Package 'ambre'

September 4, 2026

Title Multi-Barrier Approach for Water Reuse Risk Assessment

Version 2.1.2

Description Provides a Quantitative Microbial Risk Assessment (QMRA) framework for the reuse of treated wastewater in agricultural irrigation. Following a multi-barrier approach, the package simulates pathogen inflow and removal along a treatment train, estimates human exposure, and computes health risk indicators such as infection probability, illness probability, and Disability-Adjusted Life Years (DALYs). It also supports an economic analysis of the treatment scenarios considered. For more details see <doi:10.36904/20240595>.

License MIT + file LICENSE

URL https://forge.inrae.fr/reversaal/reut/ambre-package

BugReports https://forge.inrae.fr/reversaal/reut/ambre-package/-/issues

Depends R (>= 4.1.0)

Imports colorspace, cowplot, dplyr, EnvStats, formattable, ggplot2, plyr, purrr, readxl, rlang, scales, sfsmisc, stats, stringr, tibble, tidyr, utils

Suggests knitr, rmarkdown, spelling, testthat, writexl

VignetteBuilder knitr

Config/fusen/version 0.7.2

Config/roxygen2/version 8.0.0

Encoding UTF-8

Language en-US

LazyData true

RoxygenNote 7.3.2

NeedsCompilation no

Author Flora Girard [aut, cre] (ORCID: <https://orcid.org/0000-0003-0537-4530>), Nicolas Forquet [ctb] (ORCID: <https://orcid.org/0000-0003-1154-5498>), Arthur Bréant [ctb], Colin Fay [ctb] (ORCID: <https://orcid.org/0000-0001-7343-1846>)

Maintainer Flora Girard <flora.girard@inrae.fr>

Repository CRAN

Date/Publication 2026-09-04 19:50:02 UTC

Contents

`annual_cost` *Annual cost*

Description

Calculate annual cost of the scenario simulated

Usage

```
annual_cost(
  scenario,
  price_per_m3,
  membership_fee,
  grant,
  initialSituation,
  allocation_key = NULL
)
```

Arguments

scenario	data.frame scenario obtained with create_scenario function
price_per_m3	numeric, water price in euro per cubic meter
membership_fee	numeric annual membership fee in euro per hectare
grant	numeric amount of the grant in percentage
initialSituation	boolean TRUE to simulate initial situation cost, FALSE to simulation supplementary process cost
allocation_key	numeric allocation key, NULL by default

Value

result list

Examples

```
scenario_apprentissage <- create_scenario(
  system.file("input_1culture_2pop.xlsx", package = "ambre")
)
annual_cost(scenario = scenario_apprentissage,
            price_per_m3 = 0.1,
            membership_fee = 200,
            grant = 0.5,
            initialSituation = TRUE)

annual_cost(scenario = scenario_apprentissage,
            price_per_m3 = 0.1,
            membership_fee = 200,
            grant = 0.5,
            initialSituation = FALSE)

crop <- scenario_apprentissage$CropName
allocation_custom <- data.frame(CropName = crop,
                                allocation = c(0.5, 0.5))
annual_cost(scenario = scenario_apprentissage,
            price_per_m3 = 0.1,
            membership_fee = 200,
            grant = 0,
            initialSituation = TRUE,
            allocation_key = allocation_custom)

annual_cost(scenario = scenario_apprentissage,
            price_per_m3 = 0.1,
            membership_fee = 200,
            grant = 0,
            initialSituation = FALSE,
            allocation_key = allocation_custom)
```

collective_treatment_cost_calculation
Collective treatment cost calculation

Description

Calculate cost of the simulated collective treatment train

Usage

```
collective_treatment_cost_calculation(scenario, grant, allocation_key = NULL)
```

Arguments

scenario	data.frame scenario with irrigation_need_m3an column obtained with irrigation_need_calculation
grant	numeric amount of the grant in percentage
allocation_key	data.frame allocation key, NULL by default

Value

cost

Examples

```
scenario_apprentissage <- create_scenario(
  system.file("input_1culture_2pop.xlsx", package = "ambre")
)
irrigation_need <- irrigation_need_calculation(scenario_apprentissage)
collective_treatment_cost_calculation(scenario = irrigation_need,
                                      grant = 0.5)

crop <- scenario_apprentissage$CropName |> unique()
allocation_custom <- data.frame(CropName = crop,
                                allocation = c(0.5, 0.1, 0.1, 0.1, 0.1, 0.1))
collective_treatment_cost_calculation(scenario = irrigation_need,
                                      grant = 0,
                                      allocation_key = allocation_custom)
```

config_ambre *config_ambre*

Description

A structured database containing the parameters and values needed to perform Quantitative Microbial Risk Assessment (QMRA) and cost calculations.

Usage

```
config_ambre
```

Format

A list of 3 lists and 5 tibbles :

exposure Exposure condition : number of repeatings for Monte Carlo simulation, number of exposures per year, volume (L) per exposure event

inflow Pathogen concentration (number of pathogen per L)

treatment Database of log reductions for risk management measures

processes General values for the log reductions

schemes Risk management process (sequence of measures) assessed

barrier_path Specific values for log reductions based on the assessed route of exposure

barrier_specific Specific values for log reductions based on the assessed measure

barrier_decay Specific values for log reductions for the natural die off measure

doseresponse Parameters for dose response model (https://qmrawiki.org/framework/dose-response/highest-quality-models-and-parameters)

path Details on pathways of exposure

description Description of pathways of exposure

frequency Frequency values of pathways of exposure

volume Volume values of pathways of exposure

crop Description of crop

health Parameters for risk characterization

economic Parameters for economic analysis

water_need Value of irrigation need according to crop type

population Description of population exposed

cost Database of opex and capex cost of risk management measures (https://theses.hal.science/tel-05009670)

regulation_value Database of required microorganism removal performance (in terms of concentration or removal rate) by regulatory class and pathogen group

regulation_crop Database of required water class acording to crop

Examples

```
names(config_ambre)
```

`create_random_distribution` *Create random distribution*

Description

This function does the same thing as kwb.utils::create_random_distribution(), but it is bundled with *ambre* to avoid a hard dependency on the unmaintained package.

Usage

```
create_random_distribution(
  type = "uniform",
  number_of_repeatings = 1,
  number_of_events = 365,
  value = 10,
  min = 10,
  max = 1000,
  percent_within_minmax = 0.9,
  min_zero = 0.01,
  log10_min = default_min(min, max, min_zero, f = log10),
  log10_max = default_max(max, min_zero * 10, f = log10),
  log10_mean = (log10_min + log10_max)/2,
 log10_sdev = abs((log10_max - log10_mean)/get_percentile(percent_within_minmax)),
  mean = (default_min(min, max, min_zero) + default_max(max, 10 * min_zero))/2,
  sdev = abs((default_max(max, 10 * min_zero) -
    mean)/get_percentile(percent_within_minmax)),
  meanlog = mean(log(default_min(min, max, min_zero) + default_max(max, 10 *
    min_zero))/2),
 sdlog = abs(sd(c(default_min(min, max, min_zero, f = log), default_max(max, 10 *
    min_zero, f = log)))),
  mode = (default_min(min, max, min_zero) + default_max(max, 10 * min_zero))/2,
  debug = TRUE
)
```

Arguments

`type`	"uniform" calls runif(), "log10_uniform" calls 10^runif(number_of_events, log10_min, log10_max), "triangle" calls EnvStats::rtri(), "lognorm" calls rlnorm(), "norm" calls rnorm() and "log10_norm" calls 10^rnorm(number_of_events, mean = log10_mean, sdev = log10_sdev), (default: "uniform")
`number_of_repeatings`	how often should the random distribution with the same parameters be generated (default: 1)
`number_of_events`	number of events

value	constant value (no random number), gets repeated number_of_events times (if 'type' = 'value')
min	minimum value (default: 10), only used if 'type' is "runif" or "triangle"
max	maximum value (default: 1000), only used if 'type' is "runif" or "triangle"
percent_within_minmax	percent of data point within min/max (default: 0.9 i.e. 90 percent
min_zero	only used if 'type' is "log10_uniform" or "log10_norm", "norm" or "lognorm" and "min" value equal zero. In this case the zero is replaced by this value (default: 0.01), see also default_min()
log10_min	minimum value (default: default_min(min, max, min_zero, f = log10)), only used if 'type' is "log10_uniform" or "log10_norm"
log10_max	maximum value (default: ifelse(max > 0, log10(max), log10_zero_threshold), only used if 'type' is "log10_uniform" or "log10_norm"
log10_mean	mean value (default: (log10_min + log10_max)/2), only used if 'type' is "log10_norm"
log10_sdev	standard deviation (default: abs((log10_max- log10_mean) / get_percentile(0.95)), only used if 'type' is "log10_norm"
mean	mean value (default: (default_min(min, max, min_zero) / default_max(max, 10*min_zero)) / 2), only used if 'type' is "norm"
sdev	standard deviation (default: abs((default_max(max, 10*min_zero) - mean) / get_percentile(0.95))), only used if 'type' is "norm"
meanlog	log mean value (default: mean(log((min + max) / 2))), only used if 'type' is "lognorm"
sdlog	standard deviation (default: abs(sd(c(default_min(min, max, min_zero, f = log)))), only used if 'type' is "lognorm"
mode	(default: default_min(min, max, min_zero) + default_max(max, 10 * min_zero) / 2), only used if 'type' is "triangle"
debug	print debug information (default: TRUE)

Value

list with parameters of user defined random distribution and corresponding values

References

Sonnenberg H (2024). kwb.utils: General Utility Functions Developed at KWB. R package version 0.15.0 URL: https://github.com/kwb-r/kwb.utils (MIT Licence)

Examples

```
# Example usage of create_random_distribution
# Uniform distribution
uniform_dist <- create_random_distribution(
  type = "uniform",
  number_of_repeatings = 2,
  number_of_events = 10,
```

```
  min = 0,
  max = 100
)

# Log10 uniform distribution
log10_uniform_dist <- create_random_distribution(
  type = "log10_uniform",
  number_of_events = 10,
  min = 1,
  max = 1000
)

# Triangle distribution
triangle_dist <- create_random_distribution(
  type = "triangle",
  number_of_events = 10,
  min = 0,
  max = 100,
  mode = 50
)

# Normal distribution
norm_dist <- create_random_distribution(
  type = "norm",
  number_of_events = 10,
  mean = 50,
  sdev = 10
)

# Log10 normal distribution
log10_norm_dist <- create_random_distribution(
  type = "log10_norm",
  number_of_events = 10,
  log10_mean = 2,
  log10_sdev = 0.5
)

# Lognormal distribution
lognorm_dist <- create_random_distribution(
  type = "lognorm",
  number_of_events = 10,
  meanlog = 2,
  sdlog = 0.5
)

# Constant value
value_dist <- create_random_distribution(
  type = "value",
  number_of_events = 10,
  value = 42
)
```

create_scenario *Create Exposure Scenario*

Description

This function creates exposure scenarios during the two game turns.

Usage

```
create_scenario(filepath, config = config_ambre)
```

Arguments

filepath	Path to the Excel file containing input data
config	list config file default value config_ambre

Value

A list containing the scenario data and a placeholder for future data

Examples

```
scenario <- create_scenario(system.file("input_cas_apprentissage_complet.xlsx", package = "ambre"))
```

create_scenario_from_df *Create Exposure Scenario from a data.frame*

Description

Like create_scenario() but takes an in-memory data.frame instead of an Excel file, and each of the four treatment roles (STEP / Collective / Initial / Supplementary) may hold a *vector* of process names (a list-column), so a train can contain more than one process per role.

Usage

```
create_scenario_from_df(user_input, config = config_ambre)
```

Arguments

user_input	A data.frame / tibble with columns CropName, Area, PopulationName, nb_population, PathName, nb_day_decay, and the four treatment columns STEPtreatmentName, CollectiveTreatmentName, InitialProcessName, SupplementaryProcessName. Each treatment cell may be a single name, a character vector of names, NA, or empty.
config	list config file, default value config_ambre.

Details

Single-value role columns are also accepted (they behave like `create_scenario()`); `NA` / empty roles are dropped from the train.

Value

A scenario tibble, same structure as `create_scenario()`.

Examples

```
df <- data.frame(
  CropName = "Tomato", Area = 35,
  PopulationName = "Maintenance staff", nb_population = 1,
  PathName = "Consumption of the final product tomato",
  nb_day_decay = NA_real_
)
df$STEPtreatmentName <- list(c("Q.1 - Activated Sludge", "Q.2 - Maturation Pond"))
df$CollectiveTreatmentName <- list(NA_character_)
df$InitialProcessName <- list("Q.6 - Chlorination")
df$SupplementaryProcessName <- list(NA_character_)
create_scenario_from_df(df)
```

`dalys_calculation` *Dalys calculation*

Description

Calculate dalys lose per event according to illness probability

Usage

```
dalys_calculation(scenario)
```

Arguments

`scenario`	data.frame scenario with risk column

Value

scenario_updated

Examples

```
library(dplyr)
library(purrr)
scenario_test <- create_scenario(system.file("input_1culture_2pop.xlsx", package = "ambre"))

scenario_conc <- inflow_concentration(scenario = scenario_test,
 pathogenName = c("Campylobacter jejuni"))
```

```
scenario_volume <-
  scenario_conc |>
  mutate(
    volume = map(
      .x = config,
      .f = ~ simulate_exposure(config = .x)
    )
  )

scenario_dose_ini <- initial_dose_calculation(scenario_volume)

scenario_scheme <- update_treatment_scheme(scenario_dose_ini)

scenario_with_logreduc_and_co <- scenario_scheme |>
  mutate( log_reduction = map(config, simulate_treatment))

scenario_final_dose_and_co_test <- final_dose_calculation(scenario_with_logreduc_and_co)

scenario_inf_proba_and_co <- infection_probability_calculation(scenario_final_dose_and_co_test)

scenario_illness_proba_and_co <- illness_probability_calculation(scenario_inf_proba_and_co)

dalys_calculation(scenario_illness_proba_and_co)
```

`distribution_repeater` *Helper function: distribution repeater*

Description

This function does the same thing as kwb.utils::distribution_repeater(), but it is bundled with *ambre* to avoid a hard dependency on the unmaintained package.

Usage

```
distribution_repeater(
  number_of_repeatings = 10,
  number_of_events = 365,
  func,
  ...
)
```

Arguments

`number_of_repeatings`
: how often should the random distribution with the same parameters be generated (default: 1)

`number_of_events`
: number of events (a single positive integer, >= 1)

func	distribution function to be repeated (e.g. runif, rlnorm, rnorm)
...	further parameters passed to func

Value

data.frame with columns repeatID, eventID and values

References

Sonnenberg H (2024). kwb.utils: General Utility Functions Developed at KWB. R package version 0.15.0 URL: https://github.com/kwb-r/kwb.utils (MIT Licence)

Examples

```
distribution_repeater(
  number_of_repeatings = 2,
  number_of_events = 10,
  func = runif,
  min = 1,
  max = 10
)
```

---

dr.betapoisson *Dose-response model: beta-poisson*

Description

Dose-response model: beta-poisson

Usage

```
dr.betapoisson(
  dose = sfsmisc::lseq(from = 1, to = 10^10, length = 1000),
  alpha = 0.328,
  N50 = 5430
)
```

Arguments

dose	vector of dose data (default: sfsmisc::lseq(from = 0.1, to = 10^10, length = 1000))
alpha	alpha (default: 3.28E-01)
N50	N50 (default: 5.43E+03)

Value

tibble

References

Sonnenberg H (2024). kwb.utils: General Utility Functions Developed at KWB. R package version 0.15.0 URL: https://github.com/kwb-r/kwb.utils (MIT Licence)

`dr.expo` *Dose-response model: exponential*

Description

Dose-response model: exponential

Usage

```
dr.expo(dose = sfsmisc::lseq(from = 1, to = 10^10, length = 1000), k = 0.572)
```

Arguments

`dose`	vector of dose data (default: `sfsmisc::lseq(from = 0.1, to = 10^10, length = 1000)`)
`k`	k-value (default: 5.72E-01)

Value

tibble

References

Sonnenberg H (2024). kwb.utils: General Utility Functions Developed at KWB. R package version 0.15.0 URL: https://github.com/kwb-r/kwb.utils (MIT Licence)

`enumeration` *Enumerate Elements*

Description

This function does the same thing as kwb.utils::enumeration(), but it is bundled with *ambre* to avoid a hard dependency on the unmaintained package.

Usage

```
enumeration(x, type = "element", sorted = FALSE, suffix = "")
```

Arguments

x	vector of elements to be enumerated
type	type of element to appear in the message. Default: "element"
sorted	logical. Whether or not to print the available elements in lexical order. Default: FALSE
suffix	suffix to be appended to type. Can be used to distinguish between singular and plural form. Default: ""

Value

character string containing the enumeration

References

Sonnenberg H (2024). kwb.utils: General Utility Functions Developed at KWB. R package version 0.15.0 URL: https://github.com/kwb-r/kwb.utils (MIT Licence)

Examples

```
# Example usage of enumeration
cat(enumeration(c("a", "b", "c"), type = "element"))
cat(enumeration(c("x", "y"), type = "element", suffix = "s"))
```

---

final_dose_calculation

Final dose calculation

Description

Calculation of the final dose according to initial dose and log reduction of the treatments simulated

Usage

```
final_dose_calculation(scenario)
```

Arguments

scenario	dataFrame scenario with initial_dose and log_reduction columns

Value

scenario_with_final_dose

Examples

```
library(dplyr)
library(purrr)
scenario_test <- create_scenario(system.file("input_1culture_2pop.xlsx", package = "ambre"))

scenario_conc <- inflow_concentration(scenario = scenario_test,
 pathogenName = c("Campylobacter jejuni", "Escherichia coli"))

scenario_volume <-
  scenario_conc |>
  mutate(
    volume = map(
      .x = config,
      .f = ~ simulate_exposure(config = .x)
    )
  )

scenario_dose_ini <- initial_dose_calculation(scenario_volume)

scenario_scheme <- update_treatment_scheme(scenario_dose_ini)

scenario_with_logreduc_and_co <- scenario_scheme |>
  mutate( log_reduction = map(config, simulate_treatment))

final_dose_calculation(scenario_with_logreduc_and_co)
```

---

generate_random_values

Create random distribution based on configuration file

Description

This function does the same thing as kwb.utils::generate_random_values(), but it is bundled with *ambre* to avoid a hard dependency on the unmaintained package.

Usage

```
generate_random_values(
  config,
  number_of_repeatings = 1,
  number_of_events,
  debug = TRUE
)
```

Arguments

config	as retrieved by config_read()

number_of_repeatings
: how often should the random distribution with the same parameters be generated (default: 1)

number_of_events
: number of events

debug
: print debug information (default: TRUE)

Value

list random distributions based on configuration file

References

Sonnenberg H (2024). kwb.utils: General Utility Functions Developed at KWB. R package version 0.15.0 URL: https://github.com/kwb-r/kwb.utils (MIT Licence)

Examples

```
# Example usage of generate_random_values
# Create a sample config object
config <- list(
  type = "norm",
  value = 10,
  min = 5,
  max = 15,
  mean = 10,
  sd = 2,
  meanlog = log(10),
  sdlog = 0.5,
  mode = 10
)

# Call generate_random_values with the sample config
result <- generate_random_values(
  config = config,
  number_of_repeatings = 2,
  number_of_events = 10,
  debug = FALSE
)

# Print the results
cat("Simulated events:")
print(result$events)

cat("Parameters used for simulation:")
print(result$paras)
```

get_exposure_value_or_stop *Get exposure value or stop*

Description

This function does the same thing as kwb.utils::exposure_value_or_stop(), but it is bundled with *ambre* to avoid a hard dependency on the unmaintained package.

Usage

```
get_exposure_value_or_stop(config, name)
```

Arguments

config	list config
name	character name of the column

Details

Description

Value

value

References

Sonnenberg H (2024). kwb.utils: General Utility Functions Developed at KWB. R package version 0.15.0 URL: https://github.com/kwb-r/kwb.utils (MIT Licence)

Examples

```
get_exposure_value_or_stop(config_ambre, "number_of_exposures")
```

get_infection_prob *Get infection probability values*

Description

Simulate infection probability according to pathogen consider

Usage

```
get_infection_prob(tbl_risk, dose_response, health)
```

Arguments

tbl_risk	data.frame containing PathogenID and dose_perEvent columns
dose_response	data.frame database of dose_response factors presents in config_ambre
health	data.frame database of health factors presents in config_ambre

Value

result_vector

References

Sonnenberg H (2024). kwb.utils: General Utility Functions Developed at KWB. R package version 0.15.0 URL: https://github.com/kwb-r/kwb.utils (MIT Licence)

---

get_risk_total *Get risk total*

Description

Calcul the total illness probability and total dalys that are the sum of every exposure event

Usage

```
get_risk_total(scenario)
```

Arguments

scenario	data.frame with risk column

Value

scenario_updated

Examples

```
library(dplyr)
library(purrr)
scenario_test <- create_scenario(system.file("input_1culture_2pop.xlsx", package = "ambre"))

scenario_conc <- inflow_concentration(scenario = scenario_test,
 pathogenName = c("Campylobacter jejuni"))

scenario_volume <-
  scenario_conc |>
  mutate(
    volume = map(
      .x = config,
      .f = ~ simulate_exposure(config = .x)
```

```
    )
  )

scenario_dose_ini <- initial_dose_calculation(scenario_volume)

scenario_scheme <- update_treatment_scheme(scenario_dose_ini)

scenario_with_logreduc_and_co <- scenario_scheme |>
  mutate( log_reduction = map(config, simulate_treatment))

scenario_final_dose_and_co_test <- final_dose_calculation(scenario_with_logreduc_and_co)

scenario_inf_proba_and_co <- infection_probability_calculation(scenario_final_dose_and_co_test)

scenario_illness_proba_and_co <- illness_probability_calculation(scenario_inf_proba_and_co)

scenario_dalys_and_co <- dalys_calculation(scenario_illness_proba_and_co)

get_risk_total(scenario_dalys_and_co)
```

`get_scheme_events_wide` *Calculate Wide-Format Treatment Scheme Events*

Description

This function and returns the results in a wide-format data frame.

Usage

```
get_scheme_events_wide(config, treatment_events_wide)
```

Arguments

`config`	A configuration object containing treatment scheme information.
`treatment_events_wide`	A data frame in wide format containing treatment events with columns for each treatment ID and log-reduction values.

Details

This function does the same thing as kwb.qmra::get_scheme_events_wide(), but it is bundled with *ambre* to avoid a hard dependency on the unmaintained package.

Value

A data frame with columns for event ID, pathogen group, and one column for each treatment scheme containing the sum of log-reduction values for that scheme.

References

Sonnenberg H (2024). kwb.utils: General Utility Functions Developed at KWB. R package version 0.15.0 URL: https://github.com/kwb-r/kwb.utils (MIT Licence)

Examples

```
# Example usage of get_scheme_events_wide
scenario_test <- create_scenario(
  system.file("input_1culture_2pop.xlsx", package = "ambre")
)
scenario_updated <- update_pathogen(scenario = scenario_test,
 pathoName = c("Campylobacter jejuni", "Rotavirus"))
config <- scenario_updated$config[[1]]

# Simulate treatment data
treatment_data <- get_treatment_data(config$treatment$processes, n_repeatings = 5, n_events = 3)

treatment_events_wide <- treatment_data$events |> tidyr::spread(
      key = TreatmentID,
      value = logreduction
    )

# Apply get_scheme_events_wide
scheme_events_wide <- get_scheme_events_wide(config, treatment_events_wide)

# Print results
print(scheme_events_wide)
```

---

get_tbl_risk *Get table risk*

Description

Get table risk to calculate infection probability

Usage

```
get_tbl_risk(scenario)
```

Arguments

scenario	data.frame with columns inflow concentration, final dose and volume

Value

tbl_risk

Examples

```
library(dplyr)
library(purrr)
scenario_test <- create_scenario(system.file("input_1culture_2pop.xlsx", package = "ambre"))

scenario_conc <- inflow_concentration(scenario = scenario_test,
 pathogenName = c("Campylobacter jejuni", "Escherichia coli"))

scenario_volume <-
  scenario_conc |>
  mutate(
    volume = map(
      .x = config,
      .f = ~ simulate_exposure(config = .x)
    )
  )

scenario_dose_ini <- initial_dose_calculation(scenario_volume)

scenario_scheme <- update_treatment_scheme(scenario_dose_ini)

scenario_with_logreduc_and_co <- scenario_scheme |>
  mutate( log_reduction = map(config, simulate_treatment))

scenario_final_dose_and_co_test <- final_dose_calculation(scenario_with_logreduc_and_co)
tbl_risk_test <- get_tbl_risk(scenario_final_dose_and_co_test)
```

---

get_treatment_data *Simulate treatment data for pathogen processes*

Description

This function simulates treatment data for a set of pathogen processes, including random values for each process and optional parameters. It can be used to generate synthetic data for analysis or testing.

Usage

```
get_treatment_data(
  processes,
  n_repeatings,
  n_events,
  debug = TRUE,
  include_paras = TRUE
)
```

Arguments

processes	A data frame containing information about pathogen processes, including treatment names and pathogen groups.
n_repeatings	The number of times each process should be repeated.
n_events	The number of events to simulate for each process.
debug	Logical indicating whether to print debug information.
include_paras	Logical indicating whether to include parameter data.

Details

This function does the same thing as kwb.qmra::get_treatment_data(), but it is bundled with *ambre* to avoid a hard dependency on the unmaintained package.

Value

If include_paras is FALSE, returns a data frame of simulated events. If include_paras is TRUE, returns a list containing both the events data frame and a data frame of parameters.

References

Sonnenberg H (2024). kwb.utils: General Utility Functions Developed at KWB. R package version 0.15.0 URL: https://github.com/kwb-r/kwb.utils (MIT Licence)

Examples

```
# Example usage (assuming 'processes' is defined)
scenario_test <- create_scenario(
  system.file("input_1culture_2pop.xlsx", package = "ambre")
)
scenario_updated <- update_pathogen(scenario = scenario_test,
 pathoName = c("Campylobacter jejuni", "Rotavirus"))
config = scenario_updated$config[[1]]
processes_simulated <- config$treatment$processes

result <- get_treatment_data(processes_simulated, n_repeatings = 10, n_events = 5)
print(result)
```

---

illness_probability_calculation

Illness probability calculation

Description

Calculate illness probability according to infection probability and pathogen simulated

Usage

```
illness_probability_calculation(scenario)
```

Arguments

scenario	data.frame with risk column

Value

scenario_updated

Examples

```
library(dplyr)
library(purrr)
scenario_test <- create_scenario(system.file("input_1culture_2pop.xlsx", package = "ambre"))

scenario_conc <- inflow_concentration(scenario = scenario_test,
 pathogenName = c("Campylobacter jejuni"))

scenario_volume <-
  scenario_conc |>
  mutate(
    volume = map(
      .x = config,
      .f = ~ simulate_exposure(config = .x)
    )
  )

scenario_dose_ini <- initial_dose_calculation(scenario_volume)

scenario_scheme <- update_treatment_scheme(scenario_dose_ini)

scenario_with_logreduc_and_co <- scenario_scheme |>
  mutate( log_reduction = map(config, simulate_treatment))

scenario_final_dose_and_co_test <- final_dose_calculation(scenario_with_logreduc_and_co)

scenario_inf_proba_and_co <- infection_probability_calculation(scenario_final_dose_and_co_test)

illness_probability_calculation(scenario_inf_proba_and_co)
```

---

infection_probability_calculation

Infection probability calculation

Description

Simulate infection probability for each events

Usage

```
infection_probability_calculation(scenario)
```

Arguments

scenario	data.frame with columns inflow concentration, final dose and volume

Value

scenario_updated

Examples

```
library(dplyr)
library(purrr)
scenario_test <- create_scenario(system.file("input_1culture_2pop.xlsx", package = "ambre"))

scenario_conc <- inflow_concentration(scenario = scenario_test,
 pathogenName = c("Campylobacter jejuni", "Escherichia coli"))

scenario_volume <-
  scenario_conc |>
  mutate(
    volume = map(
      .x = config,
      .f = ~ simulate_exposure(config = .x)
    )
  )

scenario_dose_ini <- initial_dose_calculation(scenario_volume)

scenario_scheme <- update_treatment_scheme(scenario_dose_ini)

scenario_with_logreduc_and_co <- scenario_scheme |>
  mutate( log_reduction = map(config, simulate_treatment))

scenario_final_dose_and_co_test <- final_dose_calculation(scenario_with_logreduc_and_co)

infection_probability_calculation(scenario_final_dose_and_co_test)
```

---

inflow_concentration *Inflow concentration*

Description

To calculate inflow concentration for specified pathogen

Usage

```
inflow_concentration(scenario, pathogenName, monteCarlo = 1000)
```

Arguments

scenario	list scenario created with create_scenario
pathogenName	character names of the pathogens to be simulated
monteCarlo	integer number of simulation iterations for Monte Carlo

Value

scenario_with_concentration

Examples

```
scenario_test <- create_scenario(system.file("input_1culture_2pop.xlsx", package = "ambre"))
result <- inflow_concentration(scenario = scenario_test,
 pathogenName = c("Campylobacter jejuni", "Norovirus"),
 monteCarlo = 2000)
```

---

inflow_concentration_custom

Inflow concentration custom

Description

To calculate inflow concentration for specified pathogen

Usage

```
inflow_concentration_custom(scenario, concentration, monteCarlo = 1000)
```

Arguments

scenario	list scenario created with create_scenario
concentration	data.frame min, max and distribution law of pathogen concentration
monteCarlo	integer number of simulation iterations for Monte Carlo

Value

scenario_with_concentration

Examples

```
sc <- create_scenario(system.file("input_1culture_2pop.xlsx", package = "ambre"))
sc  <- update_volume_with_desired_value(sc,
                                        volume = data.frame(min = c(0.5,1),
                                                            max = c(1,2),
                                                        type = c("triangle", "triangle")))

sc <- update_frequency_with_desired_value(scenario = sc, frequency = c(200L, 300L))

concentration_custom <- data.frame(PathogenName = c("Campylobacter jejuni", "Norovirus"),
                                   min = c(1, 10),
                                   max = c(2,20),
                                   type = c("uniform", "uniform"))

inflow_concentration_custom(scenario = sc,
                            concentration = concentration_custom)
```

`initial_dose_calculation` *Initial dose concentration*

Description

This function calculates the initial dose that is a product of inflow concentration and volume for each exposure events.

Usage

```
initial_dose_calculation(scenario)
```

Arguments

`scenario`	dataFrame scenario with volume and inflow_concentration columns

Value

scenario_with_initial_dose

Examples

```
library(dplyr)
library(purrr)
scenario_test <- create_scenario(system.file("input_1culture_2pop.xlsx", package = "ambre"))

scenario_conc <- inflow_concentration(scenario = scenario_test,
 pathogenName = c("Campylobacter jejuni"))

scenario_volume <-
  scenario_conc |>
```

```
  mutate(
    volume = map(
      .x = config,
      .f = ~ simulate_exposure(config = .x)
    )
  )

initial_dose_calculation(scenario_volume)
```

`irrigation_need_calculation` *Irrigation need calculation*

Description

Calculate irrigation need according to irrigation need of the culture simulated and its surface

Usage

```
irrigation_need_calculation(scenario)
```

Arguments

`scenario`	data.frame scenario obtained with create_scenario function

Value

scenario_updated

Examples

```
scenario_apprentissage <- create_scenario(
  system.file("input_1culture_2pop.xlsx", package = "ambre")
)
irrigation_need <- irrigation_need_calculation(scenario_apprentissage)
print(irrigation_need$irrigation_need_m3an)
```

`percentage_irrigation_need` *Percentage irrigation need*

Description

Calculate percentage irrigation need for each culture according to total irrigation need

Usage

```
percentage_irrigation_need(scenario)
```

Arguments

scenario	data.frame with irrigation_need_m3an column obtained with irrigation_need_calculation function

Value

(scenario_updated)

Examples

```
scenario_apprentissage <- create_scenario(
  system.file("input_1culture_2pop.xlsx", package = "ambre")
)
irrigation_need <- irrigation_need_calculation(scenario_apprentissage)

irrigation_pct <- percentage_irrigation_need(irrigation_need)
print(irrigation_pct$irrigation_need_percentage)
```

---

perimeter_calculation *Perimeter calculation*

Description

Calculate perimeter (linear meter) according to surface, and add the value to the scenario dataframe

Usage

```
perimeter_calculation(scenario)
```

Arguments

scenario	data.frame scenario obtained with create_scenario function

Value

scenario_updated

Examples

```
scenario_apprentissage <- create_scenario(
  system.file("input_1culture_2pop.xlsx", package = "ambre")
)

scenario_apprentissage_updated <- perimeter_calculation(scenario_apprentissage)
print(scenario_apprentissage_updated$perimeter)
```

`plot_comparison_qmra_initial_vs_supplementary_processes` *Plot comparison QMRA calculation with initial vs supplementary processes*

Description

Compare the result of the QMRA simulation with reduction_log from initial vs supplementary processes

Usage

```
plot_comparison_qmra_initial_vs_supplementary_processes(
  scenario,
  pathogen,
  regulationLog,
  regulationConcentration
)
```

Arguments

`scenario`	data.frame, scenario obtained with create_scenario function
`pathogen`	character, Name of the pathogen to simulate
`regulationLog`	data.frame regulation value of the log reduction to compare with simulation result
`regulationConcentration`	data.frame regulation value of the log reduction to compare with simulation result

Value

combine_plots

Examples

```
library(dplyr)
scenario_example <- create_scenario(
  filepath = system.file("input_1culture_2pop.xlsx", package = "ambre")
)


regulation_reduction <- config_ambre$regulation$regulation_value |>
  dplyr::filter(Country == "France") |>
  dplyr::select(-c(Concentration, Country, RegulationID))
regulation_concentration <- config_ambre$regulation$regulation_value |>
  dplyr::filter(Country == "France") |>
  dplyr::select(-c(Country, RegulationID, Reduction))
```

```
plot_comparison_qmra_initial_vs_supplementary_processes(
  scenario = scenario_example,
  pathogen = c("Campylobacter jejuni", "Norovirus"),
  regulationLog = regulation_reduction,
  regulationConcentration = regulation_concentration
)
```

`plot_dalys` *Plot dalys*

Description

Boxplot of total dalys lost per year according to population exposed and pathogen simulated

Usage

```
plot_dalys(scenario, objective = 1e-06)
```

Arguments

`scenario`	data.frame with risk_total column obtained with get_risk_total
`objective`	numeric OMS default value 1e-6

Value

plots

Examples

```
library(dplyr)
library(purrr)
scenario_test <- create_scenario(system.file("input_1culture_2pop.xlsx", package = "ambre"))

scenario_conc <- inflow_concentration(scenario = scenario_test,
 pathogenName = c("Campylobacter jejuni", "Escherichia coli", "Norovirus", "Rotavirus"))

scenario_volume <-
  scenario_conc |>
  mutate(
    volume = map(
      .x = config,
      .f = ~ simulate_exposure(config = .x)
    )
  )

scenario_dose_ini <- initial_dose_calculation(scenario_volume)

scenario_scheme <- update_treatment_scheme(scenario_dose_ini)
```

```
scenario_with_logreduc_and_co <- scenario_scheme |>
  mutate( log_reduction = map(config, simulate_treatment))

scenario_final_dose_and_co_test <- final_dose_calculation(scenario_with_logreduc_and_co)

scenario_inf_proba_and_co <- infection_probability_calculation(scenario_final_dose_and_co_test)

scenario_illness_proba_and_co <- illness_probability_calculation(scenario_inf_proba_and_co)

scenario_dalys_and_co <- dalys_calculation(scenario_illness_proba_and_co)

scenario_risk_total_and_co <-get_risk_total(scenario_dalys_and_co)

plot_dalys(scenario_risk_total_and_co)
```

`plot_infection_probability` *Plot infection probability*

Description

Boxplot of infection probability per year according to population exposed and pathogen simulated

Usage

```
plot_infection_probability(scenario)
```

Arguments

`scenario`	with risk_total column obtained with get_risk_total

Value

plots

Examples

```
library(dplyr)
library(purrr)
scenario_test <- create_scenario(system.file("input_1culture_2pop.xlsx", package = "ambre"))

scenario_conc <- inflow_concentration(scenario = scenario_test,
 pathogenName = c("Campylobacter jejuni", "Escherichia coli", "Norovirus", "Rotavirus"))

scenario_volume <-
  scenario_conc |>
  mutate(
    volume = map(
```

```
      .x = config,
      .f = ~ simulate_exposure(config = .x)
    )
  )

scenario_dose_ini <- initial_dose_calculation(scenario_volume)

scenario_scheme <- update_treatment_scheme(scenario_dose_ini)

scenario_with_logreduc_and_co <- scenario_scheme |>
  mutate( log_reduction = map(config, simulate_treatment))

scenario_final_dose_and_co_test <- final_dose_calculation(scenario_with_logreduc_and_co)

scenario_inf_proba_and_co <- infection_probability_calculation(scenario_final_dose_and_co_test)

scenario_illness_proba_and_co <- illness_probability_calculation(scenario_inf_proba_and_co)

scenario_dalys_and_co <- dalys_calculation(scenario_illness_proba_and_co)

scenario_risk_total_and_co <-get_risk_total(scenario_dalys_and_co)

plot_infection_probability(scenario_risk_total_and_co)
```

`plot_inflow` *Plot inflow concentration*

Description

Violin plot of inflow concentration according to pathogen simulated

Usage

```
plot_inflow(scenario)
```

Arguments

`scenario`	data.frame with inflow_concentration column obtained from inflow_concentration function

Value

plot

Examples

```
scenario_test <- create_scenario(system.file("input_1culture_2pop.xlsx", package = "ambre"))
scenario_with_inflow <- inflow_concentration(scenario = scenario_test,
 pathogenName = c("Campylobacter jejuni", "Norovirus"))
plot_inflow(scenario_with_inflow)
```

plot_logreduction *Plot log reduction*

Description

Boxplot of log reduction for each barrier according to population exposed and pathogen Name

Usage

```
plot_logreduction(scenario)
```

Arguments

scenario	with log reduction column obtained with simulate_treatment function

Value

plots

Examples

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

scenario_test <- create_scenario(system.file("input_1culture_2pop.xlsx", package = "ambre"))

scenario_conc <- inflow_concentration(scenario = scenario_test,
 pathogenName = c("Campylobacter jejuni", "Norovirus"))

scenario_volume <-
  scenario_conc |>
  mutate(
    volume = map(
      .x = config,
      .f = ~ simulate_exposure(config = .x)
    )
  )

scenario_dose_ini <- initial_dose_calculation(scenario_volume)

scenario_scheme <- update_treatment_scheme(scenario_dose_ini)

scenario_with_logreduc_and_co <- scenario_scheme |>
  mutate( log_reduction = map(config, simulate_treatment))

plot_logreduction(scenario_with_logreduc_and_co)
```

plot_volumes *Plot volume*

Description

Boxplot of the volume of exposition for each population exposed

Usage

```
plot_volumes(scenario)
```

Arguments

scenario	data.frame with volume column obtained with simulate_exposure function

Value

plot

Examples

```
library(dplyr)
library(purrr)
scenario_test <- create_scenario(system.file("input_1culture_2pop.xlsx", package = "ambre"))

scenario_conc <- inflow_concentration(scenario = scenario_test,
 pathogenName = c("Campylobacter jejuni"))

scenario_volume <-
  scenario_conc |>
  mutate(
    volume = map(
      .x = config,
      .f = ~ simulate_exposure(config = .x)
    )
  )

plot_volumes(scenario_volume)
```

process_cost_calculation *Barrier Cost Calculation*

Description

Calculate cost of supplementary barriers

Usage

```
process_cost_calculation(scenario, membership_fee, charge, initialSituation)
```

Arguments

scenario	data.frame scenario with irrigation_need_m3an column obtained from irrigation_need_calculation function
membership_fee	numeric annual membership fee
charge	numeric water charge
initialSituation	boolean TRUE to simulate initial simulation cost, FALSE to simulate supplementary process cost

Value

scenario_updated

Examples

```
scenario_apprentissage <- create_scenario(
  system.file("input_1culture_2pop.xlsx", package = "ambre"))
irrigation_need <- irrigation_need_calculation(scenario_apprentissage)
process_cost_calculation(scenario = irrigation_need,
                         membership_fee = 200,
                         charge = 0.1,
                         initialSituation = TRUE)


process_cost_calculation(scenario = irrigation_need,
                         membership_fee = 200,
                         charge = 0.1,
                         initialSituation = FALSE)
```

---

query_barrier *Barrier Query*

Description

retrieve Barrier ID according to its name in the barrier database

Usage

```
query_barrier(config = config_ambre, barrierName, na.rm = FALSE)
```

Arguments

config	list configuration file default value is config_ambre dataset
barrierName	character Name of the barrier you want to find the ID
na.rm	boolean FALSE by default

Value

barrierID

Examples

```
query_barrier(barrierName = "Q.1 - Activated Sludge")
query_barrier(barrierName = c("Q.1 - Activated Sludge", "Q.2 - Maturation Pond"))
query_barrier(barrierName = "P.9 - Peeling, P.7 - Cooking")
```

---

query_crop *Crop Query*

Description

Query to return crop ID from crop Name in crop database

Usage

```
query_crop(config = config_ambre, cropName)
```

Arguments

config	list configuration file default value is config_ambre dataset
cropName	character Name of crop for which you want to find the identifier

Value

cropID integer

Examples

```
query_crop(cropName = "Corn seed")
```

---

query_crop_height *Query crop height*

Description

Retrieve height of the crop simulated

Usage

```
query_crop_height(cropName)
```

Arguments

cropName	integer, identifier of the crop simulated

Value

height

Examples

```
query_crop_height("Tomato")
```

---

query_exp_path *Path Query*

Description

Query to return path ID from path Name using the database

Usage

```
query_exp_path(config = config_ambre, pathName)
```

Arguments

config	list configuration file default value is config_ambre dataset
pathName	character Name of the path for which you want to find the ID

Value

pathID integer ID of the population define

Examples

```
query_exp_path(pathName = "Consumption of the final product potatoes")
```

---

query_frequency *Query frequency*

Description

Retrieve min and max frequency values according to path defined in the scenario (database ambre_frequence)

Usage

```
query_frequency(pathID)
```

Arguments

pathID	integer ID of the path for which we want to retrieve the frequency values

Value

frequency

Examples

```
scenario_example <- create_scenario(
  system.file("input_1culture_2pop.xlsx", package = "ambre")
)

query_frequency(pathID = scenario_example$PathID)
```

`query_irrigation_need` *Query irrigation need*

Description

Retrieve irrigation need according to crop ID

Usage

```
query_irrigation_need(scenario)
```

Arguments

`scenario`	data.frame scenario obtained with create_scenario function

Value

scenario_updated

Examples

```
scenario_apprentissage <- create_scenario(
  system.file("input_1culture_2pop.xlsx", package = "ambre")
)

irrigation_need <- query_irrigation_need(scenario_apprentissage)
print(irrigation_need$IrrigationNeed)
```

query_matrix *Query Matrix*

Description

Retrieve Matrix ID according to path

Usage

```
query_matrix(config = config_ambre, pathName)
```

Arguments

config	list configuration file default value is config_ambre dataset
pathName	character Name of the path for which you want to find the ID

Value

matrixID integer

Examples

```
query_matrix(pathName = "Consumption of the final product potatoes")
```

query_pathogen *Pathogen Query*

Description

Query to return pathogen ID from pathogen Name using the database

Usage

```
query_pathogen(config = config_ambre, pathogenName)
```

Arguments

config	list configuration file default value is config_ambre dataset
pathogenName	character Name of the pathogen for which you want to find the ID

Value

pathogenID integer ID of the pathogen define

Examples

```
query_pathogen(pathogenName = "Campylobacter jejuni")

query_pathogen(pathogenName = c("Campylobacter jejuni", "Rotavirus"))
```

---

query_pop *Population Query*

Description

Query to return population ID from population Name using the database

Usage

```
query_pop(config = config_ambre, popName)
```

Arguments

config	list configuration file default value is config_ambre dataset
popName	character Name of the population for which you want to find the ID

Value

popID integer ID of the population define

Examples

```
query_pop(popName = "Irrigation staff")
```

---

query_volume *Query volume*

Description

Retrieve min and max volume values according to path defined in the scenario (database ambre_volume)

Usage

```
query_volume(pathID)
```

Arguments

pathID	integer ID of the path for which we want to retrieve the volume min and max values

Value

volume

Examples

```
query_volume(pathID = 1)
```

`regulation_matrix_concentration`
Concentration reduction matrix calculation

Description

Calculate the difference of achieve concentration in effluent according to regulation

Usage

```
regulation_matrix_concentration(scenario, regulation)
```

Arguments

`scenario`	data.frame with log_reduction column obtained with simulate_treatment function
`regulation`	data.frame regulation table to compare to simulation results

Value

formattable

Examples

```
library(dplyr)
library(purrr)
scenario_example <- create_scenario(system.file("input_1culture_2pop.xlsx",
                                                package = "ambre"))

scenario_exposure <- scenario_example |> mutate(
     volume = map(
         .x = config,
         .f = ~ simulate_exposure(config = .x)))

scenario_patho <- update_pathogen(scenario = scenario_exposure,
                          pathoName = c("Rotavirus", "Campylobacter jejuni", "Norovirus"))
concentration = map(.x = scenario_patho$config,
                    .f= ~ simulate_inflow(.x))
scenario_concentration <- scenario_patho |> mutate(inflow_concentration = concentration)
scenario_ini_dose <- initial_dose_calculation(scenario = scenario_concentration)

scenario_barrier <- update_treatment_scheme(scenario = scenario_ini_dose, initial_situation = TRUE)
```

```
scenario_barrier$config <- purrr::pmap(
  list(
    config = scenario_barrier$config,
    barrierID = scenario_barrier$SupplementaryProcessID,
    cropHeight = scenario_barrier$CropHeight,
    matrixID = scenario_barrier$MatrixID,
    popID = scenario_barrier$PopulationID,
    decay = scenario_barrier$nb_day_decay
  ),
  function(config, barrierID, cropHeight, matrixID, popID, decay) {

    if (barrierID %in% c(24, 26, 27, 28) &&
        matrixID %in% c(3, 5)) {

      update_logreduction_specific_barrier(
        config = config,
        cropHeight = cropHeight,
        barrierID = barrierID
      )

    } else if (barrierID %in% c(12)) {

      update_logreduction_decay(
        config = config,
        cropHeight = cropHeight,
        barrierID = barrierID,
        popID = popID,
        nb_day_decay = decay
      )

    } else if (barrierID %in% c(9,13,14,15,16,17,18,24,26,27,28)) {

      update_logreduction_path(
        config = config,
        matrixID = matrixID,
        barrierID = barrierID,
        popID = popID
      )

    } else {
      config
    }
  }
)

scenario_logreduction <- scenario_barrier |>
    mutate(log_reduction = map(config, simulate_treatment))

regulation_concentration <- config_ambre$regulation$regulation_value |>
  dplyr::filter(Country == "France") |>
  dplyr::select(-c(Country, RegulationID, Reduction))
```

```
regulation_matrix_concentration(scenario = scenario_logreduction,
                                regulation = regulation_concentration)
```

`regulation_matrix_log` *Log reduction matrix calculation*

Description

Calculate the difference of maximal log reduction achieve according to regulation

Usage

```
regulation_matrix_log(scenario, regulation)
```

Arguments

`scenario`	data.frame with log_reduction column obtained with simulate_treatment function
`regulation`	data.frame regulation value to compare to simulation results

Value

formattable

Examples

```
library(dplyr)
library(purrr)
scenario_example <- create_scenario(system.file("input_1culture_2pop.xlsx",
                                                package = "ambre"))

scenario_exposure <- scenario_example |> mutate(
     volume = map(
         .x = config,
         .f = ~ simulate_exposure(config = .x)))

scenario_patho <- update_pathogen(scenario = scenario_exposure,
                                  pathoName = c("Rotavirus", "Campylobacter jejuni"))
concentration = map(.x = scenario_patho$config,
                    .f= ~ simulate_inflow(.x))
scenario_concentration <- scenario_patho |> mutate(inflow_concentration = concentration)
scenario_ini_dose <- initial_dose_calculation(scenario = scenario_concentration)

scenario_barrier <- update_treatment_scheme(scenario = scenario_ini_dose, initial_situation = TRUE)

scenario_barrier$config <- purrr::pmap(
  list(
    config = scenario_barrier$config,
```

```
    barrierID = scenario_barrier$SupplementaryProcessID,
    cropHeight = scenario_barrier$CropHeight,
    matrixID = scenario_barrier$MatrixID,
    popID = scenario_barrier$PopulationID,
    decay = scenario_barrier$nb_day_decay
  ),
  function(config, barrierID, cropHeight, matrixID, popID, decay) {

    if (barrierID %in% c(24, 26, 27, 28) &&
        matrixID %in% c(3, 5)) {

      update_logreduction_specific_barrier(
        config = config,
        cropHeight = cropHeight,
        barrierID = barrierID
      )

    } else if (barrierID %in% c(12)) {

      update_logreduction_decay(
        config = config,
        cropHeight = cropHeight,
        barrierID = barrierID,
        popID = popID,
        nb_day_decay = decay
      )

    } else if (barrierID %in% c(9,13,14,15,16,17,18,24,26,27,28)) {

      update_logreduction_path(
        config = config,
        matrixID = matrixID,
        barrierID = barrierID,
        popID = popID
      )

    } else {
      config
    }
  }
)

scenario_logreduction <- scenario_barrier |>
    mutate(log_reduction = map(config, simulate_treatment))


regulation_reduction <- config_ambre$regulation$regulation_value |>
  dplyr::filter(Country == "France") |>
  dplyr::select(-c(Concentration, Country, RegulationID))

regulation_matrix_log(scenario = scenario_logreduction,
                      regulation = regulation_reduction)
```

`retrieve_specific_barrier` *Apply barriers' specific log-reduction to a config*

Description

For one or more supplementary barriers, updates the per-config log-reduction according to each barrier's type: pathway-specific barriers use `update_logreduction_path()`, natural die-off uses `update_logreduction_decay()`, and product/soil-specific barriers use `update_logreduction_specific_barrier()`. Barriers are applied in turn so their reductions accumulate; any other barrier leaves the config unchanged. Extracted from `run_qmra_initial_situation()` / `run_qmra_supplementary_process()` (was an inline anonymous function).

Usage

```
retrieve_specific_barrier(
  config,
  barrierID,
  cropHeight,
  matrixID,
  popID,
  decay
)
```

Arguments

`config`	A per-row config (list) from the scenario.
`barrierID`	One or more supplementary barrier ids (a scalar or a vector); each is applied in turn. `NA` matches no branch and is skipped.
`cropHeight`	The crop height.
`matrixID`	The exposure-matrix id.
`popID`	The exposed-population id.
`decay`	The number of decay days (for natural die-off barriers).

Value

The updated config.

`run_economic_analysis` *Run economic analysis*

Description

Calculate annual cost and total investment fees for simulated management risk scenario

Usage

```
run_economic_analysis(
  scenario,
  price_per_m3,
  membership_fee,
  grant,
  initialSituation,
  allocation_key = NULL
)
```

Arguments

`scenario`	data.frame scenario obtained with create_scenario function
`price_per_m3`	numeric, water price in euro per cubic meter
`membership_fee`	numeric annual membership fee in euro per hectare
`grant`	numeric amount of the grant in percentage
`initialSituation`	boolean, TRUE to simulate initial situation cost, FALSE to simulate supplementary process cost
`allocation_key`	numeric allocation key, NULL by default

Value

all_plots

Examples

```
scenario <- create_scenario(
  system.file("input_1culture_2pop.xlsx", package = "ambre")
)
run_economic_analysis(scenario = scenario,
                      price_per_m3 = 0.1,
                      membership_fee = 200,
                      grant = 0.5,
                      initialSituation = TRUE)
run_economic_analysis(scenario = scenario,
                      price_per_m3 = 0.1,
                      membership_fee = 200,
                      grant = 0.5,
```

```
                      initialSituation = FALSE)

crop <- scenario$CropName
allocation_custom <- data.frame(CropName = crop,
                                allocation = c(0.5, 0.5))
run_economic_analysis(scenario = scenario,
                      price_per_m3 = 0.1,
                      membership_fee = 200,
                      grant = 0,
                      initialSituation = TRUE,
                      allocation_key = allocation_custom)
```

`run_qmra_custom` *Run QMRA custom*

Description

Run the full QMRA workflow using custom data for volume and frequency of exposure (not the default databases)

1. simulate inflow concentration, volume of exposure
2. calculate initial dose
3. simulate log reduction for treatment processes
4. calculate final dose
5. calculate infection probability, illness probability and dalys reduction
6. calculate annual total risk

Usage

```
run_qmra_custom(
  scenario,
  volume,
  frequency,
  concentration,
  objective = 1e-06,
  regulationLog,
  regulationConcentration,
  initialSituation
)
```

Arguments

`scenario`	data.frame obtain with create_scenario function
`volume`	data.frame of min and max value, same size as scenario
`frequency`	array of integer same size as scenario
`concentration`	data.frame of pathogenName, min, max and distribution law to simulate

`objective` numeric reference dalys threshold to compare simulation result on dalys plot. default value is OMS threshold 1e-6 dalys

`regulationLog` data.frame regulation value of the log reduction to compare with simulation result

`regulationConcentration` data.frame regulation value of the log reduction to compare with simulation result

`initialSituation` boolean, TRUE to consider InitialTrainID for simulation, FALSE to consider SupplementaryTrainID for simulation

Value

all_plots list

Examples

```
sc <- create_scenario(system.file("input_1culture_2pop.xlsx", package = "ambre"))

regulation_reduction <- config_ambre$regulation$regulation_value |>
  dplyr::filter(Country == "France") |>
  dplyr::select(-c(Concentration, Country, RegulationID))
regulation_concentration <- config_ambre$regulation$regulation_value |>
  dplyr::filter(Country == "France") |>
  dplyr::select(-c(Country, RegulationID, Reduction))

concentration_custom <- data.frame(PathogenName = c("Rotavirus"),
                                   min = c(1),
                                   max = c(2),
                                   type = c("uniform"))
run_qmra_custom(scenario = sc,
                  concentration = concentration_custom,
             volume = data.frame(min=c(0.5,1), max = c(1,2), type= c("triangle", "triangle")),
                  frequency = c(200L, 300L),
                  regulationLog = regulation_reduction,
                  regulationConcentration = regulation_concentration,
                  initialSituation = TRUE)

run_qmra_custom(scenario = sc,
                  concentration = concentration_custom,
             volume = data.frame(min=c(0.5,1), max = c(1,2), type= c("triangle", "triangle")),
                  frequency = c(200L, 300L),
                  regulationLog = regulation_reduction,
                  regulationConcentration = regulation_concentration,
                  initialSituation = FALSE)
```

`run_qmra_initial_situation` *Run QMRA on initial situation*

Description

Run the full QMRA workflow

1. simulate inflow concentration, volume of exposure
2. calculate initial dose
3. simulate log reduction for treatment processes
4. calculate final dose
5. calculate infection probability, illness probability and dalys reduction
6. calculate annual total risk

Usage

```
run_qmra_initial_situation(
  scenario,
  pathogen,
  regulationLog,
  regulationConcentration
)
```

Arguments

`scenario`	data.frame scenario obtained with create_scenario function path of the scenario file
`pathogen`	character Name of the pathogen to simulate
`regulationLog`	data.frame regulation value of the log reduction to compare with simulation result
`regulationConcentration`	data.frame regulation value of the log reduction to compare with simulation result

Value

all_plots list of ggplot

Examples

```
library(dplyr)
scenario <- create_scenario(system.file("input_cas_apprentissage_complet.xlsx", package = "ambre"))
regulation_reduction <- config_ambre$regulation$regulation_value |>
  dplyr::filter(Country == "France") |>
  dplyr::select(-c(Concentration, Country, RegulationID))
```

```
regulation_concentration <- config_ambre$regulation$regulation_value |>
  dplyr::filter(Country == "France") |>
  dplyr::select(-c(Country, RegulationID, Reduction))
run_qmra_initial_situation(scenario = scenario,
                   pathogen = c("Campylobacter jejuni", "Escherichia coli"),
                   regulationLog = regulation_reduction,
                   regulationConcentration = regulation_concentration)
```

`run_qmra_learning_case`

QMRA on learning case

Description

Run a learning case to evaluate risk on scenario with 6 crops (Corn Seed, Tomato, Potato, Salad, Onion) and 6 populations (maintainer, harverster, irrigation, consumer, local resident, passerby). Run QMRA calculation with barrier measures

Usage

```
run_qmra_learning_case()
```

Value

all_plots list

Examples

```
run_qmra_learning_case()
```

`run_qmra_supplementary_process`

Run QMRA on addition process

Description

Run the full QMRA workflow on additional processes

1. simulate inflow concentration, volume of exposure
2. calculate initial dose
3. simulate log reduction for barrier processes
4. calculate final dose
5. calculate infection probability, illness probability and dalys reduction
6. calculate annual total risk

Usage

```
run_qmra_supplementary_process(
  scenario,
  pathogen,
  regulationLog,
  regulationConcentration
)
```

Arguments

scenario	data.frame scenario obtained with create_scenario function
pathogen	character Name of the pathogen to simulate
regulationLog	data.frame regulation value of the log reduction to compare with simulation result
regulationConcentration	data.frame regulation value of the log reduction to compare with simulation result

Value

all_plots list of ggplot

Examples

```
library(dplyr)
scenario <- create_scenario(system.file("input_1culture_2pop.xlsx", package = "ambre"))

regulation_reduction <- config_ambre$regulation$regulation_value |>
  dplyr::filter(Country == "France") |>
  dplyr::select(-c(Concentration, Country, RegulationID))
regulation_concentration <- config_ambre$regulation$regulation_value |>
  dplyr::filter(Country == "France") |>
  dplyr::select(-c(Country, RegulationID, Reduction))

run_qmra_supplementary_process(scenario = scenario,
                               pathogen = c("Campylobacter jejuni", "Escherichia coli"),
                               regulationLog = regulation_reduction,
                               regulationConcentration = regulation_concentration)
```

---

simulate_exposure *Simulate: exposure*

Description

This function does the same thing as kwb.qmra::get_treatment_data(), but it is bundled with *ambre* to avoid a hard dependency on the unmaintained package.

Usage

```
simulate_exposure(config, debug = TRUE)
```

Arguments

config	as retrieved by config_read()
debug	print debug information (default: TRUE)

Value

list with parameters of user defined exposure scenario (number of events and volumes per Event)

References

Sonnenberg H (2024). kwb.utils: General Utility Functions Developed at KWB. R package version 0.15.0 URL: https://github.com/kwb-r/kwb.utils (MIT Licence)

Examples

```
scenario_test <- create_scenario(system.file("input_1culture_2pop.xlsx", package = "ambre"))
scenario_updated <- update_pathogen(scenario = scenario_test,
 pathoName = c("Campylobacter jejuni", "Rotavirus"))

simulate_exposure(config = scenario_updated$config[[1]])
```

---

simulate_inflow *Simulate: inflow*

Description

This function does the same thing as kwb.utils::simulate_inflow(), but it is bundled with *ambre* to avoid a hard dependency on the unmaintained package.

Usage

```
simulate_inflow(config, debug = TRUE, lean = FALSE)
```

Arguments

config	list contained in the scenario object
debug	print debug information (default: TRUE)
lean	if TRUE, only the "events" are returned (in a reduced version, i.e. without column PathogenName and with all ID columns being of class integer), otherwise a list with the events in element events" and the corresponding parameters in element paras. The default is FALSE, i.e. events and parameters are returned in a list.

Details

Simulate pathogen concentrations

Value

list with parameters of user defined random distribution and corresponding values

References

Sonnenberg H (2024). kwb.utils: General Utility Functions Developed at KWB. R package version 0.15.0 URL: https://github.com/kwb-r/kwb.utils (MIT Licence)

Examples

```
# Example usage of simulate_inflow
scenario_test <- create_scenario(system.file("input_1culture_2pop.xlsx", package = "ambre"))
scenario_updated <- update_pathogen(scenario = scenario_test,
 pathoName = c("Campylobacter jejuni", "Rotavirus"))

# Call simulate_inflow with the sample config
result_1 <- simulate_inflow(config = scenario_updated$config[[1]], debug = FALSE)

# Print the results
cat("Parameters used for simulation:")
print(result_1$paras)

# Example with lean = TRUE
lean_result_1 <- simulate_inflow(config = scenario_updated$config[[1]], debug = FALSE, lean = TRUE)
cat("Lean result (only events):")
print(lean_result_1)
```

---

simulate_treatment *Simulate Treatment*

Description

This function simulates treatment data for pathogen processes based on the provided configuration. It generates treatment events and parameters, and can return results in either long or wide format.

Usage

```
simulate_treatment(config, wide = FALSE, debug = TRUE, lean = FALSE)
```

Arguments

config	A configuration object containing treatment scheme and process information.
wide	Logical indicating whether to return results in wide format.
debug	Logical indicating whether to print debug information.
lean	Logical indicating whether to return only essential results for web applications.

Details

This function does the same thing as kwb.qmra::simulate_treatment(), but it is bundled with *ambre* to avoid a hard dependency on the unmaintained package.

Value

If `lean` is TRUE, returns a list containing only the long-format treatment events. If `wide` is TRUE, returns a list containing long-format events, wide-format events, wide-format scheme events, treatment schemes, and treatment parameters. If `wide` is FALSE, returns a list containing long-format events, treatment schemes, and treatment parameters.

References

Sonnenberg H (2024). kwb.utils: General Utility Functions Developed at KWB. R package version 0.15.0 URL: https://github.com/kwb-r/kwb.utils (MIT Licence)

Examples

```
# Example usage of simulate_treatment

scenario_test <- create_scenario(
  system.file("input_1culture_2pop.xlsx", package = "ambre")
)
scenario_updated <- update_pathogen(scenario = scenario_test,
 pathoName = c("Campylobacter jejuni","Rotavirus"))
config <- scenario_updated$config[[1]]

# Simulate treatment data
result <- simulate_treatment(config, wide = TRUE, debug = FALSE, lean = FALSE)
```

`total_area_calculation` *Total area*

Description

Calculate the total area of each culture of the simulated scenario

Usage

```
total_area_calculation(scenario)
```

Arguments

`scenario`	data.frame scenario created with create_scenario function

Value

surface_total data.frame

Examples

```
scenario_apprentissage <- create_scenario(
  system.file("input_1culture_2pop.xlsx", package = "ambre")
)
total_area_calculation(scenario = scenario_apprentissage)
```

`total_investment_cost` *Total investment cost*

Description

Calculates the total investment required to implement risk management processes relating to the reuse of wastewater for each simulated configuration

Usage

```
total_investment_cost(
  scenario,
  price_per_m3,
  membership_fee,
  grant,
  allocation_key = NULL
)
```

Arguments

`scenario`	data.frame scenario obtained with create_scenario function
`price_per_m3`	numeric, water price in euro per cubic meter
`membership_fee`	numeric annual membership fee in euro per hectare
`grant`	numeric amount of the grant in percentage
`allocation_key`	numeric allocation_key, NULL by default

Value

result list

Examples

```
scenario_apprentissage <- create_scenario(
  system.file("input_1culture_2pop.xlsx", package = "ambre")
)

total_investment_cost(scenario = scenario_apprentissage,
                      price_per_m3 = 0.1,
                      membership_fee = 200,
                      grant = 0.5)


crop <- scenario_apprentissage$CropName
allocation_custom <- data.frame(CropName = crop,
                                allocation = c(0.5, 0.5))
total_investment_cost(scenario = scenario_apprentissage,
                      price_per_m3 = 0.1,
                      membership_fee = 200,
                      grant = 0,
                      allocation_key = allocation_custom)
```

---

total_irrigation_need *Total irrigation need*

Description

Calculate total irrigation need according to irrigation need of each culture of the simulated scenario

Usage

```
total_irrigation_need(scenario)
```

Arguments

scenario	data.frame scenario with irrigation_need_m3an column obtained with irrigation_need_calculation function

Value

total_irrigation_need

Examples

```
scenario_apprentissage <- create_scenario(
  system.file("input_1culture_2pop.xlsx", package = "ambre")
)
irrigation_need <- irrigation_need_calculation(scenario_apprentissage)
total_irrigation_need(irrigation_need)
```

update_concentration *Update concentration with desired value*

Description

Update the min, max and distribution law of the concentration in the simulated configuration according to a value defined by the user

Usage

```
update_concentration(scenario, concentration)
```

Arguments

scenario	data.frame
concentration	data.frame of pathogenName, min, max and distribution law to simulate

Value

scenario_updated

Examples

```
scenario_test <- create_scenario(
  system.file("input_1culture_2pop.xlsx", package = "ambre")
)

pathogenName <- c("Norovirus", "Rotavirus")
concentration_custom <- data.frame(PathogenName = pathogenName,
                                   min = c(1, 1),
                                   max = c(2,2),
                                   type = c("uniform", "uniform"))

scenario_updated_patho <- update_pathogen(scenario = scenario_test, pathoName = pathogenName)
result <- update_concentration(scenario = scenario_updated_patho ,
                     concentration = concentration_custom)

dplyr::filter(result$config[[1]]$inflow, PathogenName == "Norovirus")
```

update_frequency *Frequency update*

Description

Update the configuration with max frequency to be simulated according to path ID

Usage

```
update_frequency(scenario)
```

Arguments

scenario	dataframe scenario obtained with function create_scenario

Value

scenario_updated list configuration updated with the min and max volumes to be simulated

Examples

```
scenario_example <- create_scenario(
  system.file("input_1culture_2pop.xlsx", package = "ambre")
)
updated_scenario <- update_frequency(scenario_example)
updated_scenario$config[[1]]$exposure
```

update_frequency_with_desired_value *Updated frequency with desired value*

Description

Update the value of the frequency in the simulated configuration according to a value defined by the user

Usage

```
update_frequency_with_desired_value(scenario, frequency)
```

Arguments

scenario	data.frame
frequency	numeric, list of max frequency to simulate

Value

scenario_updated

Examples

```
scenario_example <- create_scenario(
  system.file("input_1culture_2pop.xlsx", package = "ambre")
)
scenario_example_updated <- update_frequency_with_desired_value(scenario = scenario_example,
                                                                 frequency = c(2L,3L))
print(scenario_example_updated$config[[1]]$exposure)
print(scenario_example_updated$config[[2]]$exposure)
```

---

update_logreduction_decay

Update log reduction for decay barrier

Description

Update the log reduction value for the decay barrier according to crop and population simulated

Usage

```
update_logreduction_decay(config, cropHeight, popID, barrierID, nb_day_decay)
```

Arguments

config	list configuration simulated
cropHeight	numeric, height of th crop simulated
popID	numeric, ID of the population simulated
barrierID	numeric, ID of the barrier simulated
nb_day_decay	numeric, number of day of natural decay for the simulated crop

Value

config_updated

Examples

```
scenario_example <- create_scenario(
  system.file("input_1culture_2pop.xlsx", package = "ambre")
)

scenario_example$config <- purrr::pmap(
  list(
    config = scenario_example$config,
    barrierID = scenario_example$SupplementaryProcessID,
```

```
    cropHeight = scenario_example$CropHeight,
    popID = scenario_example$PopulationID,
    decay = scenario_example$nb_day_decay

  ),
  function(config, barrierID, cropHeight, popID, decay) {

    if (barrierID %in% c(12)) {

      update_logreduction_decay(
        config = config,
        cropHeight = cropHeight,
        barrierID = barrierID,
        popID = popID,
        nb_day_decay = decay
      )

    } else {
      config
    }
  }
)
```

`update_logreduction_path` *Update log reduction for specific path*

Description

Update the log reduction value for specific barrier according to path and population simulated

Usage

```
update_logreduction_path(config, matrixID, popID, barrierID)
```

Arguments

`config`	list configuration simulated
`matrixID`	numeric, ID of the matrix simulated
`popID`	numeric, ID of the population simulated
`barrierID`	numeric, ID of the barrier simulated

Value

config_updated

Examples

```
scenario_example <- create_scenario(
  system.file("input_1culture_2pop.xlsx", package = "ambre")
)

scenario_example$config <- purrr::pmap(
  list(
    config = scenario_example$config,
    barrierID = scenario_example$SupplementaryProcessID,
    matrixID = scenario_example$MatrixID,
    popID = scenario_example$PopulationID
  ),
  function(config, barrierID, matrixID, popID) {

    if (barrierID %in% c(9,13,14,15,16,17,18,24,26,27,28)) {

      update_logreduction_path(
        config = config,
        matrixID = matrixID,
        barrierID = barrierID,
        popID = popID
      )

    } else {
      config
    }
  }
)
```

`update_logreduction_specific_barrier` *Update log reduction with specific value*

Description

Update log reduction value according to treatment, crop, and population simulated

Usage

```
update_logreduction_specific_barrier(config, cropHeight, barrierID)
```

Arguments

`config`	list configuration simulated
`cropHeight`	numeric, height of th crop simulated
`barrierID`	numeric, ID of the barrier simulated

Value

config_updated

Examples

```
scenario_example <- create_scenario(
  system.file("input_1culture_2pop.xlsx", package = "ambre")
)

scenario_example$config <- purrr::pmap(
  list(
    config = scenario_example$config,
    barrierID = scenario_example$SupplementaryProcessID,
    matrixID = scenario_example$MatrixID,
    cropHeight = scenario_example$CropHeight
  ),
  function(config, barrierID, matrixID, cropHeight) {

    if (barrierID %in% c(24, 26, 27, 28) &&
        matrixID %in% c(3, 5)) {

      update_logreduction_specific_barrier(
        config = config,
        cropHeight = cropHeight,
        barrierID = barrierID
      )

    } else {
      config
    }
  }
)
```

`update_monte_carlo` *Update monte carlo value*

Description

Update number_of_repeating in config$exposure database

Usage

```
update_monte_carlo(scenario, monte_carlo = 1000)
```

Arguments

`scenario`	data.frame scenario obtained with create_scenario function
`monte_carlo`	integer number of simulation iterations for Monte Carlo

Value

scenario_updated

Examples

```
scenario_example <- create_scenario(
  system.file("input_1culture_2pop.xlsx", package = "ambre")
)
scenario_updated <- update_monte_carlo(scenario = scenario_example,
                    monte_carlo = 2000)
scenario_updated$config[[1]]$exposure
```

`update_pathogen` *Pathogen update*

Description

Update the configuration with list of pathogen to be simulated according to pathogen ID

Usage

```
update_pathogen(scenario, pathoName)
```

Arguments

`scenario`	list obtain from create_scenario()
`pathoName`	character name of the pathogens simulated

Value

scenario_updated list scenario updated with the pathogens to be simulated

Examples

```
scenario_test <- create_scenario(
  system.file("input_1culture_2pop.xlsx", package = "ambre")
)

scenario_updated <- update_pathogen(scenario = scenario_test,
 pathoName = c("Campylobacter jejuni", "Rotavirus"))
```

update_treatment_scheme *Update treatment scheme*

Description

Update treatment scheme list in config data frame according to scenario define

Usage

```
update_treatment_scheme(scenario, initial_situation = TRUE)
```

Arguments

scenario	created with function create_scenario
initial_situation	boolean to retrieve simulate treatment train if TRUE or from Barrier train if FALSE

Value

updated_scenario

Examples

```
scenario_test <- create_scenario(
  system.file("input_cas_apprentissage_complet.xlsx", package = "ambre")
)
scenario_ini <- update_treatment_scheme(scenario_test, initial_situation = TRUE)
scenario_supp <- update_treatment_scheme(scenario_test, initial_situation = FALSE)
```

update_volume *Volume update*

Description

Update the configuration with min and max volumes to be simulated according to path ID

Usage

```
update_volume(scenario)
```

Arguments

scenario	dataframe scenario obtained with function create_scenario

Value

scenario_updated list configuration updated with the min and max volumes to be simulated

Examples

```
scenario_example <- create_scenario(
  system.file("input_1culture_2pop.xlsx", package = "ambre")
)
scenario_updated <- update_volume(scenario_example)
scenario_updated$config[[1]]$exposure
```

---

update_volume_with_desired_value

Update volume with desired value

Description

Update the min and max value of the volume in the simulated configuration according to a value defined by the user

Usage

```
update_volume_with_desired_value(scenario, volume)
```

Arguments

scenario	data.frame
volume	numeric, named list of min and max value to simulate

Value

scenario_updated

Examples

```
scenario_example <- create_scenario(
  system.file("input_1culture_2pop.xlsx", package = "ambre")
)
volumes <- data.frame("min" = c(1.1, 1.2), "max" = c(1.3,1.4), "type" = c("triangle", "triangle"))
scenario_example_updated <- update_volume_with_desired_value(scenario = scenario_example,
                                  volume = volumes)

print(scenario_example_updated$config[[1]]$exposure)
print(scenario_example_updated$config[[2]]$exposure)
```

water_price *Water charge*

Description

Calculate cost of water charge

Usage

```
water_price(scenario, price_per_m3, membership_fee)
```

Arguments

scenario	data.frame, with irrigation_need_m3an obtained with irrigation_need_calculation function
price_per_m3	numeric, water price in euro per cubic meter
membership_fee	numeric annual membership fee in euro per hectare

Value

updated_scenario

Examples

```
scenario_apprentissage <- create_scenario(
  system.file("input_1culture_2pop.xlsx", package = "ambre")
)
irrigation_need <- irrigation_need_calculation(scenario_apprentissage)
water_price(scenario = irrigation_need,
            price_per_m3 = 0.01,
            membership_fee = 200)
```

Index