

Package: IPDMR (via r-universe)

September 26, 2026

Title Support Code for the Introduction to Practical Disease Modelling Course

Version 0.5.2-1

Date 2025-02-07

Description A collection of functions and classes that are used to illustrate fundamental concepts of disease modelling as part of a physical course series taught jointly by the University of Copenhagen and University of Sydney. The package is not intended to be used for other purposes; you are welcome to do so, but please note that the package may change without notice in order to suit changes to the associated taught course. All functions, arguments and classes are therefore subject to change without following any form of deprecation process.

License GPL (>= 3)

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Depends R (>= 4.3.0)

LinkingTo Rcpp

Imports methods, stats, rlang, dplyr, tibble, deSolve, checkmate, ggplot2, tidyr, stringr, R6, Rcpp, pbapply

Suggests knitr, rmarkdown, spelling, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

Language en-GB

Config/pak/sysreqs libicu-dev

Repository <https://ku-awdc.r-universe.dev>

Date/Publication 2025-02-07 16:40:00 UTC

RemoteUrl <https://github.com/ku-awdc/IPDMR>

RemoteRef v0.5.2

RemoteSha 1f0b57d0413a2cee28a7a6a05c1ab502d76ee1d4

Contents

ab_continuous	2
ab_discrete	2
apply_rates	3
autoplot.ipdmr_ct	4
BetweenGroupClass	5
group_models	8
make_group	10
multi_models	11
SEIRclass	13
si_continuous	15
si_discrete	16
suggest_boxes	16
Index	18

ab_continuous	<i>A super-simple continuous-time two-compartment model</i>
---------------	---

Description

A super-simple continuous-time two-compartment model

Usage

```
ab_continuous(rate = 0.1, time_points = seq(0, 10, by = 0.1))
```

Arguments

- rate the rate of progression from A to B per unit time (must be positive)
- time_points a vector of time points to include in the data frame returned

ab_discrete	<i>A super-simple discrete-time two-compartment model</i>
-------------	---

Description

A super-simple discrete-time two-compartment model

Usage

```
ab_discrete(rate = 0.1, d_time = 0.1, max_time = 10)
```

Arguments

rate	the rate of progression from A to B per unit time (must be positive)
d_time	the desired time step (delta time)
max_time	the desired maximum time point (must be greater than the time step)

apply_rates	<i>Combine compartment sizes and rates to obtain compartment change values</i>
-------------	--

Description

This is utility function taking a vector of compartments and rates, and returning a matrix of calculated (for deterministic) or sampled (for stochastic) values for the change in compartment size from each input compartment (row) to each of the output compartments represented by the input rates (columns). When only a single rate is input then the function just returns the usual compartment * 1-exp(-rated_time) (for deterministic) or rbinom(length(compartment), compartment, 1-exp(-rated_time)) (for stochastic). Where there are multiple rates, then the function corrects for the overlap in the competing rates and (for stochastic) uses a multinomial distribution instead of a binomial distribution to ensure that the sum of each row of output is less than or equal to the corresponding input compartment size.

Multiple rows are only relevant where there are multiple sub-compartments, i.e. movement from the E state. In all other cases, a matrix with a single row is returned. Likewise, multiple columns are only relevant where there are multiple rates - in all other cases, a matrix with a single column is returned. If you know that you have a single compartment and/or a single rate, then you can either explicitly select the first row and/or column or use as.numeric() on the output to remove the dimensions attribute (see the bottom of the examples).

Usage

```
apply_rates(
  compartments,
  rates,
  d_time,
  update_type = c("deterministic", "stochastic")
)
```

Arguments

compartments	a vector of compartment sizes on which to apply all rates
rates	a vector of rates representing movement from each of the compartments
d_time	the size of the time step
update_type	deterministic or stochastic

Examples

```
## A single compartment and rate:
I <- 10; gamma <- 0.1; d_time <- 1
apply_rates(I, gamma, d_time, "deterministic")
### Is the same as:
I * (1 - exp(-gamma*d_time))
### Stochastic version uses rbinom:
apply_rates(I, gamma, d_time, "stochastic")

## A single compartment and vector of rates:
cull <- 0.01
apply_rates(I, c(gamma,cull), d_time, "deterministic")
## Due to competing rates, this is NOT the same as:
I * (1 - exp(-gamma*d_time))
I * (1 - exp(-cull*d_time))
### Stochastic version uses rmultinom:
apply_rates(I, c(gamma,cull), d_time, "stochastic")

## A vector of compartments and single rate:
E <- c(0,1,2); omega <- 0.1
apply_rates(E, omega, d_time, "deterministic")
### Is the same as:
apply_rates(E[1], omega, d_time, "deterministic")
apply_rates(E[2], omega, d_time, "deterministic")
apply_rates(E[3], omega, d_time, "deterministic")
### Is the same as:
E * (1 - exp(-omega*d_time))
### Stochastic version uses rbinom:
apply_rates(E, omega, d_time, "stochastic")

## A vector of compartments and vector of rates:
repl <- 0.001
apply_rates(E, c(omega,repl), d_time, "deterministic")
apply_rates(E, c(omega,repl), d_time, "stochastic")

## If the rate vector has names, then these are carried over to the
## output column names:
apply_rates(I, c(toR=gamma,toS=cull), d_time, "deterministic")

## If you only have a single rate (and/or single compartment) then you
## probably want to remove the dimensions attribute before using, i.e.:
apply_rates(E, omega, d_time, "deterministic")[,1]
apply_rates(I, c(gamma,cull), d_time, "deterministic")[1,]
## Or:
as.numeric(apply_rates(E, omega, d_time, "deterministic"))
as.numeric(apply_rates(I, c(gamma,cull), d_time, "deterministic"))
```

Description

Autoplot methods associated with IPDMR model functions

Usage

```
## S3 method for class 'ipdmr_ct'
autoplot(object, ...)
```

Arguments

object	an object returned by an IPDMR model function
...	additional arguments (currently ignored)

BetweenGroupClass	<i>Encapsulated OO class for between-group spread models</i>
-------------------	--

Description

This is a general-purpose between-group model spread class that takes a list of within-group spread models as input, and provides methods to update/run (and save/reset) these models while taking care of between-group (density-dependent) spread by calculating and inputting `trans_external` for each of the groups as appropriate based on the `beta_matrix`, which can be modified as needed (see examples).

The input models can either be created using the [make_group](#) function, or they can be your own R6 (or C++) class that implement the necessary methods and fields (i.e. `reset`, `save`, `update`, `I`, `state`, and `trans_external`). The models need not all be the same subtype, i.e. you can mix SIR/SEIR/SI models and even R6 and C++ implementations.

Note that the between-group model stores a reference to (i.e. shallow copy of) the input models, so you can see the state of the input groups being modified as the between-group model runs. You can also change the state of these groups at any time between updates/runs, either by modifying the groups used to create the between-group model class or by using the (read-only) active binding groups to retrieve one or more reference to a group. Of course you can also change the `beta_matrix` at any time between updates/runs.

Active bindings

<code>beta_matrix</code>	a matrix representing between-group (density-dependent) spread
<code>groups</code>	a list of the internally-stored within-group models (this can be used to directly check and/or modify their state and/or parameters)
<code>time</code>	the current time point
<code>transmission_between</code>	the between-group transmission type (frequency or density)
<code>state</code>	a data frame of the current state of each group

Methods

Public methods:

- `BetweenGroupClass$new()`
- `BetweenGroupClass$update()`
- `BetweenGroupClass$save()`
- `BetweenGroupClass$reset()`
- `BetweenGroupClass$run()`
- `BetweenGroupClass$run_until()`
- `BetweenGroupClass$print()`
- `BetweenGroupClass$clone()`

Method `new()`: Create a new between-group model

Usage:

```
BetweenGroupClass$new(groups)
```

Arguments:

`groups` a list of within-group models as either R6 or C++ classes (see e.g. [make_group](#))

Returns: A new between-group model object

Method `update()`: Update the state of each group for a single time point, including setting `trans_external` based on the value of the `beta_matrix` and vector of `I` obtained from the groups (density-dependent spread).

Usage:

```
BetweenGroupClass$update(d_time)
```

Arguments:

`d_time` the desired time step (delta time)

Returns: `self`, invisibly

Method `save()`: Instruct each group to save the current state and parameter values for later retrieval using `reset()`

Usage:

```
BetweenGroupClass$save()
```

Returns: `self`, invisibly

Method `reset()`: Instruct each group to reset the current state and parameter values to their last saved state

Usage:

```
BetweenGroupClass$reset()
```

Returns: `self`, invisibly

Method `run()`: Update the state of each group for a given number of time points

Usage:

```
BetweenGroupClass$run(add_time, d_time)
```

Arguments:

`add_time` the additional time to add to the current time of the model

`d_time` the desired time step (delta time)

Returns: a data frame of the model state at each (new) time point

Method `run_until()`: Update the state of each group for several time points, until a stopping criterion is met. NOTE: this method is a stub: it has not yet been implemented!

Usage:

```
BetweenGroupClass$run_until(criterion_fun, d_time)
```

Arguments:

`criterion_fun` a function taking an input data frame of states and returning a logical scalar indicating if the simulation should be stopped (TRUE) or continue to be updated (FALSE)

`d_time` the desired time step (delta time)

Returns: a data frame of the model state at each (new) time point

Method `print()`: Print method showing the number of groups and current time

Usage:

```
BetweenGroupClass$print()
```

Returns: self, invisibly

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Create a list of groups:
groups <- list(
  make_group("stochastic", group_name="Group1"),
  make_group("stochastic", group_name="Group2")
)

## Modify the starting conditions as needed:
groups[[1]]$S <- 100
groups[[1]]$I <- 0
groups[[2]]$S <- 99
groups[[2]]$I <- 1
## etc

## Create a between-group model:
model <- BetweenGroupClass$new(groups)

## Modify beta_matrix as needed:
model$beta_matrix <- matrix(c(0,0.1,0.01,0), nrow=2, ncol=2)
```

```
## Then run the model:
output <- model$run(21, d_time=0.1)
ggplot2::autoplot(output)

## Change the beta_matrix to e.g. remove transmission from Group1->Group2:
model$beta_matrix <- matrix(c(0,0,0.01,0), nrow=2, ncol=2)
## And run for more time points:
output2 <- model$run(21, d_time=0.1)
ggplot2::autoplot(output2)

## To combine time points:
all_output <- dplyr::bind_rows(output, output2)
summary(all_output)
```

group_models

Discrete-time deterministic and stochastic SIR and SEIR models

Description

These wrapper functions create a single-group model, run the model for the indicated time steps, and then return a data frame representing the model output. Note that starting values for S/E/I/R can be continuous or discrete for the deterministic models, but must be discrete numbers for the stochastic models.

Usage

```
sir_det(
  S = 99,
  I = 1,
  R = 0,
  beta = 0.25,
  gamma = 0.2,
  delta = 0.05,
  transmission_type = c("frequency", "density"),
  d_time = 1,
  max_time = 100
)
```

```
seir_det(
  S = 99,
  E = 0,
  I = 1,
  R = 0,
  numE = 3L,
  beta = 0.25,
  omega = 0.2,
  gamma = 0.2,
```

```
    delta = 0.05,  
    vacc = 0,  
    repl = 0,  
    cull = 0,  
    transmission_type = c("frequency", "density"),  
    d_time = 1,  
    max_time = 100  
  )
```

```
sir_stoc(  
  S = 99,  
  I = 1,  
  R = 0,  
  beta = 0.25,  
  gamma = 0.2,  
  delta = 0.05,  
  transmission_type = c("frequency", "density"),  
  d_time = 1,  
  max_time = 100,  
  iterations = 1L  
)
```

```
seir_stoc(  
  S = 99,  
  E = 0,  
  I = 1,  
  R = 0,  
  numE = 3L,  
  beta = 0.25,  
  omega = 0.2,  
  gamma = 0.2,  
  delta = 0.05,  
  vacc = 0,  
  repl = 0,  
  cull = 0,  
  transmission_type = c("frequency", "density"),  
  d_time = 1,  
  max_time = 100,  
  iterations = 1L  
)
```

Arguments

S	the starting number of susceptible animals
I	the starting number of infected animals
R	the starting number of recovered animals
beta	the transmission rate parameter per unit time (must be positive)
gamma	the recovery rate parameter per unit time (must be positive)

delta	the reversion rate parameter per unit time (must be positive)
transmission_type	either frequency or density
d_time	the desired time step (delta time)
max_time	the desired maximum time point (must be greater than the time step)
E	the total starting number of exposed/latent animals (this will be distributed among the sub-compartments of E when the model is initialised)
numE	the number of sub-compartments within the exposed/latent state
omega	the latent progression rate parameter per unit time (must be positive)
vacc	the vaccination rate parameter per unit time (must be positive)
repl	the replacement rate parameter per unit time (must be positive)
cull	the targeted culling rate parameter per unit time (must be positive)
iterations	the number of iterations to run (stochastic models only)

make_group	<i>Generate an object representing a within-group spread model</i>
------------	--

Description

This is a wrapper function around the new/initialise methods for R6 and C++ implementations of within-group spread models used on the IPDMR course. For more information on available methods see the documentation for [SEIRclass](#) (the method signatures and outputs of the R6 and C++ implementations should be identical).

Usage

```
make_group(
  update_type = c("deterministic", "stochastic"),
  numE = 3L,
  numI = 1L,
  numR = 1L,
  group_name = NA_character_,
  implementation = c("C++", "R6")
)
```

Arguments

update_type	either stochastic or deterministic
numE	the number of sub-compartments desired for the E state (0 or more)
numI	the number of sub-compartments desired for the I state (1 or more)
numR	the number of sub-compartments desired for the R state (0 or more)
group_name	an optional name for the group
implementation	either C++ or R6

Examples

```
set.seed(2024-11-05)
r6res <- make_group(update_type = "stochastic", implementation = "R6")$run(10, 0.1)

set.seed(2024-11-05)
cppres <- make_group(update_type = "stochastic", implementation = "C++")$run(10, 0.1)

## Should be identical:
stopifnot(unlist(r6res) - unlist(cppres) == 0L)
```

multi_models	<i>Discrete-time deterministic and stochastic SEIR models for multiple groups</i>
--------------	---

Description

These wrapper functions create a multi-group model, run the model for the indicated time steps, and then return a data frame representing the model output. Note that starting values for S/E/I/R can be continuous or discrete for the deterministic models, but must be discrete numbers for the stochastic models.

Usage

```
multi_seir_det(
  n_groups,
  beta_matrix,
  S = 99,
  E = 0,
  I = 1,
  R = 0,
  numE = 3L,
  beta = 0.25,
  omega = 0.2,
  gamma = 0.2,
  delta = 0.05,
  vacc = 0,
  repl = 0,
  cull = 0,
  transmission_within = c("frequency", "density"),
  transmission_between = c("density", "frequency"),
  d_time = 1,
  max_time = 100
)

multi_seir_stoc(
  n_groups,
```

```

    beta_matrix,
    S = 99,
    E = 0,
    I = 1,
    R = 0,
    numE = 3L,
    beta = 0.25,
    omega = 0.2,
    gamma = 0.2,
    delta = 0.05,
    vacc = 0,
    repl = 0,
    cull = 0,
    transmission_within = c("frequency", "density"),
    transmission_between = c("density", "frequency"),
    d_time = 1,
    max_time = 100,
    iterations = 1L
)

```

Arguments

n_groups	the number of groups to generate
beta_matrix	a matrix representing between-group spread (must have nrow and ncol equal to n_groups)
S	the starting number of susceptible animals
E	the total starting number of exposed/latent animals (this will be distributed among the sub-compartments of E when the model is initialised)
I	the starting number of infected animals
R	the starting number of recovered animals
numE	the number of sub-compartments within the exposed/latent state
beta	the transmission rate parameter per unit time (must be positive)
omega	the latent progression rate parameter per unit time (must be positive)
gamma	the recovery rate parameter per unit time (must be positive)
delta	the reversion rate parameter per unit time (must be positive)
vacc	the vaccination rate parameter per unit time (must be positive)
repl	the replacement rate parameter per unit time (must be positive)
cull	the targeted culling rate parameter per unit time (must be positive)
transmission_within	the within-group transmission type (frequency or density)
transmission_between	the between-group transmission type (frequency or density)
d_time	the desired time step (delta time)
max_time	the desired maximum time point (must be greater than the time step)
iterations	the number of iterations to run (stochastic models only)

Examples

```
output <- multi_seir_det(2, matrix(0,nrow=2,ncol=2))
ggplot2::autoplot(output)

output <- multi_seir_stoc(2, matrix(0,nrow=2,ncol=2), iterations=3L)
ggplot2::autoplot(output)
```

SEIRclass

Encapsulated OO classes for within-group SEIR models

Description

This is a within-group SEIR model class that can either be run on its own or embedded in a between-group model. Both R6 and C++ implementations are available - see [make_group](#) for a wrapper function that returns either implementation. However, the method signatures and outputs of the R6 and C++ implementations are (or at least should be!) identical, so this documentation applies to the C++ implementation as well as the R6 implementation.

Active bindings

S number of susceptible animals

E total number of exposed individuals - changing this value will cause the provided total to be distributed evenly (deterministic) or randomly (stochastic) between sub-compartments

I number of infected animals

R number of recovered animals

beta the transmission rate parameter per unit time (must be positive)

omega the latent progression rate parameter per unit time (must be positive)

gamma the recovery rate parameter per unit time (must be positive)

delta the reversion rate parameter per unit time (must be positive)

vacc the vaccination rate parameter per unit time (must be positive)

repl the replacement rate parameter per unit time (must be positive)

cull the targeted culling rate parameter per unit time (must be positive)

time the current time point of the model (read-only)

N the total number of animals in the group (read-only)

state a data frame representing the current state of the model (read-only)

trans_external the external transmission parameter

transmission_type either frequency or density

Methods

Public methods:

- `SEIRclass$new()`
- `SEIRclass$save()`
- `SEIRclass$reset()`
- `SEIRclass$update()`
- `SEIRclass$run()`
- `SEIRclass$print()`
- `SEIRclass$clone()`

Method `new()`: Create a new within-group model (this would normally be called from `make_group`)

Usage:

```
SEIRclass$new(
  update_type = c("deterministic", "stochastic"),
  numE = 3L,
  group_name = NA_character_
)
```

Arguments:

`update_type` one of deterministic or stochastic

`numE` the number of sub-compartments within the exposed/latent state

`group_name` an optional name for the group (will be included in the output state, if provided)

Returns: A new within-group model object

Method `save()`: Save the current state and parameter values for later retrieval using `reset()`

Usage:

```
SEIRclass$save()
```

Returns: self, invisibly

Method `reset()`: Reset the current state and parameter values to their last saved state

Usage:

```
SEIRclass$reset()
```

Returns: self, invisibly

Method `update()`: Update the state of the group for a single time point

Usage:

```
SEIRclass$update(d_time)
```

Arguments:

`d_time` the desired time step (delta time)

Returns: self, invisibly

Method `run()`: Update the state of the group for several time points

Usage:

```
SEIRclass$run(add_time, d_time)
```

Arguments:

add_time the additional time to add to the current time of the model

d_time the desired time step (delta time)

Returns: a data frame of the model state at each (new) time point

Method print(): Print method giving an overview of the current state and parameter values

Usage:

```
SEIRclass$print()
```

Returns: self, invisibly

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

 si_continuous

A simple continuous-time SI model

Description

A simple continuous-time SI model

Usage

```
si_continuous(
  S = 9,
  I = 1,
  beta = 0.05,
  transmission_type = c("frequency", "density"),
  time_points = seq(0, 21, by = 0.1)
)
```

Arguments

S	the starting number of susceptible animals (continuous number)
I	the starting number of infected animals (continuous number)
beta	the transmission rate per unit time (must be positive)
transmission_type	either frequency or density
time_points	a vector of time points to include in the data frame returned

si_discrete	<i>A simple discrete-time SI model</i>
-------------	--

Description

A simple discrete-time SI model

Usage

```
si_discrete(
  S = 9,
  I = 1,
  beta = 0.05,
  transmission_type = c("frequency", "density"),
  d_time = 1/24,
  max_time = 21
)
```

Arguments

S	the starting number of susceptible animals (continuous number)
I	the starting number of infected animals (continuous number)
beta	the transmission rate per unit time (must be positive)
transmission_type	either frequency or density
d_time	the desired time step (delta time)
max_time	the desired maximum time point (must be greater than the time step)

Examples

```
si_discrete(S=9, I=1, transmission_type="density") |> ggplot2::autoplot()
si_discrete(S=9, I=1, transmission_type="frequency") |> ggplot2::autoplot()
```

suggest_boxes	<i>Suggest a number of sub-compartments to obtain a suitable distribution of transition times</i>
---------------	---

Description

This is a utility function that takes a desired mean and standard deviation for the distribution of transit times through a disease compartment, and suggests a suitable rate and number of sub-compartments that matches the provided mean and variance as closely as possible.

Usage

```
suggest_boxes(mean, sd)
```

Arguments

mean	the desired mean waiting time
sd	the desired standard deviation in waiting time

Value

the function returns a ggplot object illustrating the suggestions, and prints the suggested number of sub-compartments and rates to screen

Examples

```
# An average of 5 days, with sd of 2.5 days:
suggest_boxes(mean=5, sd=3)

# The number of sub-compartments is invariant to transformation of the rate
# on different time scales, i.e. we get the same if specifying in hours:
suggest_boxes(mean=5*24, sd=3*24)
```

Index

`ab_continuous`, [2](#)
`ab_discrete`, [2](#)
`apply_rates`, [3](#)
`autoplot.ipdmr_ct`, [4](#)

`BetweenGroupClass`, [5](#)

`group_models`, [8](#)

`make_group`, [5](#), [6](#), [10](#), [13](#), [14](#)
`multi_models`, [11](#)
`multi_seir_det` (`multi_models`), [11](#)
`multi_seir_stoc` (`multi_models`), [11](#)

`seir_det` (`group_models`), [8](#)
`seir_stoc` (`group_models`), [8](#)
`SEIRclass`, [10](#), [13](#)
`si_continuous`, [15](#)
`si_discrete`, [16](#)
`sir_det` (`group_models`), [8](#)
`sir_stoc` (`group_models`), [8](#)
`suggest_boxes`, [16](#)