Package 'MetaRVM'

November 6, 2026

Title Meta-Population Compartmental Model for Respiratory Virus Diseases

Version 2.2.0

Description Simulates respiratory virus epidemics using meta-population compartmental models following Fadikar et. al. (2025) <doi:10.1109/WSC68292.2025.11338996>. 'MetaRVM' implements a stochastic
SEIRD (Susceptible-Exposed-Infected-Recovered-Dead) framework with demographic stratification by user provided attributes. It supports complex
epidemiological scenarios including asymptomatic and presymptomatic transmission, hospitalization dynamics, vaccination schedules, and time-varying contact patterns
via mixing matrices.

URL https://RESUME-Epi.github.io/MetaRVM/,
https://github.com/RESUME-Epi/MetaRVM

BugReports https://github.com/RESUME-Epi/MetaRVM/issues

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

Depends R (>= 3.5.0)

Imports data.table, dplyr, ggplot2, magrittr, methods, odin, purrr, R6, tidyr, yaml

Suggests dde, doParallel, foreach, knitr, parallel, pkgdown, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr, rmarkdown

Config/testthat/edition 3

NeedsCompilation no

Author Arindam Fadikar [aut, cre, cph] (ORCID: <https://orcid.org/0000-0001-7396-0350>),
Charles Macal [ctb],
Martinez Moyano Ignacio Javier [ctb],
Ozik Jonathan [ctb]

Maintainer Arindam Fadikar <afadikar@anl.gov>

Repository CRAN

Date/Publication 2026-11-06 05:20:02 UTC

Contents

build_config 2
copy_config 5
format_metarvm_output 6
metaRVM 7
MetaRVMCheck 9
MetaRVMConfig 10
MetaRVMResults 16
MetaRVMSummary 20
meta_measles_sim 22
meta_sim 26
parse_config 32

Index **36**

`build_config` *Build a MetaRVM configuration*

Description

Constructs a MetaRVM configuration programmatically. The argument structure mirrors the YAML configuration file format. The result can be returned as a raw list, written to a YAML file, or parsed directly into a MetaRVMConfig object — or any combination of the three.

Usage

```
build_config(
  disease,
  population_init,
  mixing_matrices,
  disease_params,
  start_date,
  sim_length,
  nsim = 1L,
  nrep = 1L,
  run_id = NULL,
  vaccination = NULL,
  sub_disease_params = NULL,
  random_seed = NULL,
  delta_t = NULL,
  parallel = FALSE,
```

```
  checkpoint_dir = NULL,
  checkpoint_dates = NULL,
  restore_from = NULL,
  output_file = NULL,
  return_object = FALSE,
  verbose = FALSE,
  suppress_odin_messages = FALSE
)
```

Arguments

disease	Character. Disease to simulate. Supported values: "flu", "measles". The vaccination mechanism differs by disease: flu reads a time-varying schedule from the vaccination CSV argument; measles encodes vaccination as initial compartment counts (V10, V20) directly in the population CSV — no separate vaccination file is used.
population_init	Character. Path to the population initialization CSV.
mixing_matrices	Named list of paths to mixing-matrix CSV files. Required keys: weekday_day, weekday_night, weekend_day, weekend_night.
disease_params	Named list of disease parameters. Each element may be a scalar, a per-population vector, or a distribution-spec list such as list(dist = "lognormal", mu = -0.7, sd = 0.1).
start_date	Character ("MM/DD/YYYY") or Date. Simulation start date.
sim_length	Positive integer. Number of days to simulate.
nsim	Positive integer. Number of parameter draws (default 1).
nrep	Positive integer. Replicates per parameter draw (default 1).
run_id	Optional character identifier stored in run metadata. Auto-generated from the current timestamp when NULL.
vaccination	Optional character. Path to the vaccination schedule CSV. Used by flu only; ignored for measles (which encodes vaccination as V10/V20 columns in the population CSV).
sub_disease_params	Optional named list of per-category parameter overrides. Structure mirrors the sub_disease_params: YAML section, e.g. list(age = list(child = list(ts = 0.6))). Values can also be distribution specs, e.g. list(dist = "lognormal", mu = 2, sd = 0.5), drawn independently per simulation replicate the same way disease_params distributions are.
random_seed	Optional integer seed for reproducibility. Auto-generated when NULL.
delta_t	Numeric time-step in days. Defaults to 0.5 when NULL.
parallel	Logical. Whether metaRVM() may use a parallel foreach backend. Default FALSE. Set to TRUE only at the top-level call; leave FALSE when the config will be used inside a parallel worker to prevent nested-parallel conflicts.
checkpoint_dir	Optional character. Directory for checkpoint files. Supported for both flu and measles.

checkpoint_dates
: Optional character vector of dates ("MM/DD/YYYY") at which to write checkpoints. Supported for both flu and measles.

restore_from
: Optional character. Path to a checkpoint .Rda file to restore from. Supported for both flu and measles.

output_file
: Optional character. When provided the configuration is written to this path as a YAML file. Paths inside the config are stored as absolute paths so the file is self-contained regardless of working directory.

return_object
: Logical. When TRUE the configuration is parsed and returned as a MetaRVMConfig object. When FALSE (default) the raw configuration list is returned (invisibly when output_file is also given).

verbose
: Logical. Passed through to parse_config() when return_object = TRUE.

suppress_odin_messages
: Logical. Passed through to parse_config() when return_object = TRUE.

Value

A MetaRVMConfig object when return_object = TRUE; otherwise the raw configuration list (invisibly if output_file was also written).

See Also

parse_config(), metaRVM()

Examples

```
ext <- function(x) system.file("extdata", x, package = "MetaRVM")

# Return the raw config list for inspection / modification
cfg <- build_config(
  disease         = "flu",
  population_init = ext("population_init_n24.csv"),
  mixing_matrices = list(
    weekday_day   = ext("m_weekday_day.csv"),
    weekday_night = ext("m_weekday_night.csv"),
    weekend_day   = ext("m_weekend_day.csv"),
    weekend_night = ext("m_weekend_night.csv")
  ),
  disease_params  = list(
    ts = 0.5, ve = 0.4, dv = 180, dp = 1, de = 3,
    da = 5,   ds = 6,   dh = 8,   dr = 180,
    pea = 0.3, psr = 0.95, phr = 0.97
  ),
  vaccination     = ext("vaccination_n24.csv"),
  start_date      = "10/01/2023",
  sim_length      = 30L,
  nsim            = 2L
)

# Parse directly to a MetaRVMConfig object
```

```
cfg_obj <- build_config(
  disease         = "flu",
  population_init = ext("population_init_n24.csv"),
  mixing_matrices = list(
    weekday_day   = ext("m_weekday_day.csv"),
    weekday_night = ext("m_weekday_night.csv"),
    weekend_day   = ext("m_weekend_day.csv"),
    weekend_night = ext("m_weekend_night.csv")
  ),
  disease_params  = list(
    ts = 0.5, ve = 0.4, dv = 180, dp = 1, de = 3,
    da = 5,   ds = 6,   dh = 8,   dr = 180,
    pea = 0.3, psr = 0.95, phr = 0.97
  ),
  vaccination     = ext("vaccination_n24.csv"),
  start_date      = "10/01/2023",
  sim_length      = 30L,
  return_object   = TRUE
)
```

`copy_config` *Copy a MetaRVMConfig object*

Description

Returns an independent deep copy of a `MetaRVMConfig` object. Because R6 objects use reference semantics, assignment (`b <- a`) creates a second handle to the *same* object. Use `copy_config()` when you need a truly independent copy that can be modified without affecting the original.

Usage

```
copy_config(config)
```

Arguments

`config`	A `MetaRVMConfig` object.

Value

A new `MetaRVMConfig` object with the same configuration data.

Examples

```
## Not run:
config  <- parse_config("my_config.yaml")
config2 <- copy_config(config)
config2$set("nsim", 50L)   # does not affect config

## End(Not run)
```

`format_metarvm_output` *Format raw MetaRVM simulation output*

Description

Converts raw step-indexed output from `meta_sim()` or `meta_measles_sim()` into a tidy long-format `data.table` with calendar dates and user-defined demographic attributes.

The transformation differs by state type:

- **Compartment states** (S, E, I, H, R, D, V, ...): only integer time-step values are retained.
- **Flow states** (`n_` prefix): within-day pairs are summed to daily counts.

Usage

```
format_metarvm_output(sim_output, config)
```

Arguments

`sim_output`	`data.table` of raw simulation output from `meta_sim()` or `meta_measles_sim()`. Must contain columns `time`, `disease_state`, `population_id`, `value`, and `instance`.
`config`	A `MetaRVMConfig` object or a named list. Must supply `start_date`, `pop_map` (with a `population_id` column), and `category_names`.

Value

A `data.table` in long format with one row per (date, demographic combination, disease state, simulation instance). Columns:

`date` `Date`. Calendar date derived from `start_date` + `time`.

User category columns One column per demographic category detected in the population CSV (e.g., `age`, `zone`, `race`).

`disease_state` `character`. Compartment or flow-count state label.

`value` `numeric`. Compartment count or summed daily flow count.

`instance` `integer`. Simulation instance index.

See Also

`metaRVM()`, `meta_sim()`, `MetaRVMResults`

Examples

```
## Not run:
# Typically called internally by metaRVM(); exposed for custom pipelines.
cfg <- system.file("extdata", "example_config.yaml", package = "MetaRVM")
config <- parse_config(cfg)
raw <- meta_sim(...)  # low-level engine output
formatted <- format_metarvm_output(raw, config)
```

```
## End(Not run)
```

`metaRVM` *Run a MetaRVM epidemic simulation*

Description

`metaRVM()` is the high-level entry point for running a MetaRVM metapopulation respiratory virus simulation. It parses the configuration, runs one or more stochastic simulation instances, formats the ODIN/MetaRVM output into a tidy long table with calendar dates and demographic attributes, and returns a `MetaRVMResults` object for downstream analysis and plotting.

Usage

```
metaRVM(config_input, verbose = NULL, suppress_odin_messages = NULL)
```

Arguments

`config_input`	Configuration specification in one of three forms: • **Character string**: path to a YAML configuration file. • `MetaRVMConfig` **object**: pre-initialized configuration. • **Named list**: output from `parse_config()` with `return_object = FALSE`.
`verbose`	Optional logical. If `TRUE`, prints meaningful run progress messages. If `NULL` (default), uses `simulation_config$verbose` from the parsed config when available; otherwise defaults to `FALSE`.
`suppress_odin_messages`	Optional logical. If `TRUE`, sets `options(odin.verbose = FALSE)` for this run to suppress ODIN compilation chatter. If `NULL` (default), uses `simulation_config$suppress_odin_messa` from config when available; otherwise defaults to `FALSE`.

Details

The configuration input controls:

- **Population structure** (user-defined categories and the initial compartment counts from the initialization file)
- **Disease parameters** (disease-specific; see `?parse_config` and the disease registry for per-disease parameter lists)
- **Mixing matrices** (weekday/weekend, day/night contact patterns)
- **Vaccination** (time-varying schedule for flu; initial compartment counts for measles)
- **Simulation settings** (start date, length, number of instances, checkpointing)

Internally, `metaRVM()`:

1. Parses the YAML configuration via `parse_config()`.
2. Calls the ODIN-based simulation engine `meta_sim()` for each instance.
3. Uses `format_metarvm_output()` to convert time steps to dates and attach demographic attributes.
4. Wraps the formatted output and metadata in a `MetaRVMResults` object that supports method chaining for subsetting, summarizing, and plotting.

Value

A `MetaRVMResults` R6 object with three key components:

$results A tidy `data.table` with one row per date–subpopulation–disease state–instance combination. Typical columns include:

- `date`: calendar date (`Date`)
- user-defined demographic category columns (if present in the initialization file)
- `disease_state`: compartment or flow label (e.g., `S`, `E`, `I_symp`, `H`, `R`, `D`, `n_SE`, `n_IsympH`, etc.)
- `value`: population count or daily flow
- `instance`: simulation instance index (1, 2, ...)

$config The `MetaRVMConfig` object used for the run.

$run_info A list with metadata such as `n_instances`, `date_range`, `delta_t`, and checkpoint information.

Author(s)

Arindam Fadikar, Charles Macal, Ignacio Martinez-Moyano, Jonathan Ozik

References

Fadikar, A., et al. "Developing and deploying a use-inspired metapopulation modeling framework for detailed tracking of stratified health outcomes"

See Also

`parse_config()` for reading YAML configurations, MetaRVMConfig for configuration management, MetaRVMResults for analysis and plotting, `meta_sim()` for the low-level simulation engine.

Examples

```
options(odin.verbose = FALSE)
example_config <- system.file("extdata", "example_config.yaml",
                              package = "MetaRVM")

# Run a single-instance simulation from a YAML file
results <- metaRVM(example_config)

# Print a high-level summary
results
```

```
# Access the tidy results table
head(results$results)

# Summarize and plot hospitalizations and deaths by user-defined categories
results$summarize(
  group_by       = c("age", "zone"),
  disease_states = c("H", "D"),
  stats          = c("median", "quantile"),
  quantiles      = c(0.25, 0.75)
)$plot()

# Using a pre-parsed configuration object
cfg <- parse_config(example_config, return_object = TRUE)
results2 <- metaRVM(cfg)
```

`MetaRVMCheck` *MetaRVM Checkpoint Class*

Description

R6 class that stores a saved simulation state produced by `meta_sim()`. Used to resume or branch a simulation from an intermediate time point.

Details

Checkpoint objects are written automatically during a simulation run when `do_chk = TRUE` is set in the simulation config. Each checkpoint captures the full compartment state (S, E, I, H, R, D, V, ...) and all disease parameters at the saved time step, enabling a resumed run to pick up exactly where it left off.

`MetaRVMCheck` extends `MetaRVMConfig`: all `$get()`, `$get_category_names()`, and related accessor methods are inherited and work identically. Required checkpoint fields are disease-specific and validated against the disease registry at construction time.

Super class

`MetaRVM::MetaRVMConfig` -> `MetaRVMCheck`

Public fields

`check_data` List containing all parsed checkpoint data

Methods

Public methods:

- `MetaRVMCheck$new()`
- `MetaRVMCheck$clone()`

Method `new()`**:** Initialize a new MetaRVMCheck object

Usage:

```
MetaRVMCheck$new(input)
```

Arguments:

`input` A list containing checkpoint data.

Returns: A new `MetaRVMCheck` object.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
MetaRVMCheck$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Arindam Fadikar

See Also

`metaRVM()` for enabling checkpointing via the `simulation_config` section; `MetaRVMConfig` for the parent class and its accessor methods.

Examples

```
## Not run:
# Checkpoints are created automatically when do_chk = TRUE in the config.
# To inspect a saved checkpoint:
chk_data <- readRDS("my_checkpoint.rds")
chk <- MetaRVMCheck$new(chk_data)
chk$get("chk_time_step")
chk$get("N_pop")

## End(Not run)
```

`MetaRVMConfig` *MetaRVM Configuration Class*

Description

R6 class to handle MetaRVM configuration data with validation and methods. This class encapsulates all configuration parameters needed for MetaRVM simulations, providing methods for parameter access, validation, and introspection.

Details

The MetaRVMConfig class stores parsed configuration data from YAML files and provides structured access to simulation parameters. It automatically validates configuration completeness and provides convenient methods for accessing demographic categories, initialization-derived population metadata, and other simulation settings.

Public fields

`config_file` Path to the original YAML config file (if applicable)

`config_data` List containing all parsed configuration parameters

Methods

Public methods:

- `MetaRVMConfig$new()`
- `MetaRVMConfig$get()`
- `MetaRVMConfig$get_all()`
- `MetaRVMConfig$list_parameters()`
- `MetaRVMConfig$parameter_summary()`
- `MetaRVMConfig$set()`
- `MetaRVMConfig$print()`
- `MetaRVMConfig$get_pop_map()`
- `MetaRVMConfig$get_category_names()`
- `MetaRVMConfig$get_category_values()`
- `MetaRVMConfig$get_all_categories()`
- `MetaRVMConfig$set_nrep()`
- `MetaRVMConfig$set_random_seed()`
- `MetaRVMConfig$set_start_date()`
- `MetaRVMConfig$set_sim_length()`
- `MetaRVMConfig$set_delta_t()`
- `MetaRVMConfig$set_run_id()`
- `MetaRVMConfig$set_parallel()`
- `MetaRVMConfig$clone()`

Method `new()`**:** Initialize a new MetaRVMConfig object

Usage:

```
MetaRVMConfig$new(input)
```

Arguments:

`input` Either a file path (character) or parsed config list

Returns: New MetaRVMConfig object (invisible)

Method `get()`**:** Get a configuration parameter

Usage:

```
MetaRVMConfig$get(param)
```

Arguments:

`param` Parameter name

Returns: The requested parameter value

Method `get_all()`**:** Get all configuration parameters as a list

Usage:

```
MetaRVMConfig$get_all()
```

Returns: Named list of all configuration parameters

Method `list_parameters()`**:** List all available parameter names

Usage:

```
MetaRVMConfig$list_parameters()
```

Returns: Character vector of parameter names

Method `parameter_summary()`**:** Show summary of parameter types and sizes

Usage:

```
MetaRVMConfig$parameter_summary()
```

Returns: Data frame with parameter information

Method `set()`**:** Set a configuration parameter

Usage:

```
MetaRVMConfig$set(param, value)
```

Arguments:

`param` Character string. Parameter name to set

`value` The value to assign to the parameter

Returns: Self (invisible) for method chaining

Method `print()`**:** Print summary of configuration

Usage:

```
MetaRVMConfig$print()
```

Returns: Self (invisible)

Method `get_pop_map()`**:** Get population mapping data

Usage:

```
MetaRVMConfig$get_pop_map()
```

Returns: data.table containing population_id and user-defined demographic category columns

Method `get_category_names()`**:** Get names of all category columns

Usage:

```
MetaRVMConfig$get_category_names()
```

Details: Category columns are automatically detected from the initialization CSV file. Any column that is not a reserved column (population_id, N, S0, I0, R0, V0, etc.) is treated as a demographic category (e.g., age, zone, income_level, occupation).

Returns: Character vector of category column names, or empty vector if no categories

Examples:

```
\dontrun{
config <- MetaRVMConfig$new("config.yaml")
category_names <- config$get_category_names()  # e.g., c("age", "zone", "risk_group")
}
```

Method `get_category_values()`**:** Get unique values for a specific category

Usage:

```
MetaRVMConfig$get_category_values(category_name)
```

Arguments:

`category_name` Character string specifying the category name

Returns: Character/numeric vector of unique values for the specified category

Examples:

```
\dontrun{
config <- MetaRVMConfig$new("config.yaml")
ages <- config$get_category_values("age")  # if age is defined
income_levels <- config$get_category_values("income_level")  # if defined
}
```

Method `get_all_categories()`**:** Get all categories as a named list

Usage:

```
MetaRVMConfig$get_all_categories()
```

Returns: Named list where names are category column names and values are vectors of unique values for each category. Returns empty list if no categories.

Examples:

```
\dontrun{
config <- MetaRVMConfig$new("config.yaml")
all_cats <- config$get_all_categories()
# Returns: list(age = c("0-17", "18-64", "65+"), risk_group = c("low", "high"), ...)
}
```

Method `set_nrep()`**:** Set the number of stochastic replicates per parameter draw

Usage:

```
MetaRVMConfig$set_nrep(n)
```

Arguments:

`n` Positive integer.

Returns: Self (invisible) for method chaining

Method `set_random_seed()`**:** Set the random seed

Usage:

```
MetaRVMConfig$set_random_seed(seed)
```

Arguments:

`seed` Integer.

Returns: Self (invisible) for method chaining

Method `set_start_date()`**:** Set the simulation start date

Usage:

```
MetaRVMConfig$set_start_date(date)
```

Arguments:

`date` A Date object or character string in `"MM/DD/YYYY"` format.

Returns: Self (invisible) for method chaining

Method `set_sim_length()`**:** Set the simulation length in days

Usage:

```
MetaRVMConfig$set_sim_length(n)
```

Arguments:

`n` Positive integer.

Returns: Self (invisible) for method chaining

Method `set_delta_t()`**:** Set the time-step size

Usage:

```
MetaRVMConfig$set_delta_t(dt)
```

Arguments:

`dt` Positive numeric (days). Typically `0.5`.

Returns: Self (invisible) for method chaining

Method `set_run_id()`**:** Set the run identifier

Usage:

```
MetaRVMConfig$set_run_id(id)
```

Arguments:

`id` Character string.

Returns: Self (invisible) for method chaining

Method `set_parallel()`**:** Set whether parallel execution is allowed

Usage:

```
MetaRVMConfig$set_parallel(flag)
```

Arguments:

`flag` Logical.

Returns: Self (invisible) for method chaining

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
MetaRVMConfig$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Arindam Fadikar

Examples

```
# Initialize from YAML file
example_config <- system.file("extdata", "example_config.yaml", package = "MetaRVM")
config <- MetaRVMConfig$new(example_config)

# Access parameters
config$get("N_pop")
config$get("start_date")

# Get demographic category names (user-defined)
category_names <- config$get_category_names()  # e.g., c("age", "zone", "risk_group")

# Get values for specific categories
ages <- config$get_category_values("age")

# Get all categories as a named list
all_categories <- config$get_all_categories()


## ------------------------------------------------
## Method `MetaRVMConfig$get_category_names`
## ------------------------------------------------

## Not run:
config <- MetaRVMConfig$new("config.yaml")
category_names <- config$get_category_names()  # e.g., c("age", "zone", "risk_group")

## End(Not run)

## ------------------------------------------------
## Method `MetaRVMConfig$get_category_values`
## ------------------------------------------------

## Not run:
config <- MetaRVMConfig$new("config.yaml")
ages <- config$get_category_values("age")  # if age is defined
income_levels <- config$get_category_values("income_level")  # if defined

## End(Not run)

## ------------------------------------------------
## Method `MetaRVMConfig$get_all_categories`
## ------------------------------------------------

## Not run:
config <- MetaRVMConfig$new("config.yaml")
all_cats <- config$get_all_categories()
# Returns: list(age = c("0-17", "18-64", "65+"), risk_group = c("low", "high"), ...)
```

```
## End(Not run)
```

`MetaRVMResults` *MetaRVM Results Class*

Description

R6 class to handle MetaRVM simulation results with comprehensive analysis and visualization methods. This class stores formatted simulation results and provides methods for data summarization, subsetting, and visualization with flexible demographic groupings.

Details

The MetaRVMResults class automatically formats raw simulation output upon initialization, converting time steps to calendar dates and adding demographic attributes. It provides methods for flexible data summarization across any user-defined demographic categories, plus method chaining for streamlined analysis workflows.

Public fields

`config` MetaRVMConfig object used to generate these results

`results` data.table containing formatted simulation results

`run_info` List containing run metadata

Methods

Public methods:

- `MetaRVMResults$new()`
- `MetaRVMResults$print()`
- `MetaRVMResults$plot()`
- `MetaRVMResults$subset_data()`
- `MetaRVMResults$summarize()`
- `MetaRVMResults$get_instance_manifest()`
- `MetaRVMResults$get_instance_params()`
- `MetaRVMResults$clone()`

Method `new()`**:** Initialize a new MetaRVMResults object

Usage:

```
MetaRVMResults$new(
  raw_results,
  config,
  run_info = NULL,
  formatted_results = NULL
)
```

Arguments:

`raw_results` Raw simulation results data.table

`config` MetaRVMConfig object used for the simulation

`run_info` Optional metadata about the run

`formatted_results` formatted simulation results data.table

Returns: New MetaRVMResults object (invisible)

Method `print()`**:** Print summary of results

Usage:

```
MetaRVMResults$print()
```

Returns: Self (invisible)

Method `plot()`**:** Plot simulation trajectories directly from results

Usage:

```
MetaRVMResults$plot(
  group_by = character(0),
  disease_states = NULL,
  date_range = NULL,
  instances = NULL,
  stats = NULL,
  quantiles = c(0.25, 0.75),
  exclude_p_columns = TRUE,
  ci_level = 0.95,
  theme = theme_minimal(),
  title = NULL
)
```

Arguments:

`group_by` Vector of demographic category names to group/facet by

`disease_states` Optional disease states to include

`date_range` Optional date range for filtering

`instances` Optional instance IDs to include

`stats` Statistics for summary plotting. If NULL, plots raw trajectories.

`quantiles` Quantiles for uncertainty bands when summary plotting

`exclude_p_columns` Logical, whether to exclude p_ columns (default: TRUE)

`ci_level` Confidence level label used in summary plot title

`theme` ggplot2 theme function (default: theme_minimal())

`title` Optional custom title

Returns: ggplot object

Method `subset_data()`**:** Subset the data based on any combination of parameters

Usage:

```
MetaRVMResults$subset_data(
  ...,
  disease_states = NULL,
  date_range = NULL,
  instances = NULL,
  exclude_p_columns = TRUE
)
```

Arguments:

`...` Named arguments for category filters (e.g., age = c("0-17"), income = c("low", "high"))

`disease_states` Vector of disease states to include (default: all, excludes p_ columns)

`date_range` Vector of two dates start_date, and end_date for filtering (default: all)

`instances` Vector of instance numbers to include (default: all)

`exclude_p_columns` Logical, whether to exclude p_ columns (default: TRUE)

Returns: MetaRVMResults object with subset of results

Method `summarize()`**:** Summarize results across specified demographic characteristics

Usage:

```
MetaRVMResults$summarize(
  group_by,
  disease_states = NULL,
  date_range = NULL,
  stats = c("mean", "median", "sd"),
  quantiles = c(0.25, 0.75),
  exclude_p_columns = TRUE
)
```

Arguments:

`group_by` Vector of demographic category names to group by. Must be valid category names from the configuration (e.g., c("age", "zone"), c("income_level", "occupation")). Use config$get_category_names() to see available categories.

`disease_states` Vector of disease states to include (default: all, excludes p_ columns)

`date_range` Optional date range for filtering

`stats` Vector of statistics to calculate: c("mean", "median", "sd", "min", "max", "sum", "quantile"). If NULL, returns all instances

`quantiles` Vector of quantiles to calculate if "quantile" is in stats (default: c(0.25, 0.75))

`exclude_p_columns` Logical, whether to exclude p_ columns (default: TRUE)

Returns: data.table with summarized time series data or all instances if stats = NULL

Method `get_instance_manifest()`**:** Get per-instance simulation manifest (parameter-set id, replicate id, seed, and settings)

Usage:

```
MetaRVMResults$get_instance_manifest(instances = NULL, include_sim_args = TRUE)
```

Arguments:

`instances` Optional vector of instance ids to filter

`include_sim_args` Logical. If FALSE, omits the `sim_args` list-column for a lighter table

Returns: data.table with one row per simulation instance

Method `get_instance_params()`**:** Get per-instance settings table with parameters list-column (`params`)

Usage:

```
MetaRVMResults$get_instance_params(instances = NULL)
```

Arguments:

`instances` Optional vector of instance ids to filter

Returns: data.table with instance mapping and exact parameter values used in simulation

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
MetaRVMResults$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Arindam Fadikar

Examples

```
options(odin.verbose = FALSE)
example_config <- system.file("extdata", "example_config.yaml", package = "MetaRVM")
# Run simulation
results_obj <- metaRVM(example_config)
# Access formatted results
head(results_obj$results)

# Subset data with multiple filters
subset_data <- results_obj$subset_data(
  age = c("18-64", "65+"),
  disease_states = c("H", "D"),
  date_range = c(as.Date("2024-01-01"), as.Date("2024-02-01"))
)

# Method chaining for analysis and visualization
results_obj$subset_data(disease_states = "H")$summarize(
  group_by = c("age", "zone"),
  stats = c("median", "quantile"),
  quantiles = c(0.25, 0.75)
)$plot()
```

`MetaRVMSummary` *MetaRVM Summary Class*

Description

R6 class for summarized MetaRVM results with plotting capabilities and method chaining support. This class stores summarized simulation data and provides visualization methods that automatically adapt based on the data structure and grouping variables.

Details

The MetaRVMSummary class is designed to work seamlessly with method chaining from MetaRVM-Results. It stores either summary statistics (mean, median, quantiles, etc.) or individual instance data, and provides intelligent plotting methods that automatically determine appropriate visualizations based on the data structure and demographic groupings.

The class supports two data types:

- **Summary data**: Contains aggregated statistics across simulation instances
- **Instance data**: Contains individual trajectory data for each simulation instance

Plotting behavior adapts automatically:

- Single grouping variable: Facets by demographic category, colors by disease state
- Two grouping variables: Grid layout with both demographics as facet dimensions
- Three grouping variables: Grid layout with first two as facets, third as color

Public Fields

`data` data.table containing summarized results

`config` MetaRVMConfig object from original simulation

`type` Character string indicating data type ("summary" or "instances")

Public fields

`data` Summarized data

`config` Original MetaRVMConfig object

`type` Type of summary ("instances" or "summary")

Methods

Public methods:

- `MetaRVMSummary$new()`
- `MetaRVMSummary$print()`
- `MetaRVMSummary$plot()`
- `MetaRVMSummary$clone()`

Method `new()`**:** Initialize MetaRVMSummary object

Usage:

```
MetaRVMSummary$new(data, config, type)
```

Arguments:

`data` data.table containing summarized or instance data

`config` MetaRVMConfig object from original simulation

`type` Character string indicating data type ("summary" or "instances")

Returns: New MetaRVMSummary object (invisible)

Method `print()`**:** Print summary of the data object

Usage:

```
MetaRVMSummary$print()
```

Returns: Self (invisible)

Method `plot()`**:** Plot method that shows median with quantile bands

Usage:

```
MetaRVMSummary$plot(ci_level = 0.95, theme = theme_minimal(), title = NULL)
```

Arguments:

`ci_level` Confidence level for empirical quantiles (default: 0.95). Only used if quantile columns are not pre-specified

`theme` ggplot2 theme function (default: theme_minimal())

`title` Optional custom plot title

Details: This method creates time series plots with automatic layout adaptation based on grouping variables:

- For summary data: Shows median lines with quantile confidence bands
- Automatically determines faceting strategy based on number of grouping variables
- Uses disease states for color differentiation when appropriate

The method requires specific data structure:

- Summary data must contain 'median_value' and quantile columns (e.g., 'q25', 'q75')
- Instance data must contain 'instance' column for individual trajectory grouping

Returns: ggplot object

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
MetaRVMSummary$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Arindam Fadikar

Examples

```
options(odin.verbose = FALSE)
example_config <- system.file("extdata", "example_config_dist.yaml", package = "MetaRVM")
# Run simulation
results <- metaRVM(example_config)
# Typically created through method chaining
summary_obj <- results$subset_data(disease_states = "H")$summarize(
  group_by = c("age", "zone"),
  stats = c("median", "quantile"),
  quantiles = c(0.25, 0.75)
)

# Direct plotting
summary_obj$plot()

# Plot with custom ggplot theme and confidence level
summary_obj$plot(theme = ggplot2::theme_bw())
```

---

meta_measles_sim *Metapopulation Measles Simulator (ODIN Core)*

Description

Core ODIN-based simulation engine for a measles metapopulation model with two exposed stages (E1, E2) and two infectious stages (I1, I2), each split into quarantined (_Q) and unquarantined (_U) compartments.

Usage

```
meta_measles_sim(
  N_pop = 1,
  beta,
  S0 = rep(1000, N_pop),
  V10 = rep(0, N_pop),
  V20 = rep(0, N_pop),
  E10 = rep(0, N_pop),
  E20 = rep(0, N_pop),
  I1_Q0 = rep(0, N_pop),
  I1_U0 = rep(10, N_pop),
  I2_Q0 = rep(0, N_pop),
  I2_U0 = rep(0, N_pop),
  R0 = rep(0, N_pop),
  V1E10 = rep(0, N_pop),
  V1E20 = rep(0, N_pop),
  V1I1_Q0 = rep(0, N_pop),
  V1I1_U0 = rep(0, N_pop),
```

```
  V1I2_Q0 = rep(0, N_pop),
  V1I2_U0 = rep(0, N_pop),
  V1R0 = rep(0, N_pop),
  V2E10 = rep(0, N_pop),
  V2E20 = rep(0, N_pop),
  V2I1_Q0 = rep(0, N_pop),
  V2I1_U0 = rep(0, N_pop),
  V2I2_Q0 = rep(0, N_pop),
  V2I2_U0 = rep(0, N_pop),
  V2R0 = rep(0, N_pop),
  m_weekday_day,
  m_weekday_night,
  m_weekend_day,
  m_weekend_night,
  start_day = 0,
  delta_t,
  de1,
  de2,
  di1,
  di2,
  prop_I1_Q,
  prop_I2_Q,
  V1_eff,
  V2_eff,
  nsteps,
  is.stoch = FALSE,
  seed = NULL,
  do_chk = FALSE,
  chk_time_steps = NULL,
  chk_file_names = NULL
)
```

Arguments

N_pop	Integer. Number of subpopulations.
beta	Numeric vector or scalar. Transmission rate by subpopulation.
S0	Numeric vector of length N_pop. Initial susceptible counts.
V10	Numeric vector of length N_pop. Initial dose-1 vaccinated counts.
V20	Numeric vector of length N_pop. Initial dose-2 vaccinated counts.
E10	Numeric vector of length N_pop. Initial first exposed stage counts.
E20	Numeric vector of length N_pop. Initial second exposed stage counts.
I1_Q0	Numeric vector of length N_pop. Initial I1_Q counts.
I1_U0	Numeric vector of length N_pop. Initial I1_U counts.
I2_Q0	Numeric vector of length N_pop. Initial I2_Q counts.
I2_U0	Numeric vector of length N_pop. Initial I2_U counts.
R0	Numeric vector of length N_pop. Initial recovered counts.

V1E10	Numeric vector of length N_pop. Initial V1E1 counts (dose-1 breakthrough, first latent stage).
V1E20	Numeric vector of length N_pop. Initial V1E2 counts (dose-1 breakthrough, second latent stage).
V1I1_Q0	Numeric vector of length N_pop. Initial V1I1_Q counts.
V1I1_U0	Numeric vector of length N_pop. Initial V1I1_U counts.
V1I2_Q0	Numeric vector of length N_pop. Initial V1I2_Q counts.
V1I2_U0	Numeric vector of length N_pop. Initial V1I2_U counts.
V1R0	Numeric vector of length N_pop. Initial V1R counts.
V2E10	Numeric vector of length N_pop. Initial V2E1 counts (dose-2 breakthrough, first latent stage).
V2E20	Numeric vector of length N_pop. Initial V2E2 counts (dose-2 breakthrough, second latent stage).
V2I1_Q0	Numeric vector of length N_pop. Initial V2I1_Q counts.
V2I1_U0	Numeric vector of length N_pop. Initial V2I1_U counts.
V2I2_Q0	Numeric vector of length N_pop. Initial V2I2_Q counts.
V2I2_U0	Numeric vector of length N_pop. Initial V2I2_U counts.
V2R0	Numeric vector of length N_pop. Initial V2R counts.
m_weekday_day	Numeric matrix (N_pop x N_pop). Weekday daytime mixing matrix.
m_weekday_night	Numeric matrix (N_pop x N_pop). Weekday nighttime mixing matrix.
m_weekend_day	Numeric matrix (N_pop x N_pop). Weekend daytime mixing matrix.
m_weekend_night	Numeric matrix (N_pop x N_pop). Weekend nighttime mixing matrix.
start_day	Integer in 0:6, where 0 is Monday.
delta_t	Numeric. Simulation time step in days (typically 0.5).
de1	Numeric vector or scalar. Mean duration (days) in E1.
de2	Numeric vector or scalar. Mean duration (days) in E2.
di1	Numeric vector or scalar. Mean duration (days) in I1.
di2	Numeric vector or scalar. Mean duration (days) in I2.
prop_I1_Q	Numeric vector or scalar in [0, 1]. Proportion entering I1_Q from E2.
prop_I2_Q	Numeric vector or scalar in [0, 1]. Proportion of I1_U -> I2 transitions entering I2_Q.
V1_eff	Numeric vector or scalar in [0, 1]. Vaccine efficacy for dose-1 recipients.
V2_eff	Numeric vector or scalar in [0, 1]. Vaccine efficacy for dose-2 recipients.
nsteps	Integer. Number of simulation steps.
is.stoch	Logical. Included for API compatibility; this model currently uses stochastic binomial transitions.
seed	Integer or NULL. Included for API compatibility.
do_chk	Logical. Whether to write checkpoint files at the specified time steps.
chk_time_steps	Integer vector or NULL. Simulation steps at which to write checkpoints.
chk_file_names	Character vector/list or NULL. File paths for each checkpoint time step.

Details

Compartments evolve as:

- S -> E1 -> E2 -> I1_Q/I1_U -> I2_Q/I2_U -> R

Force of infection is computed from unquarantined infectious individuals (I1_U + I2_U) under time-varying day/night and weekday/weekend mixing.

Value

A data.table in long format with columns:

step Time-step index (0:nsteps).

time Continuous simulation time (step * delta_t).

disease_state Compartment/metric name (for example S, E1, I1_U, I_all, n_E2I1).

population_id Subpopulation id (integer).

value Compartment value at that time and subpopulation.

Examples

```
options(odin.verbose = FALSE)
N_pop <- 2
mix <- diag(N_pop)
out <- meta_measles_sim(
  N_pop = N_pop,
  beta = 0.7,
  S0 = c(1000, 900),
  I1_U0 = c(5, 3),
  m_weekday_day = mix,
  m_weekday_night = mix,
  m_weekend_day = mix,
  m_weekend_night = mix,
  delta_t = 0.5,
  de1 = 3, de2 = 3, di1 = 2, di2 = 3,
  prop_I1_Q = 0.2, prop_I2_Q = 0.4,
  V1_eff = 0.93, V2_eff = 0.97,
  nsteps = 20
)
head(out)
```

meta_sim *Metapopulation Respiratory Virus Model Simulator*

Description

The core simulation engine that implements a stochastic compartmental SEIRD (Susceptible-Exposed-Infected-Recovered-Dead) model for respiratory virus epidemics across multiple demographic subpopulations. The function compiles and executes an ODIN-based differential equation model with time-varying contact patterns, vaccination dynamics, and complex disease progression pathways.

Usage

```
meta_sim(
  N_pop,
  ts,
  S0,
  I0,
  P0,
  R0,
  H0 = rep(0, N_pop),
  D0 = rep(0, N_pop),
  Ia0 = rep(0, N_pop),
  Ip0 = rep(0, N_pop),
  E0 = rep(0, N_pop),
  V0 = rep(0, N_pop),
  m_weekday_day,
  m_weekday_night,
  m_weekend_day,
  m_weekend_night,
  start_day = 0,
  delta_t,
  vac_mat,
  dv,
  de,
  pea,
  dp,
  da,
  ds,
  psr,
  dh,
  phr,
  dr,
  ve,
  nsteps,
  is.stoch = FALSE,
  seed = NULL,
  do_chk = FALSE,
```

```
  chk_time_steps = NULL,
  chk_file_names = NULL
)
```

Arguments

N_pop	Integer. Number of demographic subpopulations in the model
ts	Numeric vector or scalar. Transmission rate for symptomatic individuals in susceptible population. If scalar, applied to all subpopulations
S0	Numeric vector of length N_pop. Initial number of susceptible individuals in each subpopulation
I0	Numeric vector of length N_pop. Initial number of symptomatic infected individuals in each subpopulation
P0	Numeric vector of length N_pop. Total population sizes for each subpopulation
R0	Numeric vector of length N_pop. Initial number of recovered individuals in each subpopulation
H0	Numeric vector of length N_pop. Initial number of hospitalized individuals in each subpopulation (default: rep(0, N_pop))
D0	Numeric vector of length N_pop. Initial number of deceased individuals in each subpopulation (default: rep(0, N_pop))
Ia0	Numeric vector of length N_pop. Initial number of asymptomatic infected individuals in each subpopulation (default: rep(0, N_pop))
Ip0	Numeric vector of length N_pop. Initial number of presymptomatic infected individuals in each subpopulation (default: rep(0, N_pop))
E0	Numeric vector of length N_pop. Initial number of exposed individuals in each subpopulation (default: rep(0, N_pop))
V0	Numeric vector of length N_pop. Initial number of vaccinated individuals in each subpopulation
m_weekday_day	Numeric matrix (N_pop × N_pop). Contact mixing matrix for weekday daytime (6 AM - 6 PM) interactions
m_weekday_night	Numeric matrix (N_pop × N_pop). Contact mixing matrix for weekday nighttime (6 PM - 6 AM) interactions
m_weekend_day	Numeric matrix (N_pop × N_pop). Contact mixing matrix for weekend daytime (6 AM - 6 PM) interactions
m_weekend_night	Numeric matrix (N_pop × N_pop). Contact mixing matrix for weekend nighttime (6 PM - 6 AM) interactions
start_day	Start day of the week expressed as an integer value between 0 and 6, 0 being Monday. Default simulation start day is Monday.
delta_t	Positive numeric. Discrete time increment in days (typically 0.5)
vac_mat	Numeric matrix. Vaccination schedule with dimensions (nsteps × (1 + N_pop)). First column contains time indices, remaining columns contain vaccination counts for each subpopulation at each time step

dv	Numeric vector or scalar. Mean duration (days) in vaccinated state before immunity waning. If scalar, applied to all subpopulations
de	Numeric vector or scalar. Mean duration (days) in exposed state. If scalar, applied to all subpopulations
pea	Numeric vector or scalar. Proportion of exposed individuals becoming asymptomatic infectious (vs. presymptomatic), values between 0 and 1. If scalar, applied to all subpopulations. If scalar, applied to all subpopulations
dp	Numeric vector or scalar. Mean duration (days) in presymptomatic infectious state. If scalar, applied to all subpopulations
da	Numeric vector or scalar. Mean duration (days) in asymptomatic infectious state. If scalar, applied to all subpopulations
ds	Numeric vector or scalar. Mean duration (days) in symptomatic infectious state. If scalar, applied to all subpopulations
psr	Numeric vector or scalar. Proportion of symptomatic individuals recovering directly (vs. hospitalization), values between 0 and 1. If scalar, applied to all subpopulations. If scalar, applied to all subpopulations
dh	Numeric vector or scalar. Mean duration (days) in hospitalized state. If scalar, applied to all subpopulations
phr	Numeric vector or scalar. Proportion of hospitalized individuals recovering (vs. death). , values between 0 and 1. If scalar, applied to all subpopulations.
dr	Numeric vector or scalar. Mean duration (days) of immunity in recovered state. If scalar, applied to all subpopulations
ve	Numeric vector or scalar. Vaccine effectiveness (proportion) , values between 0 and 1. If scalar, applied to all subpopulations
nsteps	Integer. Total number of discrete time evolution steps in simulation
is.stoch	Logical. Whether to run the simulation stochastically. Should always be TRUE; present for interface compatibility.
seed	Integer or NULL. Random seed for reproducibility. Only used when is.stoch = TRUE. Default: NULL
do_chk	Logical. Whether to save model checkpoint at simulation end. Default: FALSE
chk_time_steps	Integer vector or NULL. Time steps at which to save checkpoints.
chk_file_names	List of character vectors or NULL. File names for checkpoints. Each element of the list corresponds to a time step in chk_time_steps.

Details

The model implements a metapopulation epidemiological framework with the following features:

Compartmental Structure:

- **S**: Susceptible individuals
- **E**: Exposed (incubating) individuals
- **I_presymp**: Presymptomatic infectious individuals
- **I_asymp**: Asymptomatic infectious individuals

- **I_symp**: Symptomatic infectious individuals
- **H**: Hospitalized individuals
- **R**: Recovered individuals
- **D**: Deceased individuals
- **V**: Vaccinated individuals
- **P**: Total living population (excludes deaths)

Disease Progression Pathways:

1. **S** → **E**: Exposure through contact with infectious individuals
2. **E** → **I_asymp/I_presymp**: Progression to infectious states (proportion pea)
3. **I_presymp** → **I_symp**: Development of symptoms
4. **I_asymp** → **R**: Direct recovery from asymptomatic state
5. **I_symp** → **R/H**: Recovery or hospitalization (proportion psr)
6. **H** → **R/D**: Hospital discharge or death (proportion phr)
7. **R** → **S**: Loss of immunity
8. **S** → **V**: Vaccination
9. **V** → **S/E**: Vaccine waning or breakthrough infection

Mixing Patterns: Contact patterns vary by:

- Day of week: Weekday vs. weekend patterns
- Time of day: Day (6 AM - 6 PM) vs. night (6 PM - 6 AM) patterns
- Each pattern specified by N_pop × N_pop contact matrix

Force of Infection: Transmission occurs through contact between susceptible/vaccinated individuals and all infectious compartments (I_presymp + I_asymp + I_symp), modified by:

- Population-specific transmission rate, ts
- Time-varying contact patterns
- Vaccine effectiveness for breakthrough infections

Stochastic Mode: Demographic stochasticity is added via binomial draws at each transition.

Vaccination Implementation: Vaccination is implemented as time-varying input with:

- Scheduled vaccination counts per time step and subpopulation
- Vaccine effectiveness reducing infection probability
- Waning immunity returning individuals to susceptible state

Value

Returns a data.table with the following structure:

step Integer time step index (0 to nsteps)

time Continuous simulation time (step × delta_t)

disease_state Character vector of compartment names

population_id Character vector of subpopulation identifiers

value Numeric values representing population counts in each compartment

Available disease states in output:

- Core compartments: S, E, I_presymp, I_asymp, I_symp, H, R, D, V, P
- Derived outputs: I_all (total infectious), cum_V (cumulative vaccinations)
- Transition flows: n_SE, n_EI, n_HR, n_HD, etc. (new infections, hospitalizations, deaths)
- Debug outputs: p_SE, p_VE, I_eff (probabilities and effective populations)

Parameter Scaling

All duration parameters are automatically converted to rates (1/duration). Scalar parameters are automatically expanded to vectors of length N_pop. This allows flexible specification of homogeneous or heterogeneous parameters.

Checkpointing

When do_chk = TRUE, the function saves a checkpoint file containing:

- Final compartment states for simulation continuation
- All model parameters for reproducibility
- Vaccination schedule data
- Population structure information

Author(s)

Arindam Fadikar, Charles Macal, Ignacio Martinez-Moyano, Jonathan Ozik

References

- ODIN package: https://mrc-ide.github.io/odin/
- Fadikar, A., et al. "Developing and deploying a use-inspired metapopulation modeling framework for detailed tracking of stratified health outcomes"

See Also

metaRVM for high-level simulation interface with configuration files parse_config for configuration file processing format_metarvm_output for output formatting with demographics

Examples

```
options(odin.verbose = FALSE)
# Basic simulation
N_pop <- 2
nsteps <- 400

# Initialize populations
S0 <- rep(1000, N_pop)
I0 <- rep(10, N_pop)
P0 <- S0 + I0
R0 <- rep(0, N_pop)

# Contact matrices (simplified - identity matrices)
contact_matrix <- diag(N_pop)

# Basic vaccination schedule (10% vaccination)
vac_mat <- matrix(0, nrow = nsteps + 1, ncol = N_pop + 1)
vac_mat[, 1] <- 0:nsteps
vac_mat[1, 1 + (1:N_pop)] <- P0 * 0.1

# Run simulation
results <- meta_sim(
  N_pop = N_pop,
  ts = 0.5,
  S0 = S0,
  I0 = I0,
  P0 = P0,
  R0 = R0,
  m_weekday_day = contact_matrix,
  m_weekday_night = contact_matrix,
  m_weekend_day = contact_matrix,
  m_weekend_night = contact_matrix,
  delta_t = 0.5,
  vac_mat = vac_mat,
  dv = 365,
  de = 3,
  pea = 0.3,
  dp = 2,
  da = 7,
  ds = 7,
  psr = 0.95,
  dh = 10,
  phr = 0.9,
  dr = 180,
  ve = 0.8,
  nsteps = nsteps,
  is.stoch = FALSE
)
```

parse_config *Parse MetaRVM Configuration File*

Description

Reads and parses a YAML configuration file for MetaRVM simulations, extracting all necessary parameters for epidemic modeling including population data, disease parameters, mixing matrices, vaccination schedules, and simulation settings.

Usage

```
parse_config(config_file, return_object = FALSE)
```

Arguments

config_file	Character string. Path to a YAML configuration file containing model parameters and settings.
return_object	Logical. If TRUE, returns a MetaRVMConfig object for method chaining and enhanced functionality. If FALSE (default), returns a named list for backward compatibility.

Details

The function processes a YAML configuration file with the following main sections:

Model Selection:

- model.disease: Required. Selects the disease to simulate; supported values are "flu" and "measles". The disease choice determines which disease parameters are expected, the population initialization CSV column schema, and whether a vaccination schedule file is used (flu only; see below).

Simulation Configuration:

- random_seed: Optional random seed for reproducibility in case of stochastic simulations or stochastic parameters
- nsim: Number of simulation instances (default: 1)
- nrep: Number of stochastic replicates per parameter set (default: 1) #'
- start_date: Simulation start date in MM/DD/YYYY format
- length: Simulation length in days
- checkpoint_dir: Optional checkpoint directory for saving intermediate results
- checkpoint_dates: Optional list of dates to save checkpoints.
- restore_from: Optional path to restore simulation from checkpoint

Population Data: The required initialization CSV columns are disease-specific; any additional columns are treated as demographic categories.

- **Flu**: population_id, N, S0, I0, V0, R0 (optional: E0, Ia0, Ip0, H0, D0, defaulted to 0 when absent).
- **Measles**: population_id, N, S0, and at least one of I1_U0, I2_U0 (optional: E10, E20, I1_Q0, I2_Q0, R0, V10, V20, and the vaccinated-breakthrough compartment columns, all defaulted to 0 when absent). Measles has no I0 column.

- vaccination: CSV file with a time-varying vaccination schedule. **Flu only** — the first column must be dates in MM/DD/YYYY format, and the rest of the columns must correspond to respective subpopulations in the numeric order of population_id. Measles vaccination is specified instead via the initial V10/V20 compartment counts in the initialization file; no separate vaccination file is used.

Mixing Matrices: Contact matrices for different time periods. Each CSV file must have a matrix of order (N_pop x N_pop), where, N_pop is the number of subpopulations. It is assumed that the i-th row and j-th column correspond to i-th and j-th subpopulations.

- weekday_day, weekday_night: Weekday contact patterns
- weekend_day, weekend_night: Weekend contact patterns

Disease Parameters: Epidemiological parameters (can be scalars or distributions). The exact parameter names are disease-specific:

- **Flu**: ts, ve, dv, de, dp, da, ds, dh, dr, pea, psr, phr
- **Measles**: beta, de1, de2, di1, di2, prop_I1_Q, prop_I2_Q, V1_eff, V2_eff

Sub-population Parameters: sub_disease_params allows specification of different parameter values for specific demographic categories (e.g., age groups, races).

The function supports stochastic parameters through distribution specifications with dist, mu, sd, shape, rate, etc.

Value

If return_object = FALSE (default), returns a named list containing entries common to both diseases plus disease-specific initial compartments and parameter matrices:

disease The resolved disease name ("flu" or "measles")

N_pop Number of population groups

pop_map Data.table with population_id and user-defined demographic categories

Initial compartments For flu: S_ini, E_ini, I_asymp_ini, I_presymp_ini, I_symp_ini, H_ini, D_ini, V_ini, R_ini. For measles: S_ini, E1_ini, E2_ini, I1_Q_ini, I1_U_ini, I2_Q_ini, I2_U_ini, R_ini, V1_ini, V2_ini plus the vaccinated-breakthrough compartment initials.

vac_time_id, vac_counts, vac_mat Vaccination schedule data (flu only; not present for measles, which instead folds vaccination into the initial V1_ini/V2_ini compartments)

m_wd_d, m_wd_n, m_we_d, m_we_n Contact mixing matrices

Disease parameter matrices (nsim × N_pop) For flu: ts, ve, dv, de, dp, da, ds, dh, dr, pea, psr, phr. For measles: beta, de1, de2, di1, di2, prop_I1_Q, prop_I2_Q, V1_eff, V2_eff.

start_date Simulation start date as Date object

sim_length Simulation length in days

nsim Number of simulation instances

nrep Number of stochastic replicates per parameter set

random_seed Random seed used (auto-generated if not supplied)

delta_t Time step size (fixed at 0.5)

chk_file_names, chk_time_steps, do_chk Checkpointing configuration

If `return_object = TRUE`, returns a `MetaRVMConfig` object with methods for parameter access and validation.

Parameter Distributions

Disease parameters can be specified as distributions for stochastic modeling:

- **lognormal**: `dist: "lognormal", mu: value, sd: value`
- **gamma**: `dist: "gamma", shape: value, rate: value`
- **uniform**: `dist: "uniform", min: value, max: value`
- **beta**: `dist: "beta", shape1: value, shape2: value`
- **gaussian**: `dist: "gaussian", mean: value, sd: value`

File Requirements

Population initialization file must contain columns (see Population Data above for the full disease-specific list):

- `population_id`: Unique identifier for each population group, natural numbers
- `N`: Total population size of the subpopulation
- **Flu**: `S0`, `I0`, `V0`, `R0`
- **Measles**: `S0`, and at least one of `I1_U0`, `I2_U0`
- Optional user-defined category columns (e.g., `age`, `race`, `zone`, `income_level`, `occupation`)

Vaccination file (flu only) must contain: `date` (MM/DD/YYYY format) and vaccination counts for each population group. Not used for measles.

Author(s)

Arindam Fadikar

See Also

`metaRVM` for running simulations with parsed configuration `MetaRVMConfig` for the configuration object class `process_vac_data` for vaccination data processing

Examples

```
options(odin.verbose = FALSE)
example_config <- system.file("extdata", "example_config.yaml", package = "MetaRVM")
# Parse configuration file and return list (backward compatible)
config <- parse_config(example_config)

# Parse and return MetaRVMConfig object for method chaining
config_obj <- parse_config(example_config, return_object = TRUE)

# Access parameters from config object
config_obj$get("N_pop")
config_obj$list_parameters()

# Use with MetaRVM simulation
results <- metaRVM(config_obj)
```

Index

build_config, 2

copy_config, 5

Date, *3*, *14*

format_metarvm_output, 6, *30*
format_metarvm_output(), *8*

meta_measles_sim, 22
meta_measles_sim(), *6*
meta_sim, 26
meta_sim(), *6*, *8*, *9*
metaRVM, 7, *30*, *34*
metaRVM(), *4*, *6*, *10*
MetaRVM::MetaRVMConfig, *9*
MetaRVMCheck, 9
MetaRVMConfig, *2*, *4*, *6–10*, 10, *34*
MetaRVMResults, *6–8*, 16
MetaRVMSummary, 20

parse_config, *30*, 32
parse_config(), *4*, *7*, *8*
process_vac_data, *34*