_labor(fte = fte, cost = .08))
```

`solvecost_power` *Determine power cost*

Description

This function takes kW, % utilization, $/kWhr and determines power cost.

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)

library(dplyr)
cost_data <- tibble(
  power = seq(10, 50, 10),
  utilization = 80
) %>%
  mutate(costs = solvecost_power(power = power, utilization = utilization, cost = .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%20the%20diagram%2C%20for%20example

Examples

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

library(dplyr)
cost_data <- tibble(
  alum = seq(10, 50, 10),
  flow = 10
) %>%
  mutate(costs = solvecost_solids(alum = alum, flow = flow, turb = 2, cost = 0.05))
```

solvect_chlorine *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`). For a single water, use `solvect_chlorine`; to apply the model to a dataframe, use `solvect_chlorine_once`. For most arguments, the `_chain` and `_once` helpers "use_col" default looks for a column of the same name in the dataframe. The argument can be specified directly in the function instead or an unquoted column name can be provided.

Usage

```
solvect_chlorine(water, time, residual, baffle, use_free_cl_slot = FALSE)

solvect_chlorine_once(
  df,
  input_water = "defined_water",
  time = "use_col",
  residual = "use_col",
  baffle = "use_col",
  use_free_cl_slot = FALSE,
  water_prefix = TRUE
)
```

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.
`use_free_cl_slot`	Defaults to FALSE. When TRUE, uses free_chlorine slot in water instead of 'residual' argument.
`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".
`water_prefix`	name of the input water used for the calculation will be appended to the start of output columns. Default is TRUE.

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.

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

`solvect_chlorine` returns a data frame containing required CT (mg/L*min), actual CT (mg/*Lmin), and giardia log removal.

`solvect_chlorine_once` returns a data frame containing the original data frame and columns for 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

Examples

```
example_ct <- define_water(ph = 7.5, temp = 25) %>%
  solvect_chlorine(time = 30, residual = 1, baffle = 0.7)
library(dplyr)
ct_calc <- water_df %>%
  define_water_chain() %>%
  solvect_chlorine_once(residual = 2, time = 10, baffle = .5)

chlor_resid <- water_df %>%
  mutate(br = 50) %>%
  define_water_chain() %>%
  mutate(
    residual = seq(1, 12, 1),
    time = seq(2, 24, 2),
    baffle = 0.7
  ) %>%
  solvect_chlorine_once()
```

solvect_o3 *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 actual CT, and log removal for giardia, virus, and crypto. For a single water, use `solvect_o3`; to apply the model to a dataframe, use `solvect_o3_once`. For most arguments, the `_once` helper "use_col" default looks for a column of the same name in the dataframe. The argument can be specified directly in the function instead or an unquoted column name can be provided.

Usage

```
solvect_o3(water, time, dose, kd, baffle)

solvect_o3_once(
  df,
  input_water = "defined_water",
  time = "use_col",
  dose = "use_col",
  kd = "use_col",
  baffle = "use_col",
  water_prefix = TRUE
)
```

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.
`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".
`water_prefix`	name of the input water used for the calculation will be appended to the start of output columns. Default is TRUE.

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.

When kd is not specified, a default decay curve is used from the Water Treatment Plant Model (2002). This model does not perform well for ozone decay, so specifying the decay curve is recommended.

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

`solvect_o3` returns a data frame containing actual CT (mg/L*min), giardia log removal, virus log removal, and crypto log removal.

`solvect_o3_once` returns a data frame containing the original data frame and columns for required CT, actual CT, and giardia 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

Examples

```
# Use kd from experimental data (recommended):
define_water(ph = 7.5, temp = 25) %>%
  solvect_o3(time = 10, dose = 2, kd = -0.5, baffle = 0.9)
define_water(ph = 7.5, alk = 100, doc = 2, uv254 = .02, br = 50) %>%
  solvect_o3(time = 10, dose = 2, baffle = 0.5)


library(dplyr)
ct_calc <- water_df %>%
  mutate(br = 50) %>%
  define_water_chain() %>%
  mutate(
    dose = 2,
    O3time = 10,
  ) %>%
  solvect_o3_once(time = O3time, baffle = .7)
```

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. For a single water, use `solvedose_alk`; to apply the model to a dataframe, use `solvedose_alk_once`. For most arguments, the _once helper "use_col" default looks for a column of the same name in the dataframe. The argument can be specified directly in the function instead or an unquoted column name can be provided.

Usage

```
solvedose_alk(water, target_alk, chemical)

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

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"
`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".

Details

`solvedose_alk` uses `stats::uniroot()` on chemdose_ph to match the required dose for the requested alkalinity target.

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

`solvedose_alk` returns a numeric value for the required chemical dose.

`solvedose_alk_once` returns a data frame containing the original data frame and columns for target alkalinity, chemical dosed, and required chemical dose.

See Also

solvedose_ph

Examples

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

library(dplyr)

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


# Initialize parallel processing
library(furrr)
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

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", and user-specified chemical and target pH and returns a numeric value for the required dose in mg/L. For a single water, use `solvedose_ph`; to apply the model to a dataframe, use `solvedose_ph_once`. For most arguments, the `_once` helper "use_col" default looks for a column of the same name in the dataframe. The argument can be specified directly in the function instead or an unquoted column name can be provided.

Usage

```
solvedose_ph(water, target_ph, chemical)

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

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"
`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".

Details

`solvedose_ph` uses `stats::uniroot()` on chemdose_ph to match the required dose for the requested pH target.

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 numeric value for the required chemical dose.

`solvedose_ph_once` returns a data frame containing the original data frame and columns for target pH, chemical dosed, and required chemical dose.

See Also

chemdose_ph, solvedose_alk

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")


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


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

# 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 in MGD, 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)

library(dplyr)
mass_data <- tibble(
  dose = seq(10, 50, 10),
  flow = 10
) %>%
  mutate(mass = solvemass_chem(dose = dose, flow = flow, strength = 49))
```

`solvemass_solids` *Determine solids lb/day*

Description

This function takes coagulant doses in mg/L as chemical, removed turbidity, and plant flow as MGD to determine solids production.

Usage

```
solvemass_solids(
  alum = 0,
  ferricchloride = 0,
  ferricsulfate = 0,
  flow,
  turb,
  b = 1.5
)
```

Arguments

`alum`	Amount of hydrated aluminum sulfate added in mg/L as chemical: Al2(SO4)3*14H2O + 6HCO3 -> 2Al(OH)3(am) +3SO4 + 14H2O + 6CO2
`ferricchloride`	Amount of ferric chloride added in mg/L as chemical: FeCl3 + 3HCO3 -> Fe(OH)3(am) + 3Cl + 3CO2
`ferricsulfate`	Amount of ferric sulfate added in mg/L as chemical: 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.

Value

A numeric value for solids mass in lb/day.

Source

https://water.mecc.edu/courses/ENV295Residuals/lesson3b.htm#:~:text=From%20the%20diagram%2C%20for%20example

Examples

```
solids_mass <- solvemass_solids(alum = 50, flow = 10, turb = 20)

library(dplyr)
mass_data <- tibble(
  alum = seq(10, 50, 10),
  flow = 10
) %>%
  mutate(mass = solvemass_solids(alum = alum, flow = flow, turb = 20))
#'
```

---

solveresid_o3 *Determine ozone decay*

Description

This function applies the ozone decay model to a water from U.S. EPA (2001) equation 5-128. For a single water, use solveresid_o3; to apply the model to a dataframe, use solveresid_o3_once. For most arguments, the _once helper "use_col" default looks for a column of the same name in the dataframe. The argument can be specified directly in the function instead or an unquoted column name can be provided.

Usage

```
solveresid_o3(water, dose, time)

solveresid_o3_once(
  df,
  input_water = "defined_water",
  output_column = "o3resid",
  dose = "use_col",
  time = "use_col"
)
```

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
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_column	name of the output column storing doses in mg/L. Default is "dose_required".

Value

`solveresid_o3` returns a numeric value for the residual ozone.

`solveresid_o3_once` returns a data frame containing the original data frame and columns for ozone dosed, time, and ozone residual.

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)

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