Package `individual`

October 25, 2021

**Title**  Framework for Specifying and Simulating Individual Based Models

**Version**  0.1.7

**Description**  A framework which provides users a set of useful primitive elements for specifying individual based simulation models, with special attention models for infectious disease epidemiology. Users build models by specifying variables for each characteristic of individuals in the simulated population by using data structures exposed by the package. The package provides efficient methods for finding subsets of individuals based on these variables, or cohorts. Cohorts can then be targeted for variable updates or scheduled for events. Variable updates queued during a time step are executed at the end of a discrete time step, and the code places no restrictions on how individuals are allowed to interact. These data structures are designed to provide an intuitive way for users to turn their conceptual model of a system into executable code, which is fast and memory efficient.

**License**  MIT + file LICENSE

**Encoding**  UTF-8

**URL**  https://github.com/mrc-ide/individual,
         https://mrc-ide.github.io/individual/

**BugReports**  https://github.com/mrc-ide/individual/issues

**Imports**  R6, Rcpp

**Suggests**  ggplot2, knitr, mockery, rmarkdown, pkgdown, testthat (>= 2.1.0), xml2, bench

**RoxygenNote**  7.1.1

**VignetteBuilder**  knitr

**LinkingTo**  Rcpp, testthat

**RcppModules**  individual_cpp

**NeedsCompilation**  yes

**Author**  Giovanni Charles [aut, cre] (<https://orcid.org/0000-0002-7024-1200>),
            Sean L. Wu [aut] (<https://orcid.org/0000-0002-5781-9493>),
            Peter Winskill [aut] (<https://orcid.org/0000-0003-3001-4959>),
            Imperial College of Science, Technology and Medicine [cph]
Bernoulli process

Description

Simulate a process where individuals in a given from state advance to the to state each time step with probability rate.

Usage

bernoulli_process(variable, from, to, rate)

Arguments

variable  a categorical variable.
from  a string representing the source category.
to  a string representing the destination category.
rate  the probability to move individuals between categories.
**Descripción**

Esta es una estructura de datos que compactamente almacena la presencia de enteros en algún conjunto finito (`max_size`), y puede eficientemente realizar operaciones de conjunto (unión, intersección, complemento, diferencia simétrica, diferencia de conjuntos). **WARNING:** Todas las operaciones son in-place de modo que por favor use `$copy` si desea realizar una operación sin destruir su conjunto actual de bits.

**Campos Públicos**

- `.bitset` un puntero al objeto IterableBitset.
- `max_size` el tamaño máximo del conjunto.

**Métodos**

### Métodos Públicos:

- `Bitset$new()`
- `Bitset$insert()`
- `Bitset$remove()`
- `Bitset$size()`
- `Bitset$or()`
- `Bitset$and()`
- `Bitset$not()`
- `Bitset$xor()`
- `Bitset$set_difference()`
- `Bitset$sample()`
- `Bitset$choose()`
- `Bitset$copy()`
- `Bitset$to_vector()`
- `Bitset$clone()`

**Método new()**: crear un conjunto.

**Usos**

`Bitset$new(size, from = NULL)`

**Argumentos**

- `size` el tamaño del conjunto.
- `from` puntero a un objeto IterableBitset existente para copiar; si es NULL se crea un conjunto vacío, de lo contrario se copia un conjunto existente.
**Method** `insert()`: insert into the bitset.

*Usage:*

```perl
Bitset$insert(v)
```

*Arguments:*

- `v` an integer vector of elements to insert.

**Method** `remove()`: remove from the bitset.

*Usage:*

```perl
Bitset$remove(v)
```

*Arguments:*

- `v` an integer vector of elements (not indices) to remove.

**Method** `size()`: get the number of elements in the set.

*Usage:*

```perl
Bitset$size()
```

**Method** `or()`: to "bitwise or" or union two bitsets.

*Usage:*

```perl
Bitset$or(other)
```

*Arguments:*

- `other` the other bitset.

**Method** `and()`: to "bitwise and" or intersect two bitsets.

*Usage:*

```perl
Bitset$and(other)
```

*Arguments:*

- `other` the other bitset.

**Method** `not()`: to "bitwise not" or complement a bitset.

*Usage:*

```perl
Bitset$not(inplace)
```

*Arguments:*

- `inplace` whether to overwrite the current bitset.

**Method** `xor()`: to "bitwise xor" or get the symmetric difference of two bitset (keep elements in either bitset but not in their intersection).

*Usage:*

```perl
Bitset$xor(other)
```

*Arguments:*

- `other` the other bitset.

**Method** `set_difference()`: Take the set difference of this bitset with another (keep elements of this bitset which are not in other).

*Usage:*

```perl
Bitset$set_difference(other)
```
CategoricalVariable

Usage:
Bitset$set_difference(other)

Arguments:
other the other bitset.

Method sample(): sample a bitset.

Usage:
Bitset$sample(rate)

Arguments:
rate the success probability for keeping each element, can be a single value for all elements or a vector of unique probabilities for keeping each element.

Method choose(): choose k random items in the bitset

Usage:
Bitset$choose(k)

Arguments:
k the number of items in the bitset to keep. The selection of these k items from N total items in the bitset is random, and k should be chosen such that 0 \leq k \leq N.

Method copy(): returns a copy the bitset.

Usage:
Bitset$copy()

Method to_vector(): return an integer vector of the elements stored in this bitset.

Usage:
Bitset$to_vector()

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

Usage:
Bitset$clone(deep = FALSE)

Arguments:
deep Whether to make a deep clone.

CategoricalVariable CategoricalVariable Class

Description

Represents a categorical variable for an individual. This class should be used for discrete variables taking values in a finite set, such as infection, health, or behavioral state. It should be used in preference to IntegerVariable if possible because certain operations will be faster.
Methods

Public methods:

• CategoricalVariable$new()
• CategoricalVariable$get_index_of()
• CategoricalVariable$get_size_of()
• CategoricalVariable$get_categories()
• CategoricalVariable$queue_update()
• CategoricalVariable$.update()
• CategoricalVariable$clone()

Method new(): Create a new CategoricalVariable

Usage:
CategoricalVariable$new(categories, initial_values)

Arguments:
categories a character vector of possible values
initial_values a character vector of the initial value for each individual

Method get_index_of(): return a Bitset for individuals with the given values

Usage:
CategoricalVariable$get_index_of(values)

Arguments:
values the values to filter

Method get_size_of(): return the number of individuals with the given values

Usage:
CategoricalVariable$get_size_of(values)

Arguments:
values the values to filter

Method get_categories(): return a character vector of possible values. Note that the order of
the returned vector may not be the same order that was given when the variable was initialized,
due to the underlying unordered storage type.

Usage:
CategoricalVariable$get_categories()

Method queue_update(): queue an update for this variable

Usage:
CategoricalVariable$queue_update(value, index)

Arguments:
value the new value
index the indices of individuals whose value will be updated to the one specified in value.
This may be either a vector of integers or a Bitset.
Method `.update()`:

Usage:
CategoricalVariable$.update()

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

Usage:
CategoricalVariable$clone(deep = FALSE)

Arguments:
- deep  Whether to make a deep clone.

---

categorical_count_renderer_process

Render Categories

Description
Renders the number of individuals in each category.

Usage
categorical_count_renderer_process(renderer, variable, categories)

Arguments
- `renderer`  a `Render` object.
- `variable`  a `CategoricalVariable` object.
- `categories`  a character vector of categories to render.

Value
a function which can be passed as a process to `simulation_loop`. 
DoubleVariable

DoubleVariable Class

Description

Represents a continuous variable for an individual.

Methods

Public methods:

- DoubleVariable$new()
- DoubleVariable$get_values()
- DoubleVariable$get_index_of()
- DoubleVariable$get_size_of()
- DoubleVariable$queue_update()
- DoubleVariable$update()
- DoubleVariable$clone()

Method new(): Create a new DoubleVariable.

Usage:
DoubleVariable$new(initial_values)

Arguments:
initial_values a numeric vector of the initial value for each individual.

Method get_values(): get the variable values.

Usage:
DoubleVariable$get_values(index = NULL)

Arguments:
index optionally return a subset of the variable vector. If NULL, return all values; if passed a Bitset or integer vector, return values of those individuals.

Method get_index_of(): return a Bitset giving individuals whose value lies in an interval \([a, b]\).

Usage:
DoubleVariable$get_index_of(a, b)

Arguments:
a lower bound
b upper bound

Method get_size_of(): return the number of individuals whose value lies in an interval Count individuals whose value lies in an interval \([a, b]\).

Usage:
DoubleVariable$get_size_of(a, b)
Arguments:
a  lower bound
b  upper bound

Method queue_update(): Queue an update for a variable. There are 4 types of variable update:
1. Subset update: The argument index represents a subset of the variable to update. The argument values should be a vector whose length matches the size of index, which represents the new values for that subset.
2. Subset fill: The argument index represents a subset of the variable to update. The argument values should be a single number, which fills the specified subset.
3. Variable reset: The index vector is set to NULL and the argument values replaces all of the current values in the simulation. values should be a vector whose length should match the size of the population, which fills all the variable values in the population
4. Variable fill: The index vector is set to NULL and the argument values should be a single number, which fills all of the variable values in the population.

Usage:
DoubleVariable$queue_update(values, index = NULL)

Arguments:
values  a vector or scalar of values to assign at the index.
index  is the index at which to apply the change, use NULL for the fill options. If using indices, this may be either a vector of integers or a BitSet.

Method .update():

Usage:
DoubleVariable$.update()

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

Usage:
DoubleVariable$clone(deep = FALSE)

Arguments:
deep  Whether to make a deep clone.

<table>
<thead>
<tr>
<th>Event</th>
<th>Event Class</th>
</tr>
</thead>
</table>

Description

Describes a general event in the simulation.
Methods

Public methods:

- Event$new()
- Event$add_listener()
- Event$schedule()
- Event$clear_schedule()
- Event$.tick()
- Event$.process()
- Event$.process_listener()
- Event$.process_listener_cpp()
- Event$clone()

Method new(): Initialise an Event.

Usage:
Event$new()

Method add_listener(): Add an event listener.

Usage:
Event$add_listener(listener)

Arguments:
listener the function to be executed on the event, which takes a single argument giving the
time step when this event is triggered.

Method schedule(): Schedule this event to occur in the future.

Usage:
Event$schedule(delay)

Arguments:
delay the number of time steps to wait before triggering the event, can be a scalar or a vector
of values for events that should be triggered multiple times.

Method clear_schedule(): Stop a future event from triggering.

Usage:
Event$clear_schedule()

Method .tick():

Usage:
Event$.tick()

Method .process():

Usage:
Event$.process()

Method .process_listener():

Usage:
filter_bitset

Event$.process_listener(listener)

Method .process_listener_cpp():
  Usage:
  Event$.process_listener_cpp(listener)

Method clone(): The objects of this class are cloneable with this method.
  Usage:
  Event$clone(deep = FALSE)
  Arguments:
  deep Whether to make a deep clone.

filter_bitsetFilter a bitset

Description

This non-modifying function returns a new Bitset object of the same maximum size as the original but which only contains those values at the indices specified by the argument other. Indices in other may be specified either as a vector of integers or as another bitset. Please note that filtering by another bitset is not a "bitwise and" intersection, and will have the same behavior as providing an equivalent vector of integer indices.

Usage

filter_bitset(bitset, other)

Arguments

bitset the Bitset to filter
other the values to keep (may be a vector of integers or another Bitset)

fixed_probability_multinomial_process

Mutation process

Description

Simulates a two-stage process where all individuals in a given source_state sample whether to leave or not with probability rate; those who leave go to one of the destination_states with probabilities contained in the vector destination_probabilities.
infection_age_process

Usage

```python
fixed_probability_multinomial_process(
    variable,
    source_state,
    destination_states,
    rate,
    destination_probabilities
)
```

Arguments

- **variable**: a `CategoricalVariable` object.
- **source_state**: a string representing the source state.
- **destination_states**: a vector of strings representing the destination states.
- **rate**: probability of individuals in source state to leave.
- **destination_probabilities**: probability vector of destination states.

Value

a function which can be passed as a process to `simulation_loop`.

---

**infection_age_process  Infection process for age-structured models**

Description

Simulates infection for age-structured models, where individuals contact each other at a rate given by some mixing (contact) matrix. The force of infection on susceptibles in a given age class is computed as:

\[
\lambda_i = p \sum_j C_{i,j} \left( \frac{I_j}{N_j} \right)
\]

Where \(C\) is the matrix of contact rates, \(p\) is the probability of infection per contact. The per-capita probability of infection for susceptible individuals is then:

\[
1 - e^{-\lambda_i \Delta t}
\]
Usage

```r
infection_age_process(
    state,
    susceptible,
    exposed,
    infectious,
    age,
    age_bins,
    p,
    dt,
    mixing
)
```

Arguments

- `state`: a `CategoricalVariable` object.
- `susceptible`: a string representing the susceptible state (usually "S").
- `exposed`: a string representing the state new infections go to (usually "E" or "I").
- `infectious`: a string representing the infected and infectious state (usually "I").
- `age`: a `IntegerVariable` giving the age of each individual.
- `age_bins`: the total number of age bins (groups).
- `p`: the probability of infection given a contact.
- `dt`: the size of the time step (in units relative to the contact rates in `mixing`).
- `mixing`: a mixing (contact) matrix between age groups.

Value

A function which can be passed as a process to `simulation_loop`.

---

**IntegerVariable**

**IntegerVariable Class**

**Description**

Represents a integer valued variable for an individual. This class is similar to `CategoricalVariable`, but can be used for variables with unbounded ranges, or other situations where part of an individual’s state is better represented by an integer, such as household or age bin.

**Methods**

**Public methods:**

- `IntegerVariable$new()`
- `IntegerVariable$get_values()`
- `IntegerVariable$get_index_of()`
• \texttt{IntegerVariable\$get\_size\_of()}
• \texttt{IntegerVariable\$queue\_update()}
• \texttt{IntegerVariable\$.update()}
• \texttt{IntegerVariable\$clone()}

**Method** \texttt{new()}: Create a new IntegerVariable.

*Usage:*
\texttt{IntegerVariable\$new(initial\_values)}

*Arguments:*
initial\_values  a vector of the initial values for each individual

**Method** \texttt{get\_values()}: Get the variable values.

*Usage:*
\texttt{IntegerVariable\$get\_values(index = NULL)}

*Arguments:*
index  optionally return a subset of the variable vector. If NULL, return all values; if passed a \texttt{Bitset} or integer vector, return values of those individuals.

**Method** \texttt{get\_index\_of()}: Return a \texttt{Bitset} for individuals with some subset of values. Either search for indices corresponding to values in set, or for indices corresponding to values in range \([a, b]\). Either set or a and b must be provided as arguments.

*Usage:*
\texttt{IntegerVariable\$get\_index\_of(set = NULL, a = NULL, b = NULL)}

*Arguments:*
set  a vector of values (providing set means \(a, b\) are ignored)
a  lower bound
b  upper bound

**Method** \texttt{get\_size\_of()}: Return the number of individuals with some subset of values. Either search for indices corresponding to values in set, or for indices corresponding to values in range \([a, b]\). Either set or a and b must be provided as arguments.

*Usage:*
\texttt{IntegerVariable\$get\_size\_of(set = NULL, a = NULL, b = NULL)}

*Arguments:*
set  a vector of values (providing set means \(a, b\) are ignored)
a  lower bound
b  upper bound

**Method** \texttt{queue\_update()}: Queue an update for a variable. There are 4 types of variable update:

1. Subset update: The argument index represents a subset of the variable to update. The argument values should be a vector whose length matches the size of index, which represents the new values for that subset.
2. Subset fill: The argument index represents a subset of the variable to update. The argument values should be a single number, which fills the specified subset.
3. Variable reset: The index vector is set to NULL and the argument values replaces all of the current values in the simulation. values should be a vector whose length should match the size of the population, which fills all the variable values in the population.

4. Variable fill: The index vector is set to NULL and the argument values should be a single number, which fills all of the variable values in the population.

Usage:
IntegerVariable$queue_update(values, index = NULL)

Arguments:
values a vector or scalar of values to assign at the index
index is the index at which to apply the change, use NULL for the fill options. If using indices, this may be either a vector of integers or a Bitset.

Method .update():

Usage:
IntegerVariable$.update()

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

Usage:
IntegerVariable$clone(deep = FALSE)

Arguments:
dep Whether to make a deep clone.

multi_probability_bernoulli_process

Overdispersed Bernoulli process

Description
Simulates a Bernoulli process where all individuals in a given source state from sample whether or not to transition to destination state to with a individual probability specified by the DoubleVariable object rate_variable.

Usage
multi_probability_bernoulli_process(variable, from, to, rate_variable)

Arguments
variable a CategoricalVariable object.
from a string representing the source state.
to a string representing the destination state.
rate_variable DoubleVariable giving individual probability of each individual in source state to leave.

Value
a function which can be passed as a process to simulation_loop.
multi_probability_multinomial_process

*Overdispersed multinomial process*

**Description**

Simulates a two-stage process where all individuals in a given *source_state* sample whether to leave or not with an individual probability specified by the *DoubleVariable* object *rate_variable*; those who leave go to one of the *destination_states* with probabilities contained in the vector *destination_probabilities*.

**Usage**

```java
multi_probability_multinomial_process(
    variable,
    source_state,
    destination_states,
    rate_variable,
    destination_probabilities
)
```

**Arguments**

- **variable** a *CategoricalVariable* object.
- **source_state** a string representing the source state.
- **destination_states** a vector of strings representing the destination states.
- **rate_variable** *DoubleVariable* giving individual probability of each individual in source state to leave
- **destination_probabilities** probability vector of destination states.

**Value**

a function which can be passed as a process to *simulation_loop*.

**Description**

Class to render output for the simulation.
Methods

Public methods:

- `Render$new()`
- `Render$set_default()`
- `Render$render()`
- `Render$to_dataframe()`
- `Render$clone()`

Method `new()`: Initialise a renderer for the simulation, creates the default state renderers.

Usage:
`Render$new(timesteps)`

Arguments:
`timesteps` number of timesteps in the simulation.

Method `set_default()`: Set a default value for a rendered output renderers.

Usage:
`Render$set_default(name, value)`

Arguments:
`name` the variable to set a default for.
`value` the default value to set for a variable.

Method `render()`: Update the render with new simulation data.

Usage:
`Render$render(name, value, timestep)`

Arguments:
`name` the variable to render.
`value` the value to store for the variable.
`timestep` the time-step of the data point.

Method `to_dataframe()`: Return the render as a `data.frame`.

Usage:
`Render$to_dataframe()`

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

Usage:
`Render$clone(deep = FALSE)`

Arguments:
`deep` Whether to make a deep clone.
reschedule_listener  

Description

Schedules a follow-up event as the result of an event firing.

Usage

reschedule_listener(event, delay)

Arguments

event  
a TargetedEvent.

delay  
the delay until the follow-up event.

simulation_loop  

Description

Run a simulation where event listeners take precedence over processes for state changes.

Usage

simulation_loop(
    variables = list(),
    events = list(),
    processes = list(),
    timesteps
)

Arguments

variables  
a list of Variables

events  
a list of Events

processes  
a list of processes to execute on each timestep

timesteps  
the number of timesteps to simulate
Examples

populations <- 4
timesteps <- 5
state <- CategoricalVariable$new(c('S', 'I', 'R'), rep('S', population))
renderer <- Render$new(timesteps)

transition <- function(from, to, rate) {
  return(function(t) {
    from_state <- state$get_index_of(from)
    state$queue_update(
      to,
      from_state$sample(rate)
    )
  })
}

processes <- list(
  transition('S', 'I', .2),
  transition('I', 'R', .1),
  transition('R', 'S', .05),
  categorical_count_renderer_process(renderer, state, c('S', 'I', 'R'))
)

simulation_loop(variables=list(state), processes=processes, timesteps=timesteps)
renderer$to_dataframe()

---

TargetedEvent

TargetedEvent Class

Description

Describes a targeted event in the simulation. This is useful for events which are triggered for a sub-population.

Super class

individual::Event -> TargetedEvent

Methods

Public methods:
  * TargetedEvent$new()
  * TargetedEvent$schedule()
  * TargetedEvent$get_scheduled()
  * TargetedEvent$clear_schedule()
  * TargetedEvent$.process_listener()
  * TargetedEvent$.process_listener_cpp()
• `TargetedEvent$clone()`

**Method new()**: Initialise a `TargetedEvent`.

*Usage:*
`TargetedEvent$new(population_size)`

*Arguments:*
- `population_size` the size of the population.

**Method schedule()**: Schedule this event to occur in the future.

*Usage:*
`TargetedEvent$schedule(target, delay)`

*Arguments:*
- `target` the individuals to pass to the listener, this may be either a vector of integers or a `Bitset`.
- `delay` the number of time steps to wait before triggering the event, can be a scalar in which case all targeted individuals are scheduled for the same delay or a vector of values giving the delay for that individual.

**Method get_scheduled()**: Get the individuals who are scheduled as a `Bitset`.

*Usage:*
`TargetedEvent$get_scheduled()`

**Method clear_schedule()**: Stop a future event from triggering for a subset of individuals.

*Usage:*
`TargetedEvent$clear_schedule(target)`

*Arguments:*
- `target` the individuals to clear, this may be either a vector of integers or a `Bitset`.

**Method .process_listener()**:

*Usage:*
`TargetedEvent$.process_listener(listener)`

**Method .process_listener_cpp()**:

*Usage:*
`TargetedEvent$.process_listener_cpp(listener)`

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

*Usage:*
`TargetedEvent$clone(deep = FALSE)`

*Arguments:*
- `deep` Whether to make a deep clone.
**update_category_listener**

*Update category listener*

---

**Description**

Updates the category of a sub-population as the result of an event firing, to be used in the `TargetedEvent` class.

**Usage**

```
update_category_listener(variable, to)
```

**Arguments**

- `variable`  
  a `CategoricalVariable` object.
- `to`  
  a string representing the destination category.
Index

bernoulli_process, 2
Bitset, 3, 6, 8, 9, 11, 14, 15, 20

categorical_count_renderer_process, 7
CategoricalVariable, 5, 7, 12, 13, 15, 16, 21

data.frame, 17
DoubleVariable, 8, 15, 16

Event, 9

filter_bitset, 11
fixed_probability_multinomial_process, 11

individual::Event, 19
infection_age_process, 12
IntegerVariable, 5, 13, 13

multi_probability_bernoulli_process, 15
multi_probability_multinomial_process, 16

Render, 7, 16
reschedule_listener, 18

simulation_loop, 3, 7, 12, 13, 15, 16, 18

TargetedEvent, 18, 19, 21

update_category_listener, 21