Package ‘SPARSEMODr’

December 17, 2020

Title SPAtial Resolution-SEnsitive Models of Outbreak Dynamics

Version 1.0

URL https://github.com/NAU-CCL/SPARSEMODr

BugReports https://github.com/NAU-CCL/SPARSEMODr/issues

Description Implementation of spatially-explicit, stochastic disease models with customizable time windows that describe how parameter values fluctuate during outbreaks (e.g., in response to public health or conservation interventions).

License GPL (>= 2)

Imports Rcpp (>= 1.0.4), future.apply, data.table, future, tidyverse, lubridate, viridis, geosphere

Suggests knitr, rmarkdown, testthat

VignetteBuilder knitr

LinkingTo Rcpp

Depends R (>= 3.5.0)

NeedsCompilation yes

Author Joseph Mihaljevic [aut, cre] (C code, package development),
Toby Hocking [ctb] (R package interface),
Seth Borkovec [ctb] (package development)

Maintainer Joseph Mihaljevic <Joseph.Mihaljevic@nau.edu>

Repository CRAN

Date/Publication 2020-12-17 09:40:06 UTC

R topics documented:

SPARSEMODr-package *SPAtial Resolution-SEnsitive Models of Outbreak Dynamics*

Description

Implementation of spatially-explicit, stochastic disease models with customizable time windows that describe how parameter values fluctuate during outbreaks (e.g., in response to public health or conservation interventions).

Details

Stochastic realizations of the models can be performed in parallel using the `model_parallel` functions, and time-varying parameters can be supplied using `time_windows` objects.

Author(s)

Maintainer: Joseph Mihaljevic <joseph.mihaljevic@nau.edu>

---

covid19_control *COVID19 Parameters Control Data Structure*

Description

The COVID19 model uses parameters which are specific to the COVID19 model. This data structure affirms that required parameters are supplied, provides default values for optional parameters, and validates the data of each parameter.

Usage

```
covid19_control(
    input_N_pops=NULL,
    input_S_pops=NULL,
    input_E_pops=NULL,
    input_I_asym_pops=NULL,
    input_I_presym_pops=NULL,
    input_I_sym_pops=NULL,
    input_I_home_pops=NULL,
    input_I_hosp_pops=NULL,
    input_I_icu1_pops=NULL,
    input_I_icu2_pops=NULL,
    input_R_pops=NULL,
    input_D_pops=NULL,
    frac_beta_asym=0.55,
    frac_beta_hosp=0.05,
```

```
        delta=1/3.0,
        recov_a=1/6.0,
        recov_p=1/2.0,
        recov_s=1/6.0,
        recov_home=1/3.0,
        recov_icu1=1/8.0,
        recov_icu2=1/4.0,
        asym_rate=0.40,
        sym_to_icu_rate=0.015
    )
```

Arguments

input_N_pops
: Integer Vector representing the total population for each county.

input_S_pops
: Integer Vector representing the susceptible population for each county.

input_E_pops
: Integer Vector representing the exposed population for each county.

input_I_asym_pops
: Integer Vector representing the asymptomatic infected population for each county.

input_I_presym_pops
: Integer Vector representing the presymptomatic infected population for each county.

input_I_sym_pops
: Integer Vector representing the symptomatic infected population for each county.

input_I_home_pops
: Integer Vector representing the infected and isolated at home population for each county.

input_I_hosp_pops
: Integer Vector representing the hospitalized population for each county.

input_I_icu1_pops
: Integer Vector representing the population for each county in ICU.

input_I_icu2_pops
: Integer Vector representing the population for each county in ICU recovery.

input_R_pops
: Integer Vector representing the recovered population for each county.

input_D_pops
: Integer Vector representing the deaths for each county.

frac_beta_asym
: An adjustment to beta accounting for asymptomatic individuals being less likely to transmit than symptomatic individuals. Default value is 0.55. Must be greater than zero and less than or equal to one.

frac_beta_hosp
: An adjustment to beta accounting for hospitalized individuals being less likely to transmit than non-hospitalized individuals. Default value is 0.05. Must be greater than zero and less than or equal to one.

delta
: Incubation period. Default value is 1/3.0. Must be greater than or equal to zero. Values above one are unusual.

recov_a
: Recovery rate of asymptomatic individuals. Default value is 1/6.0. Must be greater than or equal to zero. Values above one are unusual.

recov_p
Recovery rate of presymptomatic individuals. Default value is 1/2.0. Must be greater than or equal to zero. Values above one are unusual.

recov_s
Recovery rate of symptomatic individuals. Default value is 1/6.0. Must be greater than or equal to zero. Values above one are unusual.

recov_home
Recovery rate of infected individuals in home isolation. Default value is 1/3.0. Must be greater than or equal to zero. Values above one are unusual.

recov_icu1
Recovery rate of individuals in ICU. Default value is 1/8.0. Must be greater than or equal to zero. Values above one are unusual.

recov_icu2
Recovery rate of individuals in ICU recovery. Default value is 1/4.0. Must be greater than or equal to zero. Values above one are unusual.

asym_rate
Proportion of presymtomatic individuals entering the asymptomatic stage. Default value is 0.40. Must be greater than zero and less than or equal to one.

sym_to_icu_rate
Proportion of symptomatic individuals entering ICU. Default value is 0.015. Must be greater than zero and less than or equal to one.

Details

Defines a set of parameters specific to the COVID19 model. Adjustments to the model calculations can be made by specifying any or all of the optional parameters. If an optional parameter is not set, it will use the default value as specified above. If a parameter is outside the specified limits, execution will stop and an error message will be displayed. Some parameters may have values greater than one. While these situations may be unusual, execution will not stop but a warning message will be displayed.

Note: At least one of input_N_pops or input_S_pops must be supplied. If one is not supplied, it will be calculated within the class.

Note: At least one of input_E_pops, input_I_asym_pops, input_I_presym_pops, input_I_sym_pops, input_I_home_pops, input_I_hosp_pops, input_I_icu1_pops, and input_I_icu2_pops must be supplied with a nonzero population. Any of these population parameters not supplied will be assumed to be a vector of zeroes.

Value

Returns a named list of vectors that must be supplied to model_interface.

Author(s)

Seth Borkovec, <stb224@nau.edu>; Joseph Mihaljevic, <joseph.mihaljevic@nau.edu>

See Also

seir_control

Examples

```
## Data set for the examples:
N_pops <- rep(100000, 10)
E_pops <- c(0, 1, 0, 3, 2, 0, 14, 3, 0, 0)
S_pops <- N_pops - E_pops
I_asym_pops <- rep(0, 10)
I_presym_pops <- rep(0, 10)
I_sym_pops <- c(1, 0, 11, 0, 0, 5, 0, 0, 9, 0)
I_home_pops <- rep(0, 10)
I_hosp_pops <- rep(0, 10)
I_icu1_pops <- rep(0, 10)
I_icu2_pops <- rep(0, 10)
R_pops <- rep(0, 10)
D_pops <- rep(0, 10)

## Example using the default parameters:
covid19_control <- covid19_control(input_S_pops         = S_pops,
                                   input_E_pops         = E_pops)

## Example specifying some parameters:
covid19_control <- covid19_control(input_N_pops         = N_pops,
                                   input_I_sym_pops     = I_sym_pops,
                                   input_I_home_pops    = I_home_pops,
                                   input_I_hosp_pops    = I_hosp_pops,
                                   input_I_icu2_pops    = I_icu2_pops,
                                   input_D_pops         = D_pops,
                                   frac_beta_hosp       = 0.03)

## Example specifying all possible parameters:
covid19_control <- covid19_control(input_S_pops         = S_pops,
                                   input_E_pops         = E_pops,
                                   input_I_asym_pops    = I_asym_pops,
                                   input_I_presym_pops  = I_presym_pops,
                                   input_I_sym_pops     = I_sym_pops,
                                   input_I_home_pops    = I_home_pops,
                                   input_I_hosp_pops    = I_hosp_pops,
                                   input_I_icu1_pops    = I_icu1_pops,
                                   input_I_icu2_pops    = I_icu2_pops,
                                   input_R_pops         = R_pops,
                                   input_D_pops         = D_pops,
                                   frac_beta_asym       = 0.50,
                                   frac_beta_hosp       = 0.06,
                                   delta                = 0.25,
                                   recov_a              = 0.57,
                                   recov_p              = 0.62,
                                   recov_s              = 0.11,
                                   recov_home           = 0.28,
                                   recov_icu1           = 0.12,
                                   recov_icu2           = 0.29,
                                   asym_rate            = 0.65,
                                   sym_to_icu_rate      = 0.122)
```

model_interface *Universal Model Interface to the SPARSE-MOD Models*

Description

`model_interface` determines which SPARSE-MOD model to run based on the arguments and runs the specified model.

Usage

```
model_interface(
    control,
    arg.list
)
```

Arguments

`control`	Either a `covid19_control` or `seir_control` named list. The control used will determine which model to run.
`arg.list`	A named list of arguments used in all models including: • `input_dist_mat` • `input_census_area` • `input_realz_seeds` • `input_tw`: A time window object (see `time_windows`) • `trans_type`: Transmission type (see details below) • `dd_trans_monod_k`: The Monod equation parameter 'k', for when transmission is density-dependent (see details below) • `stoch_sd`: The standard deviation of the stochastic transmission rate (see details below)

Details

This is the universal interface to all of the SPARSE-MOD models. Currently the models available are the COVID-19 Model, and the SEIR Model.

The SPARSE-MOD COVID-19 Model describes transmission using 11 classes of individuals. Please see the vignettes for a more detailed explanation of the model structure.

The SPARSE-MOD SEIR Model describes transmission using 4 classes of individuals. Please see the vignettes for a more detailed explanation of the model structure.

Transmisison types: The day-specific transmission rate (beta) is back-calculated from the user-provided value(s) of R0 (pronounced R-naught). See `Time-varying R0` for details. We allow for two transmission types:

1. Frequency-dependent transmission: In this case the transmission function is:

$$beta_scaled = beta/pop_N.$$

2. Density-dependent transmission: In this case, we allow a user-defined Monod equation to scale the beta term by population density, where $pop_density = pop_N/census_area$:

$$beta_scaled = (beta * pop_dens/(dd_trans_monod_k + pop_dens))/pop_N$$

Stochastic transmission: We implement daily stochastic variation in the transmission rate that scales with the number of infectious individuals in the focal population. In other words, as the number of infectious individuals increases, the variation in transmission rate reduces, emphasizing that stochasticity has larger effects in smaller populations (i.e., larger effects when there are few infectious individuals). To implement this stochasticity, we draw a random variate from a normal distribution with a mean of zero and a standard deviation of `stoch_sd`, and this random variate is termed `noise`. We calculate the total number of infectious individuals across infectious sub-classes (e.g., Pre-symptomatic, Hospital, etc.), and this variable is termed `infect_sum`. The functional form of stochasticity is then:

$$beta_realized = |beta_scaled * (1 + (noise/sqrt(infect_sum)))|$$

See `Movement` for details of how movement dynamics are implemented and controlled in the model.

Value

Two named lists:

1. `pops`: Integer vectors that provide the number of individuals in each model class at each time step. Different realizations of the model are distinguised by the user-provided values for the random seeds.
2. `events`: Integer vectors that provide the number of individuals that newly transitioned to specific, key model classes at each time step. Different realizations of the model are distinguised by the user-provided values for the random seeds.

   For the COVID-19 model, these event vectors are defined as:

   - `pos`: Number of newly positive individuals. Sum of new asymptomatic and pre-symptomatic individuals.
   - `sym`: Number of newly symptomatic individuals.
   - `total_hosp`: Number of newly hospitalized individuals. Sum of new Hospital admits and new Symptomatic-to-ICU1 admits.
   - `total_icu`: Number of new ICU admits. Sum of new Symptomatic-to-ICU1 admits and new Hospital-to-ICU1 admits.
   - `n_death`: Number of newly deceased individuals.

   For the SEIR model, these event vectors are defined as:

   - `birth`: Number of newly susceptible hosts through the process of reproduction.
   - `exposed`: Number of newly exposed hosts.
   - `infectious`: Number of newly infectious hosts.
   - `recov`: Number of newly recovered hosts.
   - `death`: Number of newly deceased hosts.

Author(s)

Joseph Mihaljevic, <joseph.mihaljevic@nau.edu>
Seth Borkovec, <stb224@nau.edu>

See Also

model_parallel, time_windows, covid19_control, seir_control

Examples

```
## See vignettes for more detailed work-ups.

##########################################
## See model_parallel()
## for an example to run realizations in parallel

##########################################
## Using supplied example data:

# Read in the example data:
ex_dir <- system.file(
  "extdata", "sparsemodr_example.Rdata", package="SPARSEMODr", mustWork=TRUE)
load(ex_dir)
n_pop <- length(dat_list[["pop_N"]])

# Set up realizations:
realz_seeds <- 1:2
n_realz <- length(realz_seeds)

# Set up time windows (see time_windows for other ways to do this)
input_R0 <-             c(   2.0,   2.0,   0.8,   0.8,   1.5)
input_dist_param <-     c(   200,   200,    20,   150,   150)
input_m <-              c( 0.002, 0.002, 0.002,  0.02,  0.02)
input_imm_frac <-       c(   0.0,   0.0,   0.0,  0.02,  0.02)
input_window_length <-  c(     0,    36,    10,    35,   169)

# User creates the time_windows object here
tw <- time_windows(r0 = input_R0,
                   dist_param = input_dist_param,
                   m = input_m,
                   imm_frac = input_imm_frac,
                   window_length = input_window_length)

# Randomly generate initial conditions for
# EXPOSED class:
E_pops <- vector("numeric", length = n_pop)
n_initial_E <- 40
# (more exposed in larger populations)
these_E <- sample.int(n_pop,
                      size = n_initial_E,
                      replace = TRUE,
                      prob = dat_list$pop_N)
for(i in 1:n_initial_E){
  E_pops[these_E[i]] <- E_pops[these_E[i]] + 1
}

# Inputs for the models
```

```
N_pops <- as.integer(dat_list[["pop_N"]])
S_pops <- N_pops - E_pops
I_pops <- vector("integer", length = n_pop)
R_pops <- vector("integer", length = n_pop)
I_asym_pops <- vector("integer", length = n_pop)
I_presym_pops <- vector("integer", length = n_pop)
I_sym_pops <- vector("integer", length = n_pop)
I_home_pops <- vector("integer", length = n_pop)
I_hosp_pops <- vector("integer", length = n_pop)
I_icu1_pops <- vector("integer", length = n_pop)
I_icu2_pops <- vector("integer", length = n_pop)
D_pops <- vector("integer", length = n_pop)

# User created control list of parameters
covid19_control <- covid19_control(input_N_pops = N_pops,
                                   input_E_pops = E_pops,
                                   input_I_asym_pops=I_asym_pops,
                                   input_I_presym_pops=I_presym_pops,
                                   input_I_sym_pops=I_sym_pops,
                                   input_I_home_pops=I_home_pops,
                                   input_I_hosp_pops=I_hosp_pops,
                                   input_I_icu1_pops=I_icu1_pops,
                                   input_I_icu2_pops=I_icu2_pops,
                                   input_R_pops=R_pops,
                                   input_D_pops=D_pops)

arg.list <- list(
    input_dist_mat =    dat_list$dist_vec,
    input_census_area = dat_list$census_area,
    input_tw =          tw,
    input_realz_seeds = realz_seeds
)

# Using all default parameter values,
# these are the minimum inputs
covid_model_output <-
  model_interface(
    control = covid19_control,
    arg.list
  )
```

`model_parallel` *Parallelized implementation of the SPARSE-MOD COVID-19 Model*

Description

The function uses R-level parallelization to speed up the generation of stochastic realizations of the SPARSEMODr COVID-19 model and to combine output data into a read-to-use data frame. This is the preferred method to run `model_interface`.

Usage

```
model_parallel(..., input_realz_seeds = 1:2, control)
```

Arguments

`...`	Universal model arguments passed to `model_interface`.
`input_realz_seeds`	An integer vector of user-specified random seeds to generate the stochastic realizations of the model. The number of realizations will be equal to the length of this vector.
`control`	Either a `covid19_control` or a `seir_control` named list data object.

Details

Relies on `future_lapply` to run stochastic realizations of the SPARSEMODr model in parallel.

Value

A data frame that combines the two named lists of `model_interface`.

Author(s)

Joseph Mihaljevic, <joseph.mihaljevic@nau.edu>; Toby Hocking, <toby.hocking@r-project.org>

See Also

`future_lapply`, `model_interface`, `time_windows`, `covid19_control`, `seir_control`

Examples

```
## See vignettes for more detailed work-ups.


###########################################
## Using supplied example data:

# Read in the example data:
ex_dir <- system.file(
  "extdata", "sparsemodr_example.Rdata", package="SPARSEMODr", mustWork=TRUE)
load(ex_dir)
n_pop <- length(dat_list[["pop_N"]])

# Set up realizations:
realz_seeds <- 1:2
n_realz <- length(realz_seeds)

# START FUTURE PLAN FOR PARALLELIZATION
future::plan("multisession")

# Set up time windows (see time_windows for other ways to do this)
```

```
input_R0 <-             c(   2.0,   2.0,   0.8,   0.8,   1.5)
input_dist_param <-     c(   200,   200,    20,   150,   150)
input_m <-              c( 0.002, 0.002, 0.002,  0.02,  0.02)
input_imm_frac <-       c(   0.0,   0.0,   0.0,  0.02,  0.02)
input_window_length <-  c(     0,    36,    10,    35,   169)

# User creates the time_windows object here
tw <- time_windows(r0 = input_R0,
                   dist_param = input_dist_param,
                   m = input_m,
                   imm_frac = input_imm_frac,
                   window_length = input_window_length)

# Randomly generate initial conditions for
# EXPOSED class:
E_pops <- vector("numeric", length = n_pop)
n_initial_E <- 40
# (more exposed in larger populations)
these_E <- sample.int(n_pop,
                     size = n_initial_E,
                     replace = TRUE,
                     prob = dat_list$pop_N)
for(i in 1:n_initial_E){
  E_pops[these_E[i]] <- E_pops[these_E[i]] + 1
}

# Inputs for the models
N_pops <- as.integer(dat_list[["pop_N"]])
S_pops <- N_pops - E_pops
I_pops <- vector("integer", length = n_pop)
R_pops <- vector("integer", length = n_pop)
I_asym_pops <- vector("integer", length = n_pop)
I_presym_pops <- vector("integer", length = n_pop)
I_sym_pops <- vector("integer", length = n_pop)
I_home_pops <- vector("integer", length = n_pop)
I_hosp_pops <- vector("integer", length = n_pop)
I_icu1_pops <- vector("integer", length = n_pop)
I_icu2_pops <- vector("integer", length = n_pop)
D_pops <- vector("integer", length = n_pop)

# Set up a function to use the dat_list

get_result <- function(input_realz_seeds, control = NULL){

    with(dat_list, SPARSEMODr::model_parallel(
        input_census_area = census_area,
        input_dist_mat = dist_vec,
        input_realz_seeds = input_realz_seeds,
        input_tw = tw,
        control = control)
    )
}
```

```
# User creates control list of parameters
covid19_control <- covid19_control(input_N_pops = N_pops,
                                   input_E_pops = E_pops,
                                   input_I_asym_pops=I_asym_pops,
                                   input_I_presym_pops=I_presym_pops,
                                   input_I_sym_pops=I_sym_pops,
                                   input_I_home_pops=I_home_pops,
                                   input_I_hosp_pops=I_hosp_pops,
                                   input_I_icu1_pops=I_icu1_pops,
                                   input_I_icu2_pops=I_icu2_pops,
                                   input_R_pops=R_pops,
                                   input_D_pops=D_pops)

covid_model_output <-
    get_result(
        input_realz_seeds = realz_seeds,
        control = covid19_control
    )

# Shut down parallel workers
future::plan("sequential")
```

Movement *Movement dynamics in SPARSEMODr models.*

Description

The SPARSEMODr models allow for spatially explicit movement dynamics between focal populations, and for 'visitation' from outside of the focal populations of interest.

Details

The meta-population of interest is defined by the focal populations supplied by the user in `model_interface`. Movement between focal populations within the meta-population is implemented as daily visitation (e.g., commuting). Specifically, individuals can move to a new focal population and can influence the local transmission dynamics for that day, but then individuals return to their focal population before the model simulates the next day's events. Every day, immigrants from outside of the meta-population can also visit the focal populations and influence transmission.

Movement within the meta-population

We assume that susceptible and infectious individuals can move between focal populations. In the COVID-19 model, we further assume that only individuals in the Susceptible, Asymptomatic, and Pre-symptomatic classes are moving. This is because we assume individuals that are Symptomatic, Home (isolating) or in the hospital (Hospital, ICU1, ICU2) will not be moving outside of their focal population.

In general, susceptible individuals in a focal population can become exposed to the pathogen by infectious visitors from the meta-population or by infectious visitors from outside of the meta-population (below). Similarly, susceptible individuals can visit a population within the meta-community but outside of their focal population, at which point these susceptible individuals may become exposed by resident infectious individuals.

Movement frequency is controlled by parameter m in the model, and this rate can be updated daily to simulate changes in movement patterns over time (see time_windows). In the model differential equations, m is the per-capita rate of movement outside of the focal population. The inverse of m therefore corresponds to the average number of days between an individual's movement events.

When an individual moves outside of their focal population, the model assigns this individual to a new focal population using a dispersal kernel. For now, we implement a simple distance-based dispersal kernal in the form: `prob_move[i][j] = 1 / exp(dist_mat[i][j] / dist_param)`. Here, as is convention, `prob_move[i][j]` corresponds to the probability of individuals in population `j` moving to population `i`. The `dist_param` is user-defined and can be updated daily in the simulaiton (see time_windows).

On each day in the model simulation, the tau-leaping algorithm calculates the number of individuals in each class that will move outside of their focal population. We determine which individuals will move to which outside population using a random draw from a multinomial probability distribution, using the `prob_move[i][j]` that are calculated as above. Once individuals are assigned to their new, temporary populations, then transmission can occur dependent upon the local composition of infectious individuals.

Immigration from outside of the meta-population

The model allows for outside visitors to enter the system temporarily, with visitors updated daily. In this case, the user can define parameter `imm_frac`, the value of which can be updated daily (see time_windows). The `imm_frac` is the proportion of the focal population that may constitute visitors on any given day. For example if for a given focal population, `pop_N = 1000` and `imm_frac = 0.05`, an average of 50 visitors may arrive on a given day. The exact number of visitors on a given day is determined by drawing from a Poisson distribution. Then, the number of *infectious* visitors from that group is assumed to be proportional to the number of infectious *residents* in the focal population. In other words, we assume that the pathogen is present in 'outsider' populations at similar prevalence as the focal population. The exact number of infectious visitors is then determined again by a Poisson draw. After visitors arrive at the focal population, transmission between susceptible residents and infectious visitors is determined.

See Also

model_interface, model_parallel, time_windows

`seir_control` *SEIR Parameters Control Data Structure*

Description

The SEIR model uses parameters which are specific to the SEIR model. This data structure affirms that required parameters are supplied, provides default values for optional parameters, and validates the data of each parameter.

Usage

```
seir_control(
    input_N_pops=NULL,
    input_S_pops=NULL,
    input_E_pops=NULL,
    input_I_pops=NULL,
    input_R_pops=NULL,
    birth=1/(75*365),
    incubate=1/8.0,
    recov=1/3.0
)
```

Arguments

`input_N_pops`	Integer Vector representing the total population for each county.
`input_S_pops`	Integer Vector representing the susceptible population for each county.
`input_E_pops`	Integer Vector representing the exposed population for each county.
`input_I_pops`	Integer Vector representing the infected population for each county.
`input_R_pops`	Integer Vector representing the recovered population for each county.
`birth`	Default value is 1/(75*365). Must be greater than or equal to zero. Values above one are unusual.
`incubate`	Default value is 1/8.0. Must be greater than or equal to zero. Values above one are unusual.
`recov`	Default value is 1/3.0. Must be greater than or equal to zero. Values above one are unusual.

Details

Defines a set of parameters specific to the SEIR model. If an optional parameter is not set, it will use the default value as specified above. If a parameter is outside the specified limits, execution will stop and an error message will be displayed. Some parameters may have values greater than one. While these situations may be unusual, execution will not stop but a warning message will be displayed.

Note: At least one of `input_N_pops` or `input_S_pops` must be supplied. If one is not supplied, it will be calculated within the class.

Note: At least one of `input_E_pops` and `input_I_pops` must be supplied with a nonzero population. If either of these population parameters or `input_R_pops` is not supplied, it will be assumed to be a vector of zeroes.

Value

Returns a named list of vectors that must be supplied to `model_interface`.

Author(s)

Seth Borkovec, <stb224@nau.edu>; Joseph Mihaljevic, <joseph.mihaljevic@nau.edu>

See Also

covid19_control

Examples

```
## Data set for the examples:
S_pops <- rep(100000, 10)
E_pops <- c(0, 1, 0, 3, 2, 0, 13, 3, 0, 0)
I_pops <- c(0, 0, 0, 0, 0, 0, 1, 0, 0, 0)
R_pops <- rep(0, 10)
N_pops <- S_pops + E_pops + I_pops + R_pops

## Example using the default parameters:
seir_control <- seir_control(input_S_pops = S_pops,
                             input_E_pops = E_pops)

## Example specifying one optional parameter:
seir_control <- seir_control(input_N_pops = N_pops,
                             input_I_pops = I_pops,
                             input_R_pops = R_pops,
                             recov        = 1/4.0)

## Example specifying all possible parameters:
seir_control <- seir_control(input_N_pops = N_pops,
                             input_S_pops = S_pops,
                             input_E_pops = E_pops,
                             input_I_pops = I_pops,
                             input_R_pops = R_pops,
                             birth        = 1/(65*365),
                             incubate     = 0.12,
                             recov        = 0.25)
```

`Time-varying R0` *Time-varying R0 in SPARSEMODr models.*

Description

The SPARSEMODr models allow for transmission dynamics to change over time by allowing users to specify the time-varying R0, also known as the effective reproduction number (R-eff) or the instantaneous Rt.

Details

In SPARSEMODr models, we allow the user to specify the time-varying R0 as an input (see time_windows for details). We then use these values to back-calculate the transmission rate (*beta*) for the model on any given day of the simulation. In this way, we are assuming that R0 is effectively changing only due to changes in the *beta* term, which encapsulates the effective contact rate among individuals and the probability of transmission given contact between a susceptible and infectious individual. We say 'effective' contact to define contacts that can actually lead to transmission. For instance, the wearing of masks or other face coverings may reduce the effective contact rate relevant to respiratory pathogens. As of now, in our models, we do not allow other parameters that define a pathogen's R0 to vary over time (e.g., recovery rates).

For simple models, the calcuation of *beta* from a time-varying R0 value is straightforward algebra. For more complex models with many state variables, this becomes more complicated. Therefore, to back-calculate the *beta* parameter from the time-varying R0 value, we derive the equation for time-varying R0 for each SPARSEMODr model. Then, to calculate *beta* in our C++ code, we implement a root-finding algorithm with the Brent-Dekker method, using the Gnu Scientific Library (`gsl_root_fsolver`). Thus, the user inputs the time-varying R0 value, and we calculate the value of *beta* on the back-end.

See Also

`model_interface`, `model_parallel`, `time_windows`

`time_windows` *Time Windows Data Structure*

Description

The SPARSEMODr models allow for users to dynamically update transmission rates and movement dynamics across the course of the outbreak. These time-varying parameter values must then be compiled into a `time_windows` object.

A `time_windows` object is a set of data across multiple vectors including time-varying R0, also known as the effective reproduction number (R-eff); a parameter that helps define the range of movement; a parameter that defines the frequency of movement between focal populations; a parameter that constrains the impact of hosts that immigrate from outside of the focal populations; and a method to define the dates over which the parameters fluctuate. Each vector must have the same number of elements, which corresponds to the total number of days in the model simulation.

When specifying dates for each entry, there are three options, but only one of which may be used. See details below.

1. Providing a vector for `window_length`,
2. Providing a vector each for `start_dates` and `end_dates`,
3. Providing a vector for `daily`.

Usage

```
time_windows(
    r0=NULL,
    dist_param=NULL,
    m=NULL,
    imm_frac=NULL,
    hosp_rate=NULL,
    recov_hosp=NULL,
```

```
        icu_rate=NULL,
        death_rate=NULL,
        window_length=NULL,
        start_dates=NULL,
        end_dates=NULL,
        daily=NULL
    )
```

Arguments

`r0`	*required* - A numeric vector of the time-varying R-naught.
`dist_param`	*required* - A numeric vector of the `dist_param` that is used to calculate the dispersal kernal (see details below).
`m`	*required* - A numeric vector of the `m` parameter. The inverse of `m` is the average time between individuals moving away from their focal population (see details below).
`imm_frac`	*required* - A numeric vector. This parameter corresponds to the fraction of the focal population (between 0 and 1) that may be comprised of immigrants from outside of the system (i.e., immigrants that are not from any of the supplied populations in `input_pop_N` from `model_interface`); see details below.
`hosp_rate`	A numeric vector. Proportion of symptomatic individuals entering hospitalization. Default value is 0.175. Must be greater than zero and less than or equal to one.
`recov_hosp`	A numeric vector. Recovery rate of hospitalized individuals. Default value is 1/7.0. Must be greater than or equal to zero. Values above one are unusual.
`icu_rate`	A numeric vector. Proportion of hospitalized individuals entering ICU. Default value is 0.20. Must be greater than zero and less than or equal to one.
`death_rate`	A numeric vector. Proportion of individuals who do not recover in ICU. Default value is 0.60. Must be greater than zero and less than or equal to one.
`window_length`	An integer vector supplying the number of days in each time window (see details below).
`start_dates`	A vector of Date objects that corresponds to the starting date of each time window. If supplied, `end_dates` must also be supplied (see details below).
`end_dates`	A vector of Date objects that corresponds to the ending dates of each time window. If supplied `start_dates` must also be supplied (see details below).
`daily`	A vector of Date objects that is sequential and complete, encompassing all dates from the start of the outbreak to the end of the outbreak (see details below).

Details

See `Time-varying R0` for descriptions of parameter `r0`, and see `Movement` for descriptions of `m` and `dist_param`.

Defining time window durations. One of the following options is required to define the duration of each time window: `window_length`, or `start_dates` AND `end_dates`, or `daily`.

Use window_length when you want to specify the length of each time window by the number of days.

Use start_dates AND end_dates when you want to define a time window by its starting and ending dates. A start date may not overlap with an end date, and there can be no gaps between the end date and the subsequent start date.

Use daily when you want to update parameters every day of the simulation. In this mode, each time window has a length of one day.

Value

Returns a named list of vectors that must be supplied to model_interface.

Author(s)

Seth Borkovec, <stb224@nau.edu>; Joseph Mihaljevic, <joseph.mihaljevic@nau.edu>

Examples

```
## Data set for the examples: (All examples include 5 time windows)
input_r0 <-            c(   2.5,   2.0,   0.8,   0.8,   1.5)
input_dist_param <-    c(   200,   200,    20,   150,   150)
input_m <-             c( 0.002, 0.002, 0.002,  0.02,  0.02)
input_imm_frac <-      c(   0.0,   0.0,   0.0,  0.02,  0.02)
input_window_length <- c(    10,    35,    46,    81,    40)
input_start_dates <-   c(seq(as.Date("2020-07-09"), by=10, len=5))
input_end_dates <-     c(seq(as.Date("2020-07-18"), by=10, len=5))
input_daily <-         c(seq(as.Date("2020-07-09"), by=1, len=5))

## Example using window_length:
### input_window_length defines the number of days
### that each value of the other parameters is repeated.
tw <- time_windows(r0            = input_r0,
                   dist_param    = input_dist_param,
                   m             = input_m,
                   imm_frac      = input_imm_frac,
                   window_length = input_window_length)

## Example using start_dates with end_dates:
### Five time windows, each with 10 days
tw <- time_windows(r0           = input_r0,
                   dist_param   = input_dist_param,
                   m            = input_m,
                   imm_frac     = input_imm_frac,
                   start_dates  = input_start_dates,
                   end_dates    = input_end_dates)

## Example using daily:
### Parameters are updated daily over 5 days
tw <- time_windows(r0         = input_r0,
                   dist_param = input_dist_param,
```

```
           m          = input_m,
           imm_frac   = input_imm_frac,
           daily      = input_daily)
```

Index