# Package 'pop'

June 7, 2016

**Type** Package

**Title** A Flexible Syntax for Population Dynamic Modelling

**Version** 0.1

**Date** 2016-06-07

**Author** Nick Golding

**Maintainer** Nick Golding <nick.golding.research@gmail.com>

**Description** Population dynamic models underpin a range of analyses and applications in ecology and epidemiology. The various approaches for analysing population dynamics models (MPMs, IPMs, ODEs, POMPs, PVA) each require the model to be defined in a different way. This makes it difficult to combine different modelling approaches and data types to solve a given problem. 'pop' aims to provide a flexible and easy to use common interface for constructing population dynamic models and enabling to them to be fitted and analysed in lots of different ways.

**License** MIT + file LICENSE

**Imports** igraph, MASS

**Suggests** knitr, testthat

**LazyData** TRUE

**RoxygenNote** 5.0.1

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2016-06-07 06:26:35

## R topics documented:

---

`as.transfun` *create a transition function*

---

## Description

A utility function to enable users to create bespoke transition functions (`transfun` objects) for use in `transitions`.

## Usage

```
as.transfun(fun, param, type = c("probability", "rate", "dispersal"))
```

## Arguments

| | |
|---|---|
| `fun` | an R function describing the transition. This must take only one argument: `landscape` and return a numeric vector (see `details`). |
| `param` | a named list of the parameters of `fun` (see `details`). |
| `type` | what type of transition this function represents, a probability or a rate |

## Details

`fun` must take only one argument, `landscape`, an object of class `landscape`. `landscape` objects contain three elements which may be used in the function: `population`, a dataframe giving the number of individuals of each stage (columns) in each patch (rows); `area`; a numeric vector giving the area of each patch in square kilometres; and `features`, a dataframe containing miscellaneous features (columns) of each habitat patch (rows), such as measures of patch quality or environmental variables. See examples for an illustration of how to these objects. Parameters of the transfun should be passed to `as.transfun` as a named list. These can then be used in `fun` by accessing them from this list. Note that `param` isn't an argument to `fun`, instead it's modified directly in the function's envirnment (because *reasons*).

## Examples

```
# a very simple (and unnecessary, see ?p) transfun
fun <- function(landscape) param$prob
prob <- as.transfun(fun, param = c(prob = 0.3), type = 'probability')

# a density-dependent probability
dd_fun <- function (landscape) {
    adult_density <- population(landscape, 'adult') / area(landscape)
    param$p * exp(- adult_density/param$range)
```

```
}

dd_prob <- as.transfun(dd_fun,
                       param = list(p = 0.8,
                                    range = 10),
                       type = 'probability')
```

---

`dispersal` *dispersal transfun*

---

## Description

Create a transfun object representing a relative probability of dispersal between patches. Typically used inside a call to `transition`

## Usage

```
dispersal(value)

d(value)

is.dispersal(x)
```

## Arguments

| | |
|---|---|
| `value` | the (positive) exponential rate of decay of dispersal probabilities. Large values imply shorter range dispersal. |
| `x` | an object to be tested as a dispersal transfun object |

## Details

`d()` is a shorthand for `dispersal()`. The `transfun` object returned, when applied to a `landscape` object, produces a square symmetric matrix, with zero diagonal and off-diagonals giving the relative between patch dispersal probability. This implies that *all* individuals in the state will *try* to disperse. The fraction remaining in the patch depends on `value`. To have only some fraction try to disperse, a dispersal transfun can be multiplied by a probability transfun indicating the probability of attempting dispersal.

The relative dispersal probability is given by `exp(-d * value)`, where `d` is the Euclidean distance between the origin and destination patch.

## Examples

```
# these are equivalent
disp <- dispersal(3)
disp <- d(3)

is.dispersal(disp)
```

dynamic *dynamic objects*

## Description

creates a dynamic object, comprising multiple transition objects to define a dynamical system. dynamic objects are the core of pop, since they can be created and updated using various methods (MPMs, IPMs etc.), combined (by addition of two dynamic objects to make another) and and analysed in various ways (deterministically to obtain demographic parameters, simulated to evaluate population viability etc.)

## Usage

```
dynamic(...)

is.dynamic(x)

## S3 method for class 'dynamic'
plot(x, ...)

states(x)

## S3 method for class 'dynamic'
print(x, ...)

## S3 method for class 'dynamic'
as.matrix(x, which = c("A", "P", "F", "R"), ...)

## S3 method for class 'dynamic'
parameters(x)

## S3 replacement method for class 'dynamic'
parameters(x) <- value
```

## Arguments

| | |
|---|---|
| x | a dynamic object to print, plot, convert to a transition matrix, or an object to test as a dynamic object (for is.dynamic), |
| which | which type of matrix to build: the overall population growth matrix ('A'), the probabilistic progression matrix ('P'), the fecundity matrix ('F') or the intrinsic reproduction matrix ('R') |
| value | a nested named list of parameters within each transition matching those currently defined for x |
| ... | for dynamic(): one or more transition (or other dynamic) objects making up the dynamic. For plot() and print(): further arguments passed to or from other methods |

## Examples

```
# define transitions for a simple three-stage system (with implicit
# mortality):
stasis_egg <- tr(egg ~ egg, p(0.4))
stasis_larva <- tr(larva ~ larva, p(0.3))
stasis_adult <- tr(adult ~ adult, p(0.8))
hatching <- tr(larva ~ egg, p(0.5))
fecundity <- tr(egg ~ adult, r(3))
pupation <- tr(adult ~ larva, p(0.2))

# combine these into separate dynamics
stasis <- dynamic(stasis_egg,
                  stasis_larva,
                  stasis_adult)
growth <- dynamic(hatching,
                  pupation)
reproduction <- dynamic(fecundity)

# combine these into one dynamic (the same as listing all the transitions
# separately)
all <- dynamic(stasis, growth, reproduction)

# plot these
plot(stasis)
plot(growth)
plot(all)

# get component states
states(all)

# print method
print(all)

# convert to a transition matrix
as.matrix(all)
# extract the parameters
(param_stasis <- parameters(stasis))
(param_all <- parameters(all))

# update the parameters of these transfuns
param_stasis$stasis_egg$p <- 0.6
parameters(stasis) <- param_stasis
parameters(stasis)

param_all$fecundity$r <- 15
parameters(all) <- param_all
parameters(all)
```

---

landscape *landscape objects*

---

## Description

landscape objects represent sets of patches forming a metapopulation, storing information (such as area, population and environmental features) that may impact on the dynamic transitions occurring in each component patch. dynamic objects all have a landscape object (by default a single-patch landscape) as a an attribute which can be accessed and set via the function landscape. as.landscape is used to create landscape objects, and the functions population, area, distance and features access and set each of the elements of a landscape.

## Usage

```
landscape(dynamic)

landscape(dynamic) <- value

as.landscape(patches)

is.landscape(x)

## S3 method for class 'landscape'
print(x, ...)

area(landscape)

area(landscape) <- value

population(landscape)

population(landscape) <- value

features(landscape)

features(landscape) <- value

distance(landscape)

distance(landscape) <- value

## S3 method for class 'landscape'
x[[i]]
```

## Arguments

| | |
|---|---|
| dynamic | an object of class dynamic |
| value | an object of class landscape (for landscape(dynamic) <- value) or the value to assign to the distance, area, population, or features elements of a landscape object |
| patches | an object to turn into a landscape object. Currently this can either be a dynamic, a list or NULL (see details), though more approaches will be added in the future |

| | |
|---|---|
| x | an object to print or test as a landscape object |
| landscape | an object of class landscape |
| i | index specifying the patches to include in the subset landscape object |
| ... | further arguments passed to or from other methods. |

## Details

The accessor function landscape either returns or sets the landscape structure of the dynamic, encoded as a landscape object

patches can be a list containing the following elements: population, a dataframe giving the number of individuals of each stage (columns) within each patch (rows); area, a one-column dataframe giving the areas of the patches in square kilometres; coordinates, a dataframe giving the coordinates of the habitat patches; and features, a dataframe containing miscellaneous features (columns) of the patches (rows), such as measures of patch quality or environmental variables. Alternatively, patches = NULL, will set up a 'default' one-patch landscape with area = data.frame(area =1), coordinates = data.frame(x = 0, y = 0) and blank population and features elements. The other option is to pass a dynamic object as patches, in which case the set up will be the same as for patches = NULL except that population will be a one-row dataframe of 0s, with columns corresponding to the states in the dynamic. This is what's used when analysing a dynamic object without user-specified metapopulation structure.

the accessor functions distance, area, population and features either return or set corresponding sub-dataframes of the landscape object

## Value

an object of class landscape, essentially a dataframe containing the coordinates, area, population and features (as columns) for each patch (rows)

## Examples

```
# create a default landscape
landscape <- as.landscape(NULL)

# create a marginally more interesting one-patch landscape
landscape <- as.landscape(list(coordinates = data.frame(x = c(10, 11),
                                                        y = c(11, 12)),
                               area = data.frame(area = 10),
                               population = data.frame(adult = 10,
                                                       larva = 3,
                                                       egg = 20),
                               features = data.frame(temperature = 10)))
# print method
print(landscape)

# get and set the area
area(landscape)
area(landscape) <- area(landscape) * 2
area(landscape)
```

```
# get and set the population
population(landscape)
population(landscape) <- population(landscape) * 2
population(landscape)

# get and set the features
features(landscape)
features(landscape) <- cbind(features(landscape), rainfall = 100)
features(landscape)

# get and set the distance matrix
distance(landscape)
distance(landscape) <- sqrt(distance(landscape))
distance(landscape)

# landscapes can be subsetted to get sub-landscapes of patches with double
# braces
landscape
landscape[[1]]
landscape[[1:2]]
```

---

`parameters` *get and set parameters*

---

## Description

this documents the S3 generic functions `parameters` to extract or assign parameter values from objects in the pop package. Methods of this function are defined for various object classes, including `transfun`, `transition` and `dynamic` objects.

## Usage

```
parameters(x)

parameters(x) <- value
```

## Arguments

| | |
|---|---|
| `x` | an object from which to extract parameters, or in which to set them |
| `value` | an object to assign as the parameters of `x` |

## Details

each class-specific method will return parameters in a slightly different structure, and will require `value` to be provided in a different format (though the structures returned and required will generally be the same for all classes. See the helpfile for each class for the specific details and examples.

pop *pop: A Flexible Syntax for Population Dynamic Modelling*

## Description

Models of population dynamics underpin a range of analyses and applications in ecology and epidemiology. The various approaches for fitting and analysing these models (MPMs, IPMs, ODEs, POMPs, PVA, with and without metapopulation structure) are generally fitted using different software, each with a different interface. This makes it difficult to combine various modelling approaches and data types to solve a given problem. pop aims to provide a flexible and easy to use common interface for constructing population dynamic models and enabling to them to be fitted and analysed in various ways.

`probability` *probability transfun*

## Description

Create a transfun object representing a probability of transition between states. Typically used inside a call to `transition`

## Usage

```
probability(value)

p(value)

is.probability(x)
```

## Arguments

| | |
|---|---|
| `value` | a numeric between 0 and 1 representing a probability |
| `x` | an object to be tested as a probability transfun object |

## Details

`p()` is a shorthand for `probability()`.

## Examples

```
# these are equivalent
prob <- probability(0.2)
prob <- p(0.2)

is.probability(prob)
```

`projection` *Deterministic projection*

## Description

Project a population dynamic model in discrete time, recording the number of individuals in each state at each time point.

## Usage

```
projection(dynamic, population, timesteps = 1)

is.pop_projection(x)

## S3 method for class 'pop_projection'
plot(x, states = NULL, patches = 1, ...)
```

## Arguments

| | |
|---|---|
| `dynamic` | a population dynamic model of class `dynamic` |
| `population` | a dataframe or named vector of positive integers, giving the number of individuals in each state of `dynamic`. If a dataframe, it should have only one row (as in the examples below), or as many rows as patches in the metapopulation if a multi-patch landscape has been defined for `dynamic` (using `landscape`). If a multi-patch landscape has been defined for `dynamic`, but `population` has only one row or is a vector, this population will be duplicated for all patches in the landscape. |
| `timesteps` | a positive integer giving the number of time steps (iterations) over which to simulate the model |
| `x` | a `pop_projection` object, or an object to be tested as one |
| `states` | character vector naming the states in the `dynamic` object used to run the projection that should be plotted. By default all of them are plotted. |
| `patches` | vector of positive integers identifying the patches for which to plot the projections. By default only projections for the first patch are plotted. |
| `...` | further arguments passed to or from other methods. |

## Value

an object of class `pop_projection`

## Examples

```
# set up a three-stage model
stasis_egg <- tr(egg ~ egg, p(0.6))
stasis_larva <- tr(larva ~ larva, p(0.4))
stasis_adult <- tr(adult ~ adult, p(0.9))
```

```
hatching <- tr(larva ~ egg, p(0.35))
fecundity <- tr(egg ~ adult, r(20))
pupation <- tr(adult ~ larva, p(0.2))

pd <- dynamic(stasis_egg,
              stasis_larva,
              stasis_adult,
              hatching,
              pupation,
              fecundity)

population <- data.frame(egg = 1200, larva = 250, adult = 50)

# simulate for 50 timesteps, 30 times
proj <- projection(dynamic = pd,
                   population = population,
                   timesteps = 50)

is.pop_projection(proj)

par(mfrow = c(3, 1))
plot(proj)
```

---

`rate` *rate transfun*

---

## Description

Create a transfun object representing a rate of transition between states - e.g. an expected number of offspring generated into one state from another. Typically used inside a call to `transition`

## Usage

```
rate(value)

r(value)

is.rate(x)
```

## Arguments

| | |
|---|---|
| `value` | a numeric greater than 0 representing a rate |
| `x` | an object to be tested as a rate transfun object |

## Details

`r()` is a shorthand for `rate()`.

## Examples

```
# these are equivalent
rate <- rate(0.2)
rate <- r(0.2)

is.rate(rate)
```

---

`simulation` *Stochastic Simulation*

---

## Description

Simulate a population dynamic model in discrete time, recording the number of individuals in each state at each time point.

## Usage

```
simulation(dynamic, population, timesteps = 1, replicates = 1,
  ncores = NULL)

is.simulation(x)

## S3 method for class 'simulation'
plot(x, states = NULL, patches = 1, ...)
```

## Arguments

| | |
|---|---|
| `dynamic` | a population dynamic model of class `dynamic` |
| `population` | a dataframe or named vector of positive integers, giving the number of individuals in each state of `dynamic`. If a dataframe, it should have only one row (as in the examples below), or as many rows as patches in the metapopulation if a multi-patch landscape has been defined for `dynamic` (using `landscape`). If a multi-patch landscape has been defined for `dynamic`, but `population` has only one row or is a vector, this population will be duplicated for all patches in the landscape. |
| `timesteps` | a positive integer giving the number of time steps (iterations) over which to simulate the model |
| `replicates` | a positive integer giving the number of independent time series to simulate |
| `ncores` | an optional positive integer giving the number of cpu cores to use when running simulations. By default (when `ncores = NULL`) all cores are used (or as many as `parallel::detectCores` can find). This argument is ignored is `replicates = 1` |
| `x` | a `simulation` object, or an object to be tested as a `simulation` |
| `states` | a character vector naming the states in the `dynamic` object used to run the simulation that should be plotted. By default all of them are. |

| | |
|---|---|
| patches | vector of positive integers identifying the patches for which to plot the simulations. By default only projections for the first patch are plotted. |
| ... | further arguments passed to or from other methods. |

## Details

The order of the dynamics in the simulation is defined by the order in which the transitions were passed to dynamic. I.e. if the stasis probability of a life stage (e.g. fraction surviving and remaining in the stage) was specified before the reproduction rate, then only those staying in the state will reproduce. Conversely, if reproduction was given first, individuals will reproduce before the stasis probability is applied.

## Value

an object of class simulation

## Examples

```
# set up a three-stage model
stasis_egg <- tr(egg ~ egg, p(0.6))
stasis_larva <- tr(larva ~ larva, p(0.4))
stasis_adult <- tr(adult ~ adult, p(0.9))
hatching <- tr(larva ~ egg, p(0.35))
fecundity <- tr(egg ~ adult, r(20))
pupation <- tr(adult ~ larva, p(0.2))

pd <- dynamic(stasis_egg,
              stasis_larva,
              stasis_adult,
              hatching,
              pupation,
              fecundity)

population <- data.frame(egg = 1200, larva = 250, adult = 50)

# simulate for 50 timesteps, 30 times
sim <- simulation(dynamic = pd,
                  population = population,
                  timesteps = 50,
                  replicates = 30,
                  ncores = 1)

is.simulation(sim)
par(mfrow = c(3, 1))
plot(sim)
```

transfun *transfun objects*

## Description

utility functions for the transfun class. transfun objects are created by functions such as probability.

## Usage

```
is.transfun(x)

## S3 method for class 'transfun'
print(x, ...)

## S3 method for class 'transfun'
x * y

## S3 method for class 'transfun'
parameters(x)

## S3 replacement method for class 'transfun'
parameters(x) <- value
```

## Arguments

| | |
|---|---|
| x | a transfun object to print or an object to test as a transfun object |
| y | a transfun object to be multiplied with another with the same pathway |
| value | a named list of parameters matching those currently defined for x |
| ... | further arguments passed to or from other methods. |

## Details

multiplication of transfun objects with the same pathway results in a compound transfun object (also of class transfun). When used in a stochastic model, the two stochastic transitions are evaluated one after another. When analysed deterministically, the expectation of the compound transition function is taken as the product of the expectations of the two basis transfuns.

## Examples

```
prob <- p(0.3)
is.transfun(prob)

prob
(compound <- prob * r(4.3))

# extract the transfun parameters
(param_prob <- parameters(prob))
```

```
(param_compound <- parameters(compound))

# update the parameters of these transfuns
param_prob$p <- 0.6
parameters(prob) <- param_prob
parameters(prob)

param_compound$r <- 15
parameters(compound) <- param_compound
parameters(compound)
```

---

`transition` *transition objects*

---

## Description

creates a `transition` object, encoding a transition between two states. E.g. the probability of a seed germinating, or of an individual surviving in each time step

## Usage

```
transition(formula, transfun)

tr(formula, transfun)

is.transition(x)

## S3 method for class 'transition'
print(x, ...)

## S3 method for class 'transition'
x * y

## S3 method for class 'transition'
parameters(x)

## S3 replacement method for class 'transition'
parameters(x) <- value
```

## Arguments

| | |
|---|---|
| `formula` | a two-sided formula identifying the states between which the transition occurs |
| `transfun` | a `transfun` object quantifying the transition. |
| `x` | an object to print or test as a transition object |
| `y` | a transition object to be multiplied with another with the same pathway |
| `value` | a named list of parameters matching those currently defined for `x` |
| `...` | further arguments passed to or from other methods. |

## Details

`tr` is just a shorthand for `transition`

multiplication of transition objects with the same pathway results in a transition object whose `transfun` object is a compound of the two transfuns in the transitions. See `transfun` for more details of compound transfuns.

## Examples

```
# 50/50 chance of a larva emerging from an egg
hatching <- tr(larva ~ egg, p(0.5))

# three eggs laid per adult per time step
fecundity <- tr(egg ~ adult, r(3))

# 0.1 probability of a larva pupating into an adult
pupa <- tr(adult ~ larva, p(0.1))

# print method
print(pupa)

# make a compound transition to include a probability of laying eggs
prob_laying <- tr(egg ~ adult, p(0.6))
(recruitment <- prob_laying * fecundity)

# extract the transfun parameters
(param_pupa <- parameters(pupa))
(param_recruitment <- parameters(recruitment))

# update the parameters of these transfuns
param_pupa$p <- 0.6
parameters(pupa) <- param_pupa
parameters(pupa)

param_recruitment$r <- 15
parameters(recruitment) <- param_recruitment
parameters(recruitment)
```

# Index